Formal Verification of the PCI Local Bus

Project Report for EE382M

Jayanta Bhadra, Debaleena Das and Abhijit Jas
Electrical and Computer Engineering Department
University of Texas at Austin
Austin, TX 78712
{jay,ddas,jas}@ece.utexas.edu

Abstract

Symbolic Model Checking is a successful technique for checking properties of large finite state systems. This method has been used to verify a large number of real-world hardware designs. In this project we have used the VIS model checking system to formally verify an abridged implementation of the PCI Local bus.

We have designed and implemented the PCI Bus in the subset of Verilog supported by VIS. We have scaled down the system to an extent to make sure we get results in reasonable time. We have verified properties which are implied by the protocol, properties regarding bus arbitration, deadlock and livelock.

Contents

1 Introduction

1.1 Overview of formal verification concepts

Formal verification methods are a rigorous, methodical means to prove relationships between two descriptions of behavior. The behavior may be represented in VHDL or Verilog, or it may be described using special purpose languages offering richer constructs for mathematical assertions and properties. Formal verification tools are analogous to static timing analysis tools, in that they replace dynamic simulation with mathematics, provide exhaustive analysis in lieu of vectors, and use signal constraints to eliminate pessimism [1].

Formal verification can be used in three different ways within the design process. First, to verify that a specification of the design satisfies a given set of properties. Second, to formally derive an implementation from the specification. Third, to prove that an existing implementation matches the specification, or to confirm expected differences.

The first usage scenario is supported by a class of tools known as *model checkers* , which can check whether given properties always hold within a model of the behavior. The second scenario is supported by *theorem provers*, which use inductive reasoning at each step of decomposition to guarantee a correct by construction implementation for the specification. The third class of tools is called *equivalence checkers* , because they compare two revisions of the same behavior. A combinational checker only proves that equivalent inputs produce equivalent outputs, but does not comprehend the dimension of time (e.g.; clocked memory elements). Strict sequential checkers can prove that equivalent inputs over time will produce equivalent outputs over time.

The most commonly used internal data representation is the *binary decision diagram* (*BDD*). It was quickly found that the size of a *BDD* is highly dependent on the ordering of the input signals, leading to special ordering algorithms. Later, the size of ordered *BDD*s (*OBDD*s) was further reduced by folding common subtrees into a graph form. Typed decision graphs (*TDG*) improve on the reduced, ordered *BDD* (*ROBDD*) by allowing links in the graph to invert the sense of the logic during traversal, enabling better collapsing of subtrees.

1.2 Our Approach

PCI has been an industry standard bus for a long time. There has been some work done on the specification and verification of the *PCI* bus especially performance verification [4]. The *PCI Local Bus* typically has multiple *masters* and *targets*. We have implemented and verified an abridged version of the bus which has only one master and one target. A *bus* is typically very rich in properties. Later in this report we will highlight several properties of the *PCI Local Bus* which can straightaway be identified from its protocol specifications. We thus strive to come up with a very good benchmark for the usefullness of *VIS* as a formal verification tool.

2 PCI Local Bus Overview

The PCI Local Bus is a high performance, 32-bit or 64-bit bus with multiplexed address and data lines. The bus is intended for use as an interconnect mechanism between highly integrated peripheral controller components, peripheral add-in boards, and processor/memory systems. The block diagram (Figure 1) shows a typical PCI Local Bus system architecture. This example is not intended to imply any specific architectural limits. In this example, the processor/cache/memory subsystem is connected to PCI through a PCI bridge. This bridge provides a low latency path through which the processor may directly access PCI devices mapped anywhere in the memory or I/O address spaces. It also provides a high bandwidth path allowing PCI masters direct access to main memory. The bridge may optionally include such functions as data buffering/posting and PCI central functions (e.g., arbitration).

