The Pennsylvania State University

The Graduate School

SYSTEM LEVEL POWER AND RELIABILITY MODELING

A Thesis in

Computer Science and Engineering

by

Ing-Chao Lin

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

August 2007

The thesis of Ing-Chao Lin was reviewed and approved* by the following:

Vijaykrishnan Narayanan
Professor of Computer Science and Engineering
Thesis Advisor, Chair of Committee

Mary Jane Irwin
Distinguished Professor of Computer Science and Engineering
Evan Pugh Professor, A. Robert Noll Chair of Engineering

Yuan Xie
Assistant Professor of Computer Science and Engineering

W. Kenneth Jerkins
Professor of Electrical Engineering

Nagu Dhanwada
Senior R&D Engineer and Team Lead, IBM
Special Member

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

This thesis provides system level modeling for power, reliability, and device degradation. In the system level power modeling, we use transaction level modeling. Transaction level modeling (TLM) represents the communications of IP cores as transactions and provides higher simulation speed than lower level of abstraction. We construct a hierarchical power modeling tree and augment the transaction level models with power estimation functions. We demonstrate the power estimation methodology on PCI Express transaction level models, and create various scenarios and validate the methodology on IBM CoreConnect platform. We also present experimental results to validate the accuracy and speed of our approach.

In the system level reliability modeling, we propose a transaction-based error susceptibility model for a bus-based System-on-Chip system. This reliability model provides a detailed analysis of different kinds of errors and the susceptibility of such systems to such errors on various components that comprise the bus. We inject single and multi-bit error during the execution of various transactions and examine the effect of the errors. Experimental results demonstrate error susceptibility of signals are similar across the benchmarks. Such transaction-based analysis helps us to develop an effective prediction methodology to predict the effect of a single and multi-bit error on any application running on a bus-based architecture. We demonstrate that our transaction-based prediction scheme works with an average accuracy of 91% over all the benchmarks when compared with the actual simulation results.

In the system level modeling for device degradation, we explore how Negative Bias Temperature Instability (NBTI) and Hot Carrier Effects (HCE) cause device degradation in the system. We discuss the tool we developed: a HCE and NBTI Incorporated Tool for ASICs (HANITA), for the complete analysis of circuit degradation. The tool analyzes the degradation impact on bus systems and the vulnerability of buses to such circuit degradation. We propose a hardware-based mechanism to detect the timing degradation and we further propose a PROactive BUS (PROBUS) architecture that dynamically adapts to retain the system functionality even after the system timing degrades.

Table of Contents

List of Figures

List of Tables

Acknowledgments

This dissertation has not been an individual effort. I got help from many people, and without them, I could not have completed this. I am grateful to my thesis advisor, Dr. Vijaykrishnan Narayanan. He provided guidance and constructive comments on this dissertation. His diligence and creativity also inspired me to do my best. I am also grateful to Dr. Mary Jane Irwin. Her perspective and valuable advice allowed me to look beyond the details of my research to see the bigger picture. I also would like to thank Dr. Yuan Xie and Dr. W. Kenneth Jerkins for their helpful suggestions during my Ph.D. research. Outside Penn State, I would like to thank Dr. Nagu Dhanwada, my mentor during my six-month co-op with IBM. I gained industrial and research experience while collaborating with him at IBM and became more confident in my research.

In addition, I am grateful to all my fellow colleagues in the Microsystem Design Laboratory in the Department of Computer Science and Engineering. I would especially like to thank Suresh Srinivasan and Balaji Vaidyanathan. Part of my work involves their collaboration. I also want to thank Yuh-Fang, Feihui, Raj, Aman, and other colleagues for their patience and kindness.

Finally, I am indebted to my daughters, Robyn and Ann, who were both born during the pursuit of my Ph.D. Even though it has not always been easy to raise them while completing this thesis, their presence in my life provided much needed relief from the tension of graduate life. Also, I would like to thank my parents and parents-in-law for their support. Without their help, I would not have had time to complete this dissertation. Most important of all, I would like to express my deepest gratitude to my wife, Hsueh-Wen Chow. Her patience and encouragement motivated me to continue my studies. I always knew I could count on her even in my darkest hours.

Thank you everyone for making this dissertation possible.

Chapter 1

Introduction

As CMOS technology advances, the feature size of transistors shrinks and more transistors are integrated into a single chip. With more transistors on a chip, the design complexity increases exponentially. In the meanwhile, designers apply advanced techniques such as dual supply voltage, dual threshold voltage, and voltage island to reduce power consumption or provide higher performance. These design techniques further elevate the design complexity. Figure 1.1 shows the famous design productivity gap published by the Semiconductor Industry Association. With the aid of Electronic Design Automation tools, designers' productivity increases at a rate of 28%. However, because the system complexity increases at a rate of 58%, the gap between designer's productivity and design complexity is actually growing. Current design methodologies are limited in that they are unable to significantly boost the designer's productivity. In order to bridge the gap between the designer's productivity and the system's complexity, we must raise the level of abstraction from register-transfer level to system level.

Compared to system level design, modeling and simulation at the lower levels, such as

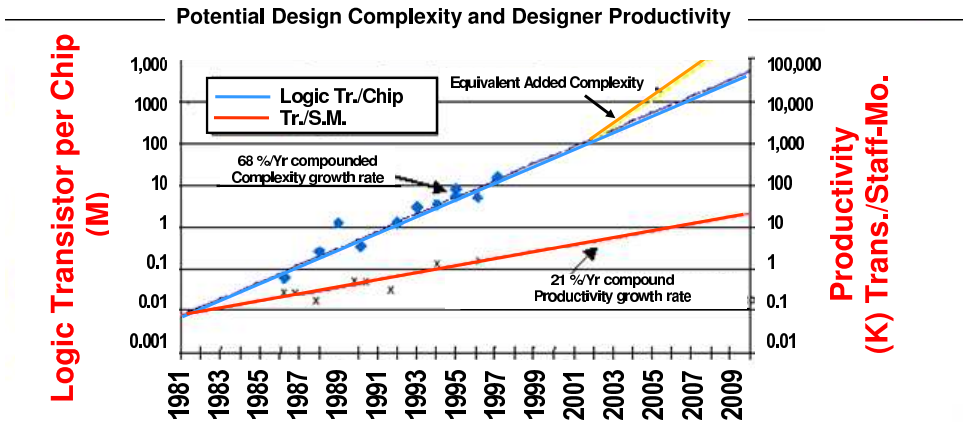

Figure 1.1. Design Productivity Gap [1]

the register-transfer level or gate level, are more complex and time-consuming. Simulation is also too slow for extensive system-wide exploration because of too much hardware detail at these levels. Power optimizations at the system level are more effective than lower level ones. Register-transfer level optimizations typically reduce power by only about 20% [7] Achieving maximum power reduction is difficult at the lower levels because of too many unnecessary hardware details. As a result, simulations are unable to find global optimizations for power reductions. In contrast, the system level design environment can achieve 10 to 20 times the power reduction by modifying the system architectures, such as memory and processors, as well as software algorithms [7]

The growing complexity of chip designs has increased design time and has made any iterative changes in the design costly. The high manufacturing cost further makes design error almost unacceptable. To achieve first-time right designs and meet time-to-market goals, it is imperative to address design issues at the system level and make informed design decisions early in the design process. Therefore, system level design has come to play a critical role in developing complex hardware and system-on-chip (SoC) systems, and the purpose of this thesis is to analyze, model, and overcome power, reliability, and

device degradation problems at the system level.

Power consumption is a growing concern as IC fabrication technology approaches the sub-100nm era. Several factors are responsible for this effect. The leakage current of a transistor, which is the main contributor of static power, increases exponentially as the channel length and gate oxide thickness reduce. Although dynamic power per transistor reduces when the transistor's size is reduced, the total chip power still grows because of increasing design sizes and transistor density. Moreover, dynamic power does not reduce at the expected rate because supply voltage does not scale proportionally to transistor size.

Many problems arise as IC chips consume more power. For example, Figure 1.2 shows the trend of cooling cost vs. thermal dissipation in a microprocessor. The cost to remove heat is directly proportional to thermal dissipation and costs about $3 per watt when power consumption is over 60 watts [2]. This power dissipation also increases packaging costs and decreases system reliability [8]. Therefore, power has emerged as an first-class design constraint.

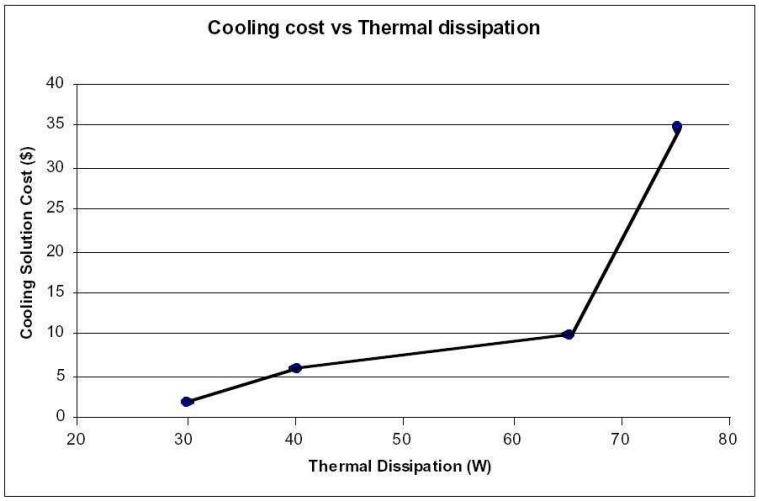

Figure 1.2. The relation between cooling cost and thermal dissipation [2]

Like power, system reliability is becoming an aggravating problem because of technology scaling. Technology scaling leads to smaller transistor size, less supply voltage, decreased noise margin, increased interconnect density, and faster clock rates [9] [10]. Circuits are exposed to more capacitive and inductive cross-talk, power supply noise, leakage noise, process variations, and other noise sources. For these reasons, both transient faults and permanent failures are all significantly accumulated. Studies have reported that the microprocessor failure rate at 65nm is 3.16 times higher than that at 180nm [8].

Currently, device degradation is not a major contributor to system unreliability. However, degradation problems caused by Negative Bias Temperature Instability (NBTI) and Hot Carrier Effect (HCE) will be more problematic and they will be the dominant causes for system degradation in the near future [11]. The NBTI phenomenon is observed in PMOS transistors when they experience stress under negative gate voltage at an elevated temperature (e.g.,$V_{gs} = -V_{dd}$). Consequently, this effect can increase the PMOS transistor's threshold voltage and reduce the absolute I_{on} current of PMOS devices, thus lowering the circuit speed. This threshold voltage increase leads to a reduced temporal performance and causes reliability issues and potential device failure. Similar to NBTI, Hot Carrier Effects (HCE), which cause electrons to be trapped in the oxide, are the main cause of NMOS transistor degradation.

The thesis will refer to System-on-chip (SoC) as a system. It proposes to estimate, model, and overcome power, reliability, and device degradation problems in the system-on-chip (SoC) at the system level, focusing on a bus-based system. In the following section, we will present our contribution about how to address and solve these problems

at the system level.

1.1 Thesis Contribution

Chapter 2 presents a power estimation methodology for transaction level models (TLMs) for SoC and implements this methodology in PCI Express TLMs. Transaction level [3][12] is a high level of abstraction above register-transfer level. This level models communication between components as channels, and transactions as function calls that exchange high level data structure through channels These functions such as bus_read() and bus_write() also provide synchronization between components. Transaction level in this thesis is at a level of abstraction equivalent to Programmer's view with timing (PV+T) level as described in [13]. This is the highest level of abstraction which can include some amount of micro-architectural detail.

Most related works for transaction level modeling focus on performance modeling, and few are done for power modeling. In this thesis, we propose a power modeling methodology for transaction level models, and demonstrates this power estimation methodology in PCI Express IP cores. We construct transaction level models and a hierarchical power modeling tree for PCI Express, and augment the transaction level models with power estimation functions. To perform power estimation and architecture exploration, we create various scenarios for both PCI Express and IBM CoreConnect SystemC TLMs. The methodology is also validated on IBM CoreConnect TLM platform.

Chapter 3 proposes a transaction-based error susceptibility for bus-based SoC reliability modeling and presents a prediction model for the system error susceptibility. This

chapter explores the fatal and deadlock error susceptibility of control and address signals in a bus-based SoC at the transaction level. We provide a detailed analysis of the types and consequences of different errors that may occur due to single-bit errors on a bus-based system during different transactions. These errors are characterized by a generic and effective transaction-based error characterization scheme for bus architectures in SoCs. Such a scheme is generic and effective, and can be extended to other systems to quickly estimate system error susceptibility. The critical measure of each signal provides an opportunity to prioritize the employment of any error correction schemes on the system, which is quite critical with the shrinking power and area budgets. We also propose a system level error prediction model to predict the probability of any single-bit or multi-bit error effect of with an accuracy of 92% on an average.

Chapter 4 focuses on device degradation of SoC bus architecture and proposes techniques to neutralize the negative impacts of degradation and improve system reliability. We look into the device degradation due to two different physical phenomena, namely Negative Bias Temperature Instability (NBTI) and Hot Carrier Effects (HCE), and present a HCE And NBTI Incorporated Tool for ASICs (HANITA) for complete analysis of the degradation of circuits over a period of time. We demonstrate a comprehensive insight into the impact of such degradation in typical shared bus-based systems based on commercial bus architectures. The degradation itself is studied on the individual components of the on-chip buses, and the tool automate the degradation analysis of each device. These device degradation are back-annotated to compute the overall system degradation over a period of time. As a result, the tool is able to analyze the degradation impact on the whole bus systems and the vulnerability of buses to such circuit degra-

dation. In addition, we propose a hardware-based dynamic scheme to detect the timing degradation. A PROactive BUS (PROBUS) architecture is presented that dynamically adapts to retain the system functionality even after the system timing degrades.

Chapter 5 summaries each work and provides recommendation for future work.

Chapter 2

Transaction Level Power Modeling

As billions of transistors are integrated into a single chip, the design complexity increases exponentially. The demands for higher speed, lower power, and lower cost chip design aggravate the design complexity. In the meanwhile, the increasingly competitive market and high manufacturing cost demands shorter time-to-market and correct designs in the first time. All of these demand a new system design approach that raises the level of abstraction and provides early modeling and verification. Transaction Level Modeling (TLM) [12] is a high level approach to model digital systems, where details of inter-module communication are separated from details of communication architecture implementation. This level of abstraction is increasingly used for System-on-Chip (SoC) architecture analysis and early embedded software development. In this chapter, we first introduce transaction level modeling and PCI Express. Next, We propose power estimation methodology for TLM, implement this methodology on PCI Express, and demonstrate its effectiveness.

2.1 Transaction Level Modeling

With the advent of advanced CMOS process technology and shrinking transistor feature sizes, more and more transistors are integrated into a single chip to provide more functions. In the same way, the design complexity increases exponentially. New methods are required to improve designers' ability to handle the design complexity and also meet the time-to-market goal. Of most well-known methodology, two methods are most popular: design reuse and raising the level of abstraction. Design reuse has been widely used for SoC design [14]. For example, IP reuse [15, 16] that SoC designers adopt IP cores that are already designed and verified to reduce the developing time for the new design. Several on-chip communication protocol such as IBM CoreConnect [17] and ARM AMBA [18] are created to provide an uniform interface between IP cores. This method, however, does not work well when new and special design are needed. The simulation of the whole system can still be very time consuming given complicated system designs; therefore, architecture exploration and system level optimization are still difficult to achieve.

Another methodology to improve designer's ability to handle complicated systems is to raise the level of abstraction. For example, the abstraction level has been raised from transistor level to gate level in the past decades to register-transfer level that is widely used in the latest ASIC and SoC design methodology. Compared to transistor and gate level, modeling at register-transfer level provides faster simulation speed. This level, however, is still not adequate to provide fast speed for modern complicated systems and again, the trend is to raise the level of abstraction to a higher level: transaction level

model.

2.1.1 Definition of Transaction Level Modeling

Grötker et al. [12] defines TLM as "a high level approach to modeling digital systems where details of communication among modules are separated from the details of the implementation of the functional units or the communication architecture." Communication methods, such as bus or FIFO, are modeled as channels that provide functional interfaces to the connecting modules. Transactions occur when modules make blocking or nonblocking function calls to the interface. The blocking functions mean the next function cannot start until the previous function is finished. In contrast, nonblocking functions can be issued simultaneously, and the return statuses of the nonblocking functions are examined by the calling modules. These functions provide a synchronization mechanism between modules at the transaction level and ignore the details of handshaking or protocol implementations. A correct amount of time is associated with each atomic function call and added up to the total simulation time, which is faster than timing at every clock. In summary, transaction level modeling focuses more on the functionality of the data transfer–what data are transfered and the destination of data transfer– rather than the actual hardware implementation, such as how the transfer protocol is implemented. This makes transaction level modeling efficient and faster than the lower level simulation and a promising solution for modern, complex SoC design.

Even though transaction level modeling (TLM) is widely discussed among industries and the academic community as an approach that can handle the complexity of SoC, no universal terminology is accepted by SoC designers. In addition, there is still dis-

agreement on the number of abstraction defined in this level. The term, 'transaction level', actually represents a continuum of abstraction levels, not any specific level. These levels vary in the degree of functional or temporal details. Gajski and Cai [3] categorized TLMs into several models based on the timing accuracy of the computation and communication in their SpecC [19]. Both computation and communication can be either un-timed (no timing information is provided), approximate-timed (approximate timing information is provided), or cycle-timed (accurate cycle information is provided). Figure 2.1 shows various TLM models defined in [3]. Shown by A in Figure 2.1, the highest

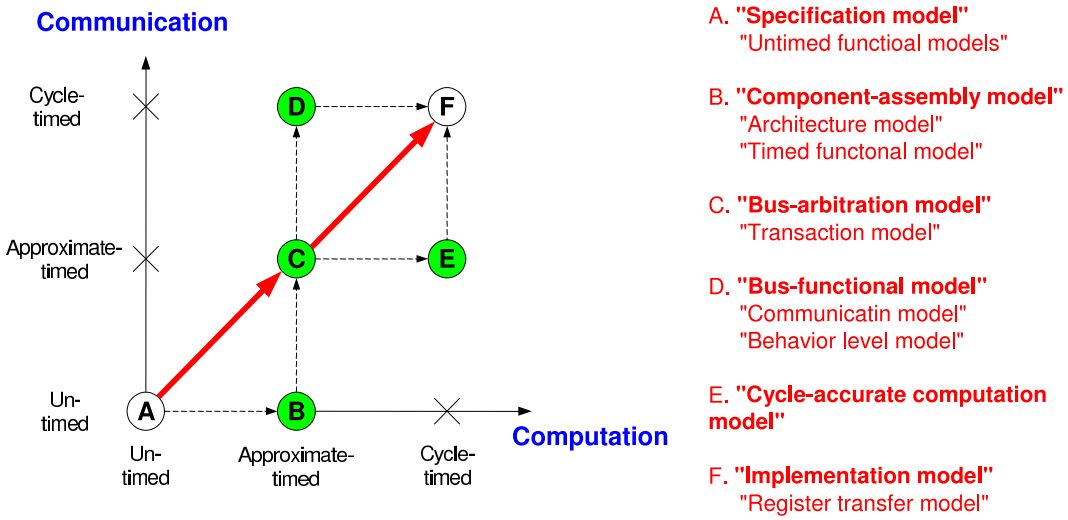

Figure 2.1. System modeling graph for TLM [3]

level is the specification model that contains no timing information on either computation or communication. The component-assembly model, shown by B, is the specification model associated with approximate timing information on computation. The component-assembly model becomes the bus-arbitration model, shown by C, if its communication is also associated with approximate timing information. When communication is modeled as cycle-accurate and computation is modeled as cycle-approximate, a bus-functional

model is built, shown by D. On the other side, if a model has cycle-accurate computation and cycle-approximate communication, it is a cycle-accurate computation model, shown by E. If both computation and computation is cycle-accurate, it is a implementation model or register-transfer model, shown by F. These definitions provide a clearer guideline and consistency on modeling timing information for TLMs.

Another more widely accepted definition of transaction level modeling is proposed by Open SystemC Initiative [4, 13]. The levels of abstraction above register-transfer level (RTL) include Algorithmic (ALG), Communicating Process (CP), Communicating Process with Time (CP+T), Programmer's View (PV), Programmers View with Time (PV+T), and Cycle Accurate (CA) level. Except Algorithmic level and register-transfer level, other levels of abstractions are considered a part of transaction level models.

In Communicating Process (CP) level, the system is composed of parallel processes that exchange high level data and parameters via point-to-point links. Generally, this level is architecture and implementation independent. However, partitioning system functions into parallel tasks do require some architectural concerns. The level can be associated with timing information to become Communicating Process + Timing (CP+T) level.

Unlike CP level that is architecture and implementation independent, Programmer' s View (PV) level contains some micro-architecture details and is more architecture specific, especially for communication. In CP level, the mechanism used for communication is point-to-point links between processes. PV level requires more elaborated communication architecture such as on-chip bus or network on chip, and arbitration mechanism such as arbitrators or routers is required. Another important characteristic for PV level

is that PV level is register-accurate. This characteristic provides an accurate programmer's representation of hardware system used in the driver software. Similar to CP level, PV level can be associated with timing information and become Programmer's View + Timing (PV+T).

