Verifying a Virtual Component Interface-based PCI Bus Wrapper Using an LSC-Based Specification

Annette Bunker and Ganesh Gopalakrishnan

UUCS-02-004

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

January 22, 2002

Abstract

Because of the high stakes involved in integrating externally developed intellectual property (IP) cores used in System on Chip (SOC) designs, methods and tool support for quick, easy, decisive standard compliance verification must be developed. Such methods and tools include formal standard specifications that are easy to read, formal definitions of standard compliance and automatic generation of model checking assertions which together imply compliance. We compare two efforts in verifying that the same register transfer level (RTL) code complies with the Virtual Sockets Interface Alliance's (VSIA) Virtual Components Interface (VCI) Standard. We show that using Live Sequence Charts (LSCs) as a formal notation for protocol specification has potential to ease the verification effort required.

1 Introduction

As designers rely more and more on externally-developed intellectual property (IP), the necessity of verifying that the IP blocks interface with one another correctly becomes a vital part of the verification task. In an effort to alleviate this concern, the Virtual Socket Interface Alliance VSIA (VSIA) produced the Virtual Component Interface (VCI) Standard. The problem, then, becomes determining whether or not a given IP block correctly implements the VCI standard.

Both the IP designer and the IP integrator face this problem, as the integrator must at least sanity check and possibly fully replicate the verification reported by the developer. Because the same

[1]This work was supported by National Science Foundation Grants CCR-9987516 and CCR-0081406.

verification may be performed twice by separate organizations the verification effort must be significantly less than the total design effort. Furthermore, because those organizations may be contractually bound, the verification must state definitively whether or not the IP block complies with the standard, and results must be reproducible.

With those three requirements in mind, we propose to use formal verification techniques to develop a system in which the compliance of a register transfer level (RTL) model with a protocol standard can be determined. Furthermore, because we use algorithmic formal methods combined with a formal specification, our system provides definitive results and our results are reproducible.

This paper presents a verification study on an RTL model using the Cadence® FormalCheck® tool. The model under test translates a VCI transaction to its equivalent Peripheral Component Interface (PCI) transaction as the VSIA expected integrators to use the VCI standard. Our control verification relies on an ad hoc specification consisting of three liveness and three safety properties. We then specify both the VCI protocol and the PCI protocol using Live Sequence Charts (LSCs) and verified the model again, using properties systematically derived from these LSCs. While the LSC-driven verification consisted of three times more properties and took far more CPU time than the ad hoc verification, it found all the bugs found in the original project and took much less time overall.

The remainder of the paper reports on the verification effort. The rest of Section 1 introduces the Virtual Component Interface Standard, the Peripheral Component Interconnect Standard, the FormalCheck verification tool and Live Sequence Charts. Section 2 reviews related research. We briefly describe the module we verified in Section 3. Section 4 presents the verification effort in detail. Appendix A contains the full Verilog code for the wrapper model, while Appendix B consists of the final verification report created by FormalCheck for the project.

1.1 The VCI Standard

The Virtual Component Interface Standard specifies a family of point-to-point communication protocols aimed at facilitating communication between virtual components [Gro00], possibly those created by separate design organizations. Three protocols currently belong to the family: the Peripheral Virtual Component Interface (PVCI), the Basic Virtual Component Interface (BVCI) and the Advanced Virtual Component Interface (AVCI).

The AVCI is a superset of the BVCI which is a superset of the PVCI. The PVCI is not a split-transaction protocol; request and response data transfers occur during a single control handshake. The BVCI, on the other hand, is a split transaction protocol. The only constraint placed on responses by the standard is that they arrive at the initiator in the same order in which the initiator generated matching requests. The AVCI is also a split-transaction protocol. AVCI requests may be tagged to allow request threads to be interleaved and transactions reordered.

All VCI standards require separate address and data busses. They allow for multiple addressing modes enabling integrators to take advantage of memory access optimizations and bus optimizations.

1.2 The PCI Standard

The Peripheral Component Interconnect Standard defines a chip-level interface for connecting I/O devices to the system's processor/cache/memory subsystem via an acyclic network of busses and bus bridges [Gro95]. The PCI standard is a split-transaction protocol based on two types of transactions, posted and delayed. A posted transaction completes on the originating bus before it completes on the destination bus. Delayed transactions, on the other hand, complete remotely before completing locally. They do so by leaving markers in each bridge along the path from source to destination which much be matched at each step by the transaction acknowledge. Only when the marker at the source is matched is a delayed transaction completed.

In an attempt to obey the producer/consumer property and remain deadlock free, PCI allows certain requests and responses to be reordered while in flight in the network. Address and data share the same bus.

1.3 FormalCheck

The FormalCheck model checker [Bel98], marketed by Cadence Design Systems, Inc., is based on language inclusion test for ω-automata. FormalCheck compiles the Verilog or VHDL design to build its system model. It provides the user with a series of templates for writing constraints (environmental assumptions) and properties. The user may choose from several verification styles ranging from bug-hunt to rigorous state-space exploration. A variety of model reduction techniques can be employed, including the default one-step reduction, iterative reduction, clock extraction, seeded and non-seeded reductions.

If the verification property holds on the system under investigation, the tool supplies the user with a message so indicating. If the model violates the property, FormalCheck supplies the user with a waveform describing the violation and the events leading to it. Violations may stem from either improperly defined properties/constraints or from issues in the RTL model.

1.4 Live Sequence Charts

Harel and Damm propose Live Sequence Charts as an extension to Message Sequence Charts (MSCs) [DH01]. Live Sequence Charts consist of sets of processes, each denoted by rectangles containing process identifiers. A process lifeline extends downward from each process. An arrow represents a message passed from a sending process to a receiving process. The time scale of each process is independent of the others with the exception of the partial order imposed on the events occurring in the sending and receiving lifelines by a passed message. Sets of events that may occur in any order are marked by coregions, dotted lines near the lifeline on which the events occur.

As the name suggests, Live Sequence Charts allow the specifier the ability to require that some or all elements on the chart occur or that a certain time within the chart be reached (liveness). Required events, messages, and points along the lifeline are denoted by solid lines, while optional events, message, and timepoints are denoted by dashed lines. Similarly, required subcharts are outlined by solid lines, while optional subcharts are outlined by dashed lines.

LSCs also offer facilities for testing conditions. Conditions are represented by bars with convex ends that cross the lifelines of relevant processes. We require that our LSC specifications begin with a precondition and end in a postcondition stating when the LSC is allowed to fire and in what state the LSC leaves the system after it executes, respectively.

2 Related Work

Work related to our case study generally resides in three categories, that motivating our current work, studies involving specifying and verifying standards at the RTL abstraction level and research involving the use of the FormalCheck model checker. We treat each category of work, here.

The importance of high-level, formal, intuitive interface specification and verification methods has increased greatly in recent years. The complexity of recently proposed standards implies that their adoption depends on development of reliable SOC cores that implement these standards [GC01]. This, in turn, relies on effective ways of specifying the standards and verifying RTL designs for compliance. A recent expert panel points out that the integration and verification of IP will depend not only on the individual tools chosen, but also the design process adopted [Mor01]. Our work can be regarded as an effort to create an automatable verification process that tries to bridge the gap between high-level standards specifications and industrial-scale model-checkers. Two key ingredients of such a process have been pointed out to be the use of high-level abstractions and interface monitors [Alb].

Most of the work previously done in the area of standards specification and verification involves the PCI standard. Shimizu, et al [SDH00], specify the PCI standard using monitors written in HDLs. Monitor-based specifications can be checked for consistency and for receptivity, however they lack conceptual cohesion and we expect them to be difficult to understand. Other work demonstrates the verification of the PCI specification at roughly the same level of abstraction as the one reported here [CCLW99, Wan99].

Xu, et al [XCS+99], describe the verification of a proprietary frame mux/demux chip by Nortel

Corporation using FormalCheck. The authors present the properties verified as well as their experiences using the model reduction features of FormalCheck. In [XCSH97], the verification of an ATM fabric switch using various theorem provers as well as model-checkers is described. The paper presents techniques to handle large queues as well as addresses the verification of latency properties specific to this chip in FormalCheck.

3 PCI bus wrapper model

The PCI bus wrapper consists of eight fifos and six state machines, as shown in 3. Each fifo is responsible for storing one element of the request or response, while earlier transactions proceed on the buses. The address, byte enables (BE), command (CMD[1:0]), write data (WDATA[31:0]) and end-of-packet (EOP) fifos contain the input transaction information from the VCI, as indicated by the name of each fifo. The response error (RERR), read data (RDATA[31:0]) and response end-of-packet (REOP) fifos return response information to the VCI. Note that all information is stored in the wrapper in its VCI-compliant format. This design decision localizes the complexity in the PCI state machine.

Of the six state machines, three make up the actual bus interfaces: two for the VCI and one for the PCI interface. The VCI Request machine reads input transactions from the VCI and inserts them into the appropriate fifos. Likewise, the VCI Response machine reads response transactions from the appropriate fifos and drives them onto the VCI. The PCI machine handles all timing relative to transacting on the PCI bus, but it is aided in certain aspects by the remaining three state machines. The Parity machine calculates even parity to be output on the PCI bus, as VCI does not use a notion of parity checking. The CMD_CVT (command convert) machine converts the VCI transaction command and byte enables into their PCI-compliant format while multiplexing them onto the the PCI command/byte enable bus (C/BE#[3:0]). Similarly, the AD_MRG (address/data merge) machine multiplexes the address and write data onto the PCI AD[31:0] bus, though it does not do any data reformatting.

If the PCI network is able to service a request without errors, the PCI state machine reformats and loads the response data into the response queues immediately. However, if the PCI network returns a retry, the PCI state machine leaves the current transaction at the head of the fifos and immediately attempts the transaction on the PCI bus again, essentially busy-waiting on the retried PCI transaction. We made this decision to simplify the wrapper's design. The only PCI errors that must be supported by this style of wrapper are target aborts, which mean that the addressed device is unable to service the request due to a fatal error. In this case, we remove the request from the fifos and return an error to the VCI initiator.

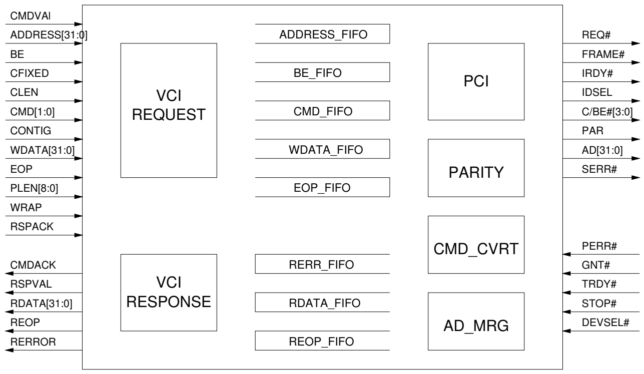

Figure 1: Structure of the PCI Bus Wrapper

4 Verifying with FormalCheck

This section discusses the process we used to verify the wrapper model in FormalCheck. Though we present our process in a linear fashion for clarity, here, we used an iterative process in which, model reductions, constraint formulation and bug tracking interacted and fed back to one another. Subsection 4.1 explains the model reductions necessary to make the model model-checkable. Subsection 4.2 enumerates the environmental constraints necessary to complete the model checking and Subsection 4.3 enumerates the properties we verified. We discuss the issues found with the design in this case study in Subsection 4.4.

