SEAS: A System for Early Analysis of SoCs

Reinaldo A. Bergamaschi[1], Youngsoo Shin[1], Nagu Dhanwada[2], Subhrajit Bhattacharya[1], William E. Dougherty[2], Indira Nair[1], John Darringer[1], Sarala Paliwal[2]

[1] IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
[2] IBM EDA Laboratory, Fishkill, NY, USA

ABSTRACT

Systems-on-chip (SoC) continue to be very complex to design and verify, despite extensive component reuse. Although reusable components are pre-designed and pre-verified, when they are assembled in an SoC there is no guarantee that the whole system will behave as expected from a performance, cost and integration point of view. In many cases this is because of faulty early design decisions regarding the architecture, core selection, floorplanning, etc. This paper presents a system for early analysis of SoCs which helps designers make early design decisions regarding performance, area, timing and power; and allows them to quickly evaluate cross-domain effects, such as the effect that an architectural decision may have on the performance and chip area.

Categories and Subject Descriptors

J.6 [**Computer-aided design**]; B7.2 [**Integrated circuits**]:Design aids

General Terms

Algorithms

Keywords

Design analysis, design space exploration, performance, power, floorplanning

1. INTRODUCTION

Systems-on-chip (SoC) built from pre-designed and pre-verified intellectual property blocks (IPs, or cores) have been hailed as the most effective means of bridging the design productivity gap [1]. However, even with extensive IP reuse, SoC design remains complex and still hampered by many of the same design problems confronting traditional large application-specific integrated circuit (ASIC) design.

These common problems are usually related to decisions made early in the design process which turned out to be either innacurate or impossible to fulfill later in the design. Such mistakes include, for example, (1) pessimistic die-size estimation which resulted in a more expensive chip, (2) architectural modifications needed for performance, which complicated the final placement, routing and timing closure of the design, (3) inaccurate early floorplanning which precluded timing problems from being detected early and later incurred in lengthy timing closure. These problems are not new and many different solutions have been attempted for ASIC designs. Nevertheless, the reuse of IP cores in SoC designs opens up new algorithmic possibilities for early analysis tools which can address the above problems. In an IP-based SOC the cores are pre-designed and pre-characterized, as opposed to a custom ASIC where many of the logic blocks are yet to be designed. The characterization of the cores provides reliable data regarding number of gates, layout size, performance parameters, timing constraints, etc. This data is reliable since it is extracted from pre-designed cores, but it is not necessarily accurate for a specific design because cores may accept parameters which can change the internal logic and its values (e.g., performance, timing) may depend on its context (e.g., external loads). The accuracy of this data also depends on whether the core is provided as a soft core (VHDL, Verilog) or a hard core (GDS-II).

This paper presents a System for Early Analysis of SoCs (SEAS) comprising a set of algorithms and a methodology for front-end design and analysis of SoCs. SEAS uses IP characterization data in conjunction with tools for performance, floorplan, timing and power analyses and is able to analyze the results in an integrated manner. The ability to analyze the cross-effects among architecture, floorplan, timing and power is crucial in preventing problems later in the design cycle.

This paper is organized as follows. Section 2 gives and overview of the tool; section 3 describes the individual algorithmic components of the tool; section 4 presents the experimental results, followed by conclusions in section 5.

2. OVERVIEW

SEAS allows users to easily specify a design in a block-diagram-like description and run types of analyses that would normally be impossible to do early in the design process with acceptable accuracy. These analyses include performance, floorplanning, timing and power. SEAS can handle core-based SoC designs, where the cores are available in a library, together with characterization data and models. The types of models needed will be described in the following sections.

A typical flow through the tool could be as follows. The designer builds the block diagram for a design (for example, an Ethernet router) and runs performance analysis, which returns values below targets, i.e., the Ethernet subsystem is not able to handle the flow of packets coming in. He/she then changes the architecture by adding another core, another Ethernet controller to double the number of channels, and as a result the performance numbers become acceptable. The modified design is then fed to the floorplanner which estimates a chip die size and global routing. It is possible that as a result of the extra added core the new chip dimensions require a larger die size which makes the chip more

CODES+ISSS'03, October 1–3, 2003, Newport Beach, California, USA.

costly; or require longer interconnections which may create critical timing paths.

