Using TLM for Exploring Bus-based SoC Communication Architectures

Sudeep Pasricha, Nikil Dutt
Center for Embedded Computer Systems
University of California, Irvine, CA
{sudeep, dutt}@cecs.uci.edu

Mohamed Ben-Romdhane
Conexant Systems Inc.
Newport Beach, CA
m.benromdhane@conexant.com

Abstract

*As billion transistor System-on-chips (SoC) become commonplace and design complexity continues to increase, designers are faced with the daunting task of meeting escalating design requirements in shrinking time-to-market windows, and have begun using an IP-based SoC design methodology that permits reuse of key SoC functional components. Since the communication architectures connecting components in these SoC designs significantly impact system performance, it is imperative that designers explore the communication design space efficiently, quickly and early in the design flow. Transaction Level Modeling (**TLM**) is an emerging abstraction that facilitates early exploration of SoC architectures. This paper outlines a typical IP-based SoC design flow, and presents the Cycle Count Accurate at Transaction Boundaries (**CCATB**) modeling abstraction which is a fast, efficient and flexible approach for exploring bus-based communication architectures in SoC designs. The CCATB models not only take less time to model but are also faster to simulate than existing modeling abstractions for communication architecture exploration such as pin-accurate BCA (PA-BCA) and transaction based BCA (T-BCA). Experimental results on several industrial SoC subsystem case studies show that CCATB models are faster than PA-BCA by as much as 120% on average and by 67% on average when compared to T-BCA, demonstrating the advantages of CCATB-based TLM abstraction for exploring bus-based SoC communication architectures.*

1. Introduction

System-on-chip (SoC) designers today are faced with incredible complexity in the light of billion transistor designs that have already become a reality [1-2] and ever increasing numbers of components (processors, memories, peripherals, custom hardware) being integrated on a single chip. The onslaught of digital convergence has resulted in requirements for SoC designs which can support more and more functionality (e.g. cell phones with built in MP3 players, digital cameras, AM/FM radios, portable gaming support and PDA functionality) even as the design cycle time keeps shrinking rapidly due to market pressures. Designers today thus have to cope with a large design complexity versus designer productivity gap (Figure 1) which continues to widen with each passing year [3].

1.1. IP-based Design

In addition to their inherent complexity, SoC designs today also have to support the added burden of being flexible and quickly adaptable to meet changing needs of increasingly fickle customers. To meet these challenges, Intellectual Property (IP) based design and reuse [4-6] has been proposed and now widely accepted as an effective way to decrease time-to-market and improve designer productivity. Designing parameterized IP cores [7-8] with a well defined interface [4] allows cores to be quickly customized and integrated into multiple design projects. The VSIA [9] and OCP-IP [10] standards have produced specifications which dictate core interfaces to promote IP reuse and speed up integration. It should be noted that there is an initial investment involved in creating reusable components and this inevitably lengthens the design time for the first project. However, productivity in subsequent designs can then be enhanced manifold as a consequence, since the reused cores do not need to be designed or verified again.

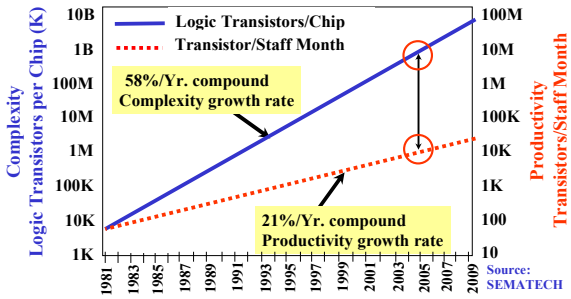

Figure 1. Design complexity versus designer productivity gap

1.2. Raising the Modeling Abstraction

Another extremely important means of increasing designer productivity has been to move towards higher levels of modeling abstractions called the system level [11]. System level models enable a unified approach for system hardware and software development. These models, usually written in high level languages such as C/C++ [12], SystemC [13] or SpecC [14] give an estimate of the system characteristics early in the design flow, well before committing to RTL development. Since these models are faster to develop and simulate than RTL models, designers can explore and verify

the system and make design decisions early in the design flow, which improves quality and reduces design time.