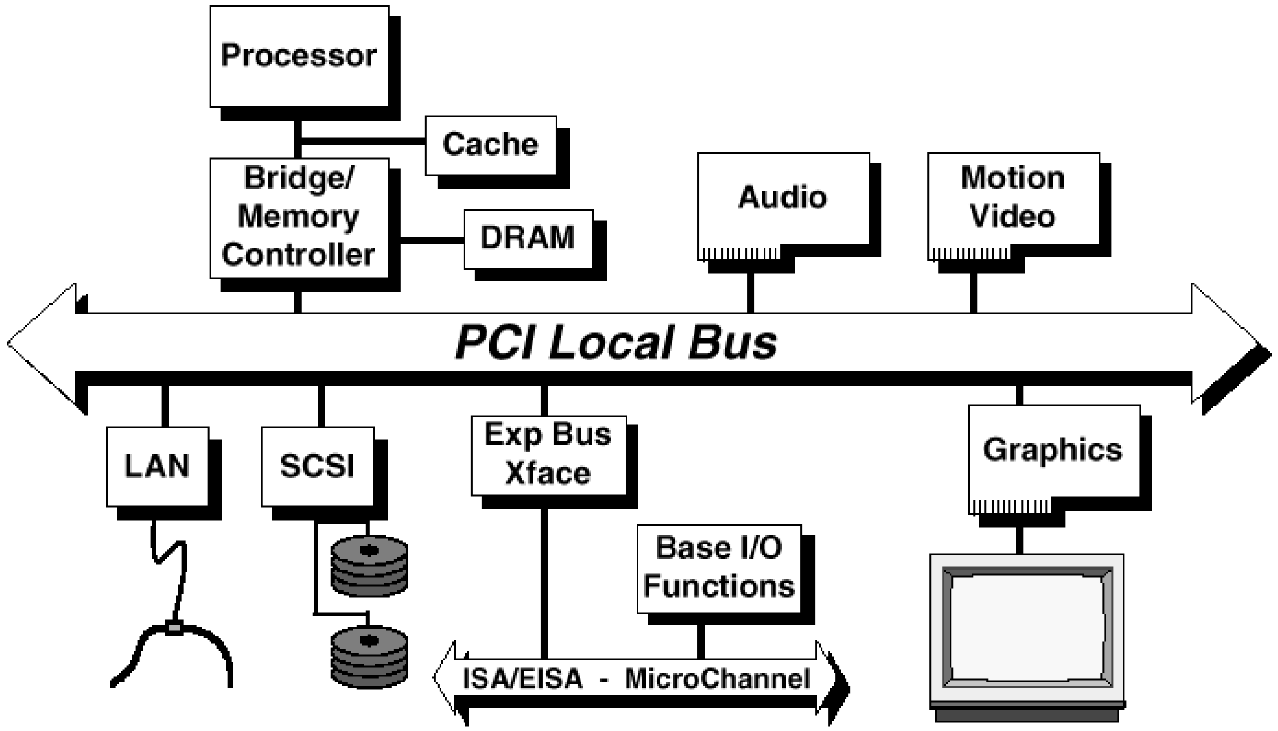


Figure 1: Block Diagram of a PCI Local Bus

The important pins of the PCI Local Bus are shown in the block diagram of Figure 2.

Abbreviation	Signal Type
in	*Input* is a standard input-only signal
out	*Output* is a standard active driver
t/s	*Tri-State* is a bi-directional, tri-state input/output pin
s/t/s	*Sustained Tri-State* is an active low tri-state signal owned and driven by one and only one agent at a time