The Cycle Accurate (CA) level contains micro-architectural details and timing is accurate to each individual clock edge. The arbitration of the communication is fully compliant to the communication protocol. However, this level usually does not model complete internal registers.

Figure 2.2 shows the level of abstraction for transaction level models and potential flow between these levels. TLMs in this work are at a level of abstraction equivalent to the Programmer's View with Timing (PV+T) level as described in [4].

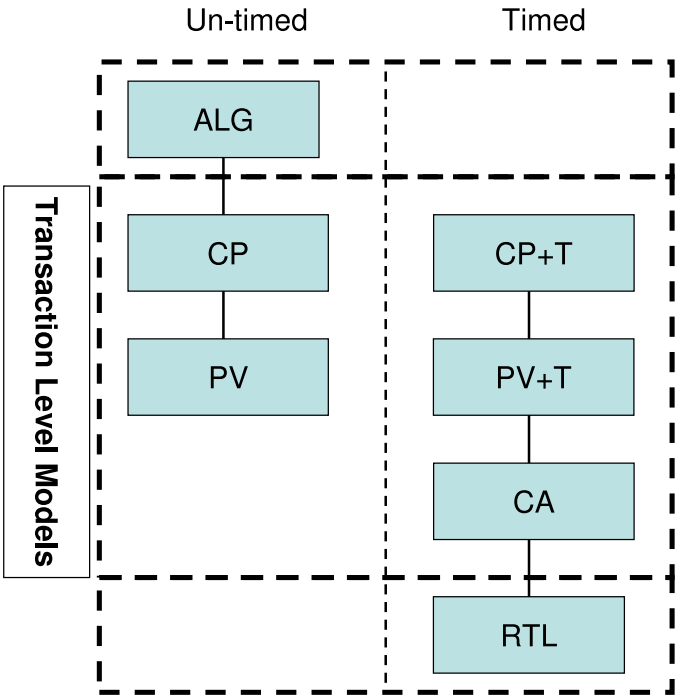

Figure 2.2. TLM abstraction levels and flow [4]

This is the highest level of abstraction which can include some amount of micro-

architectural detail.

2.1.2 Advantages of Transaction Level Modeling

Transaction level modeling has the following desirable properties that are necessary for a complicated system design:

- **Fast simulation speed:** By ignoring unnecessary hardware details, and not modeling at cycle-by-cycle basis for computation within IP cores, models at the transaction level run faster. For example, for burst-read or burst-write transactions, TLMs only consider the total required time to transfer the data. They do not have to calculate the time for each part of the data transfer, which enhances the simulation speed. From the results of [20], TLM for ARM AMBA bus is two to three orders faster than its RTL implementation.

- **Easy to develop, use, and apply:** Compared to lower level models such RTL, developing transaction level models is easier because unnecessary hardware details are ignored. For example, communication between modules is through functions calls such as bus_read() and bus_write() and does not require accurate pin information and implementation details of communication protocol. In addition, the simple function interfaces make TLM easy to use. Rather than involving complicated interfaces for each pin connection and handshaking protocol, the blocking or nonblocking TLM functions provide sufficient synchronization that is necessary for communication between modules. Furthermore, because of the simple interface the designer can easily explore different architecture or configuration by replacing

a module with another. Therefore, it is easy to apply to other modules.

- **Early construction and verification:** TLMs can be constructed early in the system design process. Because TLMs do not requre hardware details, the designer can create modules and validate the designs at an earlier phase and enable early software and firmware development. In addition, designers can explore different possibilities of system configurations and optimize the system early in the design stage.

- **Extended modeling ability:** With the advent of SystemC language, TLM is able to model both the hardware and software components of a system in the same environment. This method simplifies the design flow.

2.2 Transaction Level Power Modeling

Transaction level modeling technique is gaining more popularity with emerging standard architecture modeling languages like SystemC [21] [22]. IP core providers are beginning to provide such models for the purpose of embedded software development and early architecture analysis. These models are typically at a level where they do not capture all power-related aspects of the cores in order to optimize the simulation performance.

To enable power estimation for a system composed of such transaction level models, we incorporate power estimation techniques into a SystemC functional model designed to run embedded software. In this work, we propose a power estimation methodology for IP cores. We demonstrate the power estimation methodology in PCI Express IP cores.

PCI Express is emerging as a standard for a unified I/O architecture in terms of

providing I/O connectivity for embedded, desktop and server applications. In commercial SoC designs, a major portion of overall power consumption can be consumed in the multiple PCI Express cores required to support various devices such as graphical chip and memory. Consequently, power estimation and optimization for the PCI Express core implementation is critical. This chapter first introduces the PCI Express architecture. Next, we introduce how we implement TLM for PCI Express. We explain the power estimation methodology and augment the PCI Express TLM with the methodology. Finally, we demonstrate the simulation result for that power modeling methodology that we developed for IBM CoreConnect architecture.

2.3 PCI Express Architecture

PCI Express architecture [5][23] is the latest PCI architecture which is a low cost, high bandwidth, and highly scalable modern I/O standard communication technique. PCI Express architecture supports 2.5Gbps per lane per direction transfer rate and can achieve higher bandwidth by combining different numbers of lanes [23]. PCI Express is backward compatible to previous version of PCI architecture including PCI 1.0 and PCI 2.0 and PCI-X. This backward compatibility can help designers reuse previous design and reduce design cost.

PCI Express utilizes point-to-point links and switched protocol between IP cores instead of bus architecture. Previous PCI architectures increase the pin number and use sideband signals to provide higher performance and power management. These techniques are less favorable in modern architecture due to the increased capacity load caused

by the increased pin connection. Current pin count for PCI architecture is more than 100, which will induce significant capacity load if multiple PCI devices are used. Instead, PCI Express uses the point-to-point and in-band connection, and each component connects another component through a direct link. By using the point-to-point connection technique, each direction only requires two low-voltage differential pairs (one transmit, and one receive pair). A PCI Express lane is comprised of these two unidirectional differential pairs, each operating at 2.5Gbps to achieve a basic overall throughput of 5Gbps.

The point-to-point connection technique also provides the technique to combine several lanes into a higher speed lane. In PCI Express, x1, x2, x4, x8, x16, x32 can be used. Therefore, it provides a maximum speed of 2.5Gbps x 32 = 80Gbps in each direction.

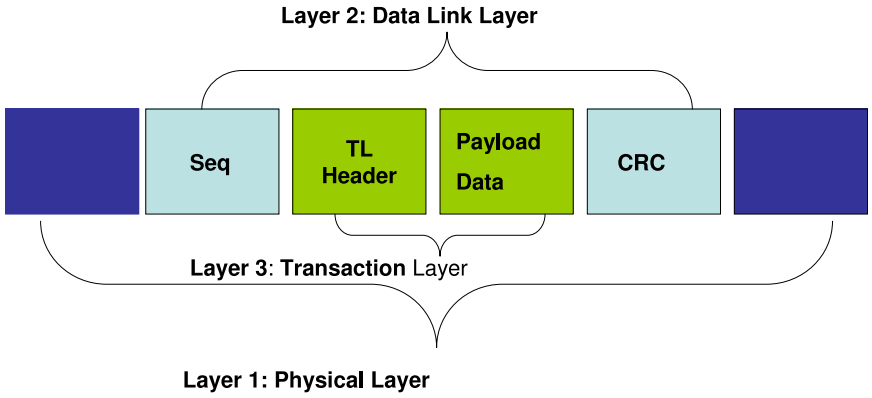

Figure 2.3. Packet format

PCI Express uses packet-based transmission. Within the PCI Express protocol, the data that need to be transferred are divided into smaller chunks and are concatenated with routing and error checking information to form a packet, as shown in Figure 2.3. Each side of the PCI Express link has an ingress port and an egress port. The ingress port processes incoming packets and the egress port processes outgoing packets.

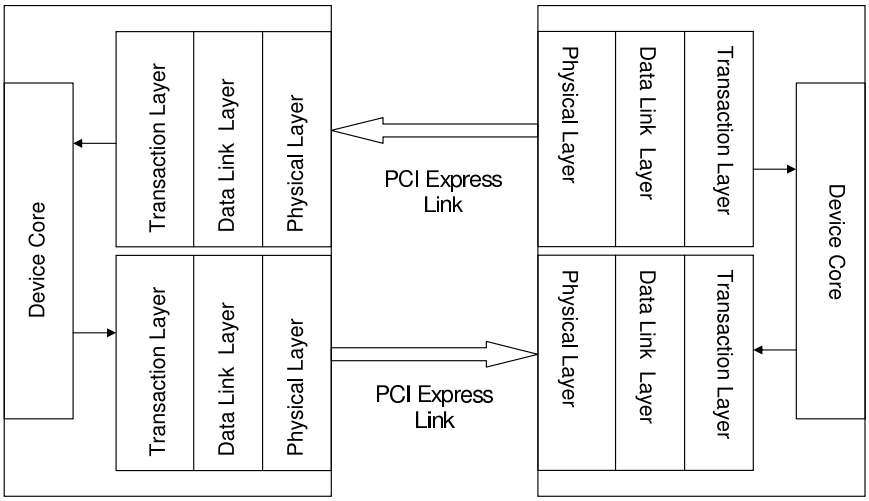

Figure 2.4. Layer Diagram of PCI Express

Figure 2.4 shows the layer diagram of PCI Express. There are three layers in both the egress port and the ingress port: the Transaction Layer, the Data Link Layer and Physical Layer. When a device starts to send data, it will send them to the Transaction Layer first. At this layer, the data will be attached with the header and CRC information to form a transaction layer packet (TLP). The Transaction Layer also provides a Virtual Channel mechanism and Virtual Channel arbitration. Then, this TLP is sent to the Data Link Layer. The Data Link Layer provides flow control and error checking mechanisms. In the Data Link Layer, a sequence number and CRC information will be attached to the TLP and transferred to the Physical Layer. After this, the Physical Layer converts the logic values into differential signals and transmits them across the PCI Express link. Inversely, the Physical Layer in the ingress port received the packet first. In the Physical Layer, the Physical Layer header is removed, and the data link layer packet is transferred to the Data link Layer. Similarly, in the Data Link layer, the Data Link Layer header is removed, and the transaction layer packet is transferred to the Transaction Layer. In

the Transaction Layer, the Transaction Layer header is removed, and the data inside the packet can be used by the IP cores.

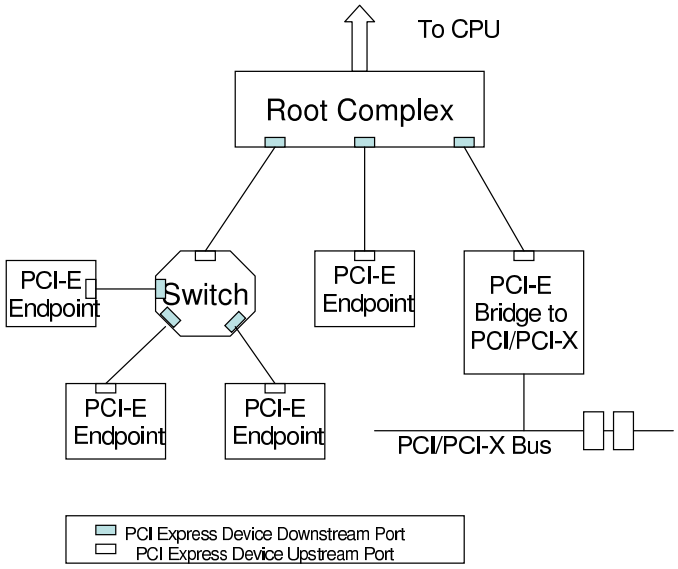

Figure 2.5. An example of PCI Express topology

An example of the PCI Express topology is shown in Figure 2.5. The topology of PCI Express is a tree structure, involving three kinds of components: Root Complex, Switch and Endpoint. If traffic is from a Root Complex to an Endpoint, then it is downstream traffic. Otherwise, it is upstream traffic. A Root Complex acts as a bridge between PCI Express architecture and other buses such as IBM CoreConnect Processor Local Bus (PLB) [17] or ARM AMBA AHB [18] bus. When a request is received from the PLB or AHB bus, it is converted to PCI Express packets and is transferred to its destination, and vice versa. An Endpoint connects to a device core, such as memory or external device, and allows only upstream traffic. A Switch can accept downstream and upstream traffic. Depending on the routing information in a packet, a Switch may transfer the packet to a Root Complex, another Switch or an Endpoint.

A Traffic Class (TC) in a PCI Express defines the priority of each packet. It is assigned by a device's application or device driver. Packets with different TCs move through the connection with different priority, resulting in varying performances. These packets are routed through the connection by utilizing Virtual Channel (VC) buffers implemented in Switches, Endpoints and Root Complexes.

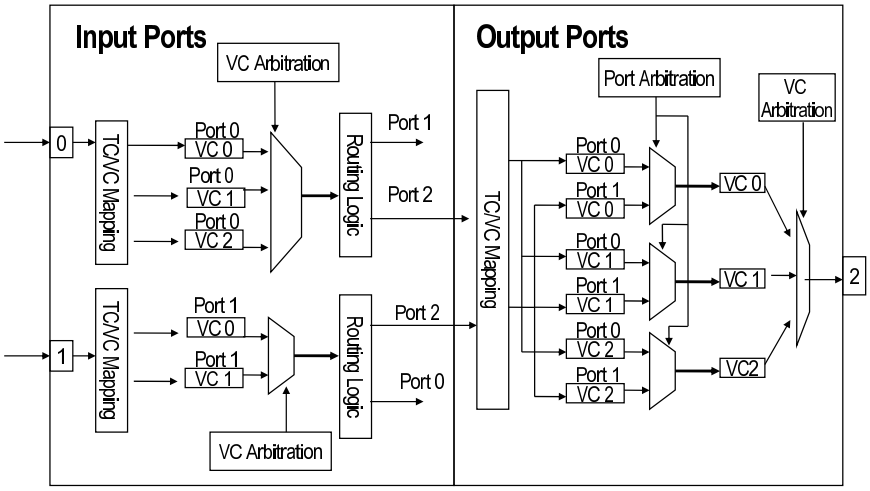

Figure 2.6. VC and port arbitration mechanism. This figure is adopted from [5]

Figure 2.6 depicts the relations between Traffic Class and Virtual Channel mapping (TC/VC mapping), virtual channel arbitration and port arbitration mechanisms in PCI Express architecture. Each Traffic Class is individually mapped to a Virtual Channel (A VC can have several TCs mapped to it, but a TC cannot be mapped to multiple VCs). The TC in each packet is used by transmitting and receiving ports to determine into which VC buffer to drop the packet. Switches are configured to arbitrate and prioritize between packets from different VCs before forwarding. This arbitration is referred to as VC arbitration. In addition, packets arriving at different ingress ports are forwarded to their own VC buffers at egress ports. These transactions are prioritized based on the ingress port number while being merged into a common VC output buffer for delivery

across the egress link. This arbitration is referred to as Port arbitration [5]. As a result, packets with different TC numbers may observe different performances when routed through PCI Express connection.

PCI Express is a low cost, high bandwidth and high flexibility solution. It is becoming a promising communication solution for system-on-chip. Therefore, we decide to choose PCI Express to demonstrate our power estimation methodology. In the next section, we explain how we construct PCI Express transaction level models.

2.4 Transaction Level Models for PCI Express

This section offers a detailed explanation about how we build PCI Express TLM model and how to interface PCI Express to PLB. Our PCI Express TLM is implemented in SystemC 2.0.1 [21] [22].

SystemC is a language that extends from C++ language and has become increasingly popular as a design and verification language. It supports designs that span from concepts to implementation in hardware and software. Using SystemC, we can eliminate the need for multiple models, and complete designs in a single modeling platform. In addtition, SysteC provides the ability to model hardware at higher levels of abstraction and enables early access to a functional virtual prototype of the hardware before the details of RTL implementations are completed.

Because our PCI Express inherits the PLB interfaces from IBM PowerPC 405 Evaluation Kits with CoreConnect SystemC TLMs (IBM PEK), IBM PEK will be explained before PCI Express TLM models.

2.4.1 IBM PowerPC 405 Evaluation Kit

IBM PowerPC 405 Evaluation Kit with CoreConnect System TLMs (IBM PEK) [6] is an industrial tool that enables designers to evaluate, build, and verify system-on-chip designs. This tool can quickly evaluate the trade-offs on performance, power, timing, and chip die size. Furthermore, it helps designers to make informed decisions and avoid mistakes. As shown in the tool's name, this tool contains IBM CoreConnect SystemC transaction level models.

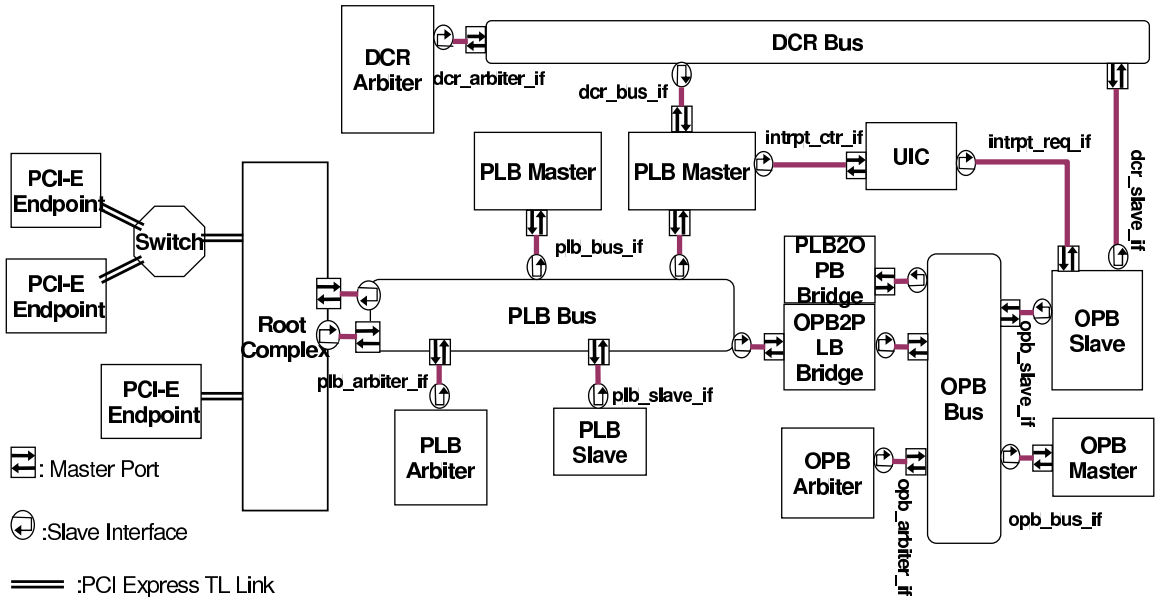

Figure 2.7. CoreConnect TL simulation platform with PCI Express TLM

Figure 2.7 shows the organization of a SystemC TLMs for a CoreConnect system with PCI Express TLMs. The right side of the figure illustrates the SystemC CoreConnect TLMs in IBM PEK, which include models for the processor local bus (PLB), on-chip peripheral bus (OPB), a bus bridge, and a device control register (DCR) bus. The left side shows the TLMs for PCI Express which we will explain in detail in the next section. This tool also implements many IP cores such as DDR Memory Controller and

UART. By using various IP cores and various connections between IP cores, designers can explore and evaluate the performance and power of the designs.

These SystemC TLMs provide PLB interface that our PCI Express TLM can connect to. Our PCI Express utilizes the PLB master and slave interface from IBM PEK.

2.4.2 PCI Express Transaction Level Models

```
Root(sc_module_name name, int type, int Width_of_VC, int Depth_of_VC
int PCIE_start_address, int PCIE_end_address,
int PLB_start_address,  int PLB_end_address);

Switch(sc_module_name name, int type, int Width_of_VC, int Depth_of_VC,
int PCIE_start_address, int PCIE_end_address);

End(sc_module_name name, int type, int Width_of_VC, int Depth_of_VC,
int PCIE_start_address, int PCIE_end_address);
```

Figure 2.8. Constructor declaration for the Root Complex, Switch, and Endpoint modules in SystemC

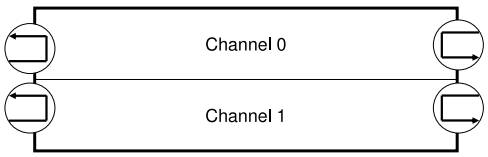

Figure 2.9. TLM for a duofifo module

As defined in PCI Express Specification [23], PCI Express has three kinds of components: Root Complex, Switch, and Endpoint. Connections between components are made through a bi-directional point-to-point link, which is emulated by a dual FIFO mechanism in our simulation platform. Figure 2.9 shows the TLM for a duofifo module. The dual FIFO is composed of two FIFOs, channel 0 and 1, each of which handles one direction of the transmission. Each channel has two PCI Express slave interfaces, read and write interfaces, that can connect to PCI Express master ports. These FIFOs were implemented by the sc_fifo channel [12] in SystemC.

