

Formal Verification of an IBM CoreConnectTM Processor Local Bus Arbiter Core *

Amit Goel
Department of Electrical and Computer
Engineering
Hammerschlag Hall
Pittsburgh, PA 15213
agoel@ece.cmu.edu

William R. Lee
System Level Design Methodology Leader
IBM, ASICs - Microelectronics Division
Mailstop YC6V-667
RTP, NC 27709
brlee@us.ibm.com

ABSTRACT

This paper describes the model checking effort for an arbiter core for the IBM CoreConnect Architecture. We present our verification methodology and describe how it was influenced by the architecture. We also present and analyze the bugs found and discuss the difficulties associated with verifying complex on-chip buses, highlighting the need for better tools and methodologies for their specification and verification.

1. INTRODUCTION

Advances in silicon densities now allow many functions to be integrated onto a complex system-on-a-chip (SOC) design. An SOC design is typically a collection of Intellectual Property (IP) cores (possibly from independent vendors) interconnected by an internal bus architecture. Re-usability of the core often requires the use of a standard on-chip bus architecture. This poses a special design and verification challenge as the cores must communicate with each other using complex protocols, and there is a large possibility of errors in the communication between different cores. The various cores need to have the same interpretation of the bus architecture specification and adhere to it.

The IP Cores need to be verified for compliance to the on-chip bus architecture. A formal specification of the architecture would aid design houses in validating their cores. Current semi-formal specifications often lead to varying interpretations by different designers, thus preventing cores from working together.

*This work was supported in part by the MARCO/DARPA Gigascale Silicon Research Center (<http://www.gigascale.org>). Their support is gratefully acknowledged. CoreConnect is a trademark of International Business Machines Corporation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC 2000, Los Angeles, California
©2000 ACM 1-58113-187-9/00/0006..\$5.00

As bus architectures become increasingly complex, there is a larger possibility for the specifications themselves to have *bugs*. These could be inconsistencies in the specification or the lack of certain *good* properties (e.g. the possibility of a deadlock occurring).

In order to better understand on-chip bus architectures and what is required to specify them precisely for verification purposes, we verified an arbiter core developed by IBM for the IBM CoreConnect Architecture.

Current formal verification techniques have been used for verifying bus protocols as demonstrated by [5, 6, 8]. Several verification efforts have concentrated on verifying the PCI bus [9, 13]. However, most of these efforts have used models built by the authors based on their interpretation of the specification specifically for the verification task. We used an RTL model developed for industrial use by the designers of the specification. This gave us an opportunity to find differences between the potential and intended interpretations.

In section 2 we discuss Symbolic Model Checking and Computation Tree Logic and introduce the model checker SMV (section 2.1). For our verification, we used RuleBase, which is described in section 2.2. Section 3 introduces the IBM CoreConnect Bus architecture with special emphasis on the Processor Local Bus (section 3.1). In section 4 we describe our verification methodology and in section 5 present our results and some of the bugs found. In section 6 we summarize and discuss future work.

2. SYMBOLIC MODEL CHECKING

Model Checking [7] is a formal verification technique for automatically verifying a *model* of a finite state concurrent system against its *specification*.

Verification using model checking consists of three tasks: *Modeling*, *Specification* and *Verification*. First, a formal model is built to represent the system. This model may use abstraction to eliminate details that are not relevant to the verification task. Non-determinism may be used to represent several possible behaviors for the system. Specifications are expressed in a propositional temporal logic. Given a model and the specification, the model checker output identifies whether the model satisfies the rules in the specification. If not, it outputs a *counterexample* that shows why the rule

was not satisfied.

2.1 SMV

SMV [11] is a temporal logic model checker that has its own built-in hardware description language for describing hierarchical finite-state systems. SMV extracts a finite-state model of an SMV program and represents it as a symbolic state-transition graph using Reduced Ordered Binary Decision Diagrams (ROBDDs) [3]. ROBDDs are a canonical representation of boolean formulae that are often more compact than other representations and can be manipulated efficiently. The use of ROBDDs for symbolic representation has been one of the major reasons for the success of symbolic model checking [4].