Table 1: Signal Types

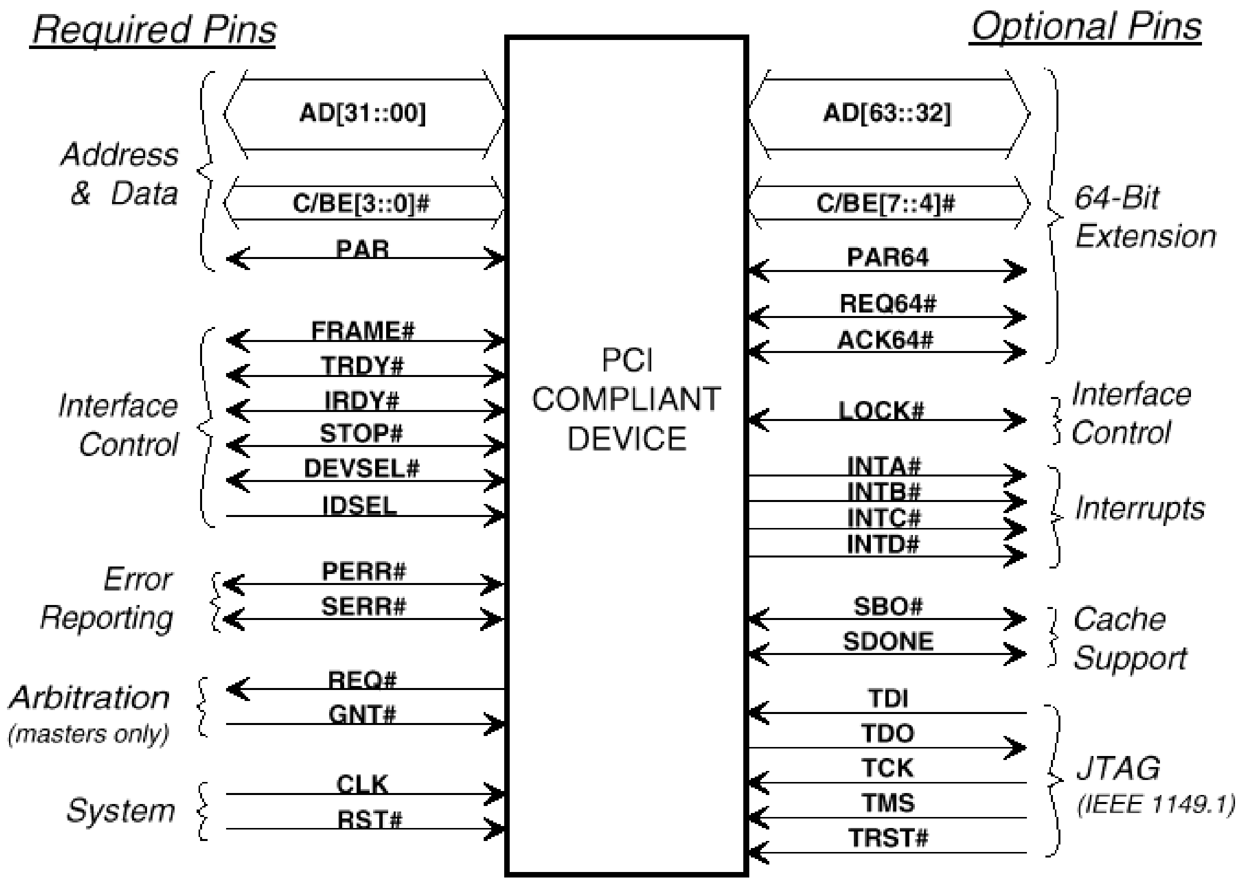


Figure 2: A block Diagram of the PCI Pin List

2.1 Some Important Signal Type Definitions

The following signal type definitions are from the view point of all devices other than the arbiter or central resource. For the arbiter, **REQ#** is an input, **GNT#** is an output, and other *PCI* signals for the arbiter have the same direction as a Master or Target. The central resource is a ”logical” device where all system type functions are located.

The agent that drives an *s/t/s* pin low must drive it high for at least one clock before letting it float. A new agent cannot start driving a *s/t/s* signal any sooner than one clock after the previous owner tri-states it.

A # symbol at the end of a signal name indicates that the active state occurs when the signal is at a low voltage. When the # symbol is absent, the signal is active at a high voltage. The signaling method used on each pin is shown below following the signal name.

- **CLK** - *in* - *Clock* provides timing for all transactions on PCI and is an input to every PCI device. All other PCI signals, except some are sampled on the rising edge of **CLK** and all other timing parameters are defined with respect to this edge.
- **RST#** - *in* - *Reset* is used to bring PCI-specific registers, sequencers, and signals to a consistent state.
- **AD[31::00]** - *t/s* - *Address and Data* are multiplexed on the same PCI pins. A bus transaction consists of an address phase followed by one or more data phases. PCI supports both read and write bursts.
- **C/BE[3::0]#** - *t/s* - *Bus Command and Byte Enables* are multiplexed on the same *PCI* pins.
- **FRAME#** - *s/t/s Frame* is asserted to indicate a bus transaction is beginning. While FRAME# is asserted, data transfers continue. When FRAME# is deasserted, the transaction is in the final data phase or has completed.
- **IRDY#** - *s/t/s* - *Initiator Ready* indicates the initiating agent's (bus master's) ability to complete the current data phase of the transaction. IRDY# is used in conjunction with TRDY#. A data phase is completed on any clock both IRDY# and TRDY# are sampled asserted. During a write, IRDY# indicates that valid data is present on AD[31::00]. During a read, it indicates the master is prepared to accept data. Wait cycles are inserted until both IRDY# and TRDY# are asserted together.
- **TRDY#** - *s/t/s* - *Target Ready* indicates the target agent's (selected device's) ability to complete the current data phase of the transaction. During a read, TRDY# indicates that valid data is present on AD[31::00]. During a write, it indicates the target is prepared to accept data.
- **STOP#** *s/t/s Stop* indicates the current target is requesting the master to stop the current transaction.
- **LOCK#** - *s/t/s* - *Lock* indicates an atomic operation that may require multiple transactions to complete. When LOCK# is asserted, non-exclusive transactions may proceed to an address that is not currently locked.
- **DEVSEL#** - *s/t/s* - *Device Select*, when actively driven, indicates the driving device has decoded its address as the target of the current access.