1.3. Modern SoC Design Flow

The concepts of IP based design-reuse and raising the modeling abstraction level, have helped redefine the modern SoC design flow, which is shown in Figure 2. After the product specifications are received from the customer (Stage 1), designers select the algorithms and perform optimizations tailored towards the particular requirements (Stage 2), and create a functional specification, which is usually an implementation and timing independent model in C/C++ that captures system functionality (Stage 3). Designers then perform hardware-software partitioning and allocate functionality to software and hardware components (Stage 4), to create an architectural model (Stage 5). Next, the communication protocol to be used between components is selected, communication channels are defined and arbitration schemes to resolve contention on shared channels selected (Stage 6) to arrive at the communication model (Stage 7). Finally, the behavior inside components and in the channels is scheduled at cycle boundaries (Stage 8) to create a cycle-accurate implementation model (Stage 9), which is either in RTL or translated to RTL and then passed to a standard back-end synthesis flow. It should be noted there are several variations of this flow that are used today, as some designers either split or merge different stages in the flow depending on design complexity, experience and the design cycle time. When the system architecture is generated from the system functionality and reaches the implementation stage by gradual addition of detail to the models shown in Figure 2, we have a top-down [14] design flow. In contrast, a meet-in-the-middle [11] design flow maps the system functionality on a predefined system architecture, instead of generating the architecture from the functionality. This approach is also referred to as platform-based design [11][15]. A third design flow, called bottom-up [6] focuses on wrapper generation and component reuse.

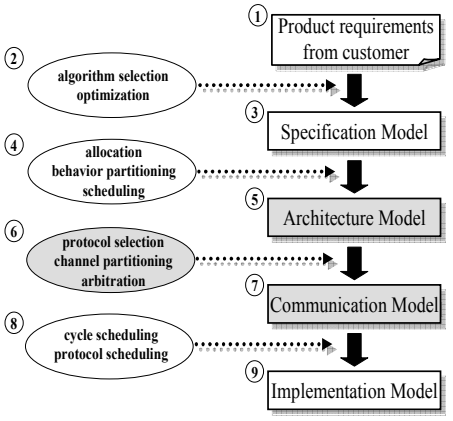

Figure 2. SoC Design Flow

Regardless of the system design flow used, supporting on-chip communication poses a major challenge for designers [16] as the number of components integrated on a single chip increases. This inter-component, on-chip communication is often in the critical path and a common source of performance bottlenecks in modern cutting-edge SoC designs [17-18]. Taking this into consideration, our emphasis in this paper will be on the fast exploration of the communication architecture space to quickly detect and eliminate these performance bottlenecks. We will focus on a section of the design flow encompassing Stages 5-7 in Figure 2, where designers generally explore the communication space.

1.4. SoC Communication Architectures

Communication architectures can be broadly classified as either being bus-based or network-on-chip (NoC) based. Bus based architectures [19-22] have been well studied and standard architectures such as AMBA [19], CoreConnect [20] and STBus [21] are widely used in current SoC designs. In contrast, NoC-based architectures [23-25] are a more recent development in the SoC domain and research in the area has just recently been gaining momentum. NoC based architectures claim to support larger bandwidths and offer better scalability, but suffer from larger area overhead, network contention delays and complex interface design issues [26]. Because of these drawbacks, the ability of bus crossbar configurations [27-28] to handle large bandwidths, and the enormous investment required to reeducate designers and port existing IPs to a new communication architecture paradigm, we believe that bus based architectures will remain the dominant communication architectures for the next few years. In the future, the transition from bus-based to NoC architectures could be eased with the use of hybrid bus-NoC architectures, as proposed in [42].

Exploration and synthesis are the two key components of bus-based communication architecture design. There is already a substantial body of work of in the area of bus architecture synthesis [43-48]. Our focus in this paper will be on the exploration phase, which involves modeling and analyzing communication flows in a SoC design to determine a suitable bus-based communication architecture for the design. The selection of a communication architecture in a design flow generally occurs after the designer has performed hardware-software partitioning and architecture mapping (Stage 6, Figure 2). The selection is complicated by the plethora of choices [19-22] available to the designer. Even after selecting a communication architecture, configuring it to meet performance requirements presents another challenge. Bus-based communication architectures such as AMBA [19] have several parameters which can be configured to improve performance: bus topology, data bus width, bus speeds, arbitration protocols, DMA burst lengths and buffer sizes have significant impact on system performance and must be considered by designers during exploration. Therefore it becomes essential for any meaningful exploration effort to comprehensively capture the bus architecture and be able to simulate the effects of changing configurable parameters at a system level [29].

