Using Model Checking for Evaluation of Arbitration Scheme in IBM's CoreConnect Bus Protocol

Ashutosh Mundra[#]
ashu615@yahoo.com

Vineet Sahula[*]
sahula@ieee.org

ABSTRACT

Advances in fabrication technology have enabled the design technology to integrate all major functions of an electronic system on a single chip. Efficiency of such a System-on-Chip (SoC) depends on the efficiency of the bus architecture being used. Hence, it is very important to verify that the bus architecture design follows the specifications mentioned in the requirement specification of the corresponding bus protocol or architecture. This paper is an attempt to verify IBM CoreConnect bus protocol using formal techniques in order to detect bugs at an early stage of design life cycle.

Model checking is a successful technique for checking properties of large finite state concurrent system. Symbolic Model Verifier (SMV) is a symbolic model checking tool, which is used to verify number of real world hardware, but unable to determine timing and performance properties directly. Formal verification of IBM CoreConnect has been performed using NuSMV and SMV. Performance of the arbitration scheme used in bus is evaluated using NuSMV. We also indicate few ambiguities in the specification-document of IBM CoreConnect Processor Local Bus (PLB).

Keywords – IBM CoreConnect, Formal verification, SMV, NuSMV.

I. INTRODUCTION

System-on-Chip technology (SoC) has evolved as one of the most widely accepted implementation technology for complex systems on a single chip. Single-chip integration takes advantage of increased bus speeds and widths. Alongside advantages, it poses a lot of technological challenges like, timing closure, capacity, physical properties and design productivity gap. In order to accelerate the design life cycle, Intellectual Property Cores (IP Cores) are widely used and reused. These IP Cores have requirement of communication among them. To integrate IP cores, bus architecture is a preferred choice, which implements the communication between them.

The evolution to SoC design has thrown new challenges to traditional verification approaches. In SoC, verification has to deal with mixed digital and analog verification and mixed hardware and hardware/software verification.

a. Scope of the work

An efficient communication system is a major performance governing factor of any SoC. Individual modules of the bus architecture run as parallel concurrent systems. Synchronization and communication are the need of such concurrent systems. These modules, which are activity agents, must follow certain rules, so that a desired overall function is performed. Although designer of the protocol takes great pain and care to document the specification and generate specification document, yet there could be ambiguities or deliberate openness in the document, for later implementation. The ambiguities may arise either because of different interpretation by the user during implementation. This paper aims to interpret requirement specification document and indicate any difference in design and development on part of protocol designer.

b. Overview

Most of the recent designs of on-chip buses borrow their ideas from standard buses, especially VME and PCI, which were designed for PCB systems. The bus architecture for a SoC should be different from PCB bus architecture because a SoC has a faster transfer rate due to shorter propagation delays and no restrictions on number of pins due to packaging or signaling constraints. The most popular bus architectures utilize hierarchical levels of buses. For example, CoreConnect has three levels of hierarchy: Processor Local Bus (PLB), On-chip Peripheral Bus (OPB), and Device Control Register (DCR). PLB provides a high performance and low latency processor bus with separate read and write transactions, while OPB provides low speed with separate read and write data buses to reduce bottlenecks caused by slow I/O devices such as serial ports, parallel ports, and UARTs. The daisy chained DCR offers a relatively low-speed data path for passing status and configuration information. An efficient arbitration scheme in bus architecture is also crucial to reducing the contention and increasing performance of the overall system.

Most current bus architectures for SoC have focused on increasing the communication efficiency between the high speed processor bus and the low speed peripheral bus. Yet, for many applications, the performance of multiprocessor system relies more on the efficient communication among processors and a balanced distribution of computation among the processors.

[#] Presently working with Oracle, India. This work was performed, when author was pursuing his B. Tech. (Electronics) at Department of ECE, Malaviya National Institute of Technology, Jaipur-302017.
[*] Reader (Asst. Prof.), Department of ECE, Malaviya National Institute of Technology, Jaipur-302017.

c. Outline

We discuss architecture, arbitration and protocol of Processor Local Bus (PLB) component of IBM CoreConnect bus in Section II. After brief introduction to model checking in Section III, we present the results of verification of PLB protocol functionality and arbitration schemes performance in Section IV. We conclude in Section V.