- **REQ#** - *t/s* - *Request* indicates to the arbiter that this agent desires use of the bus.
- **GNT#** - *t/s Grant* indicates to the agent that access to the bus has been granted.

2.2 Where We Fit

The PCI Local bus can have several device controllers sitting on it. It is illustrated by Figure 3.

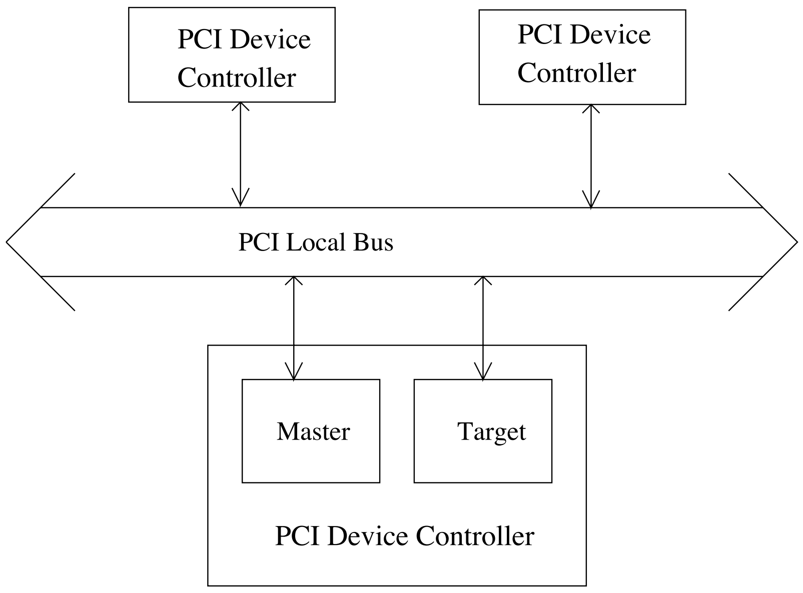


Figure 3: A PCI Local Bus Configuration

As we can see in the figure, it can have several modules, in all of which, there can be a Master and a Target machine. For simplicity sake in our implentations we will assume that there is only one Master and one Target module. The interactions between them will be in accordance to the bus protocol and for completeness we will have a rudimentary arbitrater. Since we have only one Master, having an arbitrator makes not much sense, but it is done to make the scenario more close to reality.

2.3 The Master and Target State Machines

The Master and the Target machines have definite behavior defined by the PCI 2.1 specs. We will discuss them in brief.