In this paper, we first survey popular modeling abstractions used for exploring communication architectures, before presenting the Cycle Count Accurate at Transaction Boundaries (**CCATB**) modeling abstraction [41] for on-chip communication space exploration. We position CCATB as a

substitute for the communication model in the SoC design flow shown in Figure 2. Our abstraction level speeds up simulation performance, while maintaining cycle count accuracy. To underline the effectiveness of our approach, we present an exploration case study involving an industrial strength SoC design in the multimedia application domain. We also compare simulation performance and modeling effort for CCATB with the existing PA-BCA and T-BCA models (discussed in the next section) and analyze the scalability of these approaches with design complexity.

2. Current Trend in Modeling Abstractions used for Communication Architecture Exploration

Communication architecture exploration can be performed at several different levels of abstraction. Until a few years ago, designers would often explore designs at the implementation model (or RTL) level. While this was possible designs were relatively simple, exploring today's complex SoC designs at the RTL level is an intimidating prospect. Not only is the RTL simulation speed too slow to allow adequate coverage of the large design space in modern SoC designs, but making small changes in the design can require considerable re-engineering effort due to the highly complex nature of these systems. To overcome these limitations, system designers have been forced to raise the level of abstraction of these models. Figure 3 shows the frequently used modeling abstraction levels for communication space exploration, usually captured with high level languages such as C/C++ [12]. These high level models give an early estimate of the system characteristics before committing to RTL development. In Cycle Accurate (CA) models [30-31], system components and the bus architecture are captured at a cycle and pin accurate level. While these models are extremely accurate, they are too time-consuming to model and only provide a moderate speedup over RTL models. Pin-Accurate Bus Cycle Accurate (PA-BCA) models [32] capture the system at a higher abstraction level than CA models. Behavior inside components need not be scheduled at every cycle boundary, which allows rapid system prototyping and considerable simulation speedup over RTL. The component interface and the bus are still modeled at a cycle and pin accurate level, which enables accurate communication space exploration. However, with the increasing role of embedded software and rising design complexity, even the simulation speedup gained with PA-BCA models is not enough. More recent research approaches [33-36] have focused on using concepts found in the Transaction Level Modeling (TLM) [37] domain to speed up BCA model simulation with Transaction based BCA (T-BCA) models. We will first elaborate on TLM models before describing T-BCA models and these approaches.

Transaction Level Models [37] (Figure 3) are very high level bit-accurate models of a system with specifics of the bus protocol replaced by a generic protocol-independent bus (or *channel*), and where communication takes place when components call *read()* and *write()* methods provided by the channel interface. Since detailed timing and signal-accuracy is omitted in TLM, they are extremely fast to simulate. These models can be used to gain a very high level estimate of data traffic between components for a system level communication exploration effort. Early work with TLM established SystemC 2.0 [13][37] as the modeling language of choice for the approach. In [37] we described how TLM can be used for early system prototyping, functional verification and embedded software development.