Figure 2.8 is the SystemC module declaration for the Root Complex, Switch, and Endpoint. These modules have the following parameters:

- Module Name: Each module has a unique model name, and cannot be duplicated.
- Type: The type of each component can be either 0 or 1. The function of the type is used to identify the channel that the module can use to communicate with other modules. Modules of the same type cannot connect to each other directly. Root Complex is always Type 0.
- The dimensions of the Virtual Channels: The Width_of_VC parameter specifies the number of virtual channels in the VC buffer, while the Depth_of_VC parameter specifies the depth of a virtual buffer in the VC buffer.
- Address Space: Since a Root Complex is equipped with interfaces to CoreConnect PLB bus, it requires two address spaces, PLB and PCI Express. The two address spaces are determined by $PCIE\_start\_address$, $PCIE\_end\_address$, $PLB\_start\_address$, and $PLB\_end\_address$ variables. Unlike the Root Complex, the Switch and Endpoint only require PCI Express address space. The address spaces of each components cannot overlap.

Figure 2.10 shows the TLM for a Root Complex. The Root Complex connects PCI Express architecture to CoreConnect Processor Local Bus (PLB). In this figure, the Root Complex uses a PLB master port and a PLB slave interface to connect to a PLB bus. For the connection with other PCI Express components, the Root Complex uses a read and a write master port to connect to a duofifo module. A protocol conversion

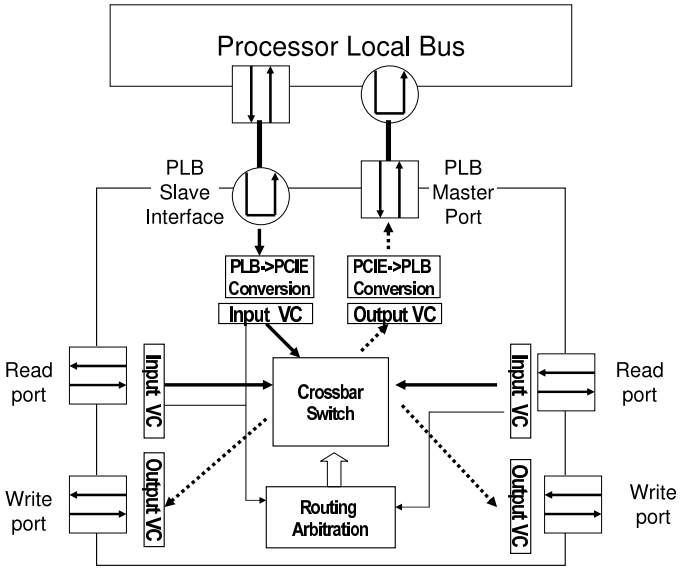

Figure 2.10. TLM for a PCI Express Root Complex that connects to a PLB bus

mechanism inside the Root Complex can convert a packet from the PLB format to the PCI Express format and vice versa. Port arbitration and Virtual Channel arbitration functions can provide correct path information for each packet, and move packets to correct destinations.

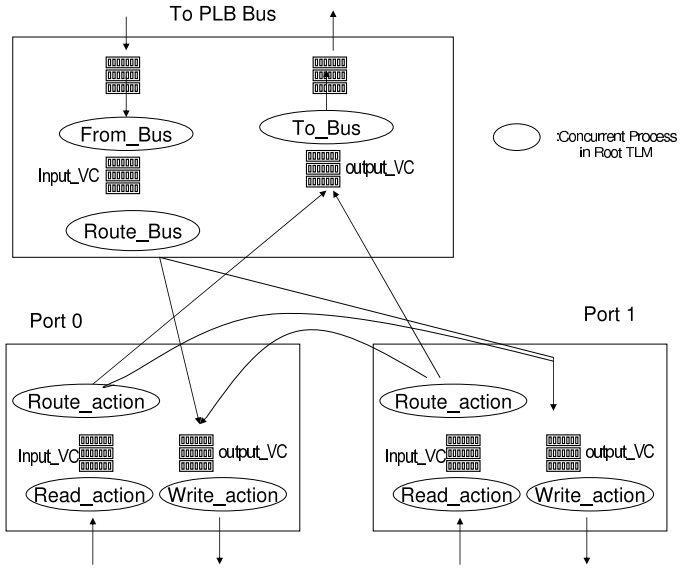

Figure 2.11. Concurrent Process Diagram for the Root Complex TLM

Figure 2.11 shows the concurrent process diagram. The concurrent processes can handle packets simultaneously. The From_PLB process retrieves data from the PLB input buffer, converts the data into PCI Express packets, and transfers the packets into a PCI Express input buffer. The Route_PLB process retrieves packets from PCI Express input buffer, and transfers the packets to output buffers according to the address inside the packet. The Write_action process retrieves the packets from the output buffer and transfers the packets to the connecting module.

```
class Root : public PLB_SLAVE_IF,  public IbmTlm::Module
{
  public:
    sc_in_clk plb_clk;
    sc_in_clk pcie_clk;
    PLB_MASTER_PORT plb_port;
    sc_port<write_if,3>out;
    sc_port<read_if,3>in;

    SC_HAS_PROCESS(Root);
    Root(sc_module_name name, int type, int Width_of_VC, int Depth_of_VCi,
    int PLB_start_address,  int PLB_end_address,
    int PCIE_start_address, int PCIE_end_address)
    {
      SC_THREAD(from_plb);
      SC_THREAD(to_plb);
      SC_THREAD(route_plb);

      SC_THREAD(write_action0);
      SC_THREAD(read_action0);
      SC_THREAD(route_action0);

      SC_THREAD(write_action1);
      SC_THREAD(read_action1);
      SC_THREAD(route_action1);
    }
  public:
      //Interface for PLB Bus
      void acknowledge_address(PLB_REQUEST* req);
      int read(PLB_REQUEST* req);
      int write(PLB_REQUEST* req);

};
```

Figure 2.12. Class definition for the Root Complex module

Figure 2.12 is the class definition for the Root Complex written in SystemC. As shown in the concurrent process diagram, the Root Complex has nine concurrent processes.

The PLB port has three processes: $from\_PLB$, to_PLB, and $route\_PLB$ process. The $from\_PLB$ process accepts requests from the PLB bus, converts PLB requests into PCI Express requests, then puts the packets into input VC buffers. The $Route\_PLB$ process retrieves packets from input VC buffers, and distributes the packets to correct output VC buffers according to the packet destination. The to_PLB process retrieves PCI Express packets from output VC buffers, converts PCI Express packets into PLB requests, and sends packets through the PLB bus. For PCI Express ports, both ports have $read\_action$, $write\_action$, and $route\_action$ processes. The $read\_action$ process accepts packets from PCI Express fabric and puts packets into input VC buffers, and the $write\_action$ retrieves packets from output VC buffers and transfers the packets through PCI Express fabric. The $route\_action$ process will read packets from input VC buffers, and put the packets into correct output buffers according to the destination address of the packets. The Root Complex is equipped with PLB master port (PLB_MASTER_PORT) and PLB slave interface (PLB_SLAVE_IF) in order to access the PLB bus.

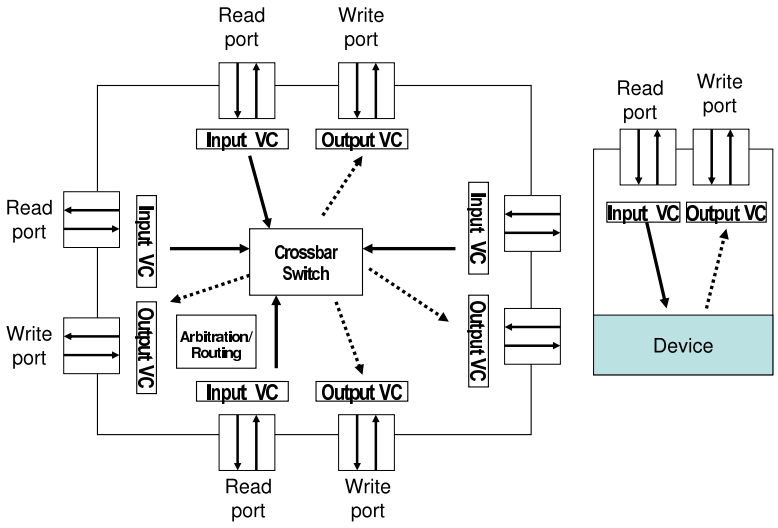

Figure 2.13. TLM for Switch and Endpoint modules

The Switch and Endpoint TLMs are simpler than the Root Complex because they only have to connect to PCI Express architecture. The left part of Figure 2.13 shows an example of a Switch TLM that can connect to four other PCI Express components. Similar to the PCI Express ports in the Root Complex TLM, each PCI Express port in the Switch TLM connects to other components through a dualfifo module. Port and VC arbitration in the Switch TLM are similar to those in the Root Complex. The right part of Figure 2.13 is a TLM for a Endpoint module. It only has one PCI Express connection with one read port and one write port. There is no routing mechanism required in Endpoint TLM.

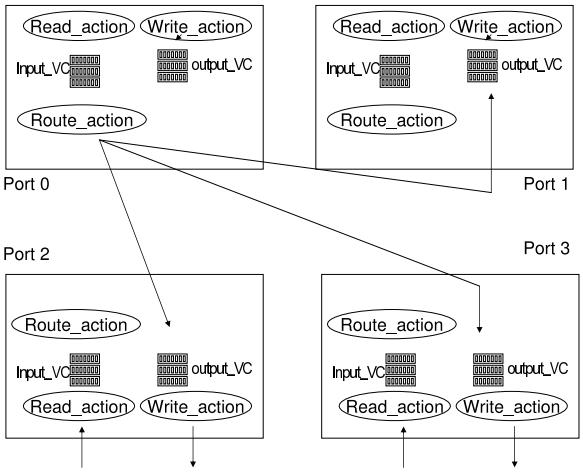

Figure 2.14. Concurrent Process Diagram for Switch TLM

Figure 2.14 is the process diagram for a Switch TLM. This Switch has four ports, each of which has three concurrent processes:*Read_action*, *Write_action*, and *Route_action*. This figure shows the path of the *Route_action* process in the port 0. The *Route_action* process in other ports have a similar function. The function of each process is similar to Root Complex's PCI Express processes.

The corresponding class definition for Switch and Endpoint TLMs are shown in

```
class Switch :  public IbmTlm::Module
{
  public:
    sc_port<write_if,3>out;
    sc_port<read_if,3>in;
    sc_in_clk pcie_clk;

    SC_HAS_PROCESS(Switch);
    Switch(sc_module_name name, int type, int Width_of_VC, int Depth_of_VC,
    int PCIE_start_address, int PCIE_end_address)
    {
      SC_THREAD(write_action0);
      SC_THREAD(read_action0);
      SC_THREAD(route_action0);

      SC_THREAD(write_action1);
      SC_THREAD(read_action1);
      SC_THREAD(route_action1);

      SC_THREAD(write_action2);
      SC_THREAD(read_action2);
      SC_THREAD(route_action2);

      SC_THREAD(write_action3);
      SC_THREAD(read_action3);
      SC_THREAD(route_action3);
    }
  private:
    vc input_buf[4];
    vc output_buf[4];

};
class Endnode : public  public IbmTlm::Module
{
  public:
    sc_port<read_if> in;
    sc_port<write_if> out;
    sc_in_clk pcie_clk;
    SC_HAS_PROCESS(Endnode);
    Endnode(sc_module_name name): sc_module(name,)
    {
      SC_THREAD(read_action);
      SC_THREAD(write_action);
      SC_THREAD(data_process);
    }
  private:
    vc buffer;
};
```

Figure 2.15. Class definition for Switch and Endpoint

Figure 2.15. The process diagram of a Endpoint TLM is not shown here because it is only a simplified version of Switch TLM.

Each PCI Express input or output port has a maximum of eight Virtual Channels,

and can be changed easily during TLM initialization. We use the C++ Vector type to store packets. The maximum number of packets that each Virtual Channel can store can be changed to explore the performance and power trade-offs of the system.

The buffer in the VC channel is a critical section that we cannot read or write at the same time. We use *sc_mutex*[12] to provide mutual exclusion mechanisms. Both the output VC buffer and the input VC buffer need to be protected. When a packet enters the Transaction Layer, the first step is to check the TC/VC mapping and find out the VC buffer that needs to store the packet. If no other process is using this buffer, then we can allocate the packet into the buffer. The VC arbitration mechanism will select a channel and transfer the packet to the receiver.

High level models of the Transaction and Data Link Layers are implemented in our transaction level model but the Physical Layer is not modeled. This is because the Physical Layer handles low level signal switching and it is usually not implemented in the high level TLM.

2.5 Power Estimation Methodology for Transaction Level Models

This section explains the TLM power estimation methodology and demonstrates this methodology in the PCI Express TLM. The power estimation methodology we proposed for transaction level modeling of SoC includes the following four steps:

1. Identify the transactions from the data book or specification of IP cores that is significant in the power modeling. The transactions are the operations that IP

cores use to communicate with other components, such as read, write, and interrupt_request operations.

2. Identify the parameters from the data book or specification. Parameters are usually the factors that will significantly affect the performance and power consumption of an IP core. Examples of parameters are execution frequency, buffer size, and the number of virtual channels.

3. Characterize the power consumption of each transaction and build hierarchical transaction level power (HTLP) trees for each transaction. The purpose of a HTLP tree is to illustrate how to calculate the total power consumption for a transaction. Each transaction is composed of many lower level procedures, and the power consumption of a transaction is the summation of those lower level procedures. Based on the specification of IP cores, we can construct HTTP trees for each transaction. The power consumption in the higher level in the diagram is based on the summation of the power consumption at the lower level.

4. Create TLMs for the IP core and parameterize the TLM. Creating parameterized designs for TLM can provide performance exploration. The parameters can be useful for power estimation.

5. Augment the TLM to extract the parameters for macro-model during the transaction level simulations and make calls to the appropriate power macro-models, thus deriving energy measures for each of the cores for that particular simulation. This can be done dynamically at run-time to derive information during simulation (trade-off between simulation accuracy and speed should be taken into account)

In the following section, we explain how to apply this methodology on our PCI Express TLM.

2.5.1 Identify Transactions and Parameters

We first identify the operations that need to be modeled as transactions. PCI Express provides four types of primary operations:

1. Memory requests, which include memory read, memory write, and memory read lock requests. These requests read data from memory devices and write data to memory devices.

2. I/O requests, which include IO Read and IO Write. These requests read from IO devices and write to IO devices.

3. Configuration requests, which include Configuration Read and Configuration Write. The configuration request can configure the address space and operation mode of different modules.

4. Message requests are message operations which provide methods for communicating within different modules.

2.5.2 Parameters for PCI Express TLM

We define these primary instructions according to the PCI Express specification. To model the power consumption for the above requests, we model the following parameters that are correlated to these requests, and build a table that contains the parameters with their corresponding power consumption.

- Width and depth of the Virtual Channel buffer. These two parameters determine the size of the VC buffer used in the PCI Express TLM. We construct a table to record the power consumption of the VC buffer with various width and depth dimensions.
- Power consumption is required to insert a packet into a VC buffer or retrieve a packet from a VC buffer. These can be used to calculate power consumption for the VC buffer control logic.
- VC buffer arbitration
- TC/VC Mapping
- Port arbitration.
- Leakage Power. This is the static power consumption that a PCI Express IP core will consume even it doesn't have any activity.

Several parameters are related to the Root Complex only. These parameters include the following power parameter :

- Power consumption for the PLB request buffer. This buffer is used to store the PLB request.
- Power consumption for protocol conversion logic.
- Power consumption for PLB master port and PLB slave interface.

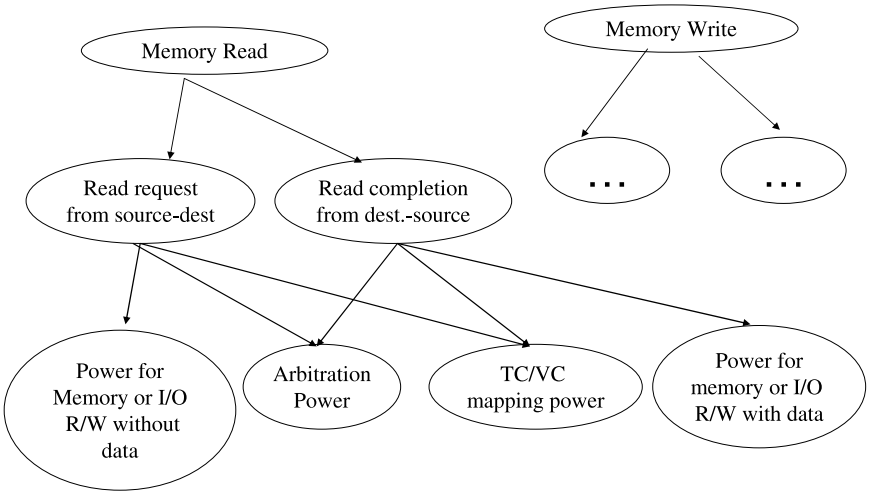

Figure 2.16. Hierarchical transaction level power tree for memory read transaction

2.5.3 Characterize Power Consumption of Each Transaction and Build HTLP Trees

Figure 2.16 shows a hierarchical power tree for the memory read transaction. The power for the memory read transaction has two components: power for read request from source to destination and power for read completion from destination to source. Each component can be divided into smaller components. After we characterize the power consumption for each transaction, we construct a power table to store all the power information. A transaction is associated with many parameters, and we use these parameters to decide the power consumption of a transaction under various circumstances. The TLM model can consult the table and determine the power consumption during execution.

2.5.4 Augment TLM with Power Estimation Functions

As the packet passes through a PCI Express component, the power consumption is obtained through table look-up. This power information can be used to calculate the average power for the transaction after the packet reaches the destination

```
void vc::read_vc(tl_packet& c, int queue) {
  queue=vc_arbitrate();
  pwr_vc_arb(c);       ...................(1)
  mutex.lock();
    if(num_elem[queue] ==0)
    {
      mutex.unlock();
      wait(wr_ev[queue]);
      mutex.lock();
    }
    c = channel[queue][first[queue]];
    --num_elem[queue];
    first[queue] = (first[queue]+1) % DEEP_VC;
    pwr_vc_dequeue(c, queue);.............(2)
    rd_ev[queue].notify();
    mutex.unlock();
}
```

Figure 2.17. VC Read function

```
void vc::write_vc(tl_packet c, int queue) {
  mutex.lock();
  queue=tc_vc_map(c);.....................(3)
  pwr_vc_map(c);,

  if(num_elem[queue]==DEEP_VC)
  {
     mutex.unlock();
     wait(rd_ev[queue]);
     mutex.lock();
  }
  channel[queue][(first[queue]+num_elem[queue])
  % NUM_VC] =c;
  ++num_elem[queue];
  pwr_vc_enqueue(c, queue);..............(4)
  wr_ev[queue].notify();
  mutex.unlock();
}
```

Figure 2.18. VC Write function

Figure 2.17 and 2.18 are the pseudo codes that show how the TLM of PCI Express is augmented with power estimation functions. We inserted timing information and power estimation functions into the TLM of PCI Express to provide power analysis.

Figure 2.17 is the read function for a Virtual Channel buffer. A power estimation function (*pwr_vc_arb*) for the VC arbitration is added after VC arbitration is complete. Also, a function (*pwr_vc_dequeue*) that estimates the power for reading a packet from the VC is added to the TLM. Both functions look up the power estimation reference table to obtain the power consumption.

Figure 2.18 is the write function for Virtual Channel. We also add power estimation functions into the TLM of VC write function. Instead of VC arbitration and de-queue function, we added the power consumption for VC mapping function(*tc_vc_map*) and enqueue function(*pwr_vc_enqueue*)

We have demonstrated the TLM for PCI Express and how to improve the TLM of PCI Express with power estimation functions. The simulation speed for TLM is very fast, but the accuracy can be improved if the TLMs capture more details of the hardware activity. By doing this, we can get the power estimation during transaction level simulation. These TLMs will be useful when we are doing system level power analysis.

2.5.5 Simulation Execution

After completing the previous methodology in PCI Express TLMs, we can instantiate TL modules and execute transaction level simulation using various scenarios or applications. Examples of scenarios are PCI Express to PCI Express traffic, PCI Express to CoreConnect PLB traffic, and CoreConnect PLB traffic to PCI Express traffic. One way to generate these scenarios is to use a random number packet generator. In addition, we can obtain traffic from real applications. One example is to use simulators

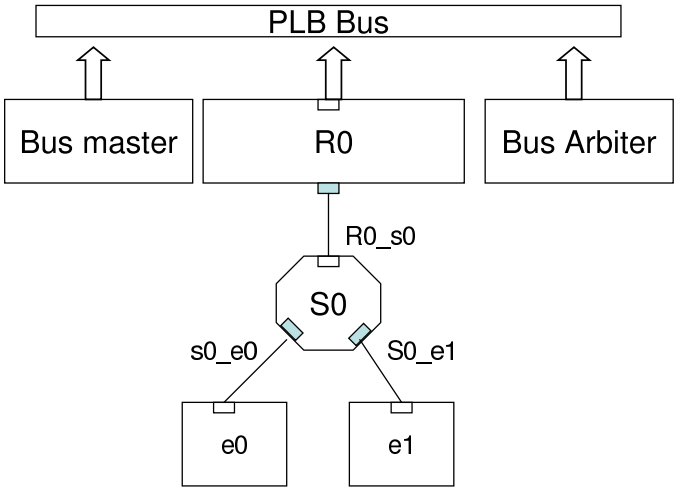

Figure 2.19. A Example of System Instantiation

such as MPARM [24] that runs real applications and generates real traffic for different applications.