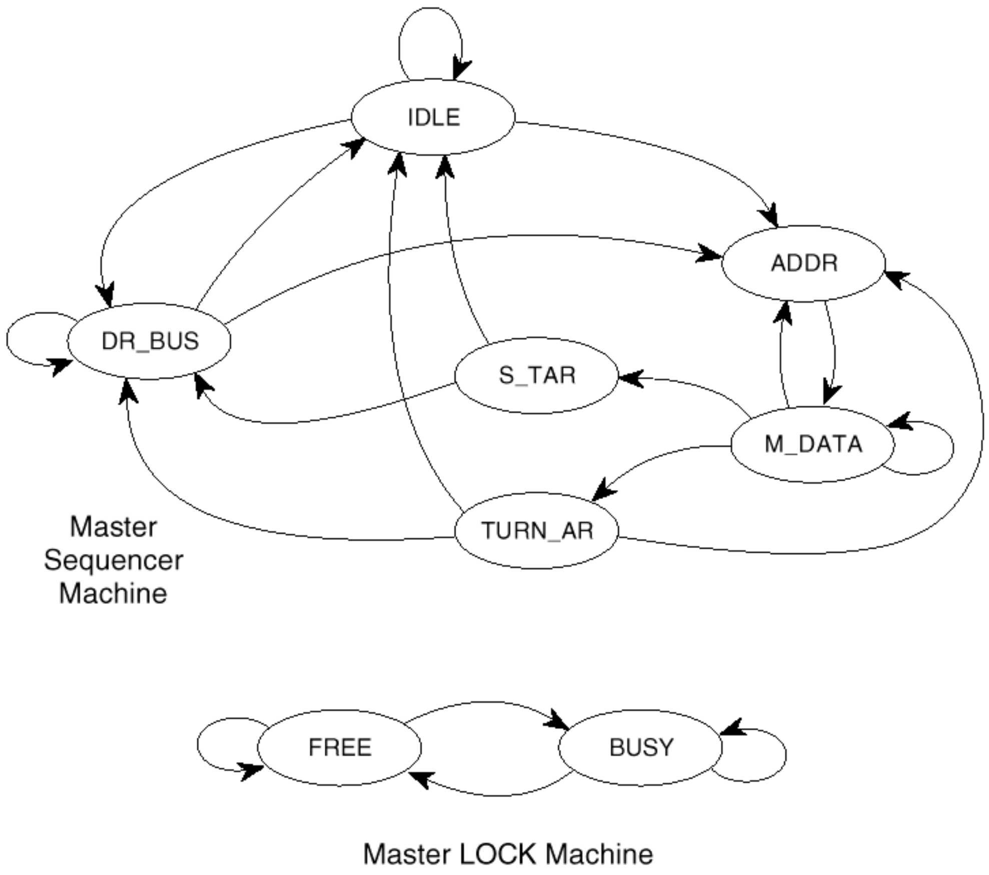


Figure 4: The Master State Machine

The Master State Machine (Figure 4) starts off from the IDLE state, where there are no transactions taking place on the bus. If it has to do a transaction on the bus, it arbitrates for the bus and waits for a grant, which is represented by the self loop to the state IDLE. If the machine gets a grant, it moves on to the ADDR state, where it has to start the transaction by driving the address lines. From then onwards a transaction starts. It does the data transaction in the state M_DATA. If it is a back to back transaction then it can move back and forth between the states ADDR and M_DATA since it has to float both data and address on the bus. From the M_DATA state, which stands for the state in which the machine can transmit, it can have a normal completion (go to state TURN_AR) or it can move to the state S_TAR which stands for the abnormal completion of the data transmission [2,3]. From the states S_TAR and TURN_AR the machine can either go to IDLE again or to DR_BUS. The state DR_BUS means that the bus is "parked", which in turn means that the Master has the grant but does not have any transaction to do. From the TURN_AR state the Master can go to the ADDR state whereby it can start a new transaction afresh.

The Master Lock machine stands for has two states FREE and LOCK. The state FREE means that it has not locked any chunk of address space in any Target machine, and it can keep looping in that state. Also, when it locks a particular chunk of memory in some Target machine it moves to the LOCK state. And when the transcation in that part of the memory is over, it can unlock the chunk and move on to the FREE state again. This state machine works in harmony with the Target lock state machine.