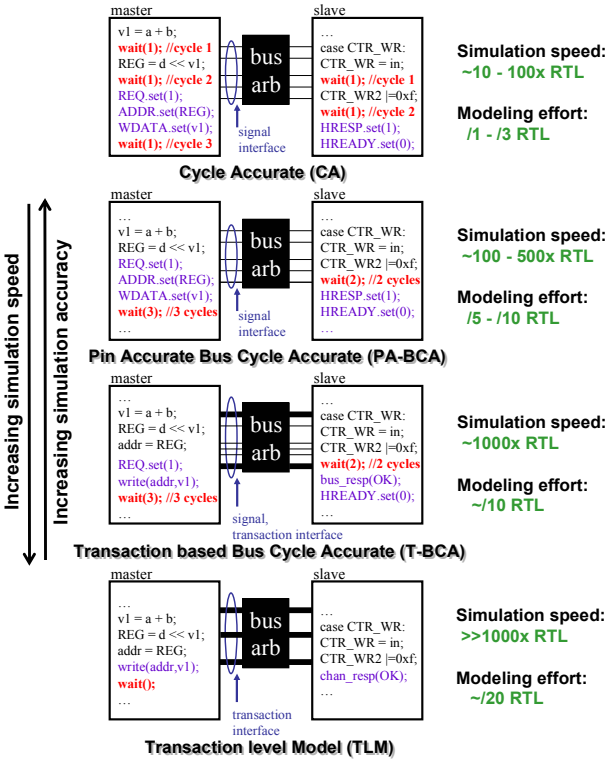

Figure 3. Modeling Abstractions for Communication Architecture Exploration

Recent research efforts [33-36] have focused on adapting TLM concepts to speed up communication architecture exploration with Transaction based BCA (T-BCA) models (Figure 3). These models make use of the read/write function call interface, optionally with a few signals to maintain bus cycle accuracy. The simpler interface reduces modeling effort and the function call semantics result in faster simulation speeds. In [33], function calls are used instead of slower signal semantics to describe models of AMBA [19] and CoreConnect [20] bus architectures at a high abstraction level. However, the resulting models are not detailed enough for accurate communication exploration. In [34], a similar attempt is made to model AMBA [19] using function calls for reads/writes on the bus, but the use of certain bus signals and SystemC *clocked threads* [13] slows down simulation speed. Similarly, in [35], data transfers in AMBA [19] are modeled using read/write transactions but low level handshaking semantics are used in the models which need not be explicitly modeled to preserve cycle accuracy and end up degrading simulation performance. Recently, ARM released the AHB Cycle-Level Interface Specification [36] which provides the definition and compliance requirements

for modeling AHB at a cycle-accurate level in SystemC. Function calls are used to replace all bus signals at the interface between components and the bus. Although using function calls speeds up simulation, there is a lot of opportunity for improving simulation speed even further by reducing the number of calls while maintaining cycle accuracy, as we show later in this paper.

3. Fast Communication Architecture Exploration

To enable fast exploration of the communication design space, we introduced a novel modeling abstraction level which is 'cycle accurate' when viewed at 'transaction boundaries'. For this reason we call our model **Cycle Count Accurate at Transaction Boundaries (CCATB)** [41]. A transaction, in this context, refers to a read or write operation issued by a master to a slave, that can either be a single data word or a multiple data burst transfer. Transactions at the CCATB level are similar to transactions at the TLM level [37] except that we additionally pass bus protocol specific control and timing information. Unlike BCA models, we do not maintain accuracy at every cycle boundary. Instead, we raise the modeling abstraction and maintain cycle count accuracy at transaction boundaries i.e., the number of bus cycles that elapse at the end of a transaction is the same when compared to cycles elapsed in a detailed cycle/pin accurate system model. A similar concept can be found in [38] where *Observable Time Windows* were defined and used for verifying results of high level synthesis. We maintain overall cycle count accuracy needed to gather statistics for accurate communication space exploration, while optimizing the models for faster simulation. Our approach essentially trades off intra-transaction visibility to gain simulation speedup.

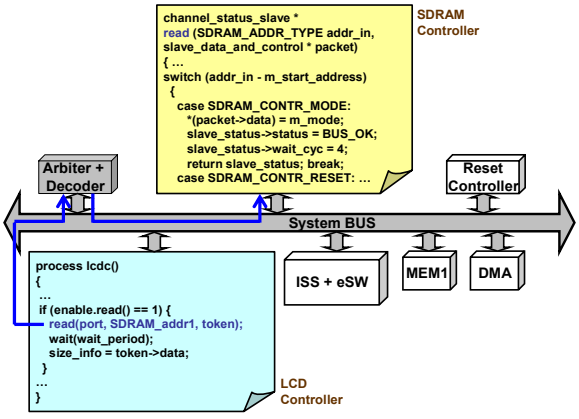

Figure 4. CCATB Transaction Example

3.1. Component Model Characteristics

We chose SystemC 2.0 [13][37] to capture designs at the CCATB abstraction level, as it provides a rich set of primitives for system modeling. Busses in CCATB are modeled by extending the generic TLM channel [37] to include bus architecture specific timing and protocol details. Arbiter and decoder modules are integrated with this channel model. Computation blocks (masters and slaves) are modeled at the behavioral abstraction level, just like TLM models in [37]. Masters are active blocks with (possibly) several computation threads and ports to interface with busses. Figure 4 shows the interface used by the master (LCD Controller) to communicate with a slave (SDRAM). In the figure, *port* specifies the port to send the read/write request on (since a master may be connected to multiple busses). *addr* is the address of the slave to send the transaction to. *token* is a structure that contains pointers to data and control. Slaves are passive entities, activated only when triggered by the arbiter on a request from the master, and have a register/memory map to handle read/write requests. The arbiter calls read() and write() functions implemented in the slave, as shown for the SDRAM controller in the figure. For more details, refer to [41].