2.6 System Instantiation and Simulation for PCI Express TLM

In this section, we demonstrate how to integrate an on-chip bus and PCI Express TLMs and simulate the system. Since our PCI Express TLMs provides interfaces to IBM PLB, we demonstrate how to instantiate PCI Express and PLB TLMs, integrate the TLMs and execute simulations. Note that PCI Express TLMs can connect to other buses if the interfaces to these bus are provided and integrate in the PCI Express TLMs. Figure 2.19 shows the diagram of the example, and Figure 2.20 is the corresponding SystemC codes for Figure 2.19. We use the following steps to instantiate the system:

1. Instantiate a PLB bus in the *sc_main*() function. The *sc_main*() function is the main function of a SystemC program. We instantiate an on-chip bus to which PCI

```
int sc_main(int, char **) {
   sc_clock plb_clock("plb_clock");
   sc_clock pcie_clock("pcie_clock");
   PLB_BUS* bus = create_bus(); // Create a PLB bus and instantiate PLB Arbiter
   plb_cpu* cpu = new plb_cpu("plb_cpu",
   0,         // master ID
   CC_128,    // data bus width
   CC_64);    // address bus width
                            // register our task with the plb_cpu
                  std::pair<plb_cpu::thread_fn, void*> tmp(&plb_cpu_task1, NULL);
   plb_memory* memory = new plb_memory("plb_memory",
        PLB_ADDRESS_BUS(0x00000), // memory address low
        PLB_ADDRESS_BUS(0x1ffff), // memory address high
        CC_128,                   // data bus width
        CC_64");                    //address bus width
   ROOTComplex* root = new ROOTComplex("root", TYPE0,   //TYPE 0
                       8,   8,   //Width and Depth of Virtual Channel
                       0x20000,  0x3ffff, //PLB memory address low and high
                       0x100000, 0x12ffff); //PCI Express memory address low and high

    bus->clock(plb_clock);
    bus->slave_port(*root);
    cpu->clock(plb_clock);
    cpu->bus(*bus);
    memory->clock(plb_clock);
    root->plbclock(plb_clock);
    root->pcieclock(pcie_clock);
  dualfifo *r0_s0;
    r0_s0 = new dualfifo("r0_s0"); r0_s0->clock(pcie_clock);
  dualfifo *s0_e0;
    s0_e0 = new dualfifo("s0_e0"); s0_e0->clock(pcie_clock);
  dualfifo *s0_e1;
    s0_e1 = new dualfifo("s0_e1"); s0_e1->clock(pcie_clock);
  Switch *switch0;
    switch0 = new Switch("Switch0", TYPE1, 4, 4, 0x100000, 0x11ffff);
    switch0->clock(pcie_clock);
    switch0->out(*r0_s0); switch0->in(*r0_s0);
    switch0->out(*s0_e0); switch0->in(*s0_e0);
    switch0->out(*s0_e1); switch0->in(*s0_e1);
  Endnode *end0;
    end0 = new Endnode("End0", TYPE0, 2, 2, 0x100000, 0x10ffff);
    end0->clock(pcie_clock); end0->out(*s0_e0); end0->in(*s0_e0);
  Endnode *end1;
    end1 = new Endnode("End1", TYPE0, 2 , 2, 0x110000, 0x11ffff);
    end1->clock(pcie_clock); end1->out(*s0_e1); end1->in(*s0_e1);

  sc_start(10000,SC_NS);
}
```

Figure 2.20. Example of System Instantiation

Express modules connect in the *sc_main*() function. In this example, we called a *create_bus*() function to instantiate a PLB bus and an PLB arbiter. A plb master device (plb_cpu) and plb slave devie (plb_memory) are also instantiated to simulate the functions of the PLB bus. More information about the PLB bus, arbiter and

```
void plb_cpu_task1(plb_master_generic* caller, void* user_data)
{
  std::vector<uint8_t> data;
  while (true)   //loop forever
  {
    for (int i = 0; i < 256; i++)  //write 256 bytes of data to memory
    {
      data.push_back(i);
    }
    memory_write(caller, 0x20000, CC_QUADWORD, 16, data);
    // read 256 bytes of data
    data = memory_read(caller, 0x20000, CC_QUADWORD, 16);
  }
}
```

Figure 2.21. Example of request generations in the *plb_cpu_task*1() function in the blocking master. Modified from [6]

other components can be found from [6].

2. Instantiate PCI Express modules, and dual FIFO modules in the *sc_main*() function. In Figure 2.19, we have one Root Complex, one Switch and two Endpoint modules. The Root Complex connects to the PLB bus directly, and provides protocol conversion mechanisms that convert PLB bus requests to and from PCI Express requests. The duofifo modules between different components serve as communication channels. Changing the topology of the system can achieve architecture exploration.

3. Define the address space and parameters. We provide the address space for the PLB slave memory and each PCI Express module, and specify the width and depth of the Virtual Channels used in each PCI Express module. In Figure 2.20, we assign different widths and depths of the Virtual Channels for the Root Complex, Switch and Endpoint modules. We also assign the address space for the PLB and PCI Express modules.

4. Generate requests to simulate the operations. We can generate requests in the

following different places:

(a) The *plb_cpu_task*1() function that hook to the PLB bus. master module. In this function, we can issue blocking and nonblocking read and write operations to the PCI Express or PLB subsystem.

(b) The *data_process*() function in the Endpoint modules can issue PCI Express operations. These operations includes Memory read, Memory write, IO read, IO write, Configuration read, and Configuration Write. The destination of these operations can be in the on-chip bus or other PCI Express modules.

With these three functions, we can generate traffic from the on-chip bus to the PCI Express subsystem and from PCI Express subsystem to the PLB bus.

We can use these functions in different modules to simulate different traffic scenarios such as from the PLB bus to PCI Express or from the PCI Express bus to the PLB bus. Figure 2.21 shows the *plb_cpu_task*1 function used by the *plb_cpu* bus master modules. In this example, the function generates ten blocking read and ten write operations. If necessary, the destination address of these operations can be changed to simulate other scenarios.

5. We can specify the simulation time in the *sc_main*() function. The *sc_start*() function to indicate how long the simulation runs. $sc\_start(10000, SC\_NS)$ means the simulation will run 10000ns. $sc\_start(-1)$ stands for executing infinitely until the program finishes.

In this section, we illustrated the TLMs and process diagrams for the Root Complex, Switch and Endpoint TLMs. We demonstrated how to instantiate PCI Express and PLB

TLMs modules. We also explained how to generate the traffic and simulate the system. The following section will discuss how we validate the power estimation methodology we proposed is section 2.5.

2.7 Validation of Power Estimation Methodology

Case	TLM power (mW)
Scenario 1 (PCI Express to PCI Express traffic 1)	24.37
Scenario 2 (PCI Express to PCI Express traffic 2)	27.92
Scenario 3 (PLB to PCI Express traffic 1)	52.21
Scenario 4 (PLB to PCI Express traffic 2)	58.73
Scenario 5 (PCI Express to PLB traffic 1)	63.48
Scenario 6 (PCI Express to PLB traffic 2)	66.24

Table 2.1. Simulation results for PCI Express

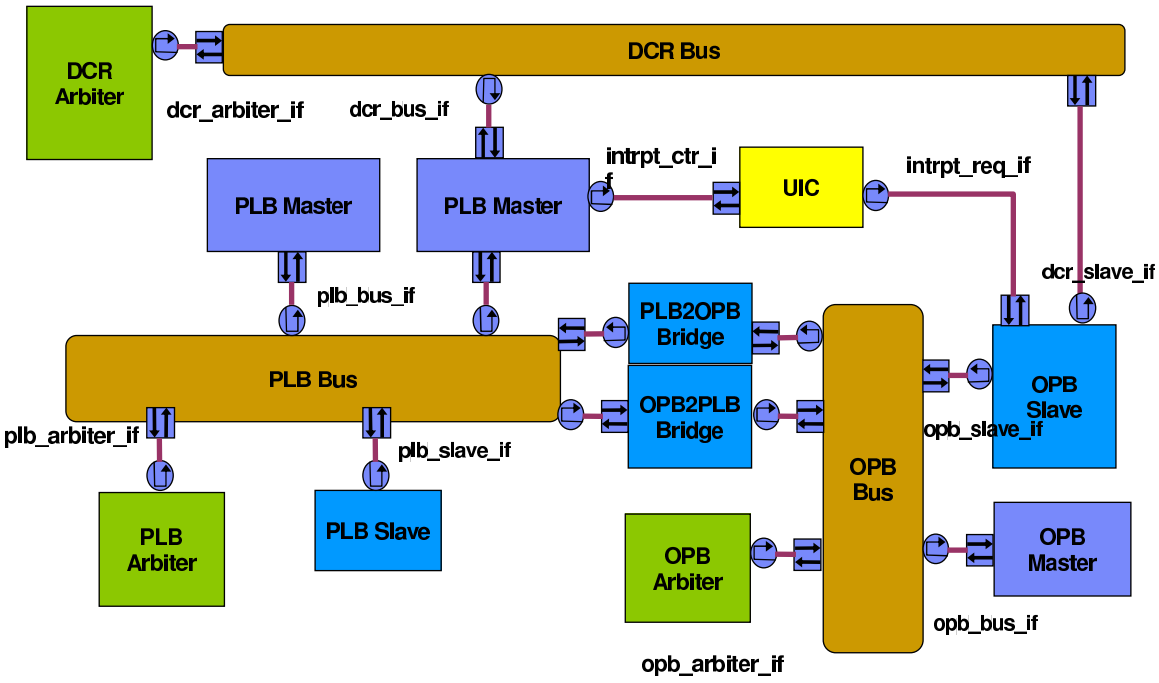

Figure 2.22. CoreConnect TL simulation platform

We have presented a TLM Power estimation methodology and demonstrated how to

implement this methodology in PCI Express architecture. We also generated different scenarios in our simulation and obtained these power estimations. We show the simulation results in Table 2.1. The validation is done by comparing the power estimation for gate level and transaction level power estimation. The validation of the methodology was performed at IBM [25] for IBM CoreConnect bus architecture (instead of PCI Express) due to the need for gate level power estimates. Figure 2.22 depicts the IBM CoreConnect transaction level platform. Elements of the CoreConnect architecture include the processor local bus (PLB), on-chip peripheral bus (OPB), a bus bridge, and a device control register (DCR) bus. High-performance peripherals connect to the high-bandwidth, low-latency PLB. Slower peripheral cores connect to the OPB, which reduces traffic on the PLB, resulting in greater overall system performance [17].

Case	GLM power (mW)	GLM runtime (min)	TLM power (mW)	TLM runtime (min)	Error
Scenario 1	57.87	22.3	56.145	0.005	-3.01%
Scenario 2	58.74	25.4	56.1194	0.01	-4.46%
Scenario 3	59.595	26.8	57.071	0.02	-2.6%
Scenario 4	22.744	35	21.975	0.02	-3.38%
Scenario 5	57	45	63.35	1.0	11.19%

Table 2.2. Comparison of Gate level and Transaction level simulation results for IBM CoreConnect Bus

We have completed the validation of IBM CoreConnect TLMs on the IBM 440 platform-based design platform [25]. To generate power consumption information for each transaction, we ran IBM Test Operating System on the IBM 440 platform-based design system. The system can produce a sequence of transaction and ran each sequence accordingly. We dumped the VCD file and feed the VCD file into a gate level power estimation tool to obtain power consumption for gate level simulation. We measured the

power consumption for each transaction including burst read, burst write, single read, single write. We integrated the gate level power information into a industrial strength CoreConnect TLM. Then we compare the gate level and transaction level power estimation. The result is shown in Table 2.2. The result shows that the power estimation for gate level and transaction level simulations are very close. However, the simulation time for transaction level simulations are much less than that for gate level simulation. The result shows our power estimation methodology can achieve desirable result.

2.8 Conclusion

This chapter presents a power estimation methodology for transaction level models written in SystemC. A related work for IBM CoreConnect SystemC TLMs is published in [26]. We demonstrate the methodology in PCI Express architecture. The main constituents of this include a power characterization approach, a hierarchical representation of transaction level data, and a mapping mechanism to augment TL simulation models with power information. The experimental results for IBM CoreConnect demonstrate the validity of the approach, providing a starting point for further exploration of transaction level power estimation.

Chapter 3

Transaction-based Reliability Modeling for Bus-based SoC

System-on-chip (SoC) architecture has traditionally relied upon bus-based interconnect for their communication needs. However, increasing bus frequencies and the load on the bus calls for added focus on reliability issues in such bus-based systems. We provide a detailed analysis of different kinds of errors and the susceptibility of such systems to such errors on various components that comprise the bus.

We first investigated the effect of single-bit errors on the bus system during the course of different transactions. The work demonstrates the fact that the vulnerability of signals is quite similar across benchmarks and only a few signals in a bus system are critical and need to be guarded. Such transaction-based analysis helps us to develop an effective prediction methodology to predict the effect of a single bit error on any application running on a bus-based architecture. As the possibility of multiple-bit errors on such systems increases, we also examined the effects of multiple-bit errors.

When compared with the actual simulation results, our results demonstrated that our transaction-based prediction scheme works with an average accuracy of 92% over all the benchmarks for single-bit errors. By expanding our transaction-based prediction scheme to multiple-bit errors, we also achieved 90% accuracy over all benchmarks, which means our prediction scheme can also be applied to multi-bit errors.

3.1 Introduction and Motivation

A bus-based interconnect has been traditionally used as a communication channel in system-on-chip architectures. However, growing demands for on-chip integration have lead to significant increases in the number of components connected to the bus, raising concerns over the architecture of such bus systems. Consequently, bus architectures have evolved over time from single shared bus models to multi-layer complex topologies for supporting the growing demands of bandwidth, concurrency, and constrained power budgets. The frequency of operation has also been on a rise, from a traditional under 100Mhz to as high as 200Mhz. Commercial buses standards like ARM AMBA [18], PCI Express [23], and IBM CoreConnect [17] have experienced nearly triple frequency increases over their original designs. Further complexity in such systems is due to different power and performance enhancing features, such as operating different buses connected by bridges at different frequencies, voltages, etc [27]. Consequently, the probability of errors occurring in such system buses increases significantly, which is further aggravated due to technology scaling. These errors may occur due to a wide range of causes, including capacitive coupling, soft errors, crosstalk, and di/dt noise, as discussed in [28, 29].

Most of these errors lead to transient errors. These errors may affect the system in many different ways, ranging from creating undetected data corruption to system crashes, which make the systems unreliable.

Making the systems more reliable is critical since it is expensive when when the systems fail and critical data are lost. Traditional work to model the system reliability is done at a lower level such as the register-transfer level or gate level. Doing modeling at the lower level is almost impossible as well as time-consuming as the complexity of modern systems increases. In addition, modeling at the lower level can not be done at the early design cycle since these models are not ready yet.

In this work, we model the reliability of the system at a higher level, the transaction level model, to provide rapid and early error susceptibility estimation. We introduced a novel transaction-based reliability model for errors. We provide a detailed analysis of the types and causes of different errors that may occur due to erroneous bit flips on a bus-based system. These errors occur during different transactions and the effects of errors on the bus-based system was characterized into four categories: fatal error, deadlock error, silent data corruption, and no effect. We also proposed a system level error susceptibility model for bus-based SoC architecture.

In our modeling, single-bit errors as well as multi-bit errors are characterized. This is because is the possibility of multi-bit errors increase significantly as the advances of technology scaling and exponential increases of system complexity happen. For example, interconnect crosstalk caused by signal swings among different wires may cause multiple unexpected bit flips on the interconnect. Soft error occurring on the bus arbiter can cause multiple bus flips as well. Additionally, technology scaling and higher system density

cause a shrinking distance between wires and transistors and an system error can easily cause multiple bit flips on wires and transistors. Therefore, it is imperative to explore the effects of multi-bit errors on the system.

We completed our experiments on a cycle-accurate simulator for the ARM AMBA AHB bus. We characterized the susceptibility of AMBA bus on errors in various signals over different transactions. Such susceptibility analysis provides an effective way to characterize the error susceptibility of a bus architecture based on the transactions in the application. The proposed prediction scheme provides results with 92% accuracy for single-bit errors and 90% accuracy for multi-bit errors as compared to the simulation results on an average for all the applications.

3.2 Related Work

On-chip communication architecture could be primarily classified into either packet-based communication or bus-based communication as shown in Figure 3.1. Packet-based communication is employed in different bus protocols like PCI Express and Xpipes [30]. Bus-based communication is adopted in commercial bus standards like AMBA AHB [18], CoreConnect and Wishbone [31]. Most packet-based communication systems have the reliability models adopted from the traditional packet-based error detection and fault tolerance schemes. PCI Express has all the packets containing CRC bits appended to them at the link layer to ensure a highly reliable data transfer. Similar approaches may not be completely applicable or suitable to the bus-based bus protocols. Various schemes for error detection and correction in bus-based systems are described in [32, 33, 34, 35].

However, with the increasingly stringent power and area budget, all such schemes are under reconsideration. Bertozzi et al. [36] provides a detailed comparison of the power and area implications of various error protection schemes. In this work, we particularly focused on the reliability issues in the bus-based protocols. We demonstrated the significance of different control signals of the bus-based on the effect of an error in any of those signals. It is, however, quite important to note that the signals in the system are not equally critical for the functionality of the system. Weaver et al. [37] presents a vulnerability analysis of various components in contemporary microarchitectures. We exploit such an observation for bus architecture under consideration to provide an effective criticality factor for each of the signals of the bus system. We provided a transaction-based error characterization model for bus systems.

We provided error susceptibility analysis for the bus system at the transaction level model. Transaction level model is a high level modeling technique that models the communication between components as transactions [3]. Transactions are modeled as cycle-accurate or at least cycle approximate, while the computation for each component is modeled as untimed. This modeling technique can provide fast simulation speed with compact models since many details of hardware information are ignored at this level. The transaction-based models for systems are getting significant attraction since the system is becoming increasingly complicated. The modeling technique can serve as a verification platform in an early design stage when detail hardware models are not ready. Such transaction-based models have been previously analyzed with respect to power consumption in [25].

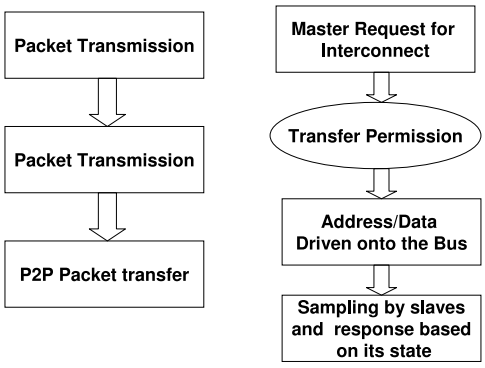

Figure 3.1. Packet-based vs bus-based protocols

3.3 Bus Architectures

Figure 3.2 shows a typical single shared bus architecture. This is the architecture of most bus-based systems. The main components in these systems are essentially as follows:

- Master and Slaves: The core components that are connected to the bus. Typically, the masters are processing cores, DMA controllers etc, and the slaves are memories, bridges and peripheral devices.

- Interconnect Structure: The logic that deals with the transfer of the data. Most of the bus control follows state machine based logic. This primarily comprises of the arbiter, decoding logic and the various control signals to control the bus protocol.

In our analysis, we dealt with errors on the interconnect structure which in turn included the errors occurring in the master and the slave ports connected to interconnect. These are the typical components of any bus-based architecture. We have provided a comprehensive error analysis for the AMBA bus architecture. Moreover, our model and characterization can be extended to any bus architecture.

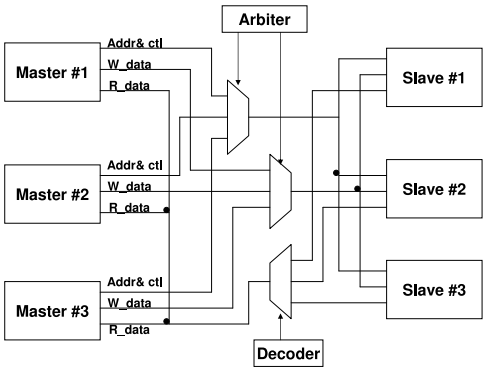

Figure 3.2. AMBA Bus Architecture

3.4 Error Characterization Model

This section explores the effects of single-bit errors and developes a generic characterization model for single-bit errors. The characterization model for multi-bit errors is similar and is postponed to later sections. Developing a generic characterization model needs to focus on the following issues:

- Determine an effective error injection model that simulates a single-bit error scenario in the best manner.
- Find the most common effects of single-bit errors in the bus system.
- Find the cause for such effects and quantify their dependency on various components of the bus system.
- Provide an effective scheme to predict the effect of any single-bit error on the whole system while executing any application.