II. PLB Architecture

IBM CoreConnect 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. A typical schematic of IBM CoreConnect Architecture is shown in Figure 1.

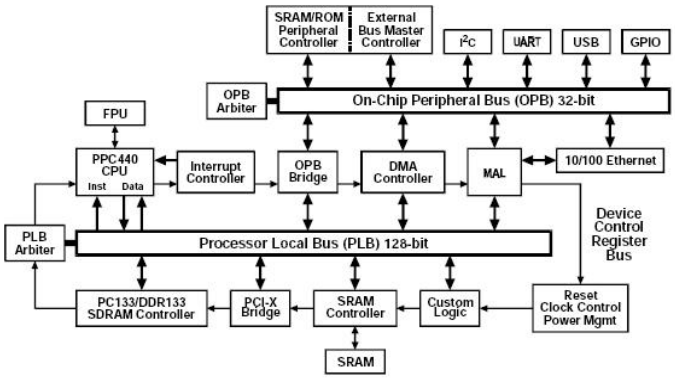

Figure 1 CoreConnect based System-on-a-chip

We focus only on PLB in the paper. PLB is on-chip bus which is used for high bandwidth devices such as processor, external memory interfaces and DMA controller. Each master is connected to the PLB through separate address, read data & write data buses and plurality of transfer qualifiers. Each slave is connected through shared but decoupled addresses read data and write data buses and a plurality of transfer qualifiers. Access to the bus is controlled by an arbiter. Arbitration mechanism is flexible enough to provide various ways of arbitration.

a. PLB Implementation

While implementing Processor Local Bus three different entities are taken into consideration, viz., PLB Master Device, PLB Slave device, PLB core. Master and Slave devices must comply with the IBM CoreConnect Processor Local Bus. These Master and Slave devices are connected to the PLB core. Sixteen masters and any number of slaves can be connected to the PLB core. However, PLB core implementation supporting less than sixteen master devices is allowed. The performance of the PLB core in a system will obviously depend on number of masters and slaves connected to the PLB core. This PLB core consists of the central bus arbiter, which is responsible for the arbitration of the request made by the masters, and the necessary bus control and gating logic.

b. PLB Arbiter and Supported configurability

This central arbitration unit plays very important role in working of any bus protocol. The arbitration is done by the bus control unit of arbiter on the basis of priority signal of all the requesting masters. But in case of tie among the requesting masters, inbuilt arbitration scheme is used in order to grant the bus. Two types of arbitration scheme are supported by the PLB core viz., Fixed and Fair. In case of fixed arbitration priority of each master is fixed and this priority doesn't change with the requests from requesting masters. But in case of Fair arbitration scheme the priority of each master is dynamic which changes in Round Robin manner with the arbiter serving each of the masters.

The architectural specification of Processor Local Bus doesn't talk about the configurability option and arbitration scheme of the arbiter. Arbitration scheme and configurability option depends on the programmer developing the core of PLB arbiter. But this arbitration scheme plays an important role ensuring FAIRNESS, LIVENESS and SAFETY or deadlock avoidance. This arbiter core for high performance Processor Local Bus designed by IBM has its own distinguishing features:

- Four levels of dynamic master request priority.
- Choice between two methods of arbitration (Fixed and Fair/Round Robin).
- PLB address pipelining.
- Programmable high bus utilization feature.

A PLB arbiter core consists of bus control unit, watch dog timer and separate address path and read & write data path units as shown in Figure 2. Bus control unit is the central unit controlling all the activities and arbitration.