4.1 Model reductions

The primary reductions that had to be made to the model to allow FormalCheck to handle the wrapper design were to reduce the counter sizes and the data bus widths. Since we were aware that we would likely make these reductions at design time, the fifos and state machines that know about the bus sizes and counter widths were all designed using Verilog parameters. Using this Verilog feature allowed us to change the size parameter at the highest level of module instantiation and the Verilog compiler managed change propagation to all necessary modules. (Later versions of FormalCheck make these reductions automatically, without the aid of parameterized designs.)

The original design contained 32-bit address and data busses as one would expect to use when interfacing with a standard PCI network. These busses were all reduced to two bits to allow for a nontrivial number of address and data element possibilities.

The original design also specified 9-bit fifo counters (for the head and tail pointers). Nine bits allows enough slots to store an entire VCI packet in the fifos at once. Though our model does all processing on the cell-level, we chose this design for flexibility. We wished to make this design easy to modify to allow full packet-based processing, if we choose to do so, later. For purposes of the verification, however, these counters were also reduced to two bits, allowing for only 4 slots in our fifos.

4.2 Constraints

It is our experience that precisely defining environmental constraints is the most difficult and time-consuming portion of a FormalCheck-based verification. We began our verification with a minimal set of constraints and added a new constraint only when it was necessary in order to avoid a false negative. We chose this approach for three reasons: to keep the verification as simple as possible, to allow our results to be as strong as possible and to mimic the industrial procedures of which we are aware.

Besides the usual clock and reset constraints, we created eight constraints during the verification project. The final set of constraints we used and their English semantics are enumerated below.

1. `Assume Never: (xlator.reset_l == 0) && (xlator.cmdval == 1)`

   `cmdval` may never be asserted while the wrapper is in reset.

2. ```
   After: (xlator.cmdval == 1) && (xlator.clk == rising)
   Assume Always: (xlator.cmdval == 1)
   Unless: (xlator.cmdval == 1) && (xlator.cmdack == 1) &&
     (xlator.clk == rising)
   ```

   `cmdval` must remain asserted until it is properly acknowledged.

3. ```
   After: (xlator.req_l == 0) && (xlator.clk == rising) &&
     (xlator.reset_l == 1)
   Assume Eventually: (xlator.gnt_l == 0) && (xlator.clk == rising)
   ```

   The PCI arbiter will eventually grant ownership of the PCI bus to the bus wrapper.

4. ```
   After: (@retry) && ((xlator.ul.curr_state == 4) ||
     (xlator.ul.curr_state == 3)) && (xlator.clk == rising)
   Assume Eventually: (xlator.stop_l == 1) &&
     (xlator.trdy_l == 0) && (xlator.ul.curr_state == 4) ||
     (xlator.ul.curr_state == 3)) && (xlator.clk == rising)
   ```

   The PCI environment will not give the wrapper a PCI retry response in all future states when the wrapper samples the PCI response.

5. ```
   After: (xlator.rspval == 1) && (xlator.clk == rising)
   Assume Eventually (xlator.rspack == 1) && (xlator.clk == rising)
   ```

   The VCI environment acknowledges every response, eventually.

6. ```
   After: (xlator.reset_l == 1) && (xlator.frame_l == 0)
   Assume Eventually: (xlator.trdy_l == 0) && (xlator.clk == rising)
   ```

   The PCI environment will eventually take ownership of every transaction the wrapper drives onto the PCI bus.
   ```

7. `Assume Never: (xlator.rspack == 1) && (xlator.clk == rising)`
   `Unless: (xlator.rspval == 1) && (xlator.clk == rising)`

   The VCI environment does not generate an acknowledge to a response unless the response has been driven.

8. ```
   After (xlator.req_l == 0) && (xlator.clk == rising)
   Assume Always: (xlator.req_l == 0)
   Unless: (xlator.req_l == 0) && (xlator.gnt_l == 0) &&
     (xlator.clk == rising)
   ```

   The PCI request remains asserted until the PCI environment grants the wrapper ownership of the bus.

Constraint 4 deserves more comment. The `@retry` term in the constraint represents a FormalCheck macro, which expands to `(xlator.stop_l == 0) && (xlator.devsel_l == 0) && (xlator.trdy_l == 1)`, the PCI signaling that indicates a retry response. Ironically, the complexity of this constraint is a direct result of our goal to keep the verification simple. We chose to model the PCI environment with as few constraints as possible. As a result, the PCI environment is allowed to drive responses at anytime, whether it is actually a moment at which the PCI machine samples the PCI inputs or not. Hence, merely stating that the PCI environment cannot give a retry forever is not strong enough. Instead, we must stipulate that retries may not be returned at all future states when the PCI transaction is sampled.

4.3 Properties

The specification we used for the verification consisted of two Live Sequence Charts describing the VCI standard and two LSCs describing the subset of the PCI standard which the wrapper supports. The example LSC shown in Figure 2 specifies VCI requests.

Because all of the elements of our specification were required behaviors (recall that LSCs can also contain optional behaviors), generating the corresponding FormalCheck specification proceeds in a systematic fashion. We first define a notion of precise previous upon which the definition of both properties depend. A precise previous message is the most recently received message that is both a required message and is not in a coregion.

Each message sent by a process results in two FormalCheck properties, one eventually property and one never property. For eventually properties, the precise previous message represents the trigger condition, while the message sent translates to the verification condition. For instance, in our wrapper verification the message leaving the VCI target labeled `cmdack` generates the following property because CMDVAL is the last required message that is not in a coregion received by the VCI target before it sends a CMDACK message.
   ```

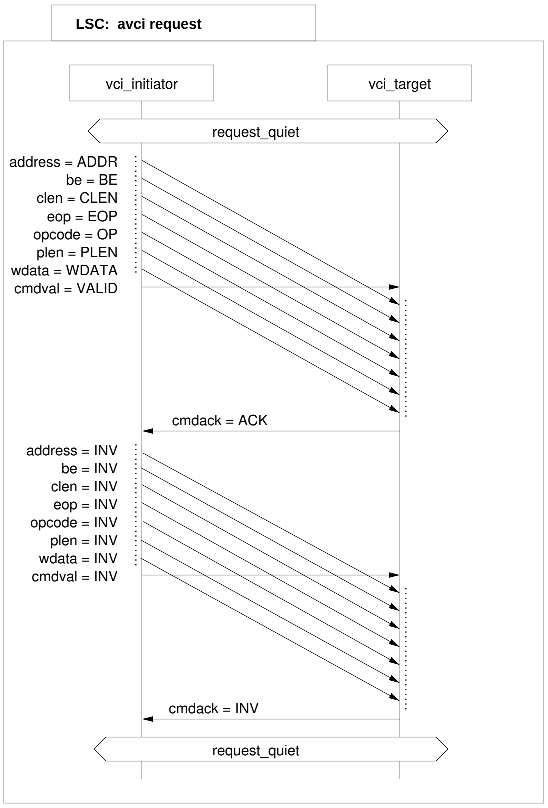

Figure 2: LSC specification of VCI requests.

```
After: (xlator.reset_L == 1) && (xlator.cmdval == 1) &&
    (xlator.clk == rising)
Eventually: (xlator.cmdack == 1) &&
    (xlator.clk == rising)
```

For never properties, the message sent becomes the verification condition, while the precise previous message becomes the discharge condition. Thus, in our wrapper verification, the `cmdack` message and its precise previous translate into the following requirement:

```
Never: (xlator.cmdack == 1) && (xlator.clk == rising)
Unless: (xlator.cmdval == 1) && (xlator.clk == rising)
```

The example properties shown above highlight an earlier observation that in order for Live Sequence Charts to be completely effective as protocol specification devices, they must have a timing model attached to them [BG01a]. Read strictly, the current LSC description does not require the clock to tick as the FormalCheck translations do. These clock ticks are, however, important aspects of the VCI Standard and must considered in the verification effort.

These examples also point out a second problem with translating the Live Sequence Charts into model checker input. User expertise is required to write the logical expressions from the charts in terms of the signal names in the implementation model. While LSC variables and implementation signals have a one-to-one mapping, the exact semantics of symbolic constants, such as ADDR are unclear.

Below, we list all the properties generated and verified in this verification project.

1. ```
   After: (xlator.reset_L == 1) && (xlator.cmdval == 1) &&
       (xlator.clk == rising)
   Eventually: (vp_xlator.cmdack == 1) && (xlator.clk == rising)
   ```
   After `cmdval` is correctly raised, eventually `cmdack` will acknowledge, correctly.
2. ```
   After: (xlator.reset_L == 1) && (xlator.cmdack == 1) &&
       (xlator.clk == rising)
   Eventually: (xlator.cmdack == 0) && (xlator.clk == rising)
   ```
   After `cmdval` is correctly asserted, it must eventually be deasserted.
3. ```
   After: (xlator.reset_L == 1) && (xlator.clk == rising)
   Never: (xlator.cmdack == 1) && (xlator.clk == rising)
   Unless: (xlator.cmdval == 1) && (xlator.clk == rising)
   ```
   No `cmdacks` are allowed without a preceding `cmdval`.
4. ```
   After: (xlator.cmdval == 1) && (xlator.clk == rising)
   Eventually: (xlator.rdata == stable) && (xlator.rspval == 1) &&
       (xlator.clk == rising)
   ```
   `rdata` should eventually be valid.

5. `After: (vp_xlator.rdata == stable) && (vp_xlator.rspval == 1) &&`
   `(vp_xlator.clk == rising)`
   `Eventually: (vp_xlator.rspval == 0) && (vp_xlator.clk == rising)`

   `rdata` must eventually invalidate.

6. `After: (vp_xlator.cmdval == 1) && (vp_xlator.clk == rising)`
   `Eventually: (vp_xlator.reop == stable) && (vp_xlator.rspval == 1) &&`
   `(vp_xlator.clk == rising)`

   `reop` is eventually valid.

7. `After: (vp_xlator.reop == stable) && (vp_xlator.rspval == 1) &&`
   `(vp_xlator.clk == rising)`
   `Eventually: (vp_xlator.rspval == 0) && (vp_xlator.clk == rising)`

   `reop` eventually invalidates.

8. `After: (vp_xlator.cmdval == 1) && (vp_xlator.clk == rising)`
   `Eventually: (vp_xlator.rerror == stable) && (vp_xlator.rspval == 1) &&`
   `(vp_xlator.clk == rising)`

   `rerror` eventually becomes valid.

9. `After: (vp_xlator.rerror == stable) && (vp_xlator.rspval == 1) &&`
   `(vp_xlator.clk == rising)`
   `Eventually: (vp_xlator.rspval == 0) && (vp_xlator.clk == rising)`

   `rerror` eventually invalida

10. `After: (vp_xlator.reset_L == 1) && (vp_xlator.cmdval == 1) &&`
    `(vp_xlator.clk == rising)`
    `Eventually: (vp_xlator.reset_L == 1) && (vp_xlator.rspval == 1) &&`
    `(vp_xlator.clk == rising)`

    `rspval` eventually becomes valid once a transaction has been inserted into the queues.

11. `After: (vp_xlator.rspval == 1) && (vp_xlator.clk == rising)`
    `Eventually: (vp_xlator.rspval == falling)`

    `rspval` eventually invalidates.

12. `Never: (vp_xlator.rspval == 1) && (vp_xlator.clk == rising)`
    `Unless: (vp_xlator.cmdval == 1) && (vp_xlator.clk == rising)`

    `rspval` may not be asserted until a `cmdval` precedes it.

13. `After: (vp_xlator.cmdval == 1) && (vp_xlator.clk == rising)`
    `Eventually: (vp_xlator.req_l == 0) && (vp_xlator.clk == rising)`

    Once a request has occured on the VCI, then the wrapper must eventually request PCI arbitration.