The main advantage of SEAS from a user point of view is the ability to describe, measure and change the specification at a very high-level of abstraction and quickly evaluate the effects in performance, area, timing and power. If the results are not satisfactory, the designer can quickly change the architecture, the floorplan, or the cores being used and run the analyses again. Figure 1 illustrates SEAS overall organization. The individual analysis algorithms are not necessarily novel, however they had to be adapted and tuned to the design representation being used (block-diagram). This tuning is critical to the accuracy of the results.

Several previous works focused on early system-level estimation of performance, area and power [2][3][4][5]. These works are different from SEAS in many ways, but primarily in the level of input description which determines how early the estimations can be obtained. In [2] and [3], the system performance is estimated, based on the estimations of its software and hardware components. Both works assume that the system specification is available and give algorithms for estimating the various system processes as implemented in hardware or software. In [4] the system power is estimated also based on the power of its hardware and software components, again assuming that a system specification is available. In [5] the system architecture and instruction set are generated automatically, and the area and timing are derived from its RTL-based hardware specification. In all these systems, the detailed system specification is required for any meaningful analysis. In SEAS this is not the case, the "early" analysis can be done before any detailed system specification is available.

3. SEAS COMPONENTS

As illustrated in Figure 1, SEAS comprises an input description similar to a block diagram and four analyses engines. Each engine has its own set of algorithms and internal model derived from the initial block diagram, and uses characterization data and models for the cores available from a core library. The following sections explain these components in detail.

3.1 SoC Representation

For an early analysis system, it is vital that the input specification be at an abstraction level easily captured by the designer in a few hours at most, and using familiar methods (e.g., schematic capture or hardware-description language). Moreover, this description should be easily modified. This immediately precludes any functional description at the register-transfer or behavioral levels,

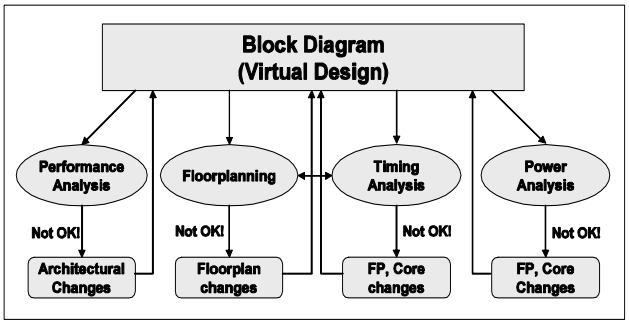

Figure 1: SEAS Overview

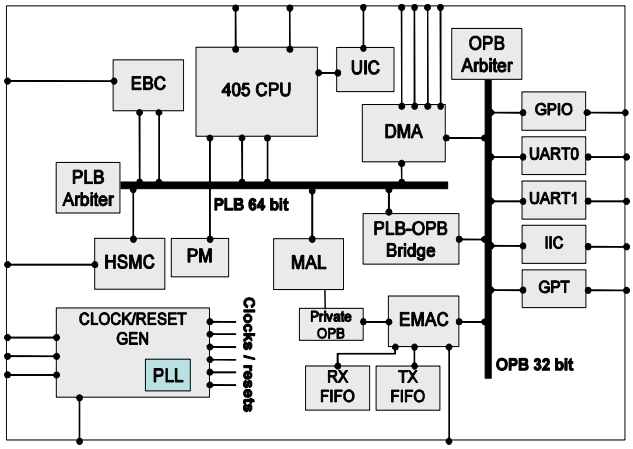

Figure 2: PowerPC 405 PBD Virtual Design

as they would take days to write up and modify. In addition, in core-based SoCs, the designer has to be familiar with the functionality and pins of the cores in order to connect and configure them properly. This again is time consuming, error-prone and inadequate for an early analysis system.