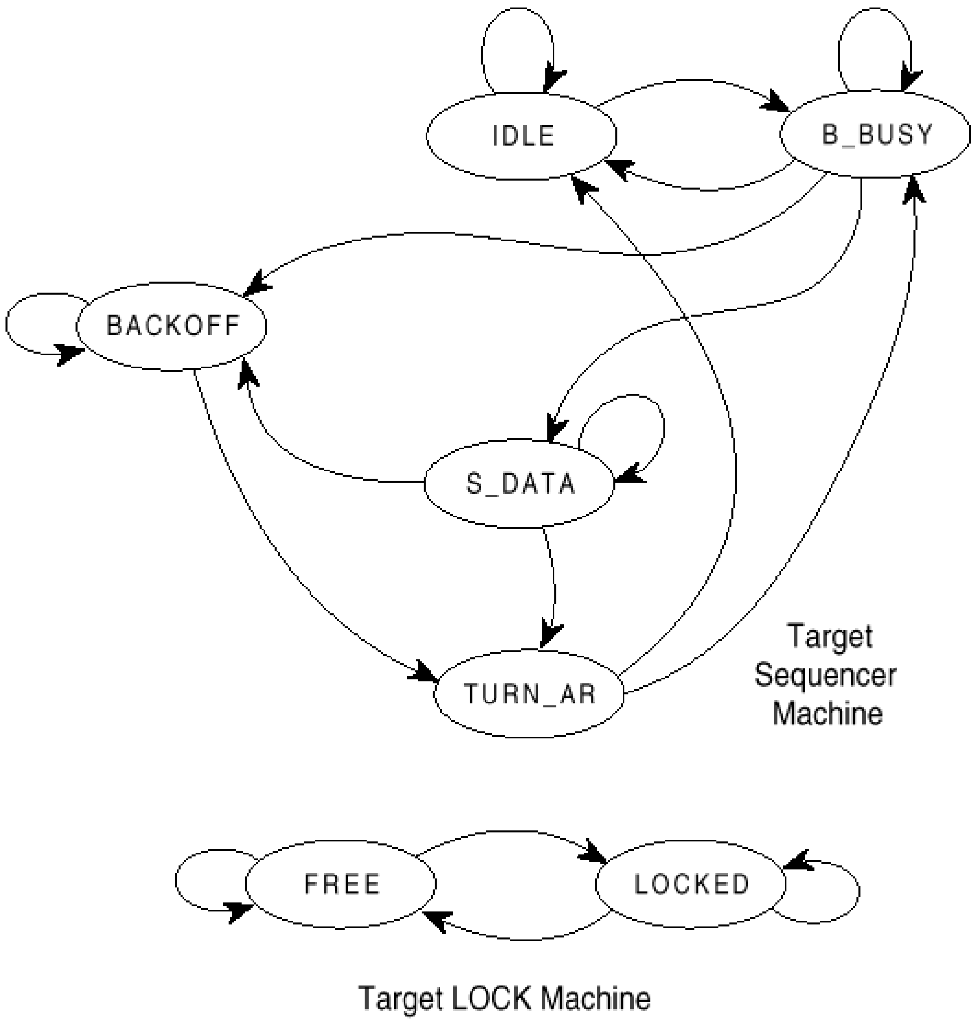


Figure 5: The Target State Machine

The Target State Machine (Figure 5) also starts off from the IDLE state, and keeps there until and unless a transaction is started on the bus by some Master. As soon as some transaction starts on the bus, it snoops in the address and decodes it. This is done after it moves to the state B_BUSY which means that the bus is busy, or in other words, has some transaction initiated on the bus. If the address floated on the bus is not meant for it, then it moves back to the IDLE state. Else, it identifies itself as the owner of the transaction and responds by asserting it's DEVSEL line. If the Target machine is not able to send or receive data (it may be busy with something else right now) it terminates the transaction (abnormally) without any data transfer, and moves to the state called BACKOFF. If it is able to do the transaction then it moves to the state called S_DATA and loops back to it. If the Target is able to finish the transaction normally (with all the data transfers complete) then it moves to the state called TURN_AR, else it moves to the state BACKOFF and tells the Master that it is an abnormal termination. This may be abnormal termination with or without data. The Target, however, has to come to the TURN_AR state even if it is an abnormal termination. From the TURN_AR state the Target can move back to the IDLE state, or, to B_BUSY, where it can try and decode another address floated on the bus.

The Target Lock Machine stands for the part of the lock synchronisation mechanism that is implemented in the Target. It can be in the FREE state which means that no chunk of memory in the Target machine is locked by any Master. It strobes the LOCK signal during the begin of every transaction. If it is asserted the machine moves on to the LOCKED state. It can remain there till the Master concerned unlocks it, and when it does, the machine moves back to the LOCK state.

3 Properties of Interest

The *PCI Local Bus Specification* asserts several functional characteristics of the bus which can immediately be identified with a set of proerties to be verified. We identified the following properties for verification using *VIS*

- Properties at the *transaction start* phase
- Properties at the *data completion* (Normal/Abnormal) phase
- Properties which should be valid *throughout* a transaction
- Properties related to the *locking* of the master and target device
- Properties related to *bus arbitration*
- Properties related to *deadlock* and *livelock* conditions

It is to be noted that not all of the properties mentioned above need to be valid at all of the different transactions. Infact, the transaction type determine the properties that need to be verified for that particular bus operation. We elaborate some of the above properties with respect to specific transactions as shown below.

3.1 Properties from the Read Transaction

Figure 6 shows the timing diagram of a typical *read transaction* on the *PCI* bus.