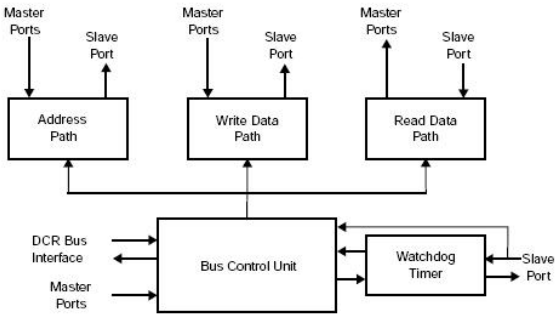

Figure 2 PLB arbiter

III. MODEL CHECKING

The verification ensures that all the requirements as mentioned in the requirement specification are met by implemented design. Verification is one of the major concerns for the designers of SoC. In order to save larger

effort at later stages of design lifecycle, it is very important to trace the bugs at very early phases of designing. Different verification methodologies are used at different levels of designing. To achieve SoC verification goal combination of simulation based technologies, static technologies, formal technologies and physical verification and analysis verification options are used. Formal verification techniques assure 100 percent coverage in a very short time. Formal verification techniques are of three types: Theorem proving, Formal Model checking and Formal Equivalence Checking.

Formal model checking is based on formal mathematical techniques to verify the behavioral properties of a design. A user defines a set a logical properties, which he may extract from the design specification. CTL is used to write properties for the verification. A CTL formula is composed of quantifiers and a logic proposition formula ϕ. Among quantifiers, A and E are path quantifiers, whereas G, F, X and U are temporal quantifiers. Some usual property specifications are mentioned below:

- **AXϕ:** ϕ is true in all the next states (along all paths, ϕ is true in next state)
- **EXϕ:** ϕ is true in at least one next states (there exists a path, where ϕ is true in next state)
- **AFϕ:** ϕ is eventually true in all outgoing paths (along all paths, ϕ is true some time in future)
- **EGϕ:** ϕ is globally true in at least one outgoing paths (there exists a path, where ϕ is true in every state)

A system's model is described in a language supported model checking tool. These properties are compared with design behavior using model checking tool. If the tool finds an error, it provides a complete trace of counterexample.

Formal model verification is suited for verification of control intensive design, but not for datapath intensive design. Datapath in designs usually have a very large state space.

IV. RESULTS

The work was divided into two parts- First being the modeling of the bus protocol and second part being its verification. SMV and NuSMV were used for the hierarchical description of the bus protocols. CTL was used to write properties for the verification.

Verification of the bus protocol takes place on its model. And modeling of this protocol can be done to the extent the document is clear to its interpreter. The level of the abstraction at which protocol is modeled, depends on the kind of properties, which are to be checked on the model. Only a part of the PLB bus has been modeled, including- Two masters; one slave; and an arbiter, with option of different schemes of arbitration- fixed, round-robin (RR) and least recently used (LRU).

Only two masters were chosen in order to reduce the overhead on the verification, due to increase in state space and hence exponential rise in verification time. As only two requests (one read and one write) can be served at a time, only one slave was chosen having pipelining capability. Our model has following capabilities and limitations.

Capabilities: Supports pipelining; Buslock signal implemented; Schemes of arbitration supported- fixed, RR, LRU; Non-determinism supported in slave behavior.

Limitations: Burst transfers and line transfers not modeled. Data path not implemented; only a subset of available signals modeled; Non-response on slave's part not modeled.

a. Property verification

Usually three different properties are checked Liveness, Safety and Fairness. First time when we verify our model we prefer to specify soft properties. In case of soft properties we specify the property in loose way. As we move ahead we check more and more stringent properties.

Soft Properties:

```
AF ((Master0_request or Master1_request)
  -> AF ((grant = 0) or (grant = 1)))
```

... (1)

In the case of property at 1, we are checking that "along all the paths eventually, if either of the masters request the bus then the grant signal will be eventually be given along all the paths". This condition was satisfied easily. But actually request from the master can also be aborted by the abort signal of the same master. So, an additional condition was added to it as shown in 2. Here we check that "along all the path globally, if any master sends request then along some of the paths eventually grant signal is given or the abort signal is given by the master requesting".

```
AG ((Master0_request or Master1_request)
-> EF ((grant for Master0)
   or (grant for Master1)
   or (Abort from Master0)
   or (Abort from Master1)))
```

... (2)

```
AF (Master0_request
-> A [Master0_request
      Until Master0 is granted])

AF (Master1_request
-> A [Master1_request
      Until Master1 is granted])
```

… (3)

In above two properties at 3, we checked that "along all the paths eventually on asserting the request signal the request signal remains asserted until the grant is given". Soft properties were easily verified without any counter examples. Some initial verification was giving false results which helped in the verification of the model.