The simplest specification format that is already very familiar to designers is the *block diagram*, or a schematic-like description with blocks representing the cores and a few lines representing the major interconnections, such as clocks, buses, etc. We have developed a system called CORAL [6] for automating the design of SoCs using cores in which we have defined a synthesizable block diagram-like specification called the *Virtual Design*. In order to make this paper self-contained we add here a brief description of the Virtual Design and its properties.

The Virtual Design is a netlist specification consisting of virtual components, virtual nets and virtual pins, and it can be created using a standard schematic editor or hardware-description languages. Virtual components are abstract representations of the actual cores in a library (called real components). For example, there is a single virtual component for a family of microprocessors (e.g., the virtual PowerPC represents an abstraction of all real PowerPC cores). The virtual component contains virtual pins, and each virtual pin is an encapsulation of a subset of the real pins in the real component. For example, the PowerPC405 core has two master interfaces to the Processor Local Bus (PLB) [7], each comprising a total of 18 pins and 188 bits. In contrast, each master interface in the virtual component is represented as a single virtual pin. Virtual pins are connected using virtual nets.

Figure 2 illustrates the virtual design for the PowerPC 405 platform-based design (PBD) [8]. As the figure shows, the virtual design resembles a block diagram. Each virtual component has a small number of pins, and each pin is marked with the type of interface it belongs to. Coral [6] understands these interface types and can automatically create a virtual design based on a set of selected virtual components and some configuration parameters set by the designer.

In SEAS each virtual component is associated with 3 other models besides a real component, namely, a performance model, an area image for floorplanning and a power state-machine model. These models are stored in a library and retrieved by SEAS during analysis.

3.2 Performance Analysis

System performance is often the most important factor used for deciding the architecture of an SoC. The overall performance of a system is dependent on the raw performance of individual components and on the interaction between them. SEAS uses a simulation-based approach in order to account for dynamic effects such as bus contention, buffer overflows, etc. These effects depend on the performance of individual components, the architecture of the SoC, the software application and the workload being run on the system.

The performance behavior of a core is captured in a high-level abstract model as a set of communicating finite state machines (FSMs). The models are created by abstracting out the functionality of the core and by considering only its performance impact. Actual data computation is not modeled, just the delay aspects of it, namely, number of cycles and latency. The FSM models for the cores communicate via token passing. The arrival of a token may trigger state transitions, or events, which in turn generate new tokens.

The performance models developed in this work can achieve very high simulation speed at the expense of some accuracy. At this early analysis stage, speed is more important than accuracy, provided that results are still meaningful and useful. The core models capture the typical behavior of the core; exceptions and corner-case behaviors are ignored. Some data-dependent delays are not accurately modeled.

The effects of the software running on the system and its interactions with the architecture have to be considered as well. In SEAS, a model for the behavior of the software is created, which generates delays and bus traffic without actually performing any function. The delay values associated with software running on the processor are obtained using profiling techniques off-line. The environment is modeled in terms of data rates or traces. For example, an Ethernet channel port would be modeled by the network speed, packet rate (number of packets/sec) and the size of each packet arriving from the network.

An important factor that affects simulation speed is the granularity of atomic operations. An atomic operation is executed in a single simulation step in the performance model but may take several clock cycles in the actual hardware. A burst transfer over a bus is an example of a single atomic operation in the performance model. The actual number of cycles of such a transfer is computed by the bus performance model depending on the number of bytes being transferred and certain core parameters (e.g., allowed burst sizes).

We have developed SystemC 2.0 [9] performance model for several of the IBM CoreConnect cores, including those used by the design in Figure 2. The virtual design is automatically converted to a SystemC program which has constructor calls for all modules representing the components and interconnections in the virtual design. To enhance reusability, the models are fully parameterized with respect to speed, width, etc. More details on performance modeling can be found in [10].