We demonstrate achieving the aforementioned goals in the context of an AMBA bus. A cycle-accurate virtual platform simulator written in SystemC for the AMBA bus-based

SoC architecture is used for simulations of our model. Details about the platform may be obtained from [24]. Table 3.1 shows the various signals in an AMBA bus, which we consider for analysis in our model. Our model has considered almost all the signals in the AMBA AHB 2.0 bus system. Signals that are not considered are either not implemented in the virtual platform or are used to support different operations, which are beyond the scope of this study.

Signal name	Function
HBURST[2:0]	This signal indicates if the transfer is a form of a burst.
HWRITE	When HIGH, this signal indicates a write transfer and when LOW, a read transfer is executed.
HTRANS[1:0]	This signal indicates the type of current transfer, which can be non-sequential, sequential, idle or busy.
HSIZE[2:0]	Indicates the size of transfer; typically byte(8-bit), halfword(16-bit) or word(32-bit)
HREADY	When HIGH, the signal indicates a transfer completed on the bus. The signals can be set on low if we need to extend a transfer.
HRESP[1:0]	Response from the slave. Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.
HBUSREQ	This signal is from a bus master to the bus arbiter which indicates that the bus master requires the bus.
HMASTER[3:0]	Arbiter indicating the bus master that is currently performing a transfer.
HGRANT	Ownership of the address and control signals changes at the end of a transfer when HREADY is HIGH, so a master gets access to the bus when both HREADY and HGRANT signals are HIGH.
HSEL	This signal indicates that the current transfer is intended for the selected slave.
HADDR[31:0]	This is the 32-bit system address bus that indicates the address to read or to write.

Table 3.1. The definition of signals and its function

3.4.1 Single-bit Error Injection

We inject a single-bit error by corrupting the various signals of interconnect during the operating phase of the bus's finite state machine. The single-bit error is injected just once by modifying the state of one of the interconnect signals. Such an error is injected randomly at any time. However, to obtain the effect of the error on a particular transaction we ensure that the error is injected during the required transaction in the bus. This precisely simulates the single-bit error in our simulation. In this chapter, we only consider two types of transactions in the bus, namely the BUS-READ and the BUS-WRITE operations. The characterization model can be further extended to other transactions such as BURST-READ and SPLIT-READ, and similarly for the write transactions.

3.4.2 Consequences of Both Single- and Multi-bit Errors

Based on the various errors observed due to single-bit and multi-bit faults, we have categorized the effects of the errors into four main classes. The errors are classified based on their effects on the system. The four main implications of the errors in the system are as follows:

- **Fatal Error (FE)**: An error that leads to system crashes. The fatal error means the system detects a fatal situation that causes the system to stop executing. Fatal errors include the following:

    1. The address cannot be mapped to any existing slave. One reason for this is the address signals are changed to incorrect status.

2. The core is trying to write an instruction that is read-only. One reason for this error is an incorrect HWRITE signal.

3. The core accepted a signal that should not have happened. For example, consider a scenario when the core expects to stop but receives a spurious grant signal from the arbiter due to an erroneous bit flip.

4. An incorrect or bad address occurred that should not have during the transfer operations. For example, during a burst transfer, the next requested address is not a correct offset address of previous address. It is, however, important to note that this error can easily be overcome if the correct address information is retained, in which case system crashes may be prevented.

- **Deadlock(DL)**: An error that leads the system into a no progress state. Examples of this error are infinite loops due to modifications in loop invariants, or incorrect arbitration so that no master can obtain the grant and continue its work. Two types of deadlock situations were found in our simulations.

  1. Program counter corruption: this case arises when the program counter jumps to an unknown location, there is an invalid instruction, and the processor keeps retrying. If the Program counter is affected and the program behavior is changed, and the *stop_simulation*() function which contains a software interrupt to terminate the program is either not executed or executed incorrectly, then the system enters an infinite loop.

  2. Undefined state: this case arises when the state machine of the bus which defines the bus protocol enters an undefined state, each of the processors wait

for the response from the bus in order to obtain access. However, at this time no one can process.

- **Silent Data Corruption (SDC)**: An error that remains unnoticed until the end of the simulation but provides incorrect results. SDC primarily occurs due to accumulation of error over many cycles without the system getting in deadlocks or crashing. One of the prime reasons for SDC is the incorrect HADDR signals that generate read-or-write transactions on incorrect addresses. This causes incorrect data processing that leads to incorrect results.

- **No Effect**: An error that does not affect the system in any way.

3.4.3 Single-bit Transaction-based Error Characterization

The error simulations are executed for multiple benchmarks and the criticality of each signal is assessed based on the effects of a single-bit error on any signal over read-and-write transactions.

We define values $P_{f|T=r}$, $P_{d|T=r}$, $P_{f|T=w}$ and $P_{d|T=w}$ for each signal, where:

- $P_{f|T=r}$ stands for the percentage of times the error in a signal leads to fatal error during a read transaction.

- $P_{f|T=w}$ stands for the percentage of times the error in a signal leads to fatal error during a write transaction.

- $P_{d|T=r}$ stands for the percentage of times the error in a signal leads to a deadlock during a read transaction.

- $P_{d|T=w}$ stands for the percentage of times the error in a signal leads to a deadlock during a write transaction.

The behavior of the error was uniform in the cases of the fatal errors and deadlocks for all the benchmarks; however, the silent data corruption was completely dependent on the nature of the benchmarks and hence characterizing it was not possible. The reason for such an abrupt behavior with respect to silent data corruption could be attributed to the fact that SDC are highly dependent on the nature of the application. This error characterization step helps us in finding the criticality of different signals and our observations are discussed in the single-error results section 3.4.5.

3.4.4 Single-bit System Level Error Prediction

The signal level susceptibility analysis is extended to system level by using the error-effect percentages of the signals described in the previous section. The errors are injected in the signals based on a probability dependent on the width of the signal. The effect of the errors are recorded and compared with the estimated values. The effect of any single-bit error on any signal is estimated using the following equation:

$$\begin{aligned} P_{sf} = Write\% \times \sum_{allsignals} p_i \times P_{f|T=w} \\ +Read\% \times \sum_{allsignals} p_i \times P_{f|T=r} \end{aligned} \tag{3.1}$$

where P_{sf} is the probability of any error leading to a fatal error in the system, p_i is the probability of an error hitting the signal and $P_{f|T=r}$ and $P_{f|T=w}$ are as defined earlier.

The value p_i is computed as the ratio of the signal width over the total number of bits in all the signals considered in our analysis. This reflects the fact that the probability of an error occurring in any signal is directly proportional to the number of bits of a signal.

Such analysis provides a near accurate estimate of the effect of any single-bit error on the execution of any application on the bus system. Similar equations for the different effects of errors may be written and their probabilities may be estimated for different applications. These probability numbers characterize the effect of errors completely in any bus-based system. Note that the accuracy of the prediction may further be improved by increasing the granularity of our analysis. We may consider more system parameters while characterizing each of the signals,such as the number of memory accesses, in order to increase the prediction accuracy of the model. In these cases, all the parameters should be incorporated into the prediction equation.

3.4.5 Single-bit Error Experimental Setup

We modify a cycle-accurate virtual platform, which models SoC architectures having ARM cores and has a capability of simulating different interconnect architectures. We analyze AMBA AHB interconnect in our work. A boot-time error file is enabled in the system during which various types of errors may be injected at different intervals on different components of the bus system. Another set of scripts is written to perform the profiling operation and determine the effects of various errors and finally observing the criticality of each of the errors. We perform the susceptibility analysis for different signals of the AMBA bus system.

We model the error susceptibility of the SoC by first identifying the error suscep-

tibility of each signal. The error susceptibility of each signal is found by executing a benchmark numerous times while injecting only one error on a signal in each execution. The results of the benchmarks are analyzed and we determine the number of executions resulting in fatal errors and deadlock situations. A deadlock is flagged when an application is not completed by a particular time. After getting the results from these executions of the benchmark, we can calculate the error susceptibility of each signal by dividing the error states over the number of executions. We do the same for each of the benchmarks and then get the average error susceptibility for each signal of all the benchmarks.

We instantiated four ARM cores in the virtual simulation platform and four private memories attached to the bus. Private memories are slaves that could be used as caches by the processor cores. The shared and semaphore memory spaces were instantiated as well. Table 3.2 shows information about the different applications we used for error characterization when executed on the SoC platform with the aforementioned specifications. Except for the first three benchmarks, which were included in the simulator and modified by us, the rest of the benchmarks were picked from the SPLASH suite.

App	C	BB(%)	BT(%)	R	W
Qsort	16576	47.4	23.53	56	743
Matrix	133503	36.59	18.29	240	23444
PIL Filter	220288	41.59	19.10	2640	7151
FFT	418920	61.21	29.97	1669	28053
DES	205307	54.01	20.61	33417	9370
LU	2051411	68.05	27.71	260717	302519

C:The number of the executing cycles. BB: The percent of time that the bus is busy. BT: The percent of time that the bus is transferring data. R: The number of read transaction W: The number of write transaction

Table 3.2. The six benchmarks used in our simulation

3.4.6 Single-bit Error Injection Results

Signal	Qsort	MATRIX	PILFILTER	FFT	DES	LU
HBURST	0.1	0.1	0.1	0.3	0.4	0.0
HWRITE	1.4	0	0.5	1.1	0.9	1.1
HTRANS	1.2	0	0.3	0.7	0.55	0.7
HSIZE	18.6	18.4	18.2	23.1	23.4	21.5
HREADY	0.5	0.1	0.9	2	2.2	1.8
HRESP	0	0	0	0.1	0.1	0
HBUSREQ	0	0	0	0	0	0
HMASTER	0.3	0	0	0	0.2	0.1
HGRANT	1.8	0.1	0.1	0.7	1.6	1.6
HSEL	22.3	24.9	19.1	29	26.9	28.3
HADDR	7.4	5.2	10.3	8	8.1	9.4

Table 3.3. Single-bit fatal error rate for each signal in benchmarks during a bus-read transaction ($P_{f|T=r}$)

To analyze the susceptibility of errors on different signals we first conducted experiments to investigate the effect of error on each signal for different benchmarks for different transaction types. Table 3.3 shows the value $P_{f|T=r}$ for different applications obtained over 10,000 runs. Clearly we can observe that, in general, the effect of the error on different signals remains similar across different benchmarks, mainly due to the fact that the error is characteristic of the protocol more than the nature of the benchmark. Similar characteristics are observed for the values $P_{d|T=r}$, $P_{f|T=w}$ and $P_{d|T=w}$, values for all the applications.

To obtain the criticality measure for each signal we obtain the average probability values for all the signals over different benchmarks for read-and-write transactions. Table 3.4 shows the average error rate of $P_{f|T=r}$, $P_{d|T=r}$, $P_{f|T=w}$ and $P_{d|T=w}$ for all the signals. We may observe that signals like HBUSREQ are very sensitive with respect to the system getting into deadlocks; on the other hand, the probability of an error in HBURST leading to malfunctioning of the system is really low. The reason for higher deadlocks due to erroneous HBUSREQ, HRESP and HREADY signals could

be attributed to the fact that all of them are quite important in establishing the right communication protocol, and consequently an error in them makes all the resources in the system in a busy wait state. HADDR is another signal that leads to system crashes with a high probability, mainly due to the system address going out of known or eligible ranges. The advantage of such an analysis is the fact that it provides a good opportunity for the reliability engineer to prioritize his/her focus on guarding the signals.

Signal	$P_{d\|T=r}$	$P_{f\|T=r}$	$P_{d\|T=w}$	$P_{f\|T=w}$
HBURST	2.57	0.17	0.51	0.33
HWRITE	11.80	0.83	0.77	0.37
HTRANS	8.29	0.58	0.70	0.29
HSIZE	0.00	20.53	0.14	23.70
HREADY	13.20	1.25	12.14	2.07
HRESP	30.01	0.03	24.49	0.00
HBUSREQ	32.62	0.00	14.25	0.00
HMASTER	0.00	0.01	0.99	0.63
HGRANT	5.53	1.07	1.56	0.50
HSEL	5.59	25.08	8.21	35.63
HADDR	4.36	8.06	0.53	9.81

Table 3.4. Average error rate for each signal

The criticality, however, is also dependent on the bit-width of each of the signals, which reflects a higher vulnerability of the signal with respect to others. For example, HADDR is more prone to errors as compared any of the other signals due to higher bit-width. To analyze the criticality of the signals while taking into account the bit-widths, we executed the benchmark circuits 20,000 times, with a single-bit error injected during each run.

The error injection was done probabilistically on all the signals, with the probability of error on any signal proportional to its width. The results were recorded and compared. Figure 3.3 and 3.4 show the percentage of fatal errors and deadlocks due to errors in different signals on different benchmarks. As expected, the probability of deadlock and

fatal errors due to the signal HADDR are the maximum due to its higher vulnerability toward getting struck by an error.

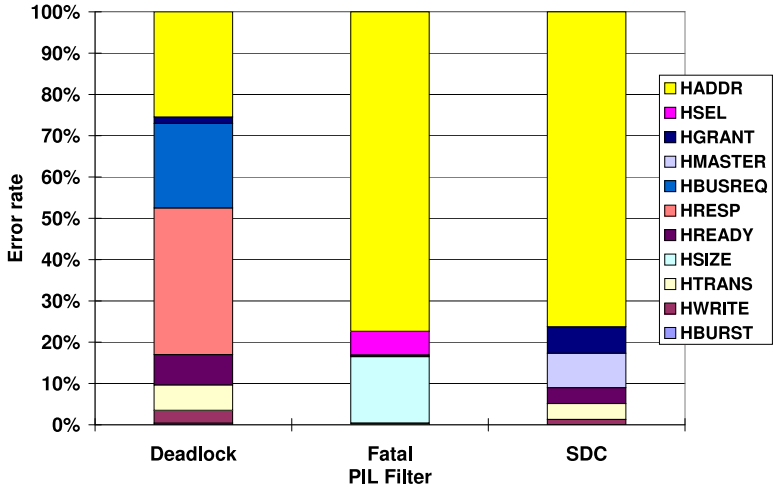

Figure 3.3. Effects of error on different signals for the PIL-Filter benchmark

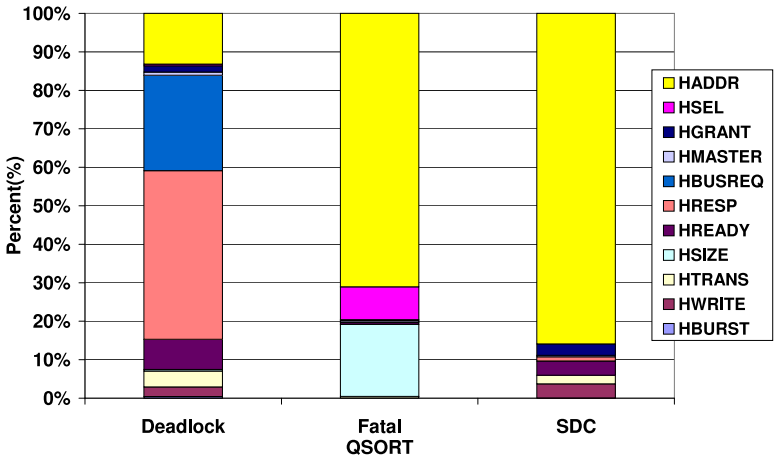

Figure 3.4. Effects of error on different signals for the Qsort benchmark

Using the probability values and the number of transactions in different benchmarks, we predict the probability of the effect of any single-bit error in the system. Such prediction is then validated with actual simulations. Figures 3.5 and 3.6 demonstrate the comparison of the predicted values and the simulated values. Our prediction scheme predicts the probability of the deadlocks with an accuracy of 94% and fatal errors with

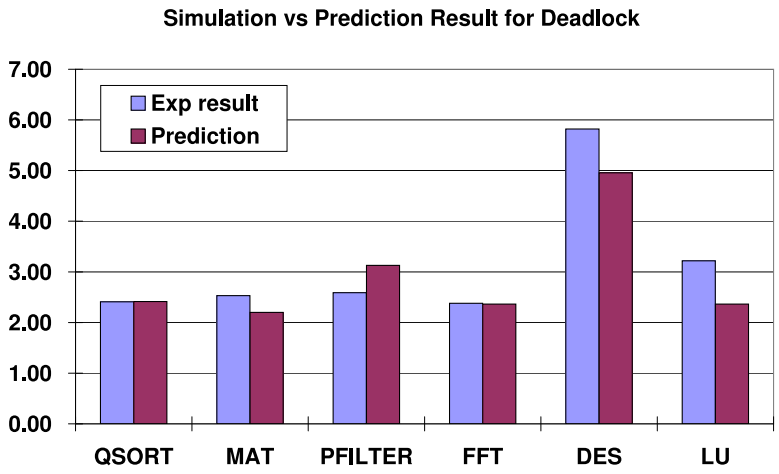

Figure 3.5. Comparing experimental values with predicted values for deadlock errors

an accuracy of 90%, respectively. It is important to note that the prediction accuracy is quite important considering the run-time of the simulations. The prediction is simply using the signal probability values from Table 3.4 and the number of transactions from Table 3.2. Table 3.5 shows the average run-time of the 20,000 simulations that we performed to obtain the system level effect of any single-bit error. Clearly the prediction scheme provides a much better methodology to obtain an estimate on the effect of any error in the whole system.

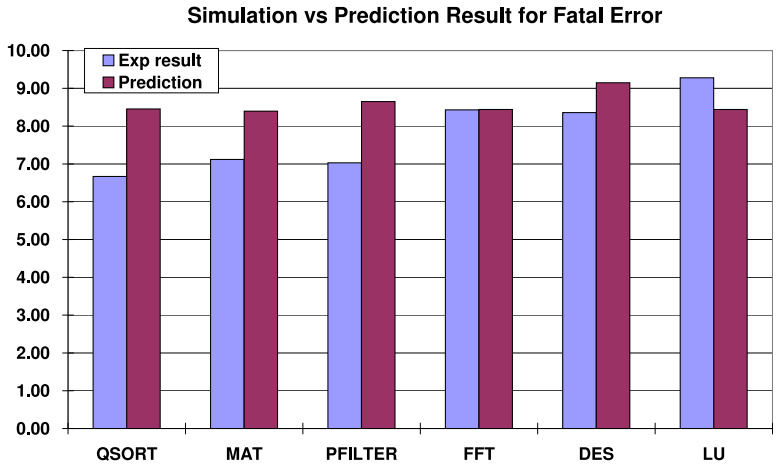

Figure 3.6. Comparing experimental values with predicted values for fatal errors

Benchmark	Time (in hours)
Qsort	5.44
MATRIX	15.93
PILFILTER	20.76
FFT	54.73
DES	56.16
LU	481.1

Table 3.5. Run time for obtaining system level effects of any single-bit error

3.5 Multi-bit Error Characterization Model

Previous sections dealt with single-bit errors on the SoC bus. Cases arise when a multi-bit error affects the system. For example, interconnect crosstalk caused by signal swings among different wires may cause multiple unexpected bit-flips on the interconnect. Soft error occurring on the bus arbiter can cause multiple bit-flips as well. To make it worse, the possibility of multi-bit errors is increasing due to technology scaling and higher system density because they cause shrinking distance between wires and transistors, and an error happening on the system can easily cause multiple bit-flips on wires and transistors. Therefore, it is critical to explore the effects of multi-bit errors on the system.

3.5.1 Multi-bit Error Injection

This section focuses on the impacts of multi-bit errors. Not all signals in the system have bit-widths greater than one. Signals, such as HBUSREQ and HGRANT, has only one bit-width. Only signals with multiple bit-widths will be affected by multi-bit errors. In the AMBA AHB, those signals with multiple bits are HBURST (2 bits), HTRANS (2 bits), HSIZE (3 bits), HRESP (2 bits), HMASTER (4 bits), and HADDR (32 bits). In addition, since the main causes of incorrect bit flip are soft error and crosstalk, and both

will cause errors that happen next to each other, we assume these errors are adjacent.

3.5.2 System Level Prediction for Multi-bit Error

We demonstrate how we extend system level prediction for single-bit errors to multi-bit errors. Similar to single-bit errors, the error simulations are executed for multiple benchmarks and the criticality of each signal is assessed based on the effects of a multi-bit error on any signal over different types of transactions. We define values $P_{f|T=r}(x)$, $P_{d|T=r}(x)$, $P_{f|T=w}(x)$ and $P_{d|T=w}(x)$ for each signal, where:

- $P_{f|T=r}(x)$ stands for the percentage of times the error in a signal leads to fatal error during a read transaction when an x-bit error happens to the signal.
- $P_{f|T=w}(x)$ stands for the percentage of times the error in a signal leads to fatal error during a write transaction when an x-bit error happens to the signal.
- $P_{d|T=r}(x)$ stands for the percentage of times the error in a signal leads to a deadlock during a read transaction when an x-bit error happens to the signal.
- $P_{d|T=w}(x)$ stands for the percentage of times the error in a signal leads to a deadlock during a write transaction when an x-bit error happens to the signal.

and

- $P(x)$ stands for the percentage of an x-bit error when errors happen to the signal.