Specifications in SMV are written in Computation Tree Logic (CTL) [7]. A computation is an infinite sequence of system states. At each state there are many branches (paths) into the future. CTL formulae are composed of:

- Atomic Propositions. An atomic proposition refers to the value of a state variable.
- Boolean connectives: conjunction, disjunction and negation ($\wedge$, $\vee$ and $\neg$).
- Temporal Operators.

Each temporal operator consists of a quantifier and a temporal modality. The quantifier **A** specifies that the property should be true of all paths from the current state whereas **E** indicates that it should be true on at least one path. The temporal modality describes the ordering of events. The basic modalities are:

- $G\varphi$ specifies that φ holds “globally” (on every state in the path).
- $X\varphi$ requires φ to be true in the “next” state (second state on the path)
- $F\varphi$ requires φ to be true sometime in the “future” (should be true on at least one state on the path)
- $\psi U\varphi$ means that ψ holds “until” φ becomes true (there is some state on the path at which φ is true and on all preceding states ψ is true)

2.2 RuleBase

RuleBase [1, 2] is an extension of the SMV model checker, developed by IBM. A model in RuleBase is specified in the Environment Description Language (EDL) which is a dialect of the SMV description language. RuleBase also allows verification of designs in traditional hardware description languages such as Verilog or VHDL.

Rules (specifications) are written in **Sugar** which adds additional operators on top of CTL to make it easier to specify properties for hardware designs, including the ability to enter regular-expression-like sequences to describe computation paths.

RuleBase was used to verify the core implementation in Verilog RTL.

3. IBM CORECONNECT

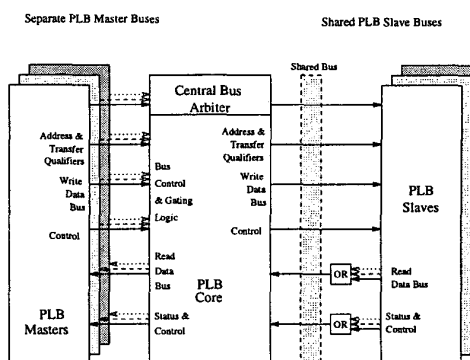


Figure 1: Processor Local Bus (PLB)

CoreConnect [10] is an open on-chip bus architecture introduced by IBM. It provides the framework to efficiently realize complex SOC designs [12]. The CoreConnect architecture provides three buses for interconnecting cores, library macros and custom logic:

- Processor Local Bus (PLB)
- On-Chip Peripheral Bus (OPB)
- Device Control Register (DCR)Bus

High-performance peripherals connect to the high bandwidth, low latency PLB. Slower or device-paced peripheral cores connect to the OPB, which reduces traffic on the PLB, resulting in greater overall system performance. The DCR Bus provides a relatively low-speed data path for initialization, control and status information.

3.1 Processor Local Bus

The PLB interconnects high-bandwidth devices such as processor cores and external memory interfaces. Its features include:

- Decoupled address, read data and write data buses
- Concurrent read and write transfers
- Address pipelining using primary and secondary requests
- Multiple masters and slaves
- Bus arbitration-locking mechanism
- Slave forced PLB re-arbitration
- A sequential burst protocol

PLB based systems require a PLB core to interconnect the various masters and slaves, as shown in figure 1. Each PLB master is attached to the PLB core via separate address, read data and write data buses and several transfer qualifier signals. PLB slaves are connected to the PLB core via a shared but decoupled address, read data and write data buses and transfer signals. The PLB core consists of a bus arbitration control unit and the logic necessary to control the address and data flow through the PLB.

The PLB is a fully synchronous bus. A PLB clock is defined and all signals are referenced to the rising edge of this clock.

The Processor Local Bus Architecture Specifications describes a system architecture and provides a detailed description of signals and transactions.

4. VERIFICATION METHODOLOGY

To verify the PLB core, we needed to do two things:

- Identify the properties required of the arbiter from the English language Processor Local Bus Architecture Specifications and formalize them using temporal logic.
- Verify that the core satisfied the identified properties.