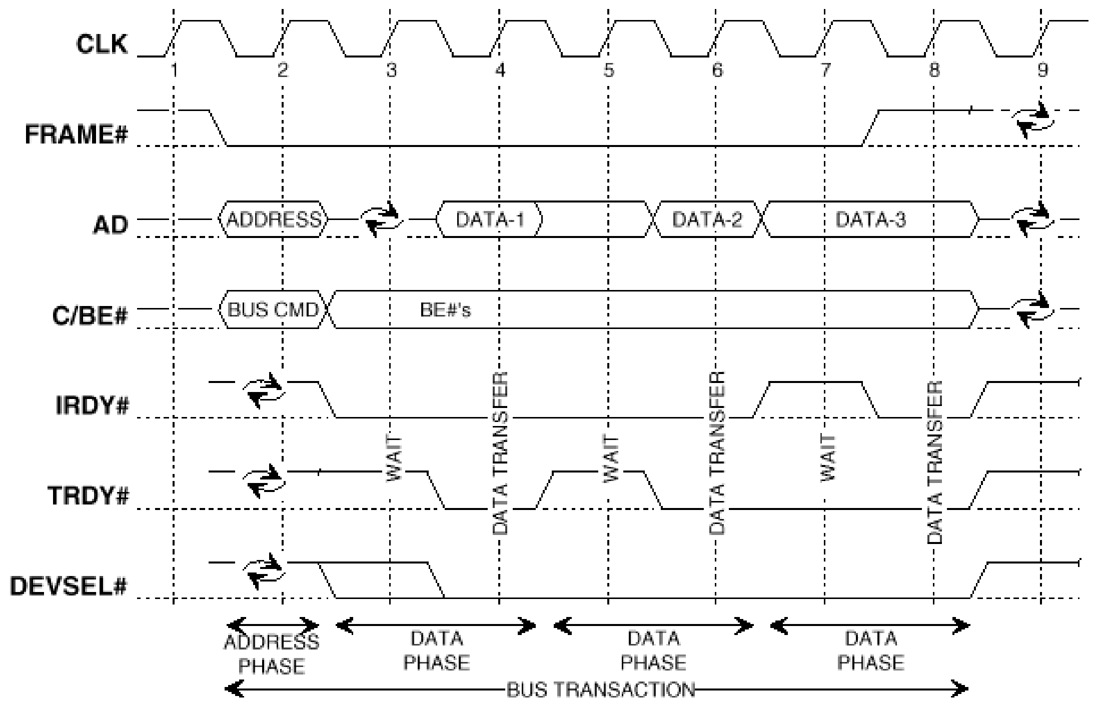


Figure 6: A Read Transaction

A *read transction* starts with an address phase which occurs when **FRAME#** is asserted for the first time and occurs on clock 2. During the address phase, **AD[31::00]** contain a valid address and C/BE[3::0]# contain a valid bus command. The first clock of the first data phase is clock 3. During the data phase, **C/BE#** indicate which byte lanes are involved in the current data phase. A data phase may consist of wait cycles and a data transfer. The **C/BE#** output buffers must remain enabled (for both read and writes) from the first clock of the data phase through the end of the transaction. The first data phase on a read transaction requires a turnaround-cycle (enforced by the target via **TRDY#**). In this case, the address is valid on clock 2 and then the master stops driving **AD**. The earliest the target can provide valid data is clock 4. The target must drive the **AD** lines following the turnaround cycle when **DEVSEL#** is asserted.

The constraints on the different signals, for example, when one can be driven after another has been asserted, can easily be identified with a set of properties (invariants). Some of these properties are given below.

- **FRAME#** assertion and **TRDY#** assertion should be seperated by at least one clock cycle
- **FRAME#** deassertion $\rightarrow$ **IRDY#** assertion
- The Byte enables (**C/BE#**s) need to be driven throughtout the transaction

3.2 Properties from the Write Transaction

Similar to the *read transaction* discussed above, a set of properties can be extracted from the timing diagram of the *write transaction* (Figure 7).

A *write transaction* starts when **FRAME#** is asserted for the first time which occurs on clock 2. A *write transaction* is similar to a *read transaction* except no turnaround cycle is required following the address phase because the master provides both address and data. Data phases work the same for both read and write transactions. The first and second data phases complete with zero wait cycles. However, the third data phase has three wait cycles inserted by the target. Notice both agents insert a wait cycle on clock 5. **IRDY#** must be asserted when **FRAME#** is deasserted indicating the last data phase. The data transfer was delayed by the master on clock 5 because **IRDY#** was deasserted. The last data phase is signaled by the master on clock 6, but it does not complete until clock 8.