The signal level susceptibility analysis is extended to system level by using the error-effect percentages of the signals described in the previous section. The errors are injected in the signals based on a probability dependent on the width of the signal. The effect

of the errors are recorded and compared with the estimated values. The effect of any multi-bit error on any signal is estimated using the following equation:

$$P_{sf} = Write\% \times \sum_{allsignals} (p_i \times \sum_{x=1}^{max} P_{f|T=w}(x) \times P(x)) \quad (3.2)$$
$$+Read\% \times \sum_{allsignals} (p_i \times \sum_{x=1}^{max} P_{f|T=r}(x) \times P(x))$$

where p_i is the width of the signal. This equation modifies the previous equations for single-bit errors and adds two variables: one stands for the number of erroneous bit-flips that an error causes, and another stands for the percentage that the error occurs.

After we add the two variables, the total system error rate will be the summation of the number of erroneous bit-flips of the error, times the percentage that the error occurs, times the error rate that the error causes. Therefore, the equation can predict the system error rate for multi-bit errors.

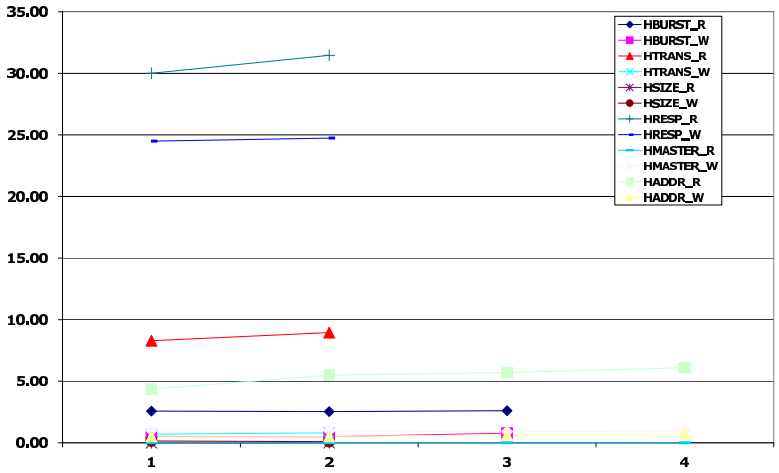

Figure 3.7. The trend of average deadlock error rate

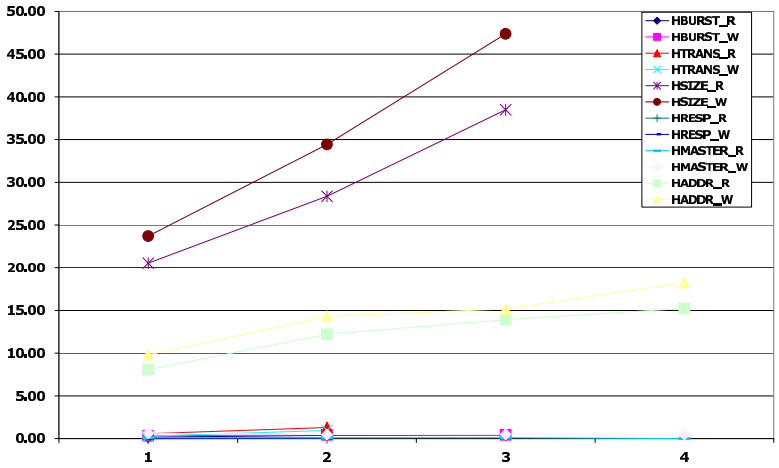

Figure 3.8. The trend of average fatal error rate

3.5.3 Multi-bit Error Injection Result

Figure 3.7 shows the result of deadlock error rate when a multi-bit error occurs during benchmark execution. Similar to the simulation for the single-bit error injection, we execute each benchmark 20,000 times, with a multi-bit error occurring in each execution. As we can see in Fig 3.7, error rates remain quite constant when the number of bit-flips increases. This result is important because it shows that a single-bit error contributes to the majority of the deadlock error. Additional bit errors do not escalate the deadlock much further. One exception is the HRESP signal; it increases about 10% during read operations. This is because the operations of the bus protocol is very sensitive to the HRESP signal. Additional errors on this signal will increase the deadlock rate significantly.

In contrast, Figure 3.8 shows fatal error rate for signals with multi-bit width. As can be seen here, the fatal error rate for the HSIZE and HADDR signal increases significantly. Our explanation for this is the HSIZE signal determines the transfer width, and if the transfer width changes, fatal errors occur. More bit-flips occurring on the transfer will

result in higher possibility to modify data that is required for transfers. Similar to the HSIZE signal, the HADDR signal is also very sensitive to multi-bit errors during transfer for the same reason.

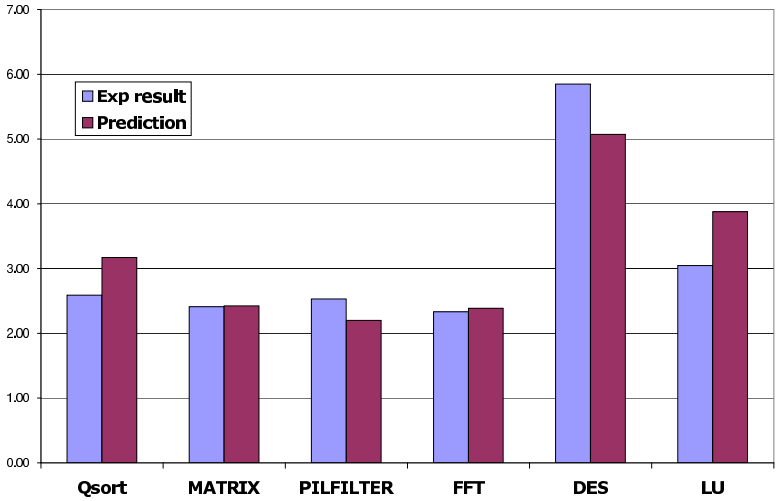

Figure 3.9. Comparing experimental values with predicted value for deadlock errors when a multi-bit occurs

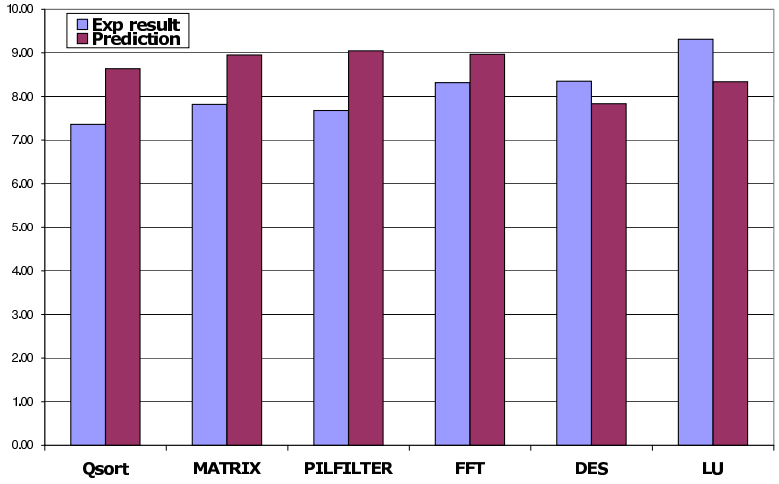

Figure 3.10. Comparing experimental values with predicted value for fatal errors when a multi-bit occurs

Figure 3.9 and Figure 3.10 are comparisons between prediction values using the system error prediction and values from experiments. In the prediction equations, we assume the possibility of single-bit errors as 5 times higher than two-bit errors, and the

possibility of two-bit errors as 5 times higher than three-bit errors and so on. From both figures, we found deadlock prediction value can achieve 89% accuracy and 91% accuracy for fatal error, and ranges from 81% accuracy to 115% accuracy for deadlock error, and 84% to 105% accuracy for fatal error.

3.6 Conclusion

In this work, we first characterized the effects of single-bit errors for bus-based systems and discovered fatal and deadlock errors are more close to protocol itself, not benchmark, which means fatal or deadlock error rates due to errors on the bus. We extended the single-bit work to multi-bit injection. Our work has demonstrated a novel, generic and effective transaction based error characterization scheme for bus architectures in SoCs. Such a model is quite generic and can be extended to other systems or bus protocol. The criticality measure of each of the signals provides an opportunity to prioritize the employment of any error correction schemes on the system, which is quite critical with the shrinking power and area budgets. From the simulations result, deadlock error rate remains quite constant even the width of bit-flips increases. However, fatal error rate for the HSIZE and HADDR signal increases significantly. The error effect prediction scheme provides the probability of the effect of any single-bit error on the system with an accuracy of 92% on an average for single-bit errors, and 90% for multi-bit errors.

Chapter 4

System Level Modeling for Device Degradation

Reliability of on-chip interconnect structures in system-on-chip (SoC) architectures has become very critical with increased integration. Scaling of feature sizes further aggravate the need for such guarding, particularly due to new emerging causes of both permanent device failures and degradation. Apart from permanent failures, systems may experience timing failures due to slow degradation of devices caused by physical phenomenon such as Negative Bias Temperature Instability (NBTI) and Hot Carrier Effects (HCE). Consequently, system buses, that form the backbone of the communication network in SoC systems, need to be designed to be aware of such permanent failures and timing degradation.

In this chapter, we present a HCE And NBTI Incorporated Tool for ASICs (HANITA) for complete analysis of degradation of circuits over a period of time due to NBTI and HCE. The tool is used to analyze the degradation impact on bus systems and the

vulnerability of buses to such circuit degradation. A hardware based dynamic schemes are proposed to detect timing degradation. A PROactive BUS (PROBUS) architecture is proposed that dynamically adapts to continue the system functionality even after the system timing degrades.

4.1 Introduction

System-on-chip (SoC) architectures are being extensively employed in many real-time systems. Scaling and increased integration in such systems is therefore quite aggressive, due to both performance and economic demands. Such an increased integration of IP components requires support for heavy on-chip communication. Consequently, the criticality of the on-chip communication system in SoC architectures necessitates an increased focus on its reliability. Particularly, in the regime of sub-100nm technologies the reliability of such communication systems becomes quite significant due to the growing concerns of not only transient faults, but permanent failures as well. Apart from the permanent failures, aging of circuits due to various physical phenomena is also increasing. Such phenomena may also shorten the lifetime of operation of circuits in the form of timing violations.

Many contemporary SoC architectures use shared bus or Network On Chip (NOC) based architectures for their communication needs. In this chapter, we look into the degradation of on-chip bus architectures in such SoC based systems, due to two different physical phenomena, namely Negative Bias Temperature Instability (NBTI) and Hot Carrier Effects (HCE). A comprehensive insight into the impact of such degradation in

typical shared bus-based systems based on commercial bus architectures is demonstrated. The degradation itself is studied on the individual components of the on-chip buses. Such an analysis is automated using a tool that takes the circuit information and computes the individual device degradation. These device degradations are back-annotated to compute the overall system degradation over a period of time. Prominent device degradation phenomenon (NBTI for PMOS and HCE for NMOS transistors), are modeled separately using empirically established existing models. The analysis we provided is also effective for upcoming communication architectures like NOCs due to the presence of similar circuit components in such systems.

NBTI has emerged as one of the dominant causes of system degradation for sub-micron technologies. The phenomenon is observed in PMOS transistors when stressed under negative gate voltage at an elevated temperature (e.g.,$V_{gs} = -V_{dd}$). Consequently, this effect can increase PMOS transistor threshold voltage and reduce absolute I_{on} current of PMOS devices, thus lowering the speed of a circuit. This threshold voltage increase leads to a reduced temporal performance and causes reliability issues and potential device failure as well. However, removal of the bias or adding positive bias helps decrease some interface traps, leading to a fractional recovery of the threshold voltage. On the other hand, degradation for NMOS transistors have primarily been dominated by the Hot Carrier Effects (HCE) causing electrons to get trapped in the oxide. Although there have been considerable work in analyzing the degradation impact on devices under continuous current flow conditions, it is necessary to analyze their impact on normal circuit operations. This analysis requires the precise device characterization in the circuit operations including the location and switching of the devices.

Although there have been efforts in isolation to solve such problems, this is the first effort to provide a well defined methodology to analyze such a degradation using automated tool flow, namely, HCE And NBTI Incorporated Tool for ASICs (HANITA). We demonstrate how combined analysis of such phenomenon significantly varies as compared to isolated circuit analysis. An important issue in countering such problems is the dynamic detection of the degradation during real time operation. We provide dynamic hardware based approaches to detect such timing violations and compare their effectiveness. Finally, based on the borrowed concept of immunity in the human body we propose a modified bus architecture, called PROactive BUS (PROBUS), that dynamically adapts to timing degradations and maintains functional correctness.

The contributions of this work are (a) Providing circuit analysis tools to model the run-time degradation of circuits due to NBTI and HCE; (b) Proposing techniques to detect timing degradation in buses with minimal hardware overheads; (c) Proposing PROactive BUS architecture that dynamically configures itself to an error resilient system.

The rest of this chapter is organized as follows. Related work and motivation is presented in Section 4.2. Section 4.3 explains the tool flow for the NBTI and HCE analysis for given designs. The impact of degradation in on-chip bus systems is presented in Section 4.4. We propose our solution to detection and the concept of PROBUS in Section 4.5. The experimental setup for our solutions are presented in Section 4.6 followed by our conclusion in Section 4.7.

4.2 Related Work and Motivation

The growing importance of SoC based systems in real time systems are depicted by their increased use in real-time systems like wireless LANs etc [38]. Such commercial products highly champion smaller feature sizes aggressively in SoC systems due to associated performance and economic benefits. However, scaling also entails aggravation of different physical phenomena leading to device failures as demonstrated by Jayanth et. al [8]. Other than permanent failures, slow degradation of devices due to phenomenon like Negative Bias Temperature Instability (NBTI) and Hot Carrier Effects (HCE) may lead to timing failures in systems [39]. Detailed analytical modeling for impact of NBTI on devices validated empirically has been described [11]. Corresponding device degradation models for HCE are presented in [40]. In [41], an analytical model is proposed to estimate the temporal performance degradation of digital circuits, and a sizing algorithm is also proposed to improve performance degradation. Since most of these aging phenomena are significantly affected by the static and switching probabilities of the circuits, there have been solutions to prolong the age by balancing circuit switching for uniform aging all throughout the devices [42]. There have been methodologies analyzing optimization due to NBTI and HCE in isolation, however, as presented by us, the actual circuit degradation needs the conglomeration of the two phenomenon during analysis.

Bus reliability has been studied from the perspective of transient errors [36, 43]. However, to the best of the author's knowledge, this work is the first aging analysis performed for bus systems. Architectural solutions to counter device degradation based timing errors and dynamic variations are presented by Kypros et.al [44]. The age de-

tection scheme we present uses the razor latches presented by Ernst et. al [45]. We use this mechanism to provide dynamically changing BUS architecture which proves to be resilient to timing errors with minimal penalty.

4.3 HANITA: HCE And NBTI Incorporated Tool for ASICs

HANITA is a completely automated ASIC design flow analysis methodology using Synopsis design tools for studying the timing impact of degradation due to HCE and NBTI. The importance of the tool is that it tightly conglomerates the analysis of aging of individual transistors and the overall timing impact of device aging on the hardware design component. Given a RTL description of circuits as input to our tool flow (shown in Figure 4.1), the average switching activity of the gates in the circuit is used to analyze the degradation of the individual devices. Such degradation information of the gates is used to compute the critical path timing of the design in a degraded state. We use Synopsis Design Compiler to obtain synthesized gate level netlist and corresponding EDIF netlist. The EDIF netlist is fed to the ACE tool that calculates static and transition probabilities at all internal nodes (gate inputs in this case) for a given set of input static probabilities. The obtained static and transition probabilities are used to compute the V_{th} degradation for each standard cells using the underlying device degradation models for HCE and NBTI. This work is done using empirically validated analytical models explained subsequently. Based on the V_{th} change, rise and fall delay degradation (percentage delay increase) values for the individual transistors are calculated using a delay model proposed by Kuroda et. al [46]. The obtained rise and fall transition delay scaling

values are updated in the TSMC timing libraries which is used by Synopsis PrimeTime analyzer for timing analysis. Figure 4.2 explains the fall and rise delay scaling calculation for an OR gate due to both NBTI and HCE. Stack effect is taken into consideration for NBTI and HCE degradation calculation for a circuit. Rise delay scaling for the NOR gate (shown in Figure 4.2) due to NBTI degradation factors the stack effect. The timing degradation calculation due to NBTI is based on static probabilities (similar to the work by Kumar et. al [47]) while the HCE involves both static and transitional probabilities.

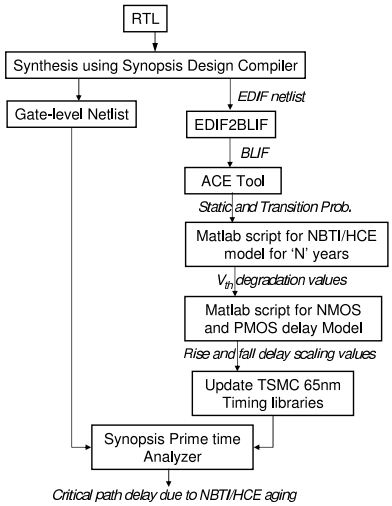

Figure 4.1. CAD flow for NBTI degradation analysis

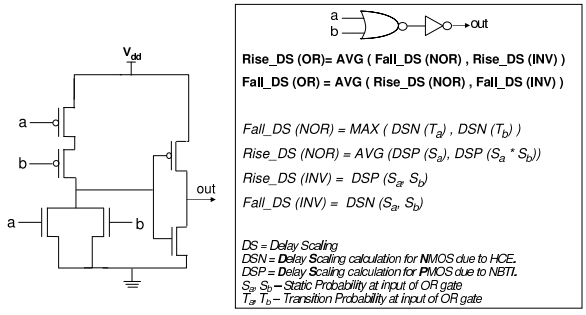

Figure 4.2. Rise and fall delay scaling calculation for an OR library gate

The effect of NBTI on threshold voltage of PMOS transistor is computed using the model presented by Rakesh et. al [11]. Equation 4.1 is used to compute the degradation

of a device under given process parameters, temperature and the duty cycle of the device.

$$\Delta V_{th} = K_v \cdot \beta^{0.25} \cdot T^{0.25} \cdot \left[\frac{1-(1-\sqrt{\eta(1-\beta)/n})^{2n}}{1-(1-\sqrt{\eta(1-\beta)/n})^{2}} \right]^{0.5} + \delta_v \tag{4.1}$$

The HCE model is based upon the model presented in [48]. The model estimates the degradation of the device threshold voltage over a period of time due to interface trap generation. NMOS transistors are observed for such degradation due a more pronounced effect of the HCE phenomena on NMOS transistors. Equation 4.2 demonstrates the threshold degradation of the NMOS device due to HCE-based degradation.

$$\Delta V_{th} \propto P_{sw} \times t^n \tag{4.2}$$

,where P_{sw} is the average switching activity of the gate and the value 'n' governing the growth of the threshold polynomial dependence on time is used as 0.45 as used in [48]. The dependence on switching activity is attributed to the average current flow through the device while the polynomial time dependence is obtained as a result of solving a first order differential equation defining dependence of threshold growth on the rate of trap generation.

	65nm technology
VDD (V)	1.0
Tox (nm)	1.0
Vth (PMOS) (V)	0.20
Vth (NMOS) (V)	0.22
T (k)	353
wire delay per mm (ps)	209

Table 4.1. Technology parameters used for NBTI and HCE analysis

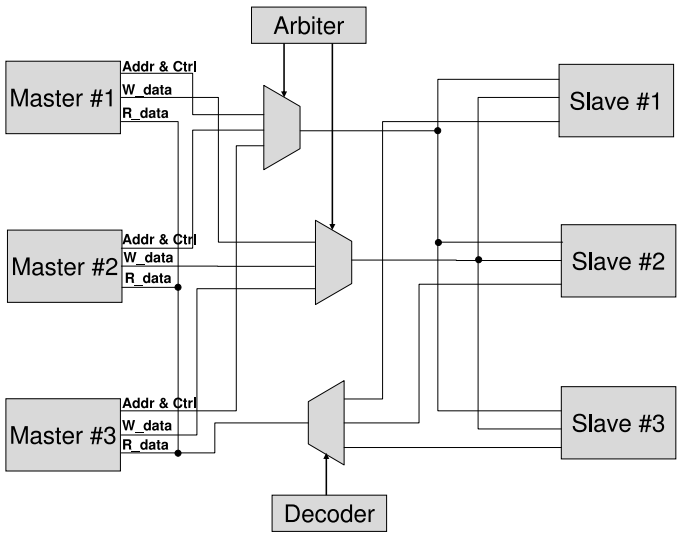

Figure 4.3. AMBA Bus Architecture

4.4 Degradation Analysis of On-chip Bus System

A typical shared bus system has four main components governing the delay of the bus system, namely, arbiter to decide the bus master, decoder unit for address decoding, multiplexers for acting as control switches and large number of link buffers. The critical path delay of such designs which govern the bus frequency, is completely dependent on the bus architecture. Figure 4.3 depicts the architecture of one such commercial bus standard, AMBA AHB [18], which we use to analyze the impact of aging. As depicted in the figure, the basic AMBA AHB bus model has a master-slave-based communication with the masters as the processors cores and the memories and other peripheral devices being the slaves is depicted in Figure 4.3. The bus request cycle from a master to the slave in such an architecture comprises the critical path, as compared to the return path through the address decoding unit. The reason for such an observation is attributed to the fact that the address decoding is performed during the request cycle itself.