To verify the properties, we needed an environment around the arbiter core, consisting of master and slave devices and modules controlling system signals such as reset. We had available to us the RTL for the PLB core and for various master and slave devices.

One methodology would have been to use the RTL of master and slave devices connected to the PLB core using the setup of a particular SOC design. The model checker could then verify if the core satisfied the identified properties for this SOC design. A similar verification task could be carried out for each new SOC design in which the PLB core was used by using a new setup and the already identified properties. However, this would be a costly procedure. Not only would the PLB core have to be verified for each new setup, it would be possible for bugs to escape detection for a long time if the initial designs it was used in did not make use of the buggy portion of the core.

What we did, instead, was to verify the real RTL PLB core once for all possible setups. To enable this, we needed an environment that encapsulates all behavior that is legal according to the architecture specification.

We did this by building master and slave models written in RuleBase's Environment Description Language. We built the models to conform to the PLB architecture specifications. The use of non-determinism enabled us to write general models that capture all legal behavior for these modules.

The PLB architecture allows multiple master devices, each connected to the arbiter using an independent interface. The interface for each master is similar and so is the set of legal behaviors for each of the masters. Hence, we built one generic model for the master and used multiple instantiations of this model in our setup.

A similar approach could have been used for slaves. However, since the slave devices interact with the arbiter using a shared interface, we wrote a model that encodes the behavior for the shared signals; only one instantiation of this model was then necessary. This model had to handle the requests that in a real system would be handled by multiple slave devices. However, since the address pipelining depth of the architecture is two, it was not too difficult to build that into our model. If more slaves could be active on the PLB concurrently, it would have been easier to write a model for individual slaves and instantiate it multiply.

The setup we used for the verification is shown in Figure 2.

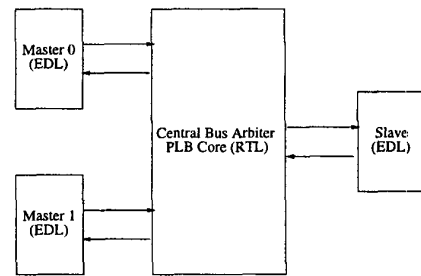


Figure 2: Verification Setup

The most important and difficult aspect of the whole exercise was the extraction of information from the Processor Local Bus Architecture Specifications. It was important because the model checker will only verify those properties that we identify and only under those behaviors that we build into our models for the environment. It was difficult because of the nature of the specification itself. Natural languages are not conducive to writing precise specifications. Also, because of the complexity of the architecture, the specifications are quite voluminous. Information about certain aspects is often distributed and appears in different locations within the specification. As the specification develops and changes are made, the distributed information is at times not updated uniformly to reflect these changes and hence, the specifications can be inconsistent.

Part of the difficulty was that the specification is targeted at designers - it has a lot of information to help them design systems. Most of these are guidelines, saying that some module *should* exhibit particular behavior. In these cases, it is never clear whether the behavior mentioned is required or recommended. In some cases, implementation issues affect the writing of the specification; this often makes it difficult to ascertain if certain details are in the specifications as requirements, guidelines or just examples. Simulation models provided with the CoreConnect toolkit, including bus protocol monitors, provide the resolution to the ambiguity found in the textual specification; however, there is no guarantee that all possible behaviors of a device's bus interface were observed with dynamic simulation methods.

These difficulties clearly highlight the need for better methods and tools to aid in the design and validation of complex protocols such as this.

We used the Processor Local Bus Architecture Specifications to obtain the following information:

- Default values for signals
- When specific signal values change
- Legal transitions

This information was used both for building the master and slave modules and for formally specifying arbiter properties. Any restriction or requirement on a signal controlled by a master or slave was built into the respective module. Any restriction or requirement on a signal controlled by the arbiter lead to a property to be verified.

Whenever there was ambiguity about behavior of a master or slave, we chose the interpretation that would give the largest set of behaviors. The aim was to not miss any legal behavior for the master or slave modules. This ensured we did not miss bugs - however, our verification was pessimistic. This was useful because it helped us identify ambiguities in the specification that could lead to bugs if misinterpreted by designers.