3.3 Floorplanning

In order to create a virtual design floorplan which is representative of the final real design floorplan, the virtual components need to be characterized for size and shape (a process called *Area Planning*). Each virtual component has a corresponding real component core in a library. This core is available either as a hardware description (in case of soft cores) or directly as a piece of layout (for hard cores). Hard cores have their shapes and sizes fixed by the layout. For soft cores, area planning is a one time step (performed as part of library generation) that involves:

- Running logic synthesis to generate a fully technology mapped netlist
- Setting appropriate image, power terminal related blockages and generating an initial outline for the component based on an area estimation method. This method uses netlist connectivity, blockage information and wiring estimates for computing an initial area and shape for the virtual component.
- Placement and global routing
- Refining initial area and shape estimates based on actual placement and global wiring information.

The area planning step sizes the components to achieve an area utilization of 65 – 70%. This is done to account for any potential increase in area that might happen later in the design process due to chip level gate sizing and buffering for timing correction. As a result of area planning all virtual components are associated with a size and shape for a given technology image, which can then be used in the floorplan of the whole virtual design.

Pin locations also need to be considered as they are important during global routing. A pin assignment algorithm is used to fix the pin locations in the virtual components based on the locations of the real pins in the corresponding hard core (if available) or based on an initial placement of the virtual components and their interconnections (for soft cores).

The inputs to the floorplanner are: the virtual design netlist, area planning characterizations of all virtual components used in the design, and the expected number of total real IO pins. An initial chip image is selected based on the total area of the virtual components and number of real IO pins. The task of floorplanning is part automatic and part manual. Designers usually know which cores contain more critical connections, such as high-speed IOs, processors, and clock generation cores, and will manually place those cores on the chip image. Given a set of such preplaced components, an automatic simulated-annealing-based floorplanning algorithm [11] is used to place the rest of the components and produce a legal non-overlapping floorplan for the virtual design.

Based on the generated floorplan, the previously fixed image is retained or is switched to an image that is a better fit to the virtual design floorplan. The result is a die size that satisfies the area and IO requirements with acceptable levels of wire congestion. This die size and the locations of the virtual components on the floorplan are later passed to the real design as a very good starting point for the back-end placement, routing and timing closure steps.

3.4 Timing Analysis

Timing closure is a critical part of any design process and cannot be ignored even during the initial architectural analysis and performance modeling stage. In current deep submicron technologies the delay of signals traveling across the chip already approaches or exceeds the cycle time, forcing designers to pay close attention to the chip floorplan and the architectural options available. New techniques like pipelining global interconnections are becoming more prevalent as means to deal with the inability to transmit a signal across the length of a chip. This makes timing

closure decisions tightly intertwined with architecture decisions and floorplanning. The challenge lies in being able to analyze timing early in the design process, at the virtual design abstraction level.

The problem in measuring timing at the virtual design level is that the virtual pins are not associated with drivers and there are no buffers inserted in the virtual design floorplan. Although we can assign average drivers for the virtual pins, the absence of real drivers and buffers make any timing measurement less reliable.

In practice the designer has a good understanding of where the major timing problems may lie, based on the real cores assertions available from the library and on his/her own experience. Hence, the main unknowns with respect to early timing are the delays of the main global interconnections (or indirectly wire length). In SEAS, as part of floorplanning the virtual design, there is a global routing step which creates global interconnections for all virtual nets and reports the wire lengths. The wire lengths and capacitances of global interconnections are reported to the designer. Timing analysis, integrated with the floorplanner, is then run using average drivers, mainly to give the designer an approximate value for the worst case delay through the wire. Virtual wires correspond to virtual nets and not to the real nets in the real design, however the system can estimate the number of real wires per virtual net and estimate routing congestion. Moreover the virtual wire lengths and delays do represent an approximation of the real wire lengths and delays, which can be very useful for flagging critical wire delays early in the design.

3.5 Power Analysis

Power analysis at the early stage of design is also an important component of SEAS. At this early stage, the typical way to estimate power consumption uses spreadsheets based on conventional or empirical power formulas. These formulas return average power consumption values for the cores, given parameters, such as the size and type of the logic block, capacitances, switching activity, frequency and power supply. In order to explore the power design space, designers have to run these spreadsheets multiple times with different values of parameters, which is very time consuming and does not provide representative coverage of the possible execution scenarios.