Consequently we determine the degradation of the arbiter, the multiplexer and the

link buffers in this work to analyze the frequency violations created by aging. A HDL description of arbiter driving a multiplexer was used for analyzing the degradation. To estimate the degradation of the link buffers we first estimated the number of repeaters and their respective sizes needed for optimal performance of the link using buffer model presented in [49]. These buffers were appropriately modeled in HDL and provided as an input to our degradation analysis tool. The technology parameters for the devices and interconnect are employed from ITRS report [10] for 65nm technology as shown in Table 4.1. The tests were performed using a 65nm delay scaled TSMC library with a scaling model similar to Jayanth et. al [8].

The degradation of a 4-input arbiter driving a multiplexer, the degradation of the link, and the total critical path is demonstrated in Figure 4.4 when operating at a frequency of 500MHz. The inputs were driven by appropriately-sized buffers to simulate the impact of cores and the static probabilities of the requests from each masters is set at 0.5. As shown in the figure, the speed of a 4-master arbiter tends to fall by 10% over a period of 1 to 5 years for a continuous usage of the bus. Such a degradation in critical path may lead to slack violations leading to either system failures or even deadlocks due to inconsistent and false transactions as demonstrated in [43]. It is important to note that these numbers project the degradation under continuous operation of the system bus which is quite a realistic scenario considering the employment of SoC based designs in real time systems, like wireless LANs, cell phones, etc. Such an observation highly motivates techniques to either prolong the age of the buses or increase the resilience of the buses to such timing violations.

Another important observation is presented in Figure 4.5. The figure demonstrates

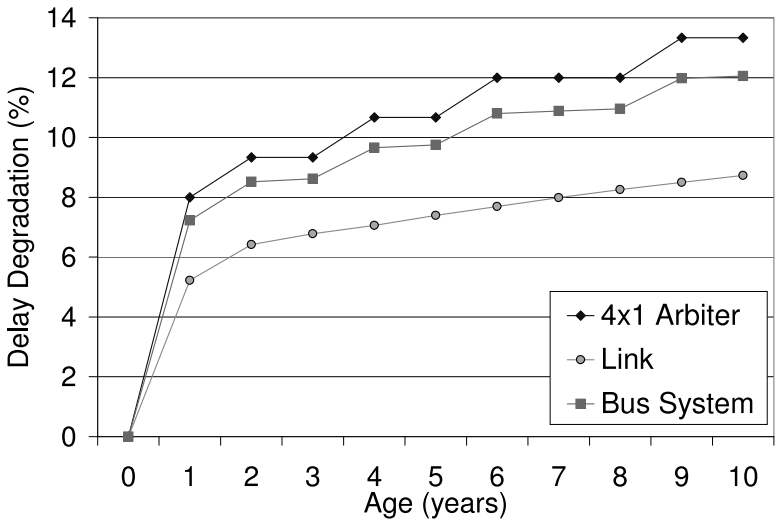

Figure 4.4. Degradation of arbiter over a period of time

the impact of NBTI and HCE individually on a 4x1 arbiter compared to the total delay impact when both of them are analyzed in conjunction with our tool. An important observation is the fact that the degradation of circuits is primarily dominated by NBTI degradation as compared to HCE for 65nm technology nodes. However, since NBTI affects the pull-up circuit and HCE affects the pull-down circuit, the critical path obtained from combined analysis will be different from separate analysis. This is evident from the different delay degradation values for the combined and individual analysis as seen from Figure 4.5. Hence, it is necessary to analyze them together to obtain delay degradation due to aging in all parts of the circuit.

We analyzed the impact of increasing the number of cores on the circuit path delay degradation. Figure 4.6 demonstrates the degradation of a single shared bus connecting 4 and 8 cores connected to same number of slaves. It is evident from Figure 4.6 that circuit degradation increases with the complexity of circuit due to the increased current flow.

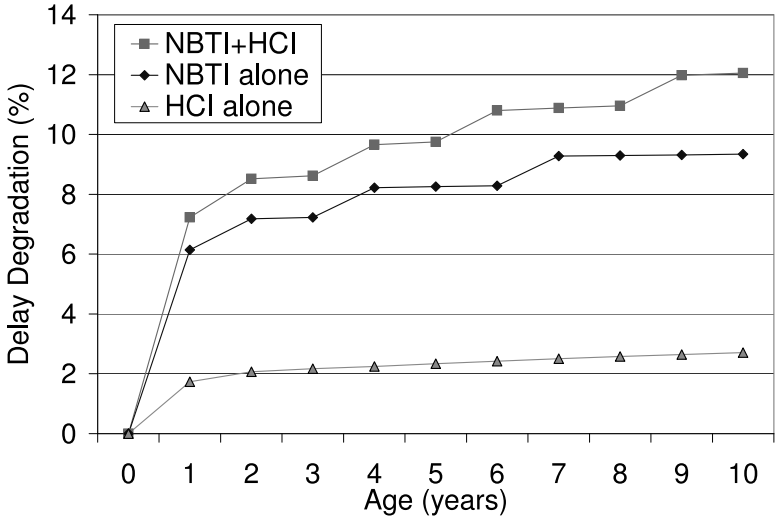

Figure 4.5. NBTI/HCE degradation in isolation and the total degradation

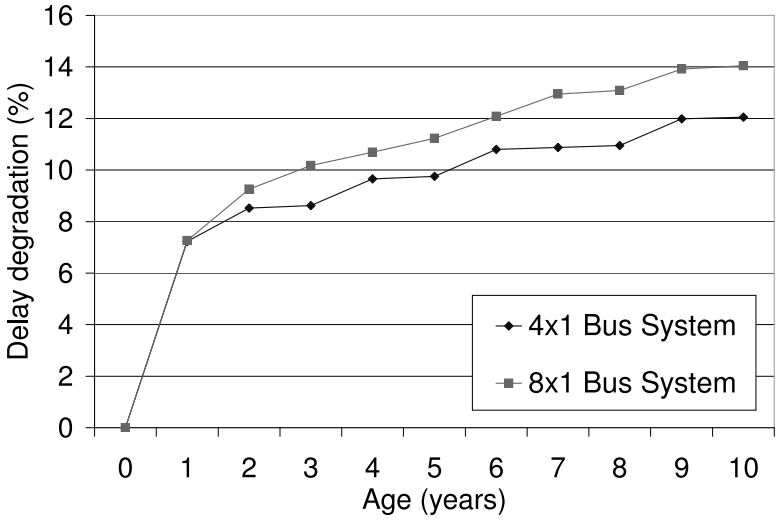

Figure 4.6. Total degradation due to NBTI/HCE for 4x1 and 8x1 arbiter bus systems

4.5 Countering Aging in Bus System

Solving the problems of degradation needs focus on the dynamic detection or a trigger mechanisms to flag the system degradation followed by a solution or response to such a trigger. We demonstrate a hardware based approach to detect timing failures. Followed by that we describe our schemes to prolong the lifetimes by continual working or recovery schemes.

4.5.1 Detecting Degradation

Software-based timing tests for buses has been explored extensively in [50]. Impact of degradation during runtime may be detected by inserting simple read-and-write transfers in a standalone system. Testing of such systems, however, is particularly complicated due to the operation of the components at different frequencies. However, simple transaction-based timing errors may be completely detected by performing consecutive write-and-read operations to some particular memory addresses and checking for the consistency of the write-and-read values. Such testing may be provided periodically to test timing violations in the system. A testing like this, however, may not be able to capture and flag the errors at run-time dynamically and may lead to the system ending up in fatal crashes due to false transactions or even deadlocks. This motivates us to look for dynamic hardware triggers which may flag timing violations.

We propose to capture the timing degradations in a critical path as soon as there is a setup or hold violation by using a Razor Flip Flop (RFF). Razor Flip Flop based low-power pipeline design techniques, which may dynamically detect and correct circuit timing errors is presented in [45]. A similar technique may be employed in bus systems; however, instead of using all flip flops as RFFs we utilize the knowledge of the critical path in the bus. Such observation enables us to capture the timing degradation of the critical path of the bus with minimal power and area overheads incurred due to RFFs.

The requirements for RFFs completely depend upon the critical path of the bus system. In this work we demonstrate how we may selectively place RFFs in a AMBA bus system to capture timing violations caused by degradation. A similar analysis may

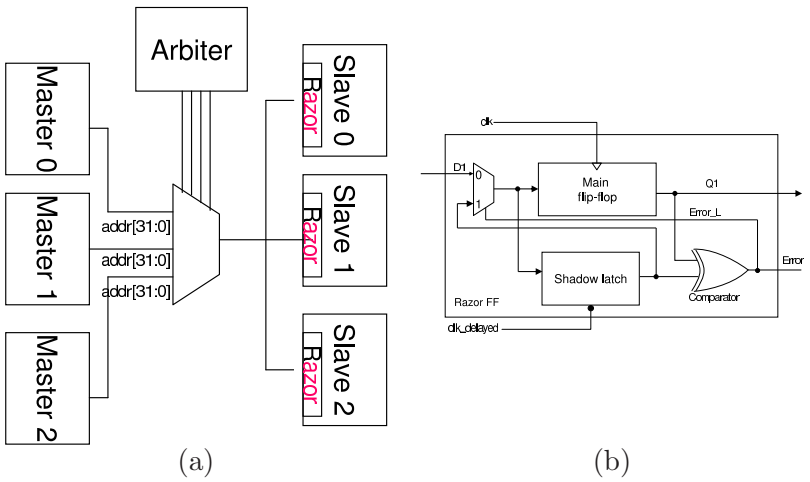

Figure 4.7. (a) AMBA AHB bus with Razor Flip Flop. (b) Razor flip-flop.

be performed in other bus systems due to the similarities in the underlying hardware of most bus architectures. As described in the previous section, the critical path in an AMBA bus system is the combinational logic path involving the arbiter, the multiplexer and the data and address link delays. This motivates us to only have RFFs at the slave end to capture the timing degradation due to NBTI or HCE in AMBA based bus systems. Such a selection of placement of RFFs may, however, vary across bus systems depending upon the design implementation of the critical paths.

A similar solution is effective for other commercial bus systems including the IBM CoreConnect and ST-buses. The number of RFFs, however, is dependent on the configuration of the system. The first timing failure of the system due to critical path degradation may be completely captured by having an RFF at the slave end of either the data bus or address bus. A 32-bit address and data bus system would therefore require $32 * NSLAVES$ RFFs, where NSLAVES is the number of slaves in the system as demonstrated in Figure 4.7. The power overhead of such a change in the hardware is primarily due to the additional delay buffers included to provide a delayed clock to the

slave latch in the RFF. Therefore, the number of such delay buffers per razor flip-flop depend upon the clock frequency. We estimated an average requirement of 300 additional buffers for a bus operating at 500MHz with 4 slaves. The percentage contribution of this to the total power consumption of a 4 core, 4 slave ARM based SoC connected by AMBA interconnect was determined to be 1.9% on average. Note that we may be able to catch transient and non-deterministic errors in non-critical paths in the bus system by having RFFs for every latch in the system. The area overhead may grow tremendously in such a case. However, a transient error in the critical path may disguise as an aging signal. Such a response is handled by having an error counter and resetting it at regular intervals to distinguish between the more severe aging related timing errors from the single event upsets.

4.5.2 PROBUS - Proactive BUS

The concept of PROBUS is based on adaptation of the bus to aging alerts produced by the aging detection trigger mechanism. Architecture depicting the PROBUS is demonstrated in Figure 4.8. PROBUS works on the concept of dynamically changing the protocol of bus transfer to adapt to the degradation of the underlying circuits due to NBTI/HCE effects. To understand the concept of PROBUS, a detailed understanding of commercial bus architectures is quite necessary.

Most commercial bus architectures like AMBA, CoreConnect and Wishbone are burst-based architectures. Architecture of a burst-based system is shown in Figure 4.8. The burst bit as demonstrated in the figure controls the input to the data and address bus multiplexer switches, using another multiplexer logic. The burst-based logic is typ-

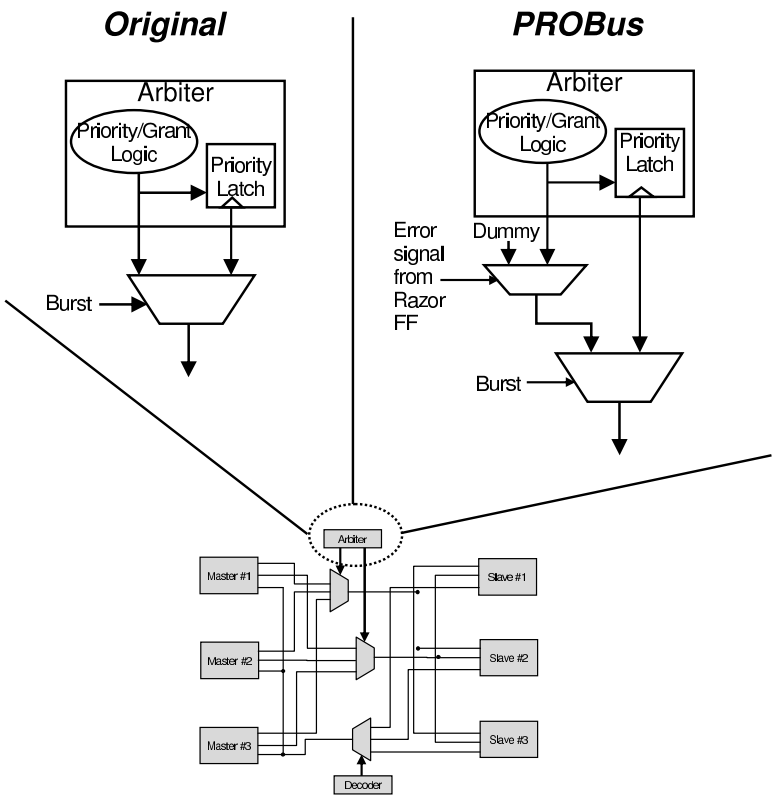

Figure 4.8. Original and PROBUS architectures

ically embedded in most arbiters of such bus systems. Consequently, in a burst-based communication the arbitration delay degradation will be logically masked due to the bus grant provided to the master in all the burst cycles except for the first. Therefore, the errors occurring in the degraded buses will primarily be during the transfer cycles due to false transfers caused by delayed arbitration. In fact, the delayed arbitrations would cause the transfers to or from the previous bus master or some corrupted master due to setup or hold violations.

The proposed PROBUS architecture is demonstrated in Figure 4.8. As shown in the figure, an additional multiplexer is used to govern the bus grant. These inputs to the multiplexer as demonstrated in the figure are the original inputs and a special grant

signal to the component is called *Dummy master*. Dummy master is typically used in most contemporary commercial bus systems for modeling a state emulating hardware. Dummy master does not send any request to the bus and, when granted by the arbiter, it signals an IDLE transfer. In cases of SPLIT transfers or no bus requests when all the masters are unable to access the bus, the arbiter signals a grant to Dummy Master. Since the degraded bus may not be able to provide the grant immediately during bus burst switching, the grant to Dummy Master acts as a shield in the first transfer cycle to prevent incorrect transactions. Therefore, to avoid the timing error explained earlier during the bus handover cycles, we propose to modify the arbiter to interleave each such hand-overs by a Dummy Master. A run-time decision is taken to pro-actively switch to the Dummy master during the interleaving cycles of the bus. This decision is based on the error signal flagged by a Razor latch on detecting a timing failure that might be caused by degradation failure. Such a concept of PROBUS may be easily adopted in commercial bus system architectures. Note that dummy grants during every transfer cycle may suffer from significant unwanted latency penalties, particularly in low burst systems, due to the introduction of idle cycles more frequently. Therefore, the PROBUS architecture is most useful when a system degrades significantly. If the system does not degrade significantly, it is better to use the frequency adjustment scheme first as will be described in Section 4.6. This observation is experimentally demonstrated in Section 4.6.

PROBUS architecture also provides an opportunity to increase the operating frequency of the bus system. This may be justified by the fact that the PROBUS system architecture splits the critical path in the original system to arbiter and the link delays separately. Note that such an increase in frequencies may be completely dependent on

the bus system design. However, implementation of the frequency change may once again be triggered by the error flag of the RFFs.

We modeled the AMBA bus system as a PROBUS using a SystemC model of AMBA bus core. AHB is a multi-master bus, and only one master is allowed to perform a transfer operation at one time. When a master is ready to transfer data, the master asserts the HBUSREQ signal. Each master has its own HBUSREQ signal, and hence several masters may assert that signals at the same time. The arbiter samples all the HBUSREQ signals at the next rising edge and decides which master has the right to access the bus by asserting the HGRANT signal of that master. Along with the regular masters, which are the system cores, the AHB system also has a dummy master [18] for reasons explained earlier. The aging-based timing failure resiliency of a PROBUS AMBA structure is demonstrated in Figure 4.9. The modified arbiter decides the next master M_{next} to be granted bus control at the end of the original bus handover cycle. However, instead of signaling a grant to M_{next}, the arbiter signals a grant to the dummy master, stores M_{next}, and in the next cycle the arbiter, signals the grant to M_{next}. Note that the only overhead of the PROBUS system is obviously the addition of the Razor latches as explained earlier in the section and an additional 2:1 multiplexer for each of the arbiters in the system. In multi-arbiter based bus systems like crossbar based models such a multiplexer is introduced for each arbiter, which however has a negligible area or delay impact.

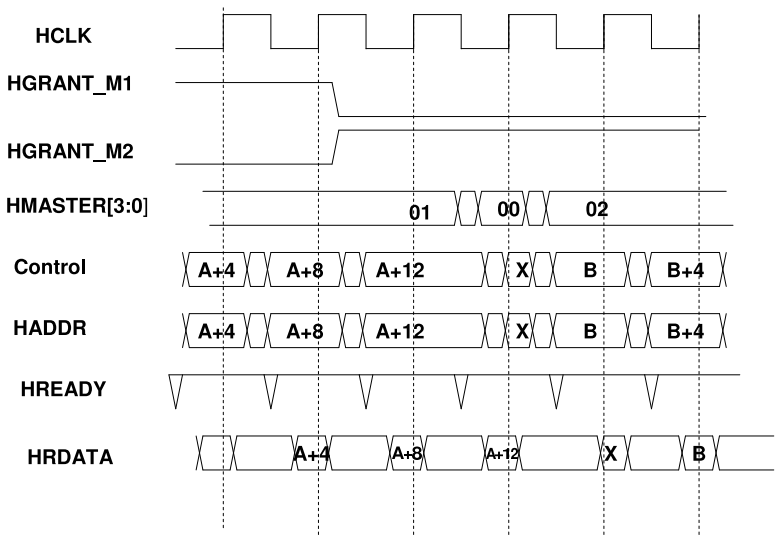

Figure 4.9. Correct transfer after one dummy cycle is added. The HMASTER signal switch from 01 to 00 , and from 00 to 02.

4.5.3 Frequency Adjustment Scheme

We propose to use a frequency adjustment scheme to prevent timing failures of the system due to device degradation. The bus system may be slowed down to ensure that the required timing is met even after degradation. The frequency reduction is triggered only when the system detects the degradation is sufficient enough to cause timing failures. In our case, we may use the trigger from the RFF latches for triggering the frequency scaling. However, such a frequency adjustment scheme entails an overhead in the form of slowing down the system and hence increases the execution time of the applications implemented on it. The impact of the frequency scaling scheme on the runtime of applications is demonstrated in Section 4.6. Hardware costs associated with such frequency adjustment scheme are presented in [51], where similar Dynamic Frequency Scaling (DFS) is achieved by using frequency dividers or PLLs (Phase Loop Lock).

4.6 Experimental Setup and Results

We experimented the proposed schemes in a cycle-accurate SoC simulator [24] which has ARM cores that communicate using an AMBA based interconnect with slave memories. The AMBA bus architecture is modified to model the PROBUS architecture as explained in Section 4.5. Six parallel applications are implemented in the bus system to demonstrate the average execution time penalty of the PROBUS and the Frequency Adjustment Scheme (FAS). Note that we consider the execution time as a metric of evaluation in such systems because different components are operated at different frequencies.

Figure 4.10 demonstrates the average increase in the execution time of the applications normalized to the original execution times for the FAS and PROBUS scheme when operated at normal and upgraded frequencies. The frequency used for the FAS scheme is 75% of the original frequency which is one of the rated frequencies of AMBA bus. Based on our degradation studies, 75% frequency is sufficiently large for preventing timing failures due to NBTI and HCE in the bus system. The figure also depicts the results of PROBUS at two different frequencies, the original and the increased frequency, which is 1.3 times the original. This new frequency is estimated based upon the new critical path delay of the PROBUS which changes as explained in Section 4.5. The execution times in the figure are normalized to a bus originally operated at 250MHz. Average performance penalty of 26% is observed across the benchmarks for PROBUS and 36% for FAS implementations at 75% of original frequency . The execution time overhead of the PROBUS scheme prevents the usage of such a bus system from the beginning. Another important observation is the fact that a PROBUS operating at a higher frequency provides a faster

execution of the same applications. However, such a design may fail much faster under degradation due to stringent timing constraints and lower slacks. Moreover, there is no rescue from such timing failures if they occur. The average lifetime of a faster PROBUS system was estimated to be 1.12 times lower than the original bus structure under similar pipeline slack assumptions.