14. `After: (vp_xlator.cmdval == 1) && (vp_xlator.clk == rising)`
    `Eventually: (vp_xlator.frame_l == 0) && (vp_xlator.clk == rising)`

    Once a request has occured on the VCI, the wrapper must initiate a PCI transaction, eventually.

15. `After: (vp_xlator.cmdval == 1) && (vp_xlator.clk == rising)`
    `Eventually: (vp_xlator.c_be_l == stable) && (vp_xlator.clk == rising)`

    Once a request has occured on the VCI, the wrapper must generate appropriate command and byte enable information on the PCI interface, eventually.

16. `After: (vp_xlator.cmdval == 1) && (vp_xlator.clk == rising)`
    `Eventually: (vp_xlator.irdy_l == 0) && (vp_xlator.clk == rising)`

    Once a transaction has occured on the VCI, the wrapper must eventually assert `irdy_l` to show its data transfer readiness.

17. `After: (vp_xlator.cmdval == 1) && (vp_xlator.cmd == 1) &&`
    `(vp_xlator.clk == rising)`
    `Eventually: (vp_xlator.trdy_l == 0) && (vp_xlator.clk == rising)`

    Once a read transaction has occured on the VCI then the wrapper must eventually assert `trdy_l` to show its data transfer readiness as the target device.

18. `Never: (vp_xlator.frame_l == 0) && (vp_xlator.clk == rising)`
    `Unless: (vp_xlator.cmdval == 1) && (vp_xlator.cmdack == 1) &&`
    `(vp_xlator.clk == rising)`

    The wrapper may not initiate a PCI transaction unless a VCI transaction preceded it.

Table 4.3 summarizes the verification statistics reported by FormalCheck on the final, passing run of each property.

4.4 Issues

Our original wrapper model contains eight issues, three of which were identified as a direct result of model checking activities. The other five were identified as a result of deeper examination of the code inspired by feedback obtained from the model-checker. Of the eight issues, six of resided in the PCI state machine. One of these six issues corrects poor coding, but was not a functional correctness issue. Another of the six relates to the fix for an earlier-identified issue.

The most interesting issue raised by this study relates to a performance optimization in the PCI machine. When the PCI network issues a target abort (fatal error) in response to the wrapper's request, the original design immediately checks the status of the queues, initiating the next transaction if there is one and idling if not. This procedure, however, does not allow the request queues enough time to discard the aborted transaction and update the empty status bit before it is checked. In the case when the aborted transaction is the last one in the queues, the wrapper tries to generate an extra, garbage transaction. Removing the optimization, forcing the PCI state machine to travel through a recovery state before testing the status bits, solves the problem.

The experimental verification found all the bugs reported in the original project [BG01b], except the buggy bug-fix. We had the advantage of experience in the second project and the bug was avoided

Property	State Vars	Time	Memory
1	71	4 min, 9 sec	481 MB
2	71	4 min, 54 sec	438 MB
3	67	2 min, 40 sec	473 MB
4	143	—	—
5	143	—	—
6	87	—	—
7	87	—	—
4	78	28 min, 56 sec	563 MB
5	78	21 min, 45 sec	530 MB
6	71	7 min, 31 sec	497 MB
7	72	6 min, 34 sec	499 MB
8	66	2 min, 15 sec	471 MB
9	72	4 min, 25 sec	483 MB
10	72	5 min, 46 sec	488 MB
11	103	9 min, 1 sec	504 MB
12	73	17 min, 8 sec	518 MB
13	72	4 min, 21 sec	481 MB
14	67	2 min, 15 sec	471 MB

Table 1: FormalCheck statistics summary

completely. All the issues found in the LSC-based verification project were found as a direct result of model checking.

5 Conclusions and future work

As the use of externally developed intellectual property becomes widespread practice, the need for formal compliance verification techniques increases. Adequate verification schemes should require significantly less time and effort than development of the IP. They should state definitively whether or not the block complies with the standard and the verification results should be reproducible.

We seek to understand if specifications written in Live Sequence Chart notation address the first requirement, above, and if so, how. While not conclusive, our study indicates that Live Sequence Chart specifications show potential for aiding in standard compliance verification tasks. The potential for large effort reductions exists, given better understanding of how LSCs can aid the production of environmental constraints. We show how model checking properties can be systematically derived from an LSC specification. The properties so generated can give the IP designer high confidence that the design correctly implements the standard. They can also give the IP integrator high confidence in and deep insight into the delivered IP component in a short amount of time.

There is much work we can still do toward developing automatic tool support for standard compliance verification. In the near term, our case study raises several specific questions. First, we should study LSC constraint generation more closely. Developing exactly the right set of the constraints for a FormalCheck verification project is difficult and time consuming. It is also the source of many errors. Finding ways to ease this task could equate to greatly reducing the total verification effort. Second, we must address the name mapping problem in order to automatically generate properties or constraints from Live Sequence Chart specifications as it is not always obvious how certain constructs of an LSC specification are expressed in the implementation model. Third, deriving schemes for addressing optional elements of protocol specifications expressed as LSCs is necessary for our methods and tools to generalize to other protocols. The Basic Virtual Component Interface is a relatively simple standard. Even describing its sister standard, the Advanced Virtual Component Interface, requires the use of optional behaviors.

In the longer term, developing a formal definition of standard compliance is a high priority. Such a definition would characterize exactly what the generated properties imply. We expect that such a definition will rely heavily upon trace inclusion concepts and techniques. Once compliance is formally defined and the issues with automatically generating properties from Live Sequence

Chart specifications are solved, we can create an automatic compliance verification system.

References

[Alb] Ken Albin. Nuts and bolts of core and soc verification.

[Bel98] Bell Labs design Automation and Lucent Technologies. *FormalCheck User's Guide*, v2.1 edition, 1998.

[BG01a] Annette Bunker and Ganesh Gopalakrishnan. Using live sequence charts for hardware protocol specification and compliance verification. In *IEEE International High Level Design Validation and Test Workshop*. IEEE Computer Society Press, November 2001.

[BG01b] Annette Bunker and Ganesh Gopalakrishnan. Verifying a virtual component interface-based pci bus wrapper using formalcheck. Technical Report UUCS-01-006, University of Utah, June 2001.

[CCLW99] Pankaj Chauhan, Edmund M. Clarke, Yuan Lu, and Dong Wang. Verifying ip-core based system-on-chip designs. In *IEEE International ASIC/SOC Conference*, pages 27–31, September 1999.

[DH01] Werner Damm and David Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, pages 45–80, 2001.

[GC01] Torbjörn Grahm and Barry Clark. Soc integration of reusable basband bluetooth ip. In *Proceedings of the 2001 Design Automation Conference*, pages 256–261, 2001.

[Gro95] PCI Special Interest Group. *PCI Local Bus Specification*. PCI Special Interest Group, 1995.

[Gro00] OCB Design Working Group. *VSI Alliance Virtual Component Interface Standard*. Virtual Sockets Interface Alliance, November 2000.

[Mor01] Gabe Moretti. Your core - my problem? integration and verification of ip. In *Proceedings of the 2001 Design Automation Conference*, pages 170–171, 2001.

[SDH00] Kanna Shimizu, David L. Dill, and Alan J. Hu. Monitor-based formal specification of PCI. In Warren A. Hunt, Jr. and Steven D. Johnson, editors, *Formal Methods in Computer-Aided Design*, volume 1954 of *Lecture Notes in Computer Science*, pages 335–352. Springer-Verlag, November 2000.

[Wan99] Dong Wang. Formal verification of the PCI local bus: A step towards ip core based system-on-chip design verification. Master's thesis, Carnegie Mellon University, May 1999.

[XCS+99] Y. Xu, E. Cerny, A. Silburt, A. Coady, Y. Liu, and P. Pownall. Practical application of formal verification techniques on a frame mux/demux chip from nortel semiconductors. In Laurence Pierre and Thomas Kropf, editors, *Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99*, volume 1703 of *Lecture Notes in Computer Science*, pages 110–124. Springer Verlag, September 1999.

[XCSH97] Ying Xu, Eduard Cerny, Allan Silburt, and Roger B. Hughes. Property verification using theorem proving and model checking. www.isdmag.com, November 1997.