3.2. Maintaining Cycle Count Accuracy

Figures 5 illustrates how CCATB maintains cycle count accuracy at transaction boundaries for different call sequences of the AMBA 2.0 [19] protocol. In the figure, after a master (M1) requests access (HBUSREQ_M1) to the bus for a write burst of length four, another master (M2) requests bus access (HBUSREQ_M2) for a write burst. While there is no delay at the master or the slave end for the first write burst, there is delay in generating the data at the master end for master M2, which is indicated by the BUSY status on the HRESP[1:0] lines. In the CCATB model, the arbiter accounts for the request (REQ), arbitration (ARB), burst length (BURST_LEN) and pipeline setup (PPL) delays for the first transaction and increments simulation time. For the subsequent transaction by master M2, the request has already been registered at the arbiter and no arbitration is required, so there is no REQ or ARB delay. Since transfers are pipelined, there is also no pipeline startup (PPL) delay like in the case of master M1. There is however delay which is dependent on the burst length (BURST_LEN) and the busy cycles (BUSY) which is accounted for by the arbiter. Thus the total time for transactions to complete in the CCATB and the CA models is the same.

3.3. Simulation Speedup using CCATB

The CCATB modeling abstraction essentially trades off intra-transaction visibility to gain simulation speedup. Since we are not concerned with maintaining visible cycle accuracy at each cycle boundary, we can perform certain optimizations in the model. We handle all delays in the system in a single module - the bus component. This prevents unnecessary invocation and activation of simulation component threads on every cycle. By clustering together the delays in the system in one module, we can exploit the opportunity for increasing simulation time in chunks which are larger than a single cycle, thus drastically speeding up simulation. For an implementation of the CCATB simulation semantics which describe in detail how speedup is obtained, refer to [40].

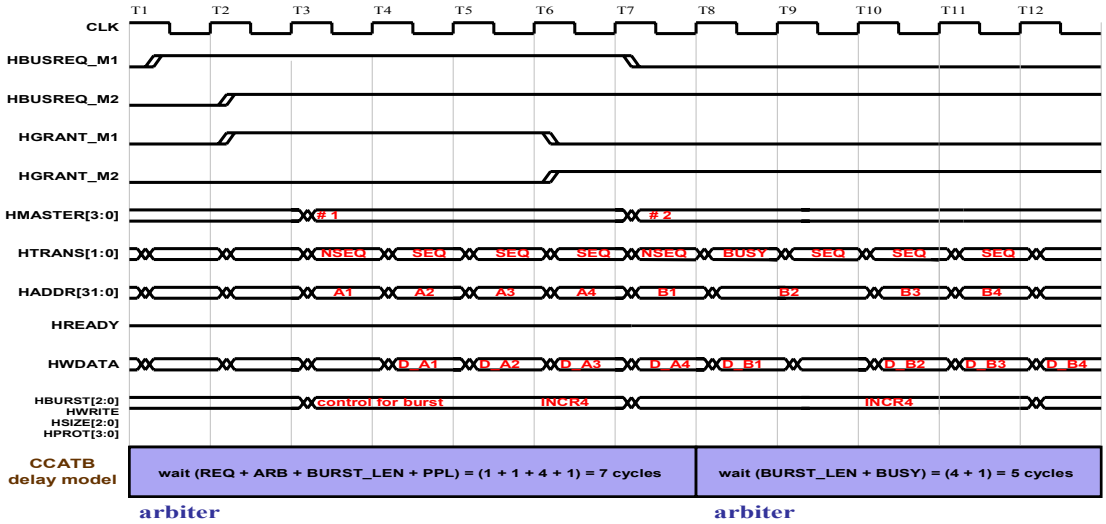

Figure 5. Reference AMBA2 call sequence

4. Case Study : Multimedia SoC Subsystem

To demonstrate the effectiveness of exploration with CCATB, we performed several case studies with industrial strength SoC subsystems. We will briefly present results from an exploration case study of a consumer multimedia SoC subsystem which performs audio and video encoding. For this case study, we assume that the choice of bus architecture has already been made and we explore different configurations of the bus architecture. For more exploration case studies involving selection and comparison of different bus architectures, refer to [41].