The models for the master and slave were built incrementally. We began with simple models that had limited functionality in order to estimate the resource requirements for the verification. The turnaround time was short enough for us to go through several iterations during a single day, thus allowing us to add behavior in small increments. Each time new behavior was added, verification was performed and in case of bugs, the results were discussed with the bus architects to determine whether the discrepancy was a bug in the implementation, an inconsistency in the specification or an ambiguity that needed cleaning up in the specification document.

We verified several properties for the arbiter. These included properties that the specification identifies the arbiter *must* satisfy for arbitration, bus-locking, timeout and pipelined transfers. Our verification was limited to the verification of control logic and did not verify the core for burst-mode transfers.

5. RESULTS

The model checking of the PLB core was part of the overall verification effort which involved verification by simulation and emulation. Since the PLB arbiter core had already been verified using traditional methods, we expected the number of bugs found to be limited. However, some were found.

One of the bugs was found when checking to see if the arbiter passes on a slave re-arbitrate request during a primary request to the current master of the bus. The CTL formula to express this property is the invariant:

$AG(\text{primary address is valid} \wedge \text{slave rearbitrate} \rightarrow \text{Assert Rearbitration for Current Master})$

The property requires that if there is a valid primary address on the address bus and the slave requests a re-arbitration then the request should be passed on to the current master (the master which requested the transfer).

The counter-example produced showed a case where the slave requests a re-arbitration for a secondary request. According to the specification this re-arbitration request should be ignored. However, the PLB core broke down on this request and when finally (after a long sequence of events) the slave issued a re-arbitrate for the request when it became a primary request, the arbiter failed to pass on the request to the master as required by the property.

Another bug could lead to large latencies for bus-locking requests. We were checking for:

$AG(\text{slave rearbitrate for master } i \rightarrow AX \text{ master } i \text{ ignored})$

$\text{master } i \text{ ignored} \equiv \text{other master request} \rightarrow \text{other master granted bus}$

$\text{other master granted bus} \equiv A[-\text{addressvalid} \cup \text{addressvalid} \wedge \text{current master} \neq i]$

which requires that if a master's request is terminated by a slave re-arbitration request, then the arbiter must ignore it during re-arbitration. "The master is ignored" means that the arbiter must allow another master to gain control of the bus if it has a request pending. This requires that the next time the arbiter indicates that there is a primary address valid on the bus (i.e. the bus is being controlled by some master) the master controlling the bus must not be the re-arbitrated master.

Since a locking request requires both the read and write data buses to be available, if one of them was busy, the bus-locking request would be held off and the master which had its request terminated could regain the bus.

Most of the bugs could be traced to problems in the specification itself. Some of the statements were not consistent with each other while at places conditions laid out were ambiguous or incomplete. Some statements could be interpreted to mean something quite different from what the architects had in mind.

For example, for the property mentioned above, the specifications stated that in case of a slave requested re-arbitration, "the PLB arbiter will ignore the original master request during re-arbitration."

This would mean that if the original master made a different request, then the PLB arbiter need not ignore that; however, the bus architects wanted the PLB arbiter to ignore the original master, not just the original request. Also, after a discussion with the architects, the above property was re-formulated as:

$AG(\text{slave rearbitrate for master } i \rightarrow AX(\text{other master request} \rightarrow \text{current master} \neq i))$

which only requires the original master to be backed off for one cycle; this interpretation was not clear from the specifications.

Problems also arose with the timeout condition which was discussed in two different sections. This led to two different interpretations, neither of which was satisfied by the PLB core. The problem was related to the specification of the slave wait signal; it was not clear from the specification if a slave could assert its wait signal multiple times during a transaction or only once. Even when this issue was clarified, it was difficult to specify the timeout condition that the architects wanted (and the PLB core satisfied) in English.

The specification included two tables outlining arbitration delays under various conditions. We found counter examples for all the cases listed. However, it turned out that these tables were meant to serve as guidelines and not absolute requirements, but this was not clear from the specification.

The problems in the specification could be classified as:

- Ambiguity : Some statements could be interpreted in different ways.
- Incompleteness : The behavior of some signals is specified incompletely.
- Redundancy : Information useful for designers, but not essential for the specification, is included.
- Inconsistency : Separate statements make differing requirements which might be inconsistent.