A Verilog Wrapper Model

```
/**********************************************************/
/*
/* Project: VCI/PCI bridge model
/* File: ad_mrg.v
/* Author: Annette Bunker
/* Date: Thu Oct  5 10:53:12 MDT 2000
/* Modifications:
/*
/**********************************************************/

/* define state names */
`define    AD     1'b0
`define    DATA     1'b1

module ad_mrg (pci_ad, vci_address, vci_data, merge, cmd, clk, reset_L);

   output[3:0]           pci_ad;
   reg   [3:0]           pci_ad;
   input [3:0]           vci_address;
   wire  [3:0]           vci_address;
   input [3:0]           vci_data;
   wire  [3:0]           vci_data;
   input   merge;
   wire   merge;
   input [1:0]  cmd;
   wire  [1:0]  cmd;
   input   clk;
   wire   clk;
   input   reset_L;
   wire   reset_L;

   reg    curr_state;
   reg    next_state;

   always @(posedge clk or negedge reset_L) begin
      if (!reset_L) begin
 next_state = `AD;
      end else begin
 curr_state = next_state;
 case (curr_state)
   `AD: begin
      if (merge) begin
 pci_ad = vci_address;
 next_state = `DATA;
      end else begin
 next_state = `AD;
```

```
      end
   end // case: `AD
   `DATA: begin
      if (cmd == `VCI_WRITE) begin
 pci_ad = vci_data;
      end else begin
 pci_ad = 32'bz;
      end
      next_state = `AD;
   end
 endcase
      end
   end
endmodule

/*********************************************************/
/*
/* Project: VCI/PCI bridge model
/* File: cmd_cvt.v
/* Author: Annette Bunker
/* Date: Tue Oct  3 11:08:45 MDT 2000
/* Modifications:
/*
/*********************************************************/

/* define state names */
`define    CMD     1'b0
`define    BE     1'b1

module cmd_cvt (pci_cmd_be, vci_cmd, vci_be, convert, clk, reset_L);

   output[3:0]           pci_cmd_be;
   reg   [3:0]           pci_cmd_be;
   input [1:0]           vci_cmd;
   wire  [1:0]           vci_cmd;
   input [3:0]           vci_be;
   wire  [3:0]           vci_be;
   input   convert;
   wire   convert;
   input   clk;
   wire   clk;
   input   reset_L;
   wire   reset_L;

   reg    curr_state;
   reg    next_state;

   always @(posedge clk or negedge reset_L) begin
      if (!reset_L) begin
```

```
 next_state = `CMD;
      end else begin
 curr_state = next_state;
 case (curr_state)
   `CMD: begin
      if (convert) begin
 case (vci_cmd)
   `VCI_NOOP: begin
   end
   `VCI_READ, `VCI_LREAD: begin
      pci_cmd_be = `PCI_READ;
   end
   `VCI_WRITE: begin
      pci_cmd_be = `PCI_WRITE;
   end
 endcase // case(vci_cmd)
 next_state = `BE;
      end else begin
 next_state = `CMD;
      end
      next_state = `BE;
   end // case: `CMD
   `BE: begin
      pci_cmd_be = ~vci_be;
      next_state = `CMD;
   end
 endcase
      end
   end
endmodule

/*******************************************************/
/*
/* Project: VCI/PCI bridge model
/* File: fifo.v
/* Author: Annette Bunker
/* Date: Tue Jan 19 12:59:28 MST 1999
/* Modifications:
/*
/*******************************************************/

/* define readable state names    */
`define DECODE 1'b1
`define CLEAN 1'b0

/* define readable last_op names  */
`define INS 1'b1
`define REM 1'b0
```

```
/* define readable command widths */
`define QCMD_WIDTH  2
`define MCMD_WIDTH  2

/* define some readable bus widths */
`define DATA_WIDTH  8

/*************************************************************/
/* This module takes the clock as input and generates
/* a binary up counter.
/*************************************************************/

module bin (cntr, new, clk, reset_L);
   parameter width = 8;

   output [width-1:0] cntr;     // create output for counter
   reg    [width-1:0] cntr;     // create a register for the counter
   input        new;      // count only when asserted
   wire        new;      // wire new to input
   input        clk;      // create clock input
   wire        clk;      // create internal wire for clock
   input        reset_L;  // create input for reset line
   wire        reset_L;  // create internal wire for reset

   always @(posedge clk) begin
      if (!reset_L) begin
 cntr <= 0;
      end
      else if (new) begin
 cntr <= cntr + 1;
      end
   end
endmodule

/*************************************************************/
/* This module implements the memory to which the
/* queue writes its data.
/*************************************************************/

module memory (data_out, write_enable, head_ptr, read_enable,
       tail_ptr, data_in, clk);
   parameter  data_width = 32;
   parameter  cntr_width = 8;

   output [data_width-1:0] data_out;    // output for queue data
   reg    [data_width-1:0] data_out;    // wire queue data to output
   input      write_enable;// allow mem to write
   wire     write_enable;// wire write enable
   input  [cntr_width-1:0] head_ptr;    // input for dest address
```

```
   wire   [cntr_width-1:0] head_ptr;    // pointer to dest
   input     read_enable; // allow mem to read
   wire     read_enable; // wire read enable
   input  [cntr_width-1:0] tail_ptr;    // input for dest address
   wire   [cntr_width-1:0] tail_ptr;    // pointer to dest
   input  [data_width-1:0] data_in;     // input for queue data
   wire   [data_width-1:0] data_in;     // wire queue data to input
   input     clk;         // input for system clock
   wire     clk;         // wire system clock

   reg [data_width-1:0]    mem[(1<<cntr_width)-1:0];    // the memory

   wire [data_width-1:0]   mem0;
   assign     mem0 = mem[0];

   always @(posedge clk) begin
      if (read_enable) begin
 data_out = mem[head_ptr];
      end
      if (write_enable) begin
 mem[tail_ptr] = data_in;
      end
   end
endmodule

/*************************************************************/
/* This module implements full and empty, signals that
/* warn the other modules that the queue is full or empty,
/* as appropriate.  Modified for simultaneous read/write.
/*************************************************************/

module dual_bounds(full, empty, head_ptr, tail_ptr, reset_L);
   parameter cntr_width = 8;

   output            full;         // signal that the queue is full
   wire            full;         // wire up the full output
   output            empty;        // signal that the queue is empty
   wire            empty;        // wire up the empty output
   input [cntr_width-1:0] head_ptr;     // input for head pointer I/O
   wire  [cntr_width-1:0] head_ptr;     // bus for head pointer I/O
   input [cntr_width-1:0] tail_ptr;     // input for tail pointer I/O
   wire  [cntr_width-1:0] tail_ptr;     // bus for tail pointer I/O
   input    reset_L;      // input for system reset
   wire    reset_L;      // wire up system reset

   wire [cntr_width-1:0]one_tail;

   assign  empty = (!reset_L) || (head_ptr == tail_ptr);
   assign  one_tail = tail_ptr+1;
```

```
   assign  full = (!reset_L) || (head_ptr == one_tail);

endmodule

/**********************************************************/
/* This module wires my system together.  Modified for
/* simultaneous read/write.
/**********************************************************/

module dual_bin_fifo (data_from_mem, full, empty, data_to_mem, insert,
      front, remove, clk, reset_L);
   parameter data_width = 32;
   parameter cntr_width = 8;

   output [data_width-1:0] data_from_mem;// bus for data I/O
   wire   [data_width-1:0] data_from_mem;// bus for data I/O
   output     full;        // wires full I/O
   wire     full;        // wires full I/O
   output     empty;       // wires empty I/O
   wire     empty;       // wires empty I/O
   input  [data_width-1:0] data_to_mem; // bus for data I/O
   wire   [data_width-1:0] data_to_mem; // bus for data I/O
   input     insert;      // insert command
   wire     insert;      // wires insert command
   input     front;       // front command
   wire     front;       // wires front command
   input     remove;      // remove command
   wire     remove;      // wires remove command
   input     clk;         // wires clock I/O
   wire     clk;         // wires clock I/O
   input     reset_L;     // wires reset -- asserted low
   wire     reset_L;     // wires reset -- asserted low
   wire  [cntr_width-1:0]  head_ptr;    // bus for head pointer I/O
   wire  [cntr_width-1:0]  tail_ptr;    // bus for tail pointer I/O

   bin #(cntr_width) head (head_ptr, remove, clk, reset_L);
   bin #(cntr_width) tail (tail_ptr, insert, clk, reset_L);
   memory #(data_width, cntr_width) mem (data_from_mem, insert,
 head_ptr, (remove || front),
 tail_ptr, data_to_mem, clk);
   dual_bounds #(cntr_width) bnd (full, empty, head_ptr, tail_ptr,
  reset_L);
endmodule

/**********************************************************/
/*
/* Project: VCI/PCI bridge model
/* File: parity.v
/* Author: Annette Bunker
```

```
/* Date: Tue Feb  1 09:34:20 MST 2000
/* Modifications:
/*
/************************************************************/

/************************************************************/
/*  This module returns an even parity over the input bus.
/************************************************************/

module even_parity (parity, bus, calc_parity, clk, reset_L);
   parameter data_width = 32;

   output                 parity;
   reg                    parity;
   input[data_width-1:0]  bus;
   wire [data_width-1:0]  bus;
   input    calc_parity;
   wire    calc_parity;
   input    clk;
   wire    clk;
   input    reset_L;
   wire    reset_L;

   always @(posedge clk) begin
      if (!reset_L) begin
 parity = 0;
      end else begin
 if (calc_parity == `TRUE) begin
    parity = ^bus;
 end
      end
   end
endmodule

/************************************************************/
/*                                                          */
/* Project: VCI/PCI bridge model     */
/* File: response.v            */
/* Author: Annette Bunker     */
/* Date: Fri Oct 13 10:18:13 MDT 2000        */
/* Modifications:                                      */
/*                            */
/************************************************************/

/* define state names */
`define    IDLE     1'b0
`define    RESPOND     1'b1

module response (rspval, reop_remove, rdata_remove, rerr_remove,
```

```
rspack, empty, clk, reset_L);
  output                rspval;
  reg   rspval;
  output                reop_remove;
  reg   reop_remove;
  output   rdata_remove;
  reg   rdata_remove;
  output   rerr_remove;