We also performed a study of the energy consumption of the different solutions, and the results are shown in Figure 4.11. It is evident from Figure 4.11 that both the PROBUS and FAS scheme have minimal energy penalty on average.

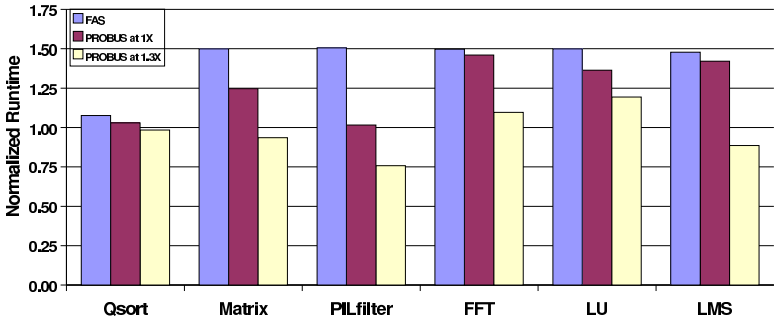

Figure 4.10. Normalized performance impact

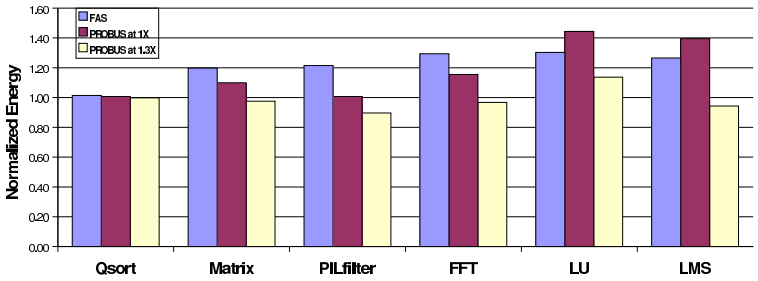

Figure 4.11. Normalized energy impact

4.7 Conclusion

This work presents a comprehensive study of device degradation impact of NBTI and HCE on SoC bus systems. We demonstrate an average increase in the critical path delay

of the bus system by 10% over a period of 5 years, which may lead to timing failures of the bus system. In addition, we present a PROactve BUS design which dynamically detects such degradation of the system and transforms to an architecture resilient to such errors at a minimal penalty on performance. We demonstrate that such a scheme may prove to be more effective as compared to a simple lowering frequency-based approach.

Chapter 5

Conclusion and Future Work

This chapter summarizes previous chapters, reviews the limitations, and makes recommendations for future research. The chapter concludes with future challenges for system level design.

5.1 Transaction Level Power Modeling

5.1.1 Chapter Summary

As the transistors on a chip increase exponentially, and power consumption becomes a firs-class design constraint. Power optimization at lower levels, such as register-transfer or gate level achieves only 20% reductions because the simulation is slow and cannot do system wide exploration and reduction. However, power optimization at the system level can achieve 10 to 20 times the reduction. Therefore, it is important to do power reduction at the system level.

In Chapter 2, the author first presented a power estimation methodology in the

transaction level models written in SystemC. We demonstrated the methodology in PCI Express architecture and integrated the PCI Express simulation platform with the IBM CoreConnect transaction level models. The IBM CoreConnect platform provides plentiful industrial-strength transaction level models for IP cores. With the ability to communicate between PCI Express and IBM CoreConnect platform, we can investigate various application scenarios and explore a greater design space. The main constituents of this work include transaction level models for PCI Express, a hierarchical representation of transaction level data, a power model mapping mechanism to augment transaction level models with power information, and a simulation platform to explore performance and power trade-offs. The experimental results from CoreConnect TLMs demonstrated the validity of this approach, providing a starting point for further exploration of transaction level power estimation. This power estimation methodology for transaction level models provides rapid power estimation and assists designers in making informed decisions at the early design stage.

5.1.2 Limitations and Future Work

Our power estimation methodology deals with power modeling at the transaction level. Contrary to traditional power modeling at the gate or register-transfer level, which is more accurate but more time consuming, our methodology provides fast but not so accurate power estimation. Cases arise when users prefer more accurate power modeling. Therefore, it becomes necessary to provide a more accurate modeling in our power estimation methodology. Future work should focus on developing a mixed model allowing an easy switch from the higher level model to the lower level model when accuracy is

desired. By integrating a lower level power modeling and providing switch mechanisms in our models, the users can decide the models they would like to use and the accuracy they would like to achieve.

5.2 Transaction-based Reliability Modeling for Bus-based SoC

5.2.1 Chapter Summary

Like Chapter 2, which proposed a power estimation methodology at the transaction level, Chapter 3 discussed the error susceptibility for bus-based SoC at the transaction level. We completed this work on a cycle-accurate multi-processor system-on-chip simulator for an AMBA AHB bus. This chapter illustrated a novel and effective transaction-based error characterization scheme for bus architectures in SoCs. The scheme characterized the error susceptibility of control and address signals. The measure of error susceptibility for each signal provides the opportunity to prioritize the employment of error correction schemes on the system, which is advantageous because of shrinking power and area budgets. In addition, the error prediction model we proposed calculates the probability of any single-bit error on the system with an accuracy of 92% on average, for multi-bit errors, we found the accuracy to be 90%.

5.2.2 Limitations and Future Work

Since there are no comparable simulation platforms for other industry on-chip buses such as IBM CoreConnect and ST-Bus, we could compare the work to other systems. Future

research can extend this work to show the error susceptibility of other bus systems.

We can improve the accuracy of power modeling by utilizing a finer-grained model. Currently, only error susceptibility for "read" and "write" transactions are identified. In the future, we can improve this by characterizing the error susceptibility at a lower level, such as "single read," "burst read" and "split read."

Our results showed the error susceptibility of each signal in the bus system. With this knowledge, a selective error detection and correction can be constructed to provide maximum reliability improvement with limited power and area resources.

5.3 System Level Modeling for Device Degradation

5.3.1 Chapter Summary

Chapter 4 addressed the degradation due to NBTI and HCE on the system level, which is the first work of this kind. Through this research, we created circuit analysis tools to model the run-time degradation of circuits due to NBTI and HCE. We analyzed the precise degradation of circuits and observed timing deterioration due to aging. We demonstrated how NBTI and HCE affected the system and how device degradation may lead to increased critical path delays of the buses in a AMBA SoC environment. We also proposed techniques to detect timing degradation in buses with minimal hardware overheads. In order to counter the degradation problem, we proposed a PROactive BUS architecture that dynamically configures itself to an error resilient system.

5.3.2 Limitations and Future Work

Degradation analyses for other industry on-chip buses, such as IBM CoreConnect or ST-Bus, are not completed due to the lack of register-transfer or gate level codes for these buses. Future work can focus on the degradation analysis for these buses if the resources are available.

Another future work is the reliability issue on the High-K gate dielectric and metal-gate technology. High-K dielectrics and metal-gates are a new process technology [52]. In this technology, traditional gate polysilicon and silicon dioxide materials are replaced with high-K dielectric and metal-gate materials, one example of which is Hafnium-based high-K material in the gate dielectrics. Intel claims they are going to produce their 45nm microprocessor using Hafnium-based High-K dielectric and metal-gate technology in mid-2007 [53].

This technology improves transistor-switching speed by 20 percent, and 30 percent reduction in transistor-switching power. In addition, this technology provides greater than 5 times reduction in source-drain leakage power and 10 times reduction in transistor gate oxide leakage [52]. However,the reliability is still an issue in this technology. For instance, the threshold voltage changes caused by the NBTI effect are more significant than the traditional NBTI in SiO_2 [54]. Future work can focus on building a tool to analyze the NBTI effect at the system level for this technology and providing techniques to dynamically switch off transistors to recover the NBTI effects.

5.4 Future Challenges

Systems are becoming more complicated and manufacturing cost is increasing. At the same time, the time-to-market pressure is growing. These factors make mistakes in design so expensive that they are almost unacceptable. Therefore, engineers must design correctly the first time and avoid any errors. However, the gap between specification and implementation is growing, making it more difficult to make accurate predictions early in the design. In addition, design at a lower level such as register-transfer level and gate level is time-consuming and cannot achieve optimal results. Therefore, system level design is becoming more important as a way to bridge this gap and avoid costly errors.

The purpose of this thesis is to address power, reliability and degradation problems at the system level. However, there are still other problems that need to be addressed. One example is process variation that changes the operational characteristics of circuits. Although researchers have proposed various design techniques to address process variations at lower levels such as circuit and logic levels, it will become imperative to address this problem earlier in the design stage as process variations increase. As technology advances, future research will have to address new and more complicated design challenges.

Bibliography

[1] SIA (1997), "The National Technology Roadmap for Semiconductors," SEMATECH.

[2] Gunther, S., F. Binns, D. Carmean, and J. Hall (2001) "Managing the Impact of Increasing Microprocessor Power Consumption," in *Intel Technology Journal, Q1.*

[3] Cai, L. and D. Gajski (2003) "Transaction Level Modeling: An Overview," in *Proceedings of International Conference on Hardware/software Codesign and System Synthesis (CODES+ISSS).*

[4] Donlin, A. (2004) "Transaction-Level Modeling: Flows and Use Models," in *Proceedings of International Conference on Hardware/software Codesign and System Synthesis (CODES+ISSS).*

[5] Anderson, D., R. Budruk, and T. Shanley (2001) *PCI Express System Architecture*, MindShare, Inc.

[6] "IBM PowerPC 405 Evaluation Kit with CoreConnect System TLMs," This software is available at http://www-128.ibm.com/developerworks/power/pek/?S_TACT=105AGX16.

[7] McNally, J., "System-level tools slash SoC dynamic power," This document is available at http://www.eetimes.com/infocus/mixedsignals/OEG20040122S0025/.

[8] Srinivasan, J., S. V. Adve, P. Bose, and J. A. Rivers (2004) "The Impact of Technology Scaling on Lifetime Reliability," in *Proceedings of International Conference on Dependable Systems and Networks (DSN).*

[9] Blish, R., T. Dellin, S. Huber, M. Johnson, J. Maiz, B. Likins, N. Lycoudes, J. McPherson, Y. Peng, C. Peridier, A. Preussger, G. Prokop, and L. Tullos (2003), "Critical Reliability Challenges for the International Technology Roadmap for Semiconductors," International Sematech Technology transfer 03024377A-TR, The International Technology Roadmap for Semiconductors.

[10] http://www.itrs.net/Links/2005ITRS/Home2005.htm (2005) *International Technology Roadmap for Semiconductors, Tech. rep.*

[11] Vattikonda, R., W. Wang, and Y. Cao (2006) "Modeling and Minimization of PMOS NBTI Effect for Robust Nanometer Design," in *Proceedings of Design Automation Conference (DAC)*.

[12] Grötker, T., S. Liao, G. Martin, and S. Swan (2002) *System Design with SystemC*, Kluwer Academic Publishers.

[13] Burton, M. and A. Donlin (2001) "Transaction-Level Modeling: Above RTL Design and Methodology," in *available at http://www.systemc.org*.

[14] Jacome, M. and H. Peixoto (2004) "A Survey of Digital Design Reuse," in *IEEE Design & Test of Computers* .

[15] Bricaud, P. (1999) "IP Reuse Creation for System-on-a-Chip Design," in *Proceedings of IEEE Custom Integrated Circuits Conference*.

[16] Savage, W. (2000) "IP Reuse in the System on a Chip Era," in *Proceedings of Internatoinal Symposium on System Synthesis*.

[17] IBM, "The CoreConnect Bus Architecture White Paper," This document is available at http://www.chips.ibm.com/products/coreconnect.

[18] "ARM AMBA Specification (rev2.0) and Multi-layer AHB specification," This document is available at http://www.arm.com/products/solutions/AMBA_Spec.html.

[19] Gajski, D., J. Zhu, R. Domer, A. Gerstlauer, and S. Zhao (2000) *SpecC: Specification Language and Methodology*, Kluwer Academic Publishers.

[20] Caldari, M., M. Conti, M. Coppola, S. Curaba, L. Pieralisi, and C. Turchetti (2003) "Transaction-Level Models for AMBA Bus Architecture Using SystemC 2.0," in *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*.

[21] The Open SystemC Initiative, "SystemC version 2.0.1," This software is available at http://www.systemc.org.

[22] ———, "IEEE 1666 SystemC Standard Language Reference Manual," This document is available at http://www.systemc.org or http://standards.ieee.org/getieee/1666/index.html.

[23] (PCI-SIG), T. P. C. I. S. I. G. (2003), "PCI Express Base Specification 1.0a," This document is available at http://www.pcisig.com/specifications/pciexpress/.

[24] Loghi, M., F. Angiolini, D. Bertozzi, L. Benini, and R. Zaffalon (2004) "Analyzing On-Chip Communication in a MPSoC," in *Proceedings of Design Automation and Test in Europe (DATE)*.

[25] Dhanwada, N., I.-C. Lin, and V. Narayanan (2005) "A power estimation methodology for SystemC transaction level models," in *Proceedings of the International conference on Hardware/software codesign and system synthesis (CODES+ISSS)*, pp. 142–147.

[26] Dhanwada, N., R. Bergamaschi, W. Dungan, I. Nair, P. Gramann, W. Dougherty1, and I.-C. Lin (2006) "Transaction-Level Modeling for Architectural and Power Analysis of PowerPC and CoreConnect based Systems," in *IEEE Journal of Design Automation for Embedded Systems (JDAES)*, vol. 10, pp. 105–125.

[27] Srinivasan, S., L. Li, and V. Narayanan (2004) "Simultaneous Partitioning and Frequency Assignment for On-Chip Bus Architectures," in *Proceedings of Design Automation and Test in Europe (DATE)*.

[28] Shepard, K. and V. Narayanan (1996) "Noise in Deep Submicron Digital Design," in *Proceeding of International Conference on Computer Aided Design (ICCAD)*, pp. 524–531.

[29] Ramanarayanan, R., V. Degalahal, N. Vijaykrishnan, M. Irwin, and D. Duarte (2003) "Analysis of Soft Error Rate in Flip-Flops and Scannable Latches," in *Proceedings of ASIC Conference*.

[30] Bertozzi, D. and L. Benini (2004) "Xpipes: A Network-on-Chip Architecture for Gigascale Network-On-Chip," in *IEEE Circuits and Systems Magazine, Vol 4, No.2*, pp. 18–32.

[31] "The WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores," This document is available at http://www.opencores.org/projects.cgi/web/wishbone/wishbone.

[32] Metra, C. and B. Ricco (2001) "Optimization of Error Detecting Codes for the Detection of Crosstalk Originated Errors," in *Proceedings of Design Automation and Test in Europe (DATE)*.

[33] Metra, C. and M. Favalli (1999) "Bus Crosstalk Fault-Detection Capabilities of Error-Detecting Codes for On-Line Testing," *IEEE Transaction on VLSI Systems*, pp. 392–396.

[34] Svensson, C. (2001) "Optimum Voltage Swing on On-Chip and Off-Chip Interconnect," *IEEE Journal of Solid-State Circuits*, pp. 1108–1112.

[35] Anghel, L. and M. Nicolaidis (2000) "Cost Reduction and Evaluation of Temporary Faults Detecting Technique," in *Proceedings of Design Automation and Test in Europe (DATE)*.

[36] Bertozzi, D., L. Benini, and G. D. Micheli (2002) "Low Power Error Resilient Encoding for On-Chip Data Buses," in *Proceedings of Design Automation and Test in Europe (DATE)*.

[37] Weaver, C., J. Emer, S. Mukherjee, and S. K. Reinhardt (2004) "Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor," in *Proceedings of International Symposium on Computer Architecture (ISCA)*.

[38] Bisdounis, L., C. Dre, S. Blionas, D. Metafas, A. Tatsaki, F. Ieromnimon, E. Macii, P. Rouzet, R. Zafalon, and L. Benini (2004) "Low-power system-on-chip architecture for wireless LANs," in *IEE Proceedings of Computers and Digital Techniques*, pp. Volume: 151, Issue: 1.

[39] Goodman, D. (2001) "Prognostic Methodology for Deep Submicron Semiconductor Failure Modes," in *IEEE Transactions on Components and Packaging Technologies*.

[40] Bouchakour, R., L. Hardy, I. Limbourg, and M. Jourdain (1996) "Modeling and characterization of the nMOS transistor stressed by hot-carrier injection," in *Proceeding of International Symposium on Circuits and Systems (ISCAS)*.

[41] Paul, B., K. Kang, H. Kufluoglu, M. Alam, and K. Roy (2006) "Temporal Performance Degradation under NBTI: Estimation and Design for Improved Reliability of Nanoscale Circuits," in *Proceedings of Design Automation and Test in Europe (DATE)*.

[42] Srinivasan, S., P. Mangalagiri, Y. Xie, and N. Vijaykrishnan (2006) "FLAW: FPGA Lifetime AWareness," in *Proceedings of Desigin Automation Conference (DAC)*.

[43] Lin, I., S. Srinivasan, V. Narayanan, and N. Dhanwada (2006) "Transaction Level Error Susceptibility Model for Bus Based SoC Architectures," in *International Symposium on Quality Electronic Design (ISQED)*.

[44] Constantinides, K., S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky (2006) "BulletProof: a defect-tolerant CMP switch architecture," in *Proceedings of High-Performance Computer Architecture (HPCA)*.

[45] Ernst, D., N. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge (2003) "Razor: A Low-power Pipeline Based on Circuit-level Timing Speculation," in *Proceedings of International Symposium on Microarchitecture (MICRO)*.

[46] Kuroda, T. (2002) "Low-Power, High-Speed CMOS VLSI Design," in *Proceedings of International Conference on Computer Design (ICCD)*, pp. 310–315.

[47] Kumar, S. V., C. H. Kim, and S. S. Sapatnekar (2007) "NBTI-Aware Synthesis of Digital Circuits," in *Proceedings of Design Automation Conference (DAC)*.

[48] Jean, Y. and C. Wu (1997) "The threshold-voltage model of MOSFET devices with localized interface charge," in *Proceedings of IEEE Transactions on Electron Devices*.

[49] Kapur, P., G. Chandra, and K. Saraswat (2002) "Power estimation in global interconnects and its reduction using a novel repeater optimization methodology," in *Proceedings of the Design Automation Conference (DAC)*, pp. 461–466.

[50] Singh, V., M. Inoue, K. Saluja, and H. Fujiwara (2003) "Software-Based Delay Fault Testing of Processor Cores," in *Proceedings of the Asian Test Symposium*.

[51] Semeraro, G., G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. Scott (2002) "Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA)*, p. 29.

[52] "Intel's 45nm Transistor Technology Breakthrough," This document is available at http://www.intel.com/pressroom/kits/45nm/index.htm.

[53] "Intel's Transistor Technology Breakthrough Represents Biggest Change to Computer Chips In 40 Years," This document is available at http://www.intel.com/pressroom/archive/releases/20070128comp.htm.

[54] Harris, H., R. Choi, B. Lee, C. Young, J. Sim, K. Mathews, P. Zeitzoff, P. Majhi, and G. Bersuker (2004) "Recovery of NBIT degradation in HfSiON/Metal Gate Transistors," in *IEEE International Integrated Reliability Workshop Final Report*.

Vita

Ing-Chao Lin

Education

The Pennsylvania State University (PSU), University Park 2002-Present
Ph.D. in Computer Science and Engineering (CSE)
Thesis Advisor: Dr. Vijaykrishnan Narayanan
Thesis Title: System Level Power and Reliability Modeling
National Taiwan University (NTU), Taipei, Taiwan 1999-2001
M.S. in Computer Science and Information Engineering
National Taiwan Normal University (NTNU), Taipei, Taiwan 1994-1998
B.Ed. in Information and Computer Education

Research Positions

Research Assistant, the Pennsylvania State University Jan. 2005-Aug. 2006
Department of Computer Science and Engineering Jan. 2004-May 2004
Created a transaction-based error characterization model for bus-based System-on-Chip
Proposed a power estimation methodology for PCI Express transaction level models

Teaching Positions

Teaching Assistant, the Pennsylvania State University Aug. 2006-Dec. 2006
Department of Computer Science and Engineering
Instructor for Digital Design Laboratory course
Teaching Assistant, the Pennsylvania State University Aug. 2003-Dec. 2003
Department of Computer Science and Engineering
Teaching Assistant for Operating Systems course

Industrial Experience

Co-Op, IBM Electronic Design Automation Laboratory May 2004-Nov. 2004
Hudson Valley Research Park, New York
Implemented a Power Estimation Methodology for IBM CoreConnect SystemC TLMs

Certificates

Certificate of System-on-Chip Design May 2006
Issued by the Department of Computer Science and Engineering, PSU
Certificate of Teacher of Computer Science July 1999
For Middle School Education
Issued by Ministry of Education, Taiwan