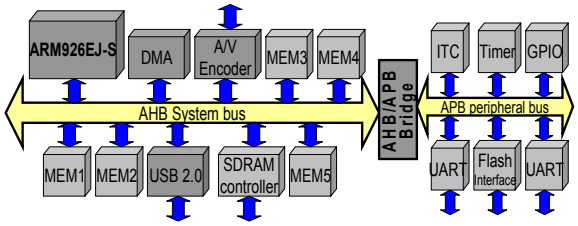

Figure 6. SoC Multimedia Subsystem

Table 1. Execution cycle count (in millions)

Arch.	Arbitration Scheme				
	RR	TDMA1	TDMA2	SP1	SP2
Arch1	27.24	24.65	25.06	25.72	26.49
Arch2	24.98	23.86	23.03	23.52	23.44
Arch3	24.73	23.74	22.96	23.11	23.05
Arch4	22.02	21.79	21.65	21.18	21.26

Figure 6 shows this system, built around the AMBA 2.0 bus architecture [19]. We consider a scenario where the designer wishes to extend the functionality of the encoder system to add support for audio/video decoding and an additional AVLink interface for streaming data. The final architecture must also meet peak bandwidth constraints for the USB component (480Mbps) and the AVLink controller interface (768Mbps). To explore the effects of changing communication architecture topology and arbitration protocols on system performance, we modeled the SoC platform at the CCATB level and simulated a test program for several interesting combinations of topology and arbitration strategies.

5. Simulation And Modeling Effort Comparison

We now present a comparison of the modeling effort and simulation performance for pin accurate BCA (PA-BCA), transaction based BCA (T-BCA) and our CCATB models. For the purpose of this study we chose the SoC platform shown in Figure 7. This platform is built around the AMBA 2.0 communication architecture and has an ARM926 processor ISS model with a test program running on it which initializes different components and then regulates data flow to and from the external interfaces such as USB, switch, external memory controller (EMC) and the SDRAM controller.

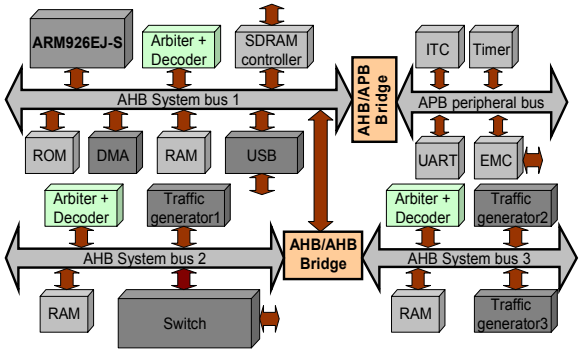

Figure 7. SoC platform

For the T-BCA model we chose the approach from [36]. Our goal was to compare not only the simulation speeds but also to ascertain how the speed changed with system complexity. We first compared speedup for a 'lightweight' system comprising of just 2 traffic generator masters along with peripherals used by these masters, such as the RAM and the EMC. We gradually increased system complexity by adding more masters and their slave peripherals. Figure 8 shows the simulation speed comparison with increasing design complexity.

Note the steep drop in simulation speed when the third master was added – this is due to the detailed non-native SystemC model of the ARM926 processor which considerably slowed down simulation. In contrast, the simulation speed was not affected as much when the DMA controller was added as the fourth master. This was because the DMA controller transferred data in multiple word bursts which can be handled very efficiently by the transaction based T-BCA and CCATB models. The CCATB particularly handles burst mode simulation very effectively and consequently has the least degradation in performance out of the three models. Subsequent steps added the USB switch and another traffic generator which put considerable communication traffic and computation load on the system, resulting in a reduction in simulation speed. Overall, the CCATB abstraction level outperforms the other two models. Table 3 gives the average speedup of the CCATB over the PA-BCA and T-BCA models. We note that on average, CCATB is faster than T-BCA by 67% and even faster than PA-BCA models by 120%.

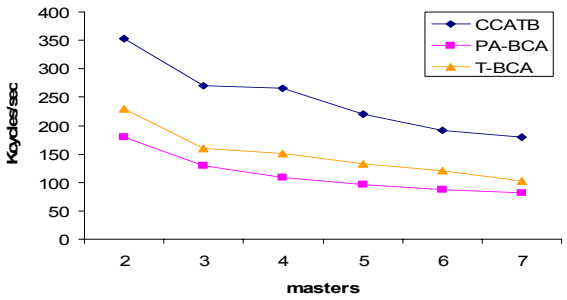

Figure 8. Simulation Speed Comparison

Table 2. Comparison of speed and modeling effort

Model Abstraction	Average CCATB speedup (x times)	Modeling Effort
CCATB	1	~3 days
T-BCA	1.67	~4 days
PA-BCA	2.2	~1.5 wks

Table 2 also shows the time taken to model the communication architecture at the three different abstraction levels by a designer familiar with AMBA 2.0. While the time taken to capture the communication architecture and model the interfaces took just 3 days for the CCATB model, it took a day more for the transaction based BCA, primarily due to the additional modeling effort to maintain accuracy at cycle boundaries for the bus system. It took almost 1.5 weeks to capture the PA-BCA model. Synchronizing and handling the numerous signals and design verification were the major contributors for the additional design effort in these models. In summary, we found that CCATB models were faster to simulate and needed less modeling effort compared to T-BCA and PA-BCA models.