  reg   rerr_remove;
  input   rspack;
  wire   rspack;
  input   empty;
  wire   empty;
  input   clk;
  wire   clk;
  input   reset_L;
  wire   reset_L;

  reg   curr_state;
  reg   next_state;

  always @(posedge clk or negedge reset_L) begin
     if (!reset_L) begin
rspval = `FALSE;
rdata_remove = `FALSE;
rerr_remove = `FALSE;
reop_remove = `FALSE;
next_state = `IDLE;
     end else begin
curr_state = next_state;
case (curr_state)
  `IDLE: begin
     rspval = `FALSE;
     if (!empty) begin
rdata_remove = `TRUE;
rerr_remove = `TRUE;
reop_remove = `TRUE;
next_state = `RESPOND;
     end else begin
next_state = `IDLE;
     end
  end
  `RESPOND: begin
     rdata_remove = `FALSE;
     rerr_remove = `FALSE;
     reop_remove = `FALSE;
     rspval = `TRUE;
     if (rspack) begin
next_state = `IDLE;
```

```
      end else begin
 next_state = `RESPOND;
      end
   end
 endcase
      end
   end
endmodule

/*********************************************************/
/*                                                       */
/* Project: VCI/PCI bridge model     */
/* File: to_pci.v     */
/* Author: Annette Bunker     */
/* Date: Tue Sep 26 10:22:55 MDT 2000        */
/* Modifications:                                    */
/*                                                       */
/*********************************************************/

/* define readable data widths */
`define D_WIDTH 4

module vp_xlator (cmdack, rspval, rdata, reop, rerror, req_l, frame_l,
  irdy_l, c_be_l, par, stop_l, perr_l, serr_l, ad,
  cmdval, address, be, cfixed, clen, cmd, contig,
  wdata, eop, plen, wrap, rspack, gnt_l, trdy_l,
  devsel_l, clk, reset_L);

   output           cmdack;// VCI ack to request
   wire          cmdack;// VCI ack to request
   output           rspval;// VCI valid response available
   wire          rspval;// VCI valid response available
   output [`D_WIDTH-1:0] rdata; // VCI response data
   wire   [`D_WIDTH-1:0] rdata; // VCI response data
   output   reop;  // VCI end of response
   wire  reop;  // VCI end of response
   output   rerror;// VCI error in request
   wire  rerror;// VCI error in request

   output   req_l; // PCI request for bus ownership
   wire  req_l; // PCI request for bus ownership
   output   frame_l;// PCI beginning of frame marker
   wire  frame_l;// PCI beginning of frame marker
   output   irdy_l;// PCI init ready to complete cycle
   wire  irdy_l;// PCI init ready to complete cycle
   output [3:0]    c_be_l;// PCI command/byte enables
   wire   [3:0]  c_be_l;// PCI command/byte enables
   output   par;   // PCI even parity
   wire  par;   // PCI even parity
```

```
inout   perr_l;// PCI parity error
inout   serr_l;// PCI system error -- fatal
inout [`D_WIDTH-1:0]  ad;    // PCI address/data

input   cmdval;        // VCI valid command available
wire   cmdval;        // VCI valid command available
input [`D_WIDTH-1:0]  address;       // VCI address
wire  [`D_WIDTH-1:0]  address;       // VCI address
input [3:0]  be;            // VCI byte enables
wire  [3:0]  be;            // VCI byte enables
input   cfixed;        // VCI opcode is fixed across chain
wire   cfixed;        // VCI opcode is fixed across chain
input   clen;          // VCI packets remaining in chain
wire   clen;          // VCI packets remaining in chain
input [1:0]   cmd;          // VCI command
wire  [1:0]   cmd;          // VCI command
input   contig;        // VCI contiguous addressing mode
wire   contig;        // VCI contiguous addressing mode
input [`D_WIDTH-1:0]  wdata;         // VCI data to write
wire  [`D_WIDTH-1:0]  wdata;         // VCI data to write
input   eop;           // VCI end of packet
wire   eop;           // VCI end of packet
input [7:0]   plen;         // VCI packet length:  0 = unspecified
wire  [7:0]   plen;         // VCI packet length:  0 = unspecified
input   wrap;          // VCI wrapping addressing mode
wire   wrap;          // VCI wrapping addressing mode
input   rspack;        // VCI response ack
wire   rspack;        // VCI response ack

input   gnt_l;         // PCI bus ownership granted
wire   gnt_l;         // PCI bus ownership granted
input   trdy_l;        // PCI target ready to complete cycle
wire   trdy_l;        // PCI target ready to complete cycle
input   stop_l;        // PCI target abends cycle
input   devsel_l;
wire   devsel_l;

input   clk;           // the global clock
wire   clk;           // the global clock
input   reset_L;       // global reset
wire   reset_L;       // global reset

wire   insert;
wire   addr_full;
wire   be_full;
wire   wdata_full;
wire   cmd_full;
wire   full;
```

```
   wire   be_empty;
   wire   addr_empty;
   wire   wdata_empty;
   wire   cmd_empty;
   wire   empty;
   wire   ad_front;
   wire   be_front;
   wire   wdata_front;
   wire   cmd_front;
   wire   calc_parity;
   wire [1:0]  new_cmd;
   wire [3:0]  new_be;
   wire [`D_WIDTH-1:0]  new_address;
   wire [`D_WIDTH-1:0]  new_wdata;
   wire   convert;
   wire   merge;
   wire   rdata_full;
   wire   rdata_empty;
   wire   rdata_insert;
   wire   rerr_full;
   wire   rerr_empty;
   wire   rerr_insert;
   wire   err;
   wire   new_eop;
   wire   eop_full;
   wire   reop_full;
   wire   eop_empty;
   wire   reop_empty;
   wire   reop_insert;
   wire   eop_remove;
   wire   reop_remove;
   wire   res_empty;

   assign   full = addr_full || be_full || wdata_full ||
 cmd_full;
   assign   empty = addr_empty || be_empty || wdata_empty
 || cmd_empty;
   assign   res_empty = rerr_empty || reop_empty ||
 rdata_empty;

   vci_loader #(2) vl (insert, cmdack, cmdval, full, cmd, clk,
       reset_L);
   dual_bin_fifo #(2, 2) cmd_q (new_cmd, cmd_full, cmd_empty, cmd,
insert, cmd_front, cmd_remove, clk,
reset_L);
   dual_bin_fifo #(4, 2) addr_q (new_address, addr_full, addr_empty,
  address,  insert, ad_front,
  ad_remove, clk, reset_L);
   dual_bin_fifo #(4, 2) be_q (new_be, be_full, be_empty, be, insert,
```

```
        be_front, be_remove, clk, reset_L);
    dual_bin_fifo #(4, 2) wdata_q (new_wdata, wdata_full, wdata_empty,
    wdata, insert, wdata_front,
    wdata_remove, clk, reset_L);
    dual_bin_fifo #(4, 2) rdata_q (rdata, rdata_full, rdata_empty, ad,
    rdata_insert, `FALSE, rdata_remove,
    clk, reset_L);
    dual_bin_fifo #(1, 2) rerr_q (rerror, rerr_full, rerr_empty, err,
  rerr_insert, `FALSE, rerr_remove,
  clk, reset_L);
    dual_bin_fifo #(1, 2) eop_q (new_eop, eop_full, eop_empty, eop,
insert, `FALSE, eop_remove, clk,
reset_L);
    dual_bin_fifo #(1, 2) reop_q (reop, reop_full, reop_empty, new_eop,
  reop_insert, `FALSE, reop_remove,
  clk, reset_L);
    cmd_cvt cvt (c_be_l, new_cmd, new_be, convert, clk, reset_L);
    ad_mrg adm (ad, new_address, new_wdata, merge, new_cmd, clk,
        reset_L);
    even_parity #(8) ep (par, {ad, c_be_l}, calc_parity, clk, reset_L);
    vci_unloader ul (irdy_l, req_l, frame_l, ad_front, wdata_front,
     be_front, cmd_front, ad_remove, eop_remove,
     wdata_remove, be_remove, cmd_remove, reop_insert,
     convert, merge, calc_parity, rdata_insert,
     rerr_insert, err, new_cmd, empty, gnt_l, trdy_l,
     stop_l, devsel_l, clk, reset_L);
    response res (rspval, reop_remove, rdata_remove, rerr_remove,
  rspack, res_empty, clk, reset_L);
endmodule

/*************************************************************/
/*
/* Project: VCI/PCI bridge model
/* File: unloadr.v
/* Author: Annette Bunker
/* Date: Wed Jul 19 10:27:57 MDT 2000
/* Modifications:
/*
/*************************************************************/

/* define state names */
`define    IDLE     3'b000
`define    REQ_ARB     3'b001
`define    ADDRESS  3'b010
`define    WRITE_DATA  3'b011
`define    READ_DATA  3'b100
`define    READ_DONE  3'b101
`define    RECOVER  3'b110
```

```
/* define PCI constants */
`define    ADDR_WIDTH 4
`define    CMD_WIDTH 4

module vci_unloader (irdy_l, req_l, frame_l, ad_front, wdata_front,
     be_front, cmd_front, ad_remove, eop_remove,
     wdata_remove, be_remove, cmd_remove, reop_insert,
     convert, merge, calc_parity, rdata_insert,
     rerr_insert, err, new_cmd, q_empty, gnt_l,
     trdy_l, stop_l, devsel_l, clk, reset_L);

   output                irdy_l;
   reg   irdy_l;
   output   req_l;
   reg   req_l;
   output   frame_l;
   reg    frame_l;
   output   ad_remove;
   reg   ad_remove;
   output   eop_remove;
   reg   eop_remove;
   output   wdata_remove;
   reg   wdata_remove;
   output   be_remove;
   reg   be_remove;
   output   cmd_remove;
   reg   cmd_remove;
   output   reop_insert;
   reg   reop_insert;
   output   ad_front;
   reg   ad_front;
   output   wdata_front;
   reg   wdata_front;
   output   be_front;
   reg   be_front;
   output   cmd_front;
   reg   cmd_front;
   output   convert;
   reg   convert;
   output   merge;
   reg   merge;
   output   calc_parity;
   reg   calc_parity;
   output   rdata_insert;
   reg   rdata_insert;
   output   rerr_insert;
   reg   rerr_insert;
   output   err;
   reg   err;
```

```
  input [1:0]   new_cmd;
  wire  [1:0]   new_cmd;
  input   q_empty;
  wire   q_empty;
  input   gnt_l;
  wire   gnt_l;
  input   trdy_l;
  wire   trdy_l;
  input   stop_l;
  wire   stop_l;
  input   devsel_l;
  wire   devsel_l;
  input   clk;
  wire   clk;
  input   reset_L;
  wire   reset_L;

  reg   [2:0]  curr_state; //current state bit
  reg   [2:0]   next_state; //next state bit

  always @(posedge clk or negedge reset_L) begin
     if (!reset_L) begin
irdy_l = `TRUE;
req_l = `TRUE;
frame_l = `TRUE;
convert = `FALSE;
merge = `FALSE;
be_front = `FALSE;
ad_front = `FALSE;
cmd_front = `FALSE;
rdata_insert = `FALSE;
err = `FALSE;
next_state = `IDLE;
     end else begin
        curr_state = next_state;
case (curr_state)
  `IDLE: begin
     rerr_insert = `FALSE;
     reop_insert = `FALSE;
     rdata_insert = `FALSE;
     ad_remove = `FALSE;
     be_remove = `FALSE;
     wdata_remove = `FALSE;
     eop_remove = `FALSE;
     cmd_remove = `FALSE;
     wdata_front = `FALSE;
     ad_front = `FALSE;
     cmd_front = `FALSE;
     calc_parity = `FALSE;
```

```
     if (!q_empty) begin
irdy_l = `TRUE;
req_l = `TRUE;
next_state = `REQ_ARB;
     end else begin
next_state = `IDLE;
     end
  end
  `REQ_ARB: begin
     rerr_insert = `FALSE;
     reop_insert = `FALSE;
     rdata_insert = `FALSE;
     ad_remove = `FALSE;
     be_remove = `FALSE;
     wdata_remove = `FALSE;
     eop_remove = `FALSE;
     cmd_remove = `FALSE;
     calc_parity = `FALSE;