Stringent Properties:

Now was the turn to check stringent properties. We started specifying following property at 4.

```
AG (Master0_request
-> AF ((Addr acknowledge for Master0
   or Rearbitrate for Master0
   or Abort from Master0)))
```

… (4)

Here, $AG(p \rightarrow q)$ means along "along all the paths globally $\overline{p} + q$ is true". Obviously, that the above property won't be satisfied for the model, because of the fact that in model description there is a time when Master0_request is true and address acknowledgement, rearbitration and abort signal remain false. As this property can not hold in the model, we redefined property as in 5, which were then satisfied.

```
AG (Master0_request
-> AF ((Addr acknowledge for Master0 or
Rearbitrate for Master0 or Abort from
Master0)
   or (Master0_request and (! Addrack
for Master0 or! Rearbitrate for Master0
or! Abort from Master0))))
```

… (5)

When we checked for following strong properties as in 6 and 7, they were not satisfied. The model building may not be solely the reason. There were few considerations about master/slave behaviors. Since, model considered non-determinism for all the responses; it was possibly the slave response in non-deterministic way, which caused violation of the properties at 6 and 7. There is some delay in receiving address acknowledgement signal from slave and the grant signal from the arbiter.

```
AG ((Master0_request or Master1_request)
-> AF ((grant for Master0)
   or (grant for Master1)
   or (Abort from Master0)
   or (Abort from Master1)))
```

… (6)

```
AG (Master0_request
-> A [Master0_request
       Until grant for Master0])
```

… (7)

The above mentioned properties should be modified, to adapt and to include non-determinism. We infer that modeling of the protocol appropriately is very important during verification, in order to derive correct results. We present in the Table 1, the results of verifying the system's two different models using same properties. They show considerable difference in the utilization of system resources.

Table 1 System resources comparison

Case	*User time(sec.)*	*System time(sec.)*	*BDD nodes*
I	69.42	0.25	1918147
II	2.6	0.05	306303

b. Arbitration schemes

We explored three schemes of arbitration within PLB protocol. Results are presented in Table 2.

Table 2 Performance of various arbitration schemes

	Priority [Min, Max]		
Master	***Fixed***	***RR***	***LRU***
M0	[0,1]	[0,7]	[0,7]
M1	[0,∞]	[0,7]	[0,7]
M2	[0,∞]	[0,7]	[0,7]
M3	[0,∞]	[0,7]	[0,7]
M4	[0,∞]	[0,7]	[0,7]
M5	[0,∞]	[0,7]	[0,7]
M6	[0,∞]	[0,7]	[0,7]
M7	[0,∞]	[0,7]	[0,7]

As shown in the Table 2, with fixed priority for masters, the granting of bus is not fair, and there is starvation for other masters, which are having lower priority. In case of round robin scheme, the priority of masters changes as they are served, according to a static criterion. The criterion is that the served master will have lowest priority, whereas others' priority would be upgraded one up. The priority of each master will be at a difference of one from each other, irrespective of which master is served. This means the priority reference is with respect to neighborhood. We predicted that if we use a dynamic criterion of reassignment of priority, like least recently used scheme (LRU), where masters would have priority reference from the time instant they are served, not their neighborhood. But surprisingly, for the model build, this scheme too has the identical performance to that of RR, as is visible from third column of Table 2.

Table 3 Performance of arbitration scheme upon assertion

	Priority [Min, Max]	
Master	***RR***	***LRU***
M0	[0,7]	[0,8]
M1	[0,7]	[0,8]
M2	[0,7]	[0,8]
M3	[0,7]	[0,8]

M4	[0,7]	[0,8]
M5	[0,7]	[0,7]
M6	[0,7]	[0,7]
M7	[0,7]	[0,6]

When we asserted some master signals. The present priority list is M7, M6, M5, M4, M3, M2, M1, and M0. At first the M6 requests the bus, when M7 and M5 are negated. After M6 is granted the bus M7 and M5 are asserted and all other master requests are asserted non-deterministically. The results generated have shown Round robin to be better if the condition for master request is followed. In this assertion the performance of all the masters were same in case of round robin. But in case of least recently used the performance for all the masters were not same. We present the traces of results generated in Table 3. We present in Table 4, the comparison of various system dependent parameters in case of verifying two different arbitration scheme using NuSMV on Linux OS on Pentium-III 833 MHz system.