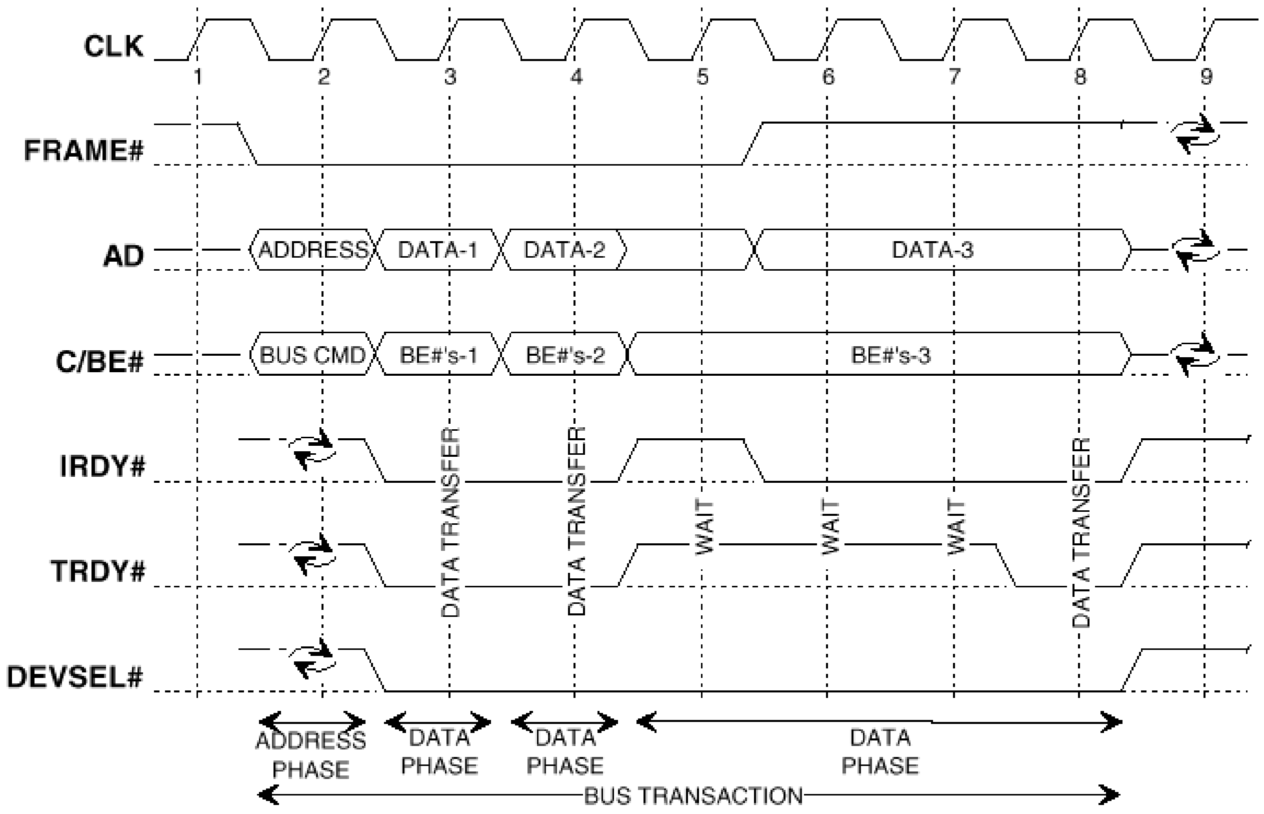


Figure 7: A Write Transaction

Some of the properties are enumerated below.

- **TRDY#** should not be asserted before the **DEVSEL#** is asserted
- **FRAME#** deassertion $\rightarrow$ **IRDY#** assertion

- Once **FRAME#** is asserted it cannot be deasserted again in the same transaction.

3.3 Approach

3.3.1 Scaling Down

The inherent problem with any model checking tool is the blowing up of the state space as mentioned earlier. To keep things in check several techniques were used.

- One Master - One Target module configuration

  This is the minimum configuration in which all the bus protocols can be verified, with the exception of arbitration. In the PCI bus, the controllers are resident on the Master and Target, there is no central controller. Thus, the design for the Master and Target controllers is of considerable complexity. We did not opt for more than one Master and Target as it would have resulted in a state space explosion.

- Subset of transactions chosen