     wdata_front = `FALSE;
     irdy_l = `TRUE;
     req_l = `FALSE;
     if (!gnt_l) begin
cmd_front = `TRUE;
be_front = `TRUE;
convert = `TRUE;
ad_front = `TRUE;
wdata_front = `TRUE;
merge = `TRUE;
next_state = `ADDRESS;
     end else begin
next_state = `REQ_ARB;
     end
  end
  `ADDRESS: begin
     cmd_front = `FALSE;
     be_front = `FALSE;
     wdata_front = `FALSE;
     ad_front = `FALSE;
     convert = `FALSE;
     merge = `FALSE;
     calc_parity = `TRUE;
     frame_l = `FALSE;
     if (new_cmd == `VCI_WRITE) begin
eop_remove = `TRUE;
next_state = `WRITE_DATA;
     end else begin
next_state = `READ_DATA;
     end
  end
```

```
  `WRITE_DATA: begin
     frame_l = `TRUE;
     irdy_l = `FALSE;
     eop_remove = `FALSE;
     if (devsel_l && !stop_l) begin      // TARGET-ABORT
ad_remove = `TRUE;
be_remove = `TRUE;
wdata_remove = `TRUE;
eop_remove = `TRUE;
cmd_remove = `TRUE;
err = `TRUE;
rerr_insert = `TRUE;
reop_insert = `TRUE;
rdata_insert = `TRUE;
next_state = `RECOVER;
     end else if (!devsel_l && !stop_l && trdy_l) begin   // RETRY
err = `FALSE;
rerr_insert = `TRUE;
reop_insert = `TRUE;
rdata_insert = `TRUE;
next_state = `REQ_ARB;
     end else if (!trdy_l) begin        // NORMAL TERMINATION
ad_remove = `TRUE;
be_remove = `TRUE;
wdata_remove = `TRUE;
eop_remove = `TRUE;
cmd_remove = `TRUE;
err = `FALSE;
rerr_insert = `TRUE;
reop_insert = `TRUE;
rdata_insert = `TRUE;
next_state = `RECOVER;
     end else begin                     // NO INPUT
next_state = `WRITE_DATA;
     end
  end
  `READ_DATA: begin
     irdy_l = `FALSE;
     calc_parity = `FALSE;
     if (devsel_l && !stop_l) begin   // TARGET-ABORT
ad_remove = `TRUE;
be_remove = `TRUE;
wdata_remove = `TRUE;
eop_remove = `TRUE;
cmd_remove = `TRUE;
err = `TRUE;
rerr_insert = `TRUE;
next_state = `RECOVER;
     end else if (!devsel_l && !stop_l && trdy_l) begin // RETRY
```

```
 err = `FALSE;
 rerr_insert = `TRUE;
 next_state = `REQ_ARB;
      end else if (!trdy_l) begin       // NORMAL TERMINATION
 ad_remove = `TRUE;
 be_remove = `TRUE;
 wdata_remove = `TRUE;
 eop_remove = `TRUE;
 cmd_remove = `TRUE;
 err = `FALSE;
 rerr_insert = `TRUE;
 next_state = `READ_DONE;
      end else begin
 next_state = `READ_DATA;      // NO INPUT
      end
   end
   `READ_DONE: begin
      ad_remove = `FALSE;
      be_remove = `FALSE;
      wdata_remove = `FALSE;
      eop_remove = `FALSE;
      cmd_remove = `FALSE;
      reop_insert = `TRUE;
      rdata_insert = `TRUE;
      rerr_insert = `FALSE;
      frame_l = `TRUE;
      next_state = `IDLE;
   end // case: `READ_DONE
   `RECOVER: begin
      ad_remove = `FALSE;
      be_remove = `FALSE;
      wdata_remove = `FALSE;
      eop_remove = `FALSE;
      cmd_remove = `FALSE;
      next_state = `IDLE;
   end
 endcase
      end
   end
endmodule

/*************************************************************/
/*
/* Project: VCI/PCI bridge model
/* File: unloadr.v
/* Author: Annette Bunker
/* Date: Wed Jul 19 10:27:57 MDT 2000
/* Modifications:
/*
```

```
/*************************************************************/

/* define state names */
`define    IDLE     3'b000
`define    REQ_ARB     3'b001
`define    ADDRESS  3'b010
`define    WRITE_DATA  3'b011
`define    READ_DATA  3'b100
`define    READ_DONE  3'b101
`define    RECOVER  3'b110

/* define PCI constants */
`define    ADDR_WIDTH 4
`define    CMD_WIDTH 4

module vci_unloader (irdy_l, req_l, frame_l, ad_front, wdata_front,
     be_front, cmd_front, ad_remove, eop_remove,
     wdata_remove, be_remove, cmd_remove, reop_insert,
     convert, merge, calc_parity, rdata_insert,
     rerr_insert, err, new_cmd, q_empty, gnt_l,
     trdy_l, stop_l, devsel_l, clk, reset_L);

   output                irdy_l;
   reg   irdy_l;
   output   req_l;
   reg   req_l;
   output   frame_l;
   reg    frame_l;
   output   ad_remove;
   reg   ad_remove;
   output   eop_remove;
   reg   eop_remove;
   output   wdata_remove;
   reg   wdata_remove;
   output   be_remove;
   reg   be_remove;
   output   cmd_remove;
   reg   cmd_remove;
   output   reop_insert;
   reg   reop_insert;
   output   ad_front;
   reg   ad_front;
   output   wdata_front;
   reg   wdata_front;
   output   be_front;
   reg   be_front;
   output   cmd_front;
   reg   cmd_front;
   output   convert;
```

```
  reg   convert;
  output   merge;
  reg   merge;
  output   calc_parity;
  reg   calc_parity;
  output   rdata_insert;
  reg   rdata_insert;
  output   rerr_insert;
  reg   rerr_insert;
  output   err;
  reg   err;
  input [1:0]   new_cmd;
  wire  [1:0]   new_cmd;
  input   q_empty;
  wire   q_empty;
  input   gnt_l;
  wire   gnt_l;
  input   trdy_l;
  wire   trdy_l;
  input   stop_l;
  wire   stop_l;
  input   devsel_l;
  wire   devsel_l;
  input   clk;
  wire   clk;
  input   reset_L;
  wire   reset_L;

  reg   [2:0]  curr_state; //current state bit
  reg   [2:0]   next_state; //next state bit

  always @(posedge clk or negedge reset_L) begin
     if (!reset_L) begin
irdy_l = `TRUE;
req_l = `TRUE;
frame_l = `TRUE;
convert = `FALSE;
merge = `FALSE;
be_front = `FALSE;
ad_front = `FALSE;
cmd_front = `FALSE;
rdata_insert = `FALSE;
err = `FALSE;
next_state = `IDLE;
     end else begin
        curr_state = next_state;
case (curr_state)
  `IDLE: begin
     rerr_insert = `FALSE;
```

```
      reop_insert = 'FALSE;
      rdata_insert = 'FALSE;
      ad_remove = 'FALSE;
      be_remove = 'FALSE;
      wdata_remove = 'FALSE;
      eop_remove = 'FALSE;
      cmd_remove = 'FALSE;
      wdata_front = 'FALSE;
      ad_front = 'FALSE;
      cmd_front = 'FALSE;
      calc_parity = 'FALSE;
      if (!q_empty) begin
irdy_l = 'TRUE;
req_l = 'TRUE;
next_state = 'REQ_ARB;
      end else begin
next_state = 'IDLE;
      end
  end
  'REQ_ARB: begin
      rerr_insert = 'FALSE;
      reop_insert = 'FALSE;
      rdata_insert = 'FALSE;
      ad_remove = 'FALSE;
      be_remove = 'FALSE;
      wdata_remove = 'FALSE;
      eop_remove = 'FALSE;
      cmd_remove = 'FALSE;
      calc_parity = 'FALSE;
      wdata_front = 'FALSE;
      irdy_l = 'TRUE;
      req_l = 'FALSE;
      if (!gnt_l) begin
cmd_front = 'TRUE;
be_front = 'TRUE;
convert = 'TRUE;
ad_front = 'TRUE;
wdata_front = 'TRUE;
merge = 'TRUE;
next_state = 'ADDRESS;
      end else begin
next_state = 'REQ_ARB;
      end
  end
  'ADDRESS: begin
      cmd_front = 'FALSE;
      be_front = 'FALSE;
      wdata_front = 'FALSE;
      ad_front = 'FALSE;
```

```
      convert = `FALSE;
      merge = `FALSE;
      calc_parity = `TRUE;
      frame_l = `FALSE;
      if (new_cmd == `VCI_WRITE) begin
eop_remove = `TRUE;
next_state = `WRITE_DATA;
      end else begin
next_state = `READ_DATA;
      end
  end
  `WRITE_DATA: begin
      frame_l = `TRUE;
      irdy_l = `FALSE;
      eop_remove = `FALSE;
      if (devsel_l && !stop_l) begin      // TARGET-ABORT
ad_remove = `TRUE;
be_remove = `TRUE;
wdata_remove = `TRUE;
eop_remove = `TRUE;
cmd_remove = `TRUE;
err = `TRUE;
rerr_insert = `TRUE;
reop_insert = `TRUE;
rdata_insert = `TRUE;
next_state = `RECOVER;
      end else if (!devsel_l && !stop_l && trdy_l) begin   // RETRY
err = `FALSE;
rerr_insert = `TRUE;
reop_insert = `TRUE;
rdata_insert = `TRUE;
next_state = `REQ_ARB;
      end else if (!trdy_l) begin        // NORMAL TERMINATION
ad_remove = `TRUE;
be_remove = `TRUE;
wdata_remove = `TRUE;
eop_remove = `TRUE;
cmd_remove = `TRUE;
err = `FALSE;
rerr_insert = `TRUE;
reop_insert = `TRUE;
rdata_insert = `TRUE;
next_state = `RECOVER;
      end else begin                     // NO INPUT
next_state = `WRITE_DATA;
      end
  end
  `READ_DATA: begin
      irdy_l = `FALSE;
```

```
     calc_parity = `FALSE;
     if (devsel_l && !stop_l) begin   // TARGET-ABORT
ad_remove = `TRUE;
be_remove = `TRUE;
wdata_remove = `TRUE;
eop_remove = `TRUE;
cmd_remove = `TRUE;
err = `TRUE;
rerr_insert = `TRUE;
next_state = `RECOVER;
     end else if (!devsel_l && !stop_l && trdy_l) begin // RETRY
err = `FALSE;
rerr_insert = `TRUE;
next_state = `REQ_ARB;
     end else if (!trdy_l) begin       // NORMAL TERMINATION
ad_remove = `TRUE;
be_remove = `TRUE;
wdata_remove = `TRUE;
eop_remove = `TRUE;
cmd_remove = `TRUE;
err = `FALSE;
rerr_insert = `TRUE;
next_state = `READ_DONE;
     end else begin
next_state = `READ_DATA;      // NO INPUT
     end
  end
  `READ_DONE: begin
     ad_remove = `FALSE;
     be_remove = `FALSE;
     wdata_remove = `FALSE;
     eop_remove = `FALSE;
     cmd_remove = `FALSE;
     reop_insert = `TRUE;
     rdata_insert = `TRUE;
     rerr_insert = `FALSE;
     frame_l = `TRUE;
     next_state = `IDLE;
  end // case: `READ_DONE
  `RECOVER: begin
     ad_remove = `FALSE;
     be_remove = `FALSE;
     wdata_remove = `FALSE;
     eop_remove = `FALSE;
     cmd_remove = `FALSE;
     next_state = `IDLE;
  end
endcase
     end
```

```
   end
endmodule

/**********************************************************/
/*
/* Project: VCI/PCI bridge model
/* File: vciloadr.v
/* Author: Richard Sharp
/* Date: Wed Jul 19 10:27:57 MDT 2000
/*
/**********************************************************/

/* define state names */
`define    NOOP    1'b0
`define    LOAD    1'b1

module vci_loader (q_insert, cmdack, cmdval, full, cmd, clk, reset_L);
   parameter cmd_width = 2;

   output                q_insert; // insert data to FIFO
   reg   q_insert; // register insert signal
   output   cmdack;   // command acknowledge to VCI
   reg   cmdack;   // register ACK signal
   input   cmdval;   // command valid signal from VCI
   wire   cmdval;   // wire command valid signal
   input   full;     // FIFO full signal
   wire   full;     // wire FIFO full signal
   input [cmd_width-1:0] cmd;      // VCI command
   wire  [cmd_width-1:0] cmd;      // wire VCI command
   input   clk;      // system clock
   wire   clk;      // wire system clock
   input   reset_L;  // reset signal asserted low
   wire   reset_L;  // wire reset signal

   reg   curr_state; //current state bit
   reg   next_state; //next state bit

   always @(posedge clk) begin
      if (!reset_L) begin
 q_insert = `FALSE;
    cmdack = `FALSE;
 next_state = `NOOP;
      end else begin
         curr_state = next_state;
 case (curr_state)
   `NOOP: begin
      if (cmdval && !full) begin
 q_insert = `TRUE;
 cmdack = `TRUE;
```

```
 next_state = `LOAD;
      end else begin
 next_state = `NOOP;
      end
   end
   `LOAD: begin
      q_insert = `FALSE;
      cmdack = `FALSE;
      next_state = `NOOP;
   end
 endcase
      end
   end
endmodule
```