Table 4 System resources comparison

Scheme	***User time (sec.)***	***System time (sec.)***	***BDD nodes***	***BDD Nodes in initial state***
LRU	258	0.52	6103352	42
RR	248	0.84	6218093	41

c. Inferences about PLB documentation

A number of ambiguities were found in the specification document. Moreover, some of the signals have been described at multiple places in the document. Few other observed and evaluated issues are as follows:

Statement: "…in order to avoid a possible deadlock scenario, the PLB arbiter will ignore the original master during rearbitration"

It was not clear, whether the present master would be ignored OR only present request from the master would be ignored. If master is ignored, then it should be so for how many clocks. If it is the only master, requesting, then will it be assigned the bus very next clock, OR else if ignored till any other master is first granted the bus, a near starvation situation for the present master may arise.

Statement: "…in order to avoid a possible deadlock scenario, the PLB arbiter will ignore the original master during rearbitration"

After receiving the re-arbitration signal from the slave, what will be the priority of the current ignored master?

Statement: "…MasterID signal indicate to the slaves identification of the master of the current transfer (page 20)"

Sometimes when both read and write requests of different masters are served, which ID is stored by mastered signal?

Statement: "…if Buslock signal is raised, the master requesting Buslock can not be granted the bus without both read and write data buses being free"

This would result in increased latency for the system. We suggest that a separate priority assignment be devised for masters raising Buslock signals.

Statement: "…once Mn_request has been asserted, this signal, the address and all of the transfer qualifiers must retain their values until {slave has terminated address cycle OR the master has aborted the request OR PLB arbiter has asserted timeout signal} (page 12)"

It is not clear whether this condition be hold, even when master is not granted the bus. We propose that it should hold even when master is not granted the bus.

Statement: "…table 5 on page 17"

In Table 5, which mention about PLB_SAValid assertion, there should be a correlation between Note 2 on the same page with the signal having Y on their Requesting buslock column. PLB_SAValid will only be asserted for pipelined requests generated by the locking master.

V. CONCLUSIONS

We have presented a verification based evaluation of arbitration schemes for IBM CoreConnect PLB protocol. From our experience of the formal verification of bus protocols, we can conclude that:

- Formal verification can be used by the designers in the early design phase to find out the bugs in protocol specifications which may be difficult to trace otherwise.
- Formal verification doesn't imply complete correctness in the absence of certain errors, so it must not be regarded as the debugging tool. FV can help the same way an exhaustive simulation would, certainly detecting a specified error if it does exist.
- Modeling of the system at proper abstraction level is very important while verifying model using formal model checking. During the work, most of the time was spent in improving the model implemented. The approach we have adopted is to model the protocol in Nu/SMV as close to its specification as possible, as we didn't have lower level implementation available, viz. RTL or gate circuit. Many a times, we felt that we should have gone for some higher-level description in order to first check for the completeness of the protocol. But this would in turn move us away from the detailed description of these bus protocols and it would have kept us away from using the trace-based tracking of states in a model using NuSMV.

- System description is obviously downscaled because of the limited capability of the model checking tool in respect to number of possible system states it can handle.

ACKNOWLEDGEMENT

Help of S. Ramesh Babu, who has been working on verification of AMBA bus protocol, is thankfully acknowledged.

REFERENCES

[1] D. L. Dill, A. J. Drexler, A. J. Hu, C. H. Yang. *Protocol Verification as a Hardware Design Aid*. Conference on Computer Design, VLSI in Computers and Processors, pages 522--525, Los Alamitos, Ca., USA, 1992.