In SEAS we have implemented a simulation-based approach intended to compute power for specific performance simulation scenarios. The power for each core is still computed based on spreadsheet formulas which were expanded to account for different power modes. Each core is associated with a power state machine (PSM) [12] which defines its power states and transitions and the transition delays (or how many cycles it takes to change power states).

The performance models described in section 3.2 were extended to capture the power state machine for the cores. During the performance analysis, the simulation records the active cores, i.e., involved in the execution of the transaction, as well as cores in idle or sleep states. By adding up the power values for the states which the cores transition to during simulation, we can compute the average power for the whole system for a given simulation scenario. A power management unit (PMU), which manages the power consumption of a system by controlling each core, is also modeled with parameters that can be controlled during simulation.

By combining the performance analysis simulation with power computation, SEAS can evaluate dynamic effects such as the overhead of delay and power when cores change their power states, and the effects of power management policy on performance.

3.6 Real Design

SEAS is dedicated to early analysis of SoCs. However, for the design process to complete successfully, the early analysis estimates and parameters have to be honored by the rest of the design process.

SEAS operates on the virtual design. The Coral tool [6] is capable of understanding this block-diagram-like description and automatically maps it to an RTL description for the complete SoC. This description, along with the RTL for all the actual cores used in the design, is loaded into synthesis for optimization and technology mapping. The mapped netlist is fed to a floorplanner which uses the die size and core locations estimated by SEAS. The floorplanner has to deal with all real interconnections and may adapt the floorplan accordingly. This is followed by detailed placement and routing and timing closure.

4. EXPERIMENTAL RESULTS

In order to validate the SEAS approach we devised an experiment to evaluate the following: (1) how easy is it to create and change a design using the virtual design representation, (2) how useful are the early analyses proposed, (3) can the effects of architectural changes be estimated across multiple domains, (4) can the lower-level tools effectively honor the early analysis decisions, and (5) can the early analyses results be correlated with post-physical design results.

The design used in this experiment is shown in Figure 2. It is a platform-based design developed around the PowerPC 405 CPU, the CoreConnect Bus architecture and IBM's Blue Logic core library. It contains an Ethernet subsystem represented by the Ethernet controller (EMAC), the Media Access Layer (MAL) core, and receive and transmit FIFOs. It also contains a high-speed memory controller (HSMC), an external bus controller (EBC), DMA controller, interrupt controller and various peripherals including 2 UARTs, 1 IIC and 1 timer. The cores are all connected to either the high-speed Processor Local Bus (PLB), or the On-chip Peripheral Bus (OPB). The experiment consists of evaluating this design for Ethernet packet processing purposes. Performance analysis will be used for measuring the system throughput and CPU utilization, after which the architecture will be changed by adding a second Ethernet controller and the performance analysis repeated. The floorplan for both designs will be generated and die sizes estimated, along with wire length and power information.

The first part of the experiment consisted of creating the virtual design representation and running performance analysis, floorplanning, timing and power analysis. Since all virtual components are available in a component library, creating a virtual design is a matter of selecting the desired components and interconnecting the pins, which can be done very quickly (e.g., a couple of hours) using a schematic editor, a Tcl script, or directly using VHDL or Verilog [6].

The cores involved in packet processing are the EMAC, MAL, PLB Arbiter, CPU, and HSMC. The packets arrive from the network to the EMAC input and are received into its receive buffer. The MAL works as a dedicated DMA and transfers the packet through the PLB, to the memory controller and finally into