B FormalCheck Verification Report

```
                          FormalCheck Project Report

/home/abunker/projects/vci/fc2/lsc_assert.fpj
Tue Jan 22 10:22:58 MST 2002

Title: VCI, using LSC assertions

-------------------------------------------------------------------------
FormalCheck Query Report

Query: eventually_cmdack
-------------------------------------------------------------------------
PROPERTIES:
-------------------------------------------------------------------------
Property: eventually_cmdack_prop
  Type: Eventually

  After: (xlator.reset_L == 1) && (xlator.cmdval == 1) &&
(xlator.clk == rising)
  Eventually: (xlator.cmdack == 1) && (xlator.clk == rising)

  Options:(None)
-------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_rspack_wo_rspval read_or_write_only req_until_gnt
  reset_L
-------------------------------------------------------------------------
```

```
STATE VARIABLES: (None)

RUN OPTIONS:
-------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
-------------------------------------------------------------------------
REDUCTION OPTIONS:
-------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
-------------------------------------------------------------------------
RESULT:
-------------------------------------------------------------------------
Verification Query eventually_cmdack VERIFIED! Tue Jan 15 19:35:06
2002 on server: hanuman

Query Data:
820 combinational variables
256 Possible input combinations per state
71 State variables: 3.54e+21 states

Verification Data:
Reachable space: 4.81e+06 states
4.81e+06 states reached.
State variable coverage: 71 variables, 100.00% average coverage
Search Depth: 134
Real time: 4 minutes 9 seconds
Memory Usage: 480.805 megabytes

-------------------------------------------------------------------------
FormalCheck Query Report

Query: cmdack_invalidates
-------------------------------------------------------------------------
PROPERTIES:
-------------------------------------------------------------------------
Property: cmdack_invalidates_prop
  Type: Eventually

  After: (xlator.reset_L == 1) && (xlator.cmdack == 1) &&
(xlator.clk == rising)
  Eventually: (xlator.cmdack == 0) && (xlator.clk == rising)

  Options:(None)
-------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_rspack_wo_rspval read_or_write_only req_until_gnt
  reset_L
```

```
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
------------------------------------------------------------------------
REDUCTION OPTIONS:
------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
------------------------------------------------------------------------
RESULT:
------------------------------------------------------------------------
Verification Query cmdack_invalidates VERIFIED! Tue Jan 15 19:39:40
2002 on server: hanuman

Query Data:
820 combinational variables
256 Possible input combinations per state
71 State variables: 3.54e+21 states

Verification Data:
Reachable space: 6.16e+06 states
6.16e+06 states reached.
State variable coverage: 71 variables, 100.00% average coverage
Search Depth: 134
Real time: 4 minutes 54 seconds
Memory Usage: 482.984 megabytes

------------------------------------------------------------------------
FormalCheck Query Report

Query: no_extra_cmdacks
------------------------------------------------------------------------
PROPERTIES:
------------------------------------------------------------------------
Property: no_extra_cmdacks_prop
  Type: Never

  After: (xlator.reset_L == 1) && (xlator.clk == rising)
  Never: (xlator.cmdack == 1) && (xlator.clk == rising)
  Unless: (xlator.cmdval == 1) && (xlator.clk == rising)

  Options:(None)
------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_cmdack_wo_cmdval no_rspack_wo_rspval
```

```
  read_or_write_only req_until_gnt reset_L
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
------------------------------------------------------------------------
REDUCTION OPTIONS:
------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
------------------------------------------------------------------------
RESULT:
------------------------------------------------------------------------
Verification Query no_extra_cmdacks VERIFIED! Tue Jan 15 19:44:58 2002
on server: hanuman

Query Data:
806 combinational variables
256 Possible input combinations per state
67 State variables: 4.4e+20 states

Verification Data:
8.56e+05 states reached.
State variable coverage: 67 variables, 99.25% average coverage
Search Depth: 84
Real time: 2 minutes 40 seconds
Memory Usage: 473.473 megabytes

------------------------------------------------------------------------
FormalCheck Query Report

Query: eventually_rdata
------------------------------------------------------------------------
PROPERTIES:
------------------------------------------------------------------------
Property: eventually_rdata_prop
  Type: Eventually

  After: (xlator.cmdval == 1) && (xlator.clk == rising)
  Eventually: (xlator.rdata == stable) && (xlator.rspval == 1) &&
    (xlator.clk == rising)

  Options:(None)
------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_rspack_wo_rspval read_or_write_only req_until_gnt
```

```
  reset_L
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
------------------------------------------------------------------------
REDUCTION OPTIONS:
------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
------------------------------------------------------------------------
RESULT:
------------------------------------------------------------------------
Verification Query eventually_rdata Terminated - At user request

Query Data:
1.29e3 combinational variables
1.04e6 Possible input combinations per state
143 State variables: 1.67e+43 states

Real time: 9 minutes 10 seconds
Memory Usage: 472.162 megabytes

------------------------------------------------------------------------
FormalCheck Query Report

Query: rdata_invalidates
------------------------------------------------------------------------
PROPERTIES:
------------------------------------------------------------------------
Property: rdata_invalidates_prop
  Type: Eventually

  After: (xlator.rdata == stable) && (xlator.rspval == 1) &&
    (xlator.clk == rising)
  Eventually: (xlator.rspval == 0) && (xlator.clk == rising)

  Options:(None)
------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_rspack_wo_rspval read_or_write_only req_until_gnt
  reset_L
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
```

```
-----------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
-----------------------------------------------------------------------
REDUCTION OPTIONS:
-----------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
-----------------------------------------------------------------------
RESULT:
-----------------------------------------------------------------------
Verification Query Data:
1.29e3 combinational variables
1.04e6 Possible input combinations per state
143 State variables: 1.67e+43 states
Real time: 60 minutes 7 seconds
Memory Usage: 472.187 megabytes

-----------------------------------------------------------------------
FormalCheck Query Report

Query: eventually_reop
-----------------------------------------------------------------------
PROPERTIES:
-----------------------------------------------------------------------
Property: eventually_reop_prop
  Type: Eventually

  After: (xlator.cmdval == 1) && (xlator.clk == rising)
  Eventually: (xlator.reop == stable) && (xlator.rspval == 1) &&
    (xlator.clk == rising)

  Options:(None)
-----------------------------------------------------------------------
CONSTRAINTS:  clk cmdval_after_reset cmdval_until_cmdack eventually_arbitrate
    eventually_not_retry eventually_rspack eventually_trdy no_rspack_wo_rspval
    read_or_write_only req_until_gnt reset_L
-----------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
-----------------------------------------------------------------------
  Algorithm: Symbolic (BDD)-
----------------------------------------------------------------------
REDUCTION OPTIONS:
-----------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
-----------------------------------------------------------------------
RESULT:
```

```
------------------------------------------------------------------------
Verification Query Data:
960 combinational variables
512 Possible input combinations per state
87 State variables: 2.32e+26 states
Real time: 65 minutes 18 seconds
Memory Usage: 471.302 megabytes

------------------------------------------------------------------------
FormalCheck Query Report

Query: reop_invalidates
------------------------------------------------------------------------
PROPERTIES:
------------------------------------------------------------------------
Property: reop_invalidates_prop
  Type: Eventually

  After: (xlator.reop == stable) && (xlator.rspval == 1) &&
    (xlator.clk == rising)
  Eventually: (xlator.rspval == 0) && (xlator.clk == rising)

  Options:(None)
------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_rspack_wo_rspval read_or_write_only req_until_gnt
  reset_L
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
------------------------------------------------------------------------
REDUCTION OPTIONS:
------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
------------------------------------------------------------------------
RESULT:
------------------------------------------------------------------------
Verification Query Data:
960 combinational variables
512 Possible input combinations per state
87 State variables: 2.32e+26 states
Real time: 76 minutes 4 seconds
Memory Usage: 471.302 megabytes
```

```
------------------------------------------------------------------------
FormalCheck Query Report

Query: eventually_rerror
------------------------------------------------------------------------
PROPERTIES:
-----------------------------------------------------------------------
Property: eventually_rerror_prop
  Type: Eventually

  After: (xlator.cmdval == 1) && (xlator.clk == rising)
  Eventually: (xlator.rerror == stable) && (xlator.rspval == 1) &&
    (xlator.clk == rising)

  Options:(None)
------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_rspack_wo_rspval read_or_write_only req_until_gnt
  reset_L
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
------------------------------------------------------------------------
REDUCTION OPTIONS:
------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
------------------------------------------------------------------------
RESULT:
------------------------------------------------------------------------
Verification Query eventually_rerror VERIFIED! Wed Jan 16 14:21:14
2002 on server: hanuman

Query Data:
868 combinational variables
256 Possible input combinations per state
78 State variables: 4.53e+23 states

Verification Data:
Reachable space: 5.68e+07 states
5.68e+07 states reached.
State variable coverage: 78 variables, 100.00% average coverage
Search Depth: 146
Real time: 28 minutes 56 seconds
Memory Usage: 562.995 megabytes
```

```
-------------------------------------------------------------------------
FormalCheck Query Report

Query: rerror_invalidates
-------------------------------------------------------------------------
PROPERTIES:
-------------------------------------------------------------------------
Property: rerror_invalidates_prop
  Type: Eventually

  After: (xlator.rerror == stable) && (xlator.rspval == 1) &&
    (xlator.clk == rising)
  Eventually: (xlator.rspval == 0) && (xlator.clk == rising)

  Options:(None)
-------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_rspack_wo_rspval read_or_write_only req_until_gnt
  reset_L
-------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
-------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
-------------------------------------------------------------------------
REDUCTION OPTIONS:
-------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
-------------------------------------------------------------------------
RESULT:
-------------------------------------------------------------------------
Verification Query rerror_invalidates VERIFIED! Wed Jan 16 14:50:35
2002 on server: hanuman

Query Data:
868 combinational variables
256 Possible input combinations per state
78 State variables: 4.53e+23 states

Verification Data:
Reachable space: 5.22e+07 states
5.22e+07 states reached.
State variable coverage: 78 variables, 100.00% average coverage
Search Depth: 146
Real time: 21 minutes 45 seconds
```

```
Memory Usage: 530.416 megabytes

------------------------------------------------------------------------
FormalCheck Query Report

Query: eventually_rspval
------------------------------------------------------------------------
PROPERTIES:
------------------------------------------------------------------------
Property: eventually_rspval_prop
  Type: Eventually

  After: (xlator.reset_L == 1) && (xlator.cmdval == 1) &&
(xlator.clk == rising)
  Eventually: (xlator.reset_L == 1) && (xlator.rspval == 1) &&
    (xlator.clk == rising)

  Options:(None)
------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_rspack_wo_rspval read_or_write_only req_until_gnt
  reset_L
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
------------------------------------------------------------------------
REDUCTION OPTIONS:
------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
------------------------------------------------------------------------
RESULT:
------------------------------------------------------------------------
Verification Query eventually_rspval VERIFIED! Tue Jan 15 18:11:42
2002 on server: hanuman

Query Data:
820 combinational variables
256 Possible input combinations per state
71 State variables: 3.54e+21 states

Verification Data:
Reachable space: 7.34e+06 states
7.34e+06 states reached.
State variable coverage: 71 variables, 100.00% average coverage
```

```
Search Depth: 136
Real time: 7 minutes 31 seconds
Memory Usage: 496.886 megabytes

------------------------------------------------------------------------
FormalCheck Query Report

Query: rspval_invalidates
------------------------------------------------------------------------
PROPERTIES:
------------------------------------------------------------------------
Property: rspval_invalidates_prop
  Type: Eventually

  After: (xlator.rspval == 1) && (xlator.clk == rising)
  Eventually: (xlator.rspval == falling)

  Options:(None)
------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_rspack_wo_rspval read_or_write_only req_until_gnt
  reset_L
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
------------------------------------------------------------------------
REDUCTION OPTIONS:
------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
------------------------------------------------------------------------
RESULT:
------------------------------------------------------------------------
Verification Query rspval_invalidates VERIFIED! Wed Jan 16 03:55:15 2002 on
server: hanuman

Query Data:
820 combinational variables
256 Possible input combinations per state
72 State variables: 7.08e+21 states

Verification Data:
Reachable space: 6.62e+06 states
6.62e+06 states reached.
State variable coverage: 72 variables, 100.00% average coverage
```

```
Search Depth: 134
Real time: 6 minutes 34 seconds
Memory Usage: 499.024 megabytes

------------------------------------------------------------------------
FormalCheck Query Report

Query: no_extra_rspvals
------------------------------------------------------------------------
PROPERTIES:
------------------------------------------------------------------------
Property: no_extra_rspvals_prop
  Type: Never

  Never: (xlator.rspval == 1) && (xlator.clk == rising)
  Unless: (xlator.cmdval == 1) && (xlator.clk == rising)

  Options:(None)
------------------------------------------------------------------------
CONSTRAINTS:  clk cmdval_after_reset cmdval_until_cmdack eventually_arbitrate
    eventually_not_retry eventually_rspack eventually_trdy no_cmdack_wo_cmdval
    no_rspack_wo_rspval read_or_write_only req_until_gnt reset_L
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
------------------------------------------------------------------------
REDUCTION OPTIONS:
------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
------------------------------------------------------------------------
RESULT:
------------------------------------------------------------------------
Verification Query no_extra_rspvals VERIFIED! Wed Jan 16 04:02:13 2002
on server: hanuman

Query Data:
805 combinational variables
256 Possible input combinations per state
66 State variables: 2.21e+20 states

Verification Data:
7 states reached.
State variable coverage: 66 variables, 53.03% average coverage
Search Depth: 7
Real time: 2 minutes 15 seconds
```

```
Memory Usage: 470.786 megabytes

--------------------------------------------------------------------------
FormalCheck Query Report

Query: eventually_req
--------------------------------------------------------------------------
PROPERTIES:
--------------------------------------------------------------------------
Property: eventually_req_prop
  Type: Eventually

  After: (xlator.cmdval == 1) && (xlator.clk == rising)
  Eventually: (xlator.req_l == 0) && (xlator.clk == rising)

  Options:(None)
--------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_cmdack_wo_cmdval no_rspack_wo_rspval
  read_or_write_only req_until_gnt reset_L
--------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
--------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
--------------------------------------------------------------------------
REDUCTION OPTIONS:
--------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
--------------------------------------------------------------------------
RESULT:
--------------------------------------------------------------------------
Verification Query eventually_req VERIFIED! Wed Jan 16 04:04:53 2002
on server: hanuman

Query Data:
821 combinational variables
256 Possible input combinations per state
72 State variables: 7.08e+21 states

Verification Data:
Reachable space: 6.04e+06 states
6.04e+06 states reached.
State variable coverage: 72 variables, 100.00% average coverage
Search Depth: 136
Real time: 4 minutes 25 seconds
```

```
Memory Usage: 483.222 megabytes

------------------------------------------------------------------------
FormalCheck Query Report

Query: eventually_frame
------------------------------------------------------------------------
PROPERTIES:
------------------------------------------------------------------------
Property: eventually_frame_prop
  Type: Eventually

  After: (xlator.cmdval == 1) && (xlator.clk == rising)
  Eventually: (xlator.frame_l == 0) && (xlator.clk == rising)

  Options:(None)
------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_cmdack_wo_cmdval no_rspack_wo_rspval
  read_or_write_only req_until_gnt reset_L
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
------------------------------------------------------------------------
REDUCTION OPTIONS:
------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
------------------------------------------------------------------------
RESULT:
------------------------------------------------------------------------
Verification Query eventually_frame VERIFIED! Wed Jan 16 04:09:42 2002
on server: hanuman

Query Data:
821 combinational variables
256 Possible input combinations per state
72 State variables: 7.08e+21 states

Verification Data:
Reachable space: 7.23e+06 states
7.23e+06 states reached.
State variable coverage: 72 variables, 100.00% average coverage
Search Depth: 141
Real time: 5 minutes 46 seconds
```

```
Memory Usage: 487.776 megabytes

--------------------------------------------------------------------------
FormalCheck Query Report

Query: eventually_cbe
--------------------------------------------------------------------------
PROPERTIES:
--------------------------------------------------------------------------
Property: eventually_cbe_prop
  Type: Eventually

  After: (xlator.cmdval == 1) && (xlator.clk == rising)
  Eventually: (xlator.c_be_l == stable) && (xlator.clk == rising)

  Options:(None)
--------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_cmdack_wo_cmdval no_rspack_wo_rspval
  read_or_write_only req_until_gnt reset_L
--------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
--------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
--------------------------------------------------------------------------
REDUCTION OPTIONS:
--------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
--------------------------------------------------------------------------
RESULT:
--------------------------------------------------------------------------
Verification Query eventually_cbe VERIFIED! Wed Jan 16 09:21:40 2002
on server: hanuman

Query Data:
1.02e3 combinational variables
4.09e3 Possible input combinations per state
103 State variables: 1.52e+31 states

Verification Data:
Reachable space: 7.73e+11 states
7.73e+11 states reached.
State variable coverage: 103 variables, 100.00% average coverage
Search Depth: 136
Real time: 9 minutes 1 seconds
```

```
Memory Usage: 504.381 megabytes

--------------------------------------------------------------------------
FormalCheck Query Report

Query: eventually_irdy
--------------------------------------------------------------------------
PROPERTIES:
--------------------------------------------------------------------------
Property: eventually_irdy_prop
  Type: Eventually

  After: (xlator.cmdval == 1) && (xlator.clk == rising)
  Eventually: (xlator.irdy_l == 0) && (xlator.clk == rising)

  Options:(None)
--------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_cmdack_wo_cmdval no_rspack_wo_rspval
  read_or_write_only req_until_gnt reset_L
--------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
--------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
--------------------------------------------------------------------------
REDUCTION OPTIONS:
--------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
--------------------------------------------------------------------------
RESULT:
--------------------------------------------------------------------------
Verification Query eventually_irdy VERIFIED! Wed Jan 16 09:31:06 2002
on server: hanuman

Query Data:
830 combinational variables
256 Possible input combinations per state
73 State variables: 1.42e+22 states

Verification Data:
Reachable space: 6.82e+06 states
6.82e+06 states reached.
State variable coverage: 73 variables, 100.00% average coverage
Search Depth: 134
Real time: 17 minutes 8 seconds
```

```
Memory Usage: 518.283 megabytes

------------------------------------------------------------------------
FormalCheck Query Report

Query: eventually_trdy
------------------------------------------------------------------------
PROPERTIES:
------------------------------------------------------------------------
Property: eventually_trdy_prop
  Type: Eventually

  After: (xlator.cmdval == 1) && (xlator.cmd == 1) &&
(xlator.clk == rising)
  Eventually: (xlator.trdy_l == 0) && (xlator.clk == rising)

  Options:(None)
------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_cmdack_wo_cmdval no_rspack_wo_rspval
  read_or_write_only req_until_gnt reset_L
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
------------------------------------------------------------------------
REDUCTION OPTIONS:
------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
------------------------------------------------------------------------
RESULT:
------------------------------------------------------------------------
Verification Query eventually_trdy VERIFIED! Wed Jan 16 09:48:40 2002
on server: hanuman

Query Data:
821 combinational variables
256 Possible input combinations per state
72 State variables: 7.08e+21 states

Verification Data:
Reachable space: 9.34e+06 states
9.34e+06 states reached.
State variable coverage: 72 variables, 100.00% average coverage
Search Depth: 134
```

```
Real time: 4 minutes 21 seconds
Memory Usage: 481.01 megabytes

------------------------------------------------------------------------
FormalCheck Query Report

Query: no_extra_frames
------------------------------------------------------------------------
PROPERTIES:
------------------------------------------------------------------------
Property: no_extra_frames_prop
  Type: Never

  Never: (xlator.frame_l == 0) && (xlator.clk == rising)
  Unless: (xlator.cmdval == 1) && (xlator.cmdack == 1) &&
(xlator.clk == rising)

  Options:(None)
------------------------------------------------------------------------
CONSTRAINTS: clk cmdval_after_reset cmdval_until_cmdack
  eventually_arbitrate eventually_not_retry eventually_rspack
  eventually_trdy no_cmdack_wo_cmdval no_rspack_wo_rspval
  read_or_write_only req_until_gnt reset_L
------------------------------------------------------------------------
STATE VARIABLES: (None)

RUN OPTIONS:
------------------------------------------------------------------------
  Algorithm: Symbolic (BDD)
------------------------------------------------------------------------
REDUCTION OPTIONS:
------------------------------------------------------------------------
  Reduction Technique: 1-Step
  Reduction Seed: Empty
------------------------------------------------------------------------
RESULT:
------------------------------------------------------------------------
Verification Query no_extra_frames VERIFIED! Wed Jan 16 10:02:54 2002
on server: hanuman

Query Data:
814 combinational variables
256 Possible input combinations per state
67 State variables: 4.4e+20 states

Verification Data:
10 states reached.
State variable coverage: 67 variables, 58.96% average coverage
Search Depth: 8
```

```
Real time: 2 minutes 15 seconds
Memory Usage: 470.778 megabytes

-------------------------------------------------------------------------
CONSTRAINTS:
-------------------------------------------------------------------------
Constraint address_stable_cmdval
  Type: Always

  After: (xlator.cmdval == 1)
  Assume Always: (xlator.address == stable)
  Unless: (xlator.cmdval == 0)

  Options: (None)
-------------------------------------------------------------------------
Clock Constraint: clk
  Signal: xlator.clk
  Extract: No
  Default: Yes

  Start: Low
  1st Duration: 1
  2nd Duration: 1
-------------------------------------------------------------------------
Constraint cmdval_after_reset
  Type: Never

  Assume Never: (xlator.reset_L == 0) && (xlator.cmdval == 1)

  Options: Default
-------------------------------------------------------------------------
Constraint cmdval_until_cmdack
  Type: Always

  After: (xlator.cmdval == 1) && (xlator.clk == rising)
  Assume Always: (xlator.cmdval == 1)
  Unless: (xlator.cmdval == 1) && (xlator.cmdack == 1) &&
(xlator.clk == rising)

  Options: Default
-------------------------------------------------------------------------
Constraint eventually_arbitrate
  Type: Eventually

  After: (xlator.req_l == 0) && (xlator.clk == rising) &&
    (xlator.reset_L == 1)
  Assume Eventually: (xlator.gnt_l == 0) && (xlator.clk == rising)

  Options: Default
```

```
---------------------------------------------------------------------------
Constraint eventually_not_retry
  Type: Eventually

  After: (@retry) && ((xlator.ul.curr_state == 4) ||
    (xlator.ul.curr_state == 3)) && (xlator.clk == rising)
  Assume Eventually: (xlator.stop_l == 1) &&
    (xlator.trdy_l == 0) && ((xlator.ul.curr_state == 4) ||
    (xlator.ul.curr_state == 3)) && (xlator.clk == rising)

  Options: Default
---------------------------------------------------------------------------
Constraint eventually_rspack
  Type: Eventually

  After: (xlator.rspval == 1) && (xlator.clk == rising)
  Assume Eventually: (xlator.rspack == 1) && (xlator.clk == rising)

  Options: Default
---------------------------------------------------------------------------
Constraint eventually_trdy
  Type: Eventually

  After: (xlator.reset_L == 1) && (xlator.frame_l == 0)
  Assume Eventually: (xlator.trdy_l == 0) && (xlator.clk == rising)

  Options: Default
---------------------------------------------------------------------------
Constraint no_cmdack_wo_cmdval
  Type: Never

  Assume Never: (xlator.cmdack == 1) && (xlator.clk == rising)
  Unless: (xlator.cmdval == 1) && (xlator.clk == rising)

  Options: Default
---------------------------------------------------------------------------
Constraint no_rspack_wo_rspval
  Type: Never

  Assume Never: (xlator.rspack == 1) && (xlator.clk == rising)
  Unless: (xlator.rspval == 1) && (xlator.clk == rising)

  Options: Default
---------------------------------------------------------------------------
Constraint read_or_write_only
  Type: Always

  Assume Always: (xlator.cmd == 1) || (xlator.cmd == 2)
```

```
  Options: Default
------------------------------------------------------------------------
Constraint req_until_gnt
  Type: Always

  After: (xlator.req_l == 0) && (xlator.clk == rising)
  Assume Always: (xlator.req_l == 0)
  Unless: (xlator.req_l == 0) && (xlator.gnt_l == 0) &&
(xlator.clk == rising)

  Options: Default
------------------------------------------------------------------------
Reset Constraint: reset_L
  Signal: xlator.reset_L
  Default: Yes

  Start: Low

  Transition Duration Value
       Start        4     0
              forever     1

-----------------------------------------------------------------------

-----------------------------------------------------------------------
EXPRESSION MACROS:
-----------------------------------------------------------------------
retry: ((xlator.stop_l == 0) && (xlator.devsel_l == 0)) &&
        (xlator.trdy_l == 1)
```