The verification effort took 10 engineering-weeks. The majority (60%) of this time was spent in understanding the specification and building the master and slave modules. 10% of the time was spent in writing the temporal logic properties. The rest of the time was spent in the actual verification runs, in understanding the counterexamples produced by RuleBase and in discussions with the designers of the arbiter.

Most of the verification runs were performed on an IBM RISC System/6000 model with an 80MHz Power PC 601 Processor with 256 MB of main memory. The machine had a performance rating of 20.3 DP-MFLOPS, 90.5 SPECint92, 100.8 SPECfp92. On this machine, RuleBase took 175.71633 seconds CPU time and used 40 MB to verify a set of properties related to re-arbitration, with a setup using two master modules connected to the PLB core.

The verification did not scale very well with additional masters added. For example, we verified a property that checks that every request is properly terminated. On the above machine and with a two master setup, RuleBase took 866.683 seconds CPU time and used 41 MB. The same property on an IBM RISC System/6000 200 MHz Power PC 630 with 4 GB of main memory ran for 3828.48333 seconds CPU time and used 148 MB with four masters connected to the PLB core. With eight masters connected (the maximum supported by the PLB core), RuleBase ran overnight and still did not complete the verification of the property.

6. CONCLUSIONS

We verified the control logic of the PLB core for the IBM CoreConnect Architecture using RuleBase, a model checker based on SMV. Properties for the PLB core and models for master and slave modules were derived from the Processor Local Bus Architecture Specifications. Burst mode transfers on the PLB were not covered by our models. We were able to perform verification for the PLB core using a setup with two masters. However, the model checking did not scale for an eight master setup.

The bugs found in the PLB core using model checking demonstrate that model checking applied to on-chip bus interface verification is a valuable addition to the overall verification methodology.

We also found inconsistencies and ambiguities in the specification itself, most of which happen because it is difficult to specify a protocol as complex as this in English. Our work highlights the need for better methods and tools to aid in the design and validation of bus architectures.

7. REFERENCES

- [1] I. Beer, S. Ben-David, C. Eisner, D. Geist, L. Gluhovsky, T. Heyman, A. Landver, P. Paanah, Y. Rodeh, G. Ronin, and Y. Wolfsthal. Rulebase: Model Checking at IBM. In *Conference on Computer Aided Verification CAV'97*, pages 480–483, 1997.
- [2] I. Beer, S. Ben-David, C. Eisner, and A. Landver. Rulebase: an Industry-Oriented Formal Verification Tool. In *Proceedings of the 33rd Conference on Design Automation*, pages 655–660, 1996.
- [3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):253–267, August 1986.
- [4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} states and beyond. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 428–439, June 1992.
- [5] S. Campos, E. Clarke, W. Marrero, and M. Minea. Verifying the performance of the PCI Local Bus using symbolic techniques. In *IEEE International Conference on Computer Design*, 1995.
- [6] P. Chauhan, E. Clarke, Y. Lu, and D. Wang. Verifying IP core based System on Chip designs. In *Proceedings of the IEEE ASIC conference*, September 1999.
- [7] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
- [8] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the Futurebus+ Cache Coherence Protocol. Technical Report CMU-CS-92-206, Carnegie Mellon University, OCTOBER 1992.
- [9] P. S. I. Group. *PCI Local Bus Specification Rev. 2.2*. December 1998.
- [10] IBM. The CoreconnectTM Bus Architecture. http://www.chips.ibm.com/product/coreconnect/docs/crcon_wp.pdf.
- [11] K. L. McMillan. Symbolic Model Checking: An Approach to the State Explosion Problem. Technical Report CMU-CS-92-131, Carnegie Mellon University, MAY 1992.
- [12] A. M. Rincon, W. R. Lee, and M. Slattery. The Changing Landscape of System-on-a-Chip Design. In *IEEE 1999 Custom Integrated Circuits Conference*, May 1999.
- [13] E. Solari and G. Willse. *PCI Hardware and Software - Architecture and Design*. Annabooks, 1998.