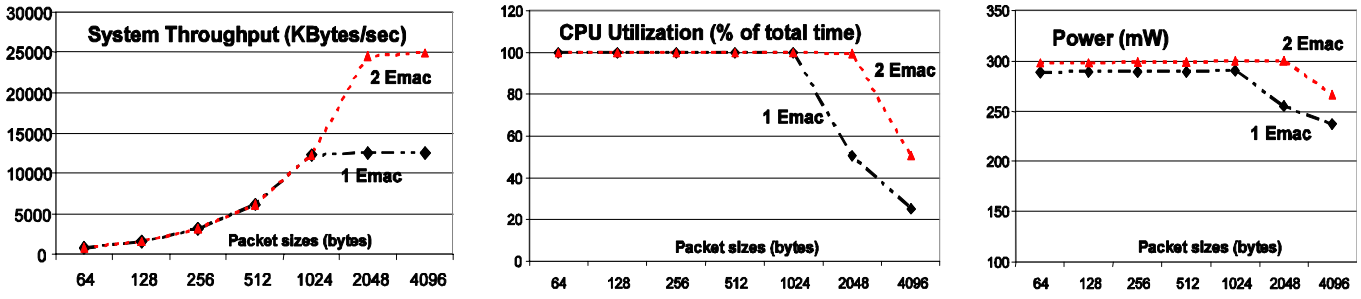

Figure 3: Performance and power analysis results for 1-EMAC and 2-EMAC designs

an external memory. The time it takes for receiving a packet into memory depends on the data rate, the size of the packet, the capacity of the MAL (size of burst transfer, number of bursts needed per packet) and some constant delays associated with the EMAC and HSMC. After the packet is received in memory, the CPU processes it by reading the header, computing a new address and writing back a new header. In this example, it is assumed that this CPU header processing is constant and does not depend on the size of the packet. This CPU time is measured off-line by profiling techniques. The packet is then read by the MAL and transmitted, through the EMAC, back to the network.

The system throughput and CPU utilization are shown in Figure 4 for various packet sizes. Throughput is defined as the number of Kbytes per second transmitted by the system. This depends on the number of packets, size of the packet, processing capacity, and it is limited by the maximum channel capacity, that is, the maximum number of bits/sec that the EMAC can transmit. CPU utilization is a measure of how much time the CPU is busy vs. idle. With small packets and 1 EMAC, the CPU is 100% busy and throughput increases with packet size, up to the maximum allowed by the channel. In this example the EMAC is limited to 100 Mbits/sec. Above a certain packet size, throughput remains capped at 100Mbits/sec, and the CPU starts becoming idle as the time to receive a packet becomes larger than the CPU processing time.

In order to increase throughput beyond 100Mbits/sec, the only alternative is to add one or more extra EMACs to the design. Adding one extra EMAC doubles the maximum throughput to 200Mbits/sec which allows for higher packet rates and bigger packets to be processed.

A new version of the design was created by adding an extra EMAC to the first virtual design. Both EMACs are connected to the MAL, which arbitrates (in a simple round-robin scheme) and alternates between the two EMACs for transferring their packets to memory. As soon as a first packet P1 is received (through EMAC-1 and MAL), the CPU can start processing it. The two emacs operate concurrently, hence depending on the packet size, a second packet P2 will be received and available to the CPU after P1 is received, but before the CPU completes processing P1. Therefore, once the CPU is done with P1, it is readily available to process P2 and there will be no CPU idle time, or 100% utilization. As the packet size increases, it will take longer to receive a packet, and P2 may not be available when the CPU completes processing P1, thus the CPU may be underutilized even in the 2 EMAC design. Throughput and utilization for the 2 EMAC design are shown in Figure 4.

In practice, the system may be doing something else besides packet processing (e.g., computing statistics and printing a report through the UART), which requires some CPU utilization. In this case the goal will not be to keep the CPU 100% busy with packet processing, but perhaps make sure that the target throughput is achieved with 80% CPU utilization, thus leaving some CPU capacity for other processing. This example demonstrates the ability in SEAS to change the architecture to meet the requirements and quickly verify it using performance analysis.

Power analysis is run with the performance simulation. Figure 4 shows the power consumed by the system during packet processing for the two virtual designs. It can be seen that power does not increase significantly in the 2-EMAC case, which is expected since most of the power is dissipated by the CPU when active. It also shows that when the CPU becomes partly idle the power decreases accordingly. This simulation assumes that the EMAC, MAL and CPU will be active when in use, and idle otherwise, and all other cores are in sleep mode.

Given these two architectural design points, they now need to be evaluated for size and timing. We generated the floorplan for both virtual designs and estimated their required die sizes. Based on floorplan area alone, the 1 EMAC version fit into a 5.57x5.57mm image, whereas the 2 EMAC version needed the next larger size, a 6.05x6.05mm image. However, this design was actually pin-limited, that is, the required number of pins (437) for the 1-EMAC version exceed the maximum number of pins in the 5.57mm image, hence the 6.05x6.05mm image had to be used, also for the 1-EMAC version. Both floorplans are shown in Figure 5. SEAS benefit in this case was to show that the higher performance design (2 EMACs) could fit in the same die size, with the same silicon cost.

In order to complete the design process we submitted the first design (1 EMAC) through the rest of the design flow, that is, RTL generation, logic synthesis, floorplan, detailed placement and routing and timing closure (repowering, buffer insertion). The core locations and areas in the virtual design floorplan were fixed in the final floorplan. Detailed placement was not allowed to modify core locations, but it could alter the aspect ratio of the area for each core in order to suit specific layout requirements for hard macros. The final layout, shown in Figure 5, was able to meet the timing constraints. As expected the final layout is very similar to the virtual design floorplan since the locations of the cores were fixed. The fact that the timing constraints could be met indicates that the early virtual design floorplan was indeed realistic enough to be used in the real design.

Comparing the timing of individual paths between the virtual design and the real design proved less accurate, mainly because the virtual design has no concept of driver strength and buffers, which are essential for accurate timing. Instead of comparing

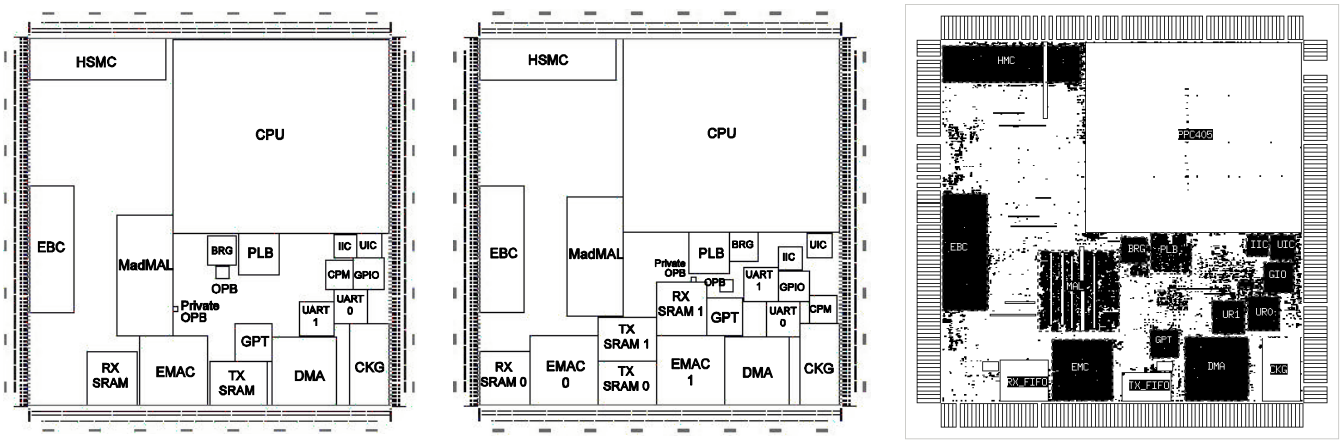

Figure 4: Virtual design floorplans and corresponding real layout

timings on individual paths, we compared the wire length of a virtual net (in the virtual floorplan, after global routing) and the average wire length for all corresponding real nets in the final real layout. The results, given in Table 1, show a good correlation between wire lengths estimated on the virtual design and the actual wire lengths in the final layout, in most cases. The two cases were the differences are over 40% are due to differences in pin locations between the virtual design layout and the final layout. Designers can look at the wire lengths for the major interconnections in the virtual design and determine if they may be too long for timing purposes.

We also compared this final design with an independently generated version of the same design produced using a traditional RTL-based design methodology: the die-size was the same and the timing constraints were met in both designs, although the actual most critical paths were different due to different layouts.

Table 1: Wire lengths of global interconnections in the virtual floorplan and in the real detailed layout (um)

Nets	Virtual FP	Detailed Layout	% Diff
PLB to DMA	1131	1420	20.4%
DMA to PLB	1131	1994	43.3%
PLB to ICU/CPU	1852	2115	12.4%
ICU/CPU to PLB	1852	2151	13.9%
PLB to DCU/CPU	1244	1459	14.7%
DCU/CPU to PLB	1244	1547	19.6%
PLB to EBC	2448	2727	10.2%
EBC to PLB	2448	2939	16.7%
PLB to MAL	1102	763	-44.4%
MAL to PLB	1102	1169	5.8%

5. CONCLUSIONS

Early design decisions, if misguided, can have a significant impact on the ability to meet performance, area, timing and power constraints on SoCs. This paper presented a system which helps designers make these early decisions. The advantages of the approach include: (1) a simple block-diagram-like notation for design specification which allows the designer to create and modify the design quickly, (2) integrated analyses algorithms for performance, floorplan, timing and power, which allow the designer to change the architecture, the core selection or the floorplan of the design and quickly evaluate the effect on other domains. Results have shown that estimations based on our approach are accurate enough to guide early design decisions as well as used by lower-level tools.

6. REFERENCES

[1] "International Roadmap for Semiconductors, 2001 Edition", available from http://public.itrs.net.

[2] J. Grode and J. Madsen, "A Unified Component Modeling Approach for Performance Estimation in Hardware/Software Codesign", *Proceedings of the 24th Euromicro Conference,* August 1998, pp.65 -69.

[3] A. Allara, C. Brandolese, W. Forniciari, F. Salice, D. Sciuto, "System-level Performance Estimation Strategy for Sw and Hw", *Proceedigns of the International Conference on Computer Design (ICCD '98)*, Oct. 1998, pp.48 -53.

[4] M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno, "Cosimulation-Based Power Estimation for System-on-Chip Design", *IEEE Transactions on Very Large Scale Integration Systems*, v. 10, no.3, June 2002, pp.253-266.

[5] G. Hadjiyiannis, P. Russo, S. Devadas, "A methodology for accurate performance evaluation in architecture exploration", *Proceedings of the 36th Design Automation Conference*, June 1999, pp.927 -932.

[6] R. Bergamaschi, S. Bhattacharya, R. Wagner, C. Fellenz, W. R. Lee, F. White, M. Muhlada and J-M. Daveau, "Automating the Design of SoCs Using Cores", *IEEE Design & Test of Computers*, v.18, no.5, Sept/Oct. 2001, p.32-45.

[7] "The CoreConnect™ Bus Architecture", white paper, available from http://www.ibm.com/chips/products/coreconnect/.

[8] "IBM Platform-Based Design Kit", described in http://www.ibm.com/chips/products/asics/methodology/design_kit.html.

[9] T. Grotker, S. Liao, G. Martin, and S. Swan , "System Design with SystemC", Kluwer Academic Publishers, 2002.

[10] J. Darringer, R. Bergamaschi, S. Bhattacharya, D. Brand, A. Herkersdorf, J. Morrell, I. Nair, P. Sagmeister, Y. Shin, "Early Analysis Tools for System-on-a-Chip Design", *IBM Journal of Research and Development*, v.46, no.6, November 2001, 691-707.

[11] J. Sayah et. al., "Design Planning for High-Performance ASICs", *IBM Journal of Research and Development*, v.40, no.3, 1996, 431-452.

[12] L. Benini, R. Hodgson, and P. Siege, "System-level power estimation and optimization"', in *Proceedings of the* International *Symposium on Low Power Electronics and Design (ISLPD)*, 173-178, ACM, August 1998.