6. Conclusion and Future Work

The demanding requirements of modern SoC designs are forcing fundamental changes in the way SoC systems are designed. The only way to design billion transistor chips when faced with shrinking design cycle times is to allow design teams to reuse pre-designed components, and to raise the level of modeling abstraction to the system level. Since the communication architecture which connects the components in a SoC design significantly impacts system performance, it becomes essential to explore the communication architecture space early in the design flow (e.g., at the transaction level) so that bottlenecks can be avoided and performance constraints met. In this paper we presented the *Cycle Count Accurate at Transaction Boundaries* (**CCATB**) modeling abstraction which is a fast, efficient and flexible approach for exploring the vast communication space for shared-bus architectures in SoC designs. Our model enables plug-and-play exploration of various facets of the communication space, allowing master, slave and bus IPs to be easily replaced with their architecture variants, and quickly estimating the impact on system performance. We have successfully applied our approach for exploring several industrial strength SoC subsystems. One such case study from the multimedia domains is briefly presented in this paper. We also showed that the CCATB models not only take less time to model, but are also faster to simulate than pin-accurate BCA (PA-BCA) models by as much as 120% on average and transaction based BCA (T-BCA) models by 67% on average. We have already made use of the fast CCATB models in the automated synthesis of communication architectures in [48]. Our ongoing work is focusing on automatic refinement of CCATB models from high level architectural (TLM) models; and interface refinement from CCATB down to the CA implementation models and subsequent RTL generation. Our future work will focus on understanding and exploring hybrid bus-NoC architectures [42] in detail.

7. Acknowledgements

This research was partially supported by grants from Conexant Systems Inc., UC Micro (03-029) and NSF grants CCR 0203813 and CCR 0205712.

References

[1] S. Naffziger, T. Grutkowski, B. Stackhouse, "The Implementation of a 2-core Multi-Threaded Itanium® Family Processor", *In Proc of ISSCC,* 2005

[2] D. Burger, J. R. Goodman, “Billion-Transistor Architectures: There and Back Again”, *In IEEE Computer,* March 2004

[3] The International Technology Roadmap for Semiconductors, *SIA*, 1999

[4] J. Rowson, A. L. Sangiovanni-Vincentelli, “Interface-based Design”, *In Proc of DAC*, 1997

[5] R. Bergamaschi, W. R. Lee, “Designing Systems-on-Chip Using Cores“, *In Proc of DAC,* 2000

[6] W. Cesário et al, “Component-Based Design Approach for Multicore SoCs,” *In Proc. of DAC*, 2002.

[7] T. Givargis, F. Vahid, “Parameterized System Design”, *In Proc of CODES*, 2000

[8] Mohamed Ben-Romdhane et al. “Quick-Turnaround ASIC Design in VHDL: Core-Based Behavioral Synthesis” *Kluwer Academic Publishers*, 1996

[9] Virtual Socket Interface Alliance Component Interface Standard (OCB 2 1.0), *VSI Alliance*, 2000

[10] Open Core Protocol International Partnership (OCP-IP). OCP datasheet, http://www.ocpip.org

[11] K. Keutzer et al. “System-level design: Orthogonalization of concerns and platform-based design,” In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Dec. 2000

[12] “System-on-Chip Specification and Modeling Using C++: Challenges and Opportunities”, In IEEE D&T May/ June 2001

[13] T. Grötker, S. Liao, G. Martin, S. Swan. “System Design with SystemC”. *Kluwer Academic Publishers*, 2002

[14] D. Gajski et al., “SpecC: Specification Language and Methodology”, *Kluwer Academic Publishers*, 2000

[15] L. P. Carloni et al. “The Art and Science of Integrated Systems Design”, *In Proc of the 28th European Solid-State Circuits Conference*, 2002

[16] M. Sgroi et. al, “Addressing the System-on-Chip Interconnect woes through Communication-Based Design,” *In Proc. of DAC*, June 2001