[2] A. Goel, W. R. Lee. *Formal Verification of an IBM CoreConnect Processor Local Bus Arbiter Core*. DAC 2000

[3] S. Campos, E. Clarke, W. Marrero, M. Minea. *Verifying the Performance of the PCI Local Bus using Symbolic Techniques*. IEEE 1995

[4] A. Roychoudhary, T. Mitra, S. R. Karri. *Using Formal techniques to debug the AMBA SoC Bus Protocol*. Design, Automation & Test in Europe, 2004

[5] IBM. *64-bit Processor Local Bus Architecture Specification version 3.5*. http://www-306.ibm.com/chips/techlib/techlib.nsf/products/. May 2001.

[6] IBM. *64-bit PLB Arbiter Core User's Manual version 3.6*. www-306.ibm.com/chips/techlib/techlib.nsf/products/. April 2002

[7] *Whitepaper on IBM CoreConnect bus Architecture*. http://www.chips.ibm.com/products/coreconnect/. 1999

[8] R. Usselmann. *OpenCores SoC Bus Review version 1.0*, January 2001

[9] Alur, R. and Yannakakis, M. (2000). *Model checking of hierarchical state machines*, *ACM Trans. on Programming Languages and Systems*, pp 1-31.

[10] Burch, J. R., Clarke, E. M., Long, D. E., McMillan, K. L., Dill, D. L., (1994). *Symbolic model checking for sequential circuit verification*, IEEE Trans on Computer –Aided Design, vol.13, no.4, pp.401-424, April 1994

[11] GUPTA, A. (1992). *Formal Hardware Verification Methods: A Survey*, Formal Methods in System Design, VOL. 1, PP. 151-238.

[12] K.L. McMillan. *Symbolic Model Checking: An approach to the state explosion problem*. Ph.D. Thesis, SCS, Carnegie Mellon University. May 1992

[13] B. Chen, M. Yamazaki, M. Fujita. *Bug Identification of a Real Chip Design by Symbolic Model Checking*. IEEE 1994

[14] K. K. Ryu, E. Shin, V. J. Mooney. *A Comparison of Five different Multiprocessor SoC Bus Architectures*. Euromicro Symposium on Digital Systems Design, pages 202. 2001

[15] A. S. Lee, N. W. Bergmann. *On-chip Communication Architectures for Reconfigurable System-on-Chip*. http://eprint.uq.edu.au/archive/00000787/01/Lee_ICFPT2003_poster.pdf

[16] A. Chakraborthi, P. Dasgupta, P. P. Chakrabarti, A. Benerjee. *Formal Verification of Module Interfaces against Real Time Specifications*. 39th conference on Design automation, pages 141-145. 2002

[17] *Getting started with SMV*. http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/, March, 1999

[18] *NuSMV 2.2 User Manual*. http://nusmv.irst.itc.it/.

[19] *NuSMV 2.2 Tutorial*. http://nusmv.irst.itc.it/.

[20] P. Rashinkar, P. Paterson, L. Singh. *System-on-a-chip Verification: Methodology and Techniques*. Kluwer Academic Publishers. 2001

[21] List and link of various tools available for model checking and verification. http://members.fortunecity.com/boroday/Automatatools.html

[22] Homepage of Prof. S. Arun Kumar of IIT, Delhi. http://www.iitd.ernet.in/iitd-cwb

[23] Webpage of Formal Verification Research Group at IIT, Kharagpur.

[24] http://www.facweb.iitkgp.ernet.in/~pallab/forverif.html

[25] CoreConnect Bus Simulator - An Automatic Synthesized Simulator for Cycle-Accurate IBM CoreConnect Bus Model. http://www.princeton.edu/~xzhu/coreconnect.html

[26] Tool for Verification interacting with synthesis from Berkeley University. http://www-cad.eecs.berkeley.edu/~vis/getting_VIS.html

[27] Virtual Socket Interface. http://www.vsi.org/.

[28] Forte Design System. http://www.forteds.com/

[29] Murphi description language and automatic verifier. http://verify.stanford.edu/dill/murphi.html

[30] A community of Verification Professionals. http://www.verificationguild.com/