[17] J. A Davis, J. D Meindl, “Is interconnect the weak link?” *In Circuit and Device Magazine. pp. 30-36*, March 1998

[18] R. Otten, P. Stravers, “Challenges in physical chip design”, *In Proc of ICCAD, pp. 84-91*, Nov. 2000

[19] ARM Specification Rev 2.0, *ARM Ltd*., 1999

[20] IBM CoreConnect. http://www.chips.ibm.com/products/powerpc/cores

[21] STMicroelectrinics STBus Specification, http://mcu.st.com/inchtml-pages-STBus_intro.html

[22] Sonics uNetworks Technical Overview, *A21-1, Sonics Inc*, 2000

[23] W. J. Dally, B. Towles, “Route packets, not wires: On-chip interconnection networks”, *In Proc of DAC,* 2001

[24] L. Benini, G. D. Micheli, “Network on-chips”, In *IEEE Computer vol. 1 pp 70-78,* Jan 2002

[25] A.Jantsch, H.Tenhunen, “Networks on Chip”, *Kluwer Academic Publishers*, 2003.

[26] P. Guerrier, A. Greiner, “A Generic Architecture for On-Chip Packet-Switched Interconnections”, *In Proc of DATE* 2000

[27] B. Mathewson, J. Morris , “Matrix holds key to on-chip Interconnect”, *In EEdesign, available at http://www.eedesign.com/article/showArticle.jhtml?articleId =16503702*, October 2001

[28] S. Murali, G. De Micheli, “An Application-Specific Design Methodology for STbus Crossbar Generation”, *In Proc of DATE,* 2005

[29] M. Loghi et al. “Analyzing On-Chip Communication in a MPSoC Environment”, *In Proc of DATE,* 2004

[30] Joon-Seo Yim et al. “A C-Based RTL Design Verification Methodology for Complex Microprocessor”, *In Proc of DAC,* 1997

[31] H. Jang et al., “High-Level System Modeling and Architecture Exploration with SystemC on a Network SoC: S3C2510 Case Study”, *In Proc of DATE,* 2004

[32] Luc Séméria, Abhijit Ghosh, "Methodology for Hardware/ Software Co-verification in C/C++", *In Proc of ASP-DAC,* 2000

[33] X. Zhu , S. Malik, “A hierarchical modeling framework for on-chip communication architectures”, *In Proc of ICCAD,* 2002

[34] M. Caldari et al “Transaction-Level Models for AMBA Bus Architecture Using SystemC 2.0”, *In Proc of DATE,* 2003

[35] O. Ogawa et al. “A Practical Approach for Bus Architecture Optimization at Transaction Level”, *In Proc of DATE 2003*

[36] AHB CLI Specification http://www.arm.com/armtech/ahbcli

[37] S. Pasricha, “Transaction Level Modeling of SoC with SystemC 2.0”, *In Synopsys User Group Conference (SNUG)*, 2002

[38] R. A. Bergamaschi, S. Raje, “Observable Time Windows: Verifying the Results of High-Level Synthesis”, *In Proc. of DATE*, 1996

[39] AMBA AXI Specification www.arm.com/armtech/AXI

[40] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Fast Exploration of Bus-based On-chip Communication Architectures", *In Proc of CODES+ISSS*, 2004

[41] S. Pasricha, N. Dutt, M. Ben-Romdhane, “Extending the Transaction Level Modeling Approach for Fast Communication Architecture Exploration", *In Proc of DAC,* 2004

[42] T. Dumitras, S. Kerner, R. Marculescu, “Enabling On-Chip Diversity Through Architectural Communication Design”, *In Proc. of ASP-DAC*, 2004

[43] S. Narayan and D. Gajski, “Synthesis of system level bus interfaces”, *In Proc. of DATE*, 1994

[44] M. Gasteier, M. Glesner “Bus-based communication synthesis on system level”, *In ACM TODAES*, January 1999

[45] M. Drinic et al. “Latency-guided on-chip bus network design”, *In Proc. of ICCAD* 2000

[46] A. Pinto et al “Constraint-driven communication synthesis”, In *Proc. of DAC* 2002

[47] K. K. Ryu, V. J. Mooney III “Automated Bus Generation for Multiprocessor SoC Design”, *In Proc. of DATE* 2003

[48] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Automated Throughput-driven Synthesis of Bus-based Communication Architectures", *In Proc of ASP-DAC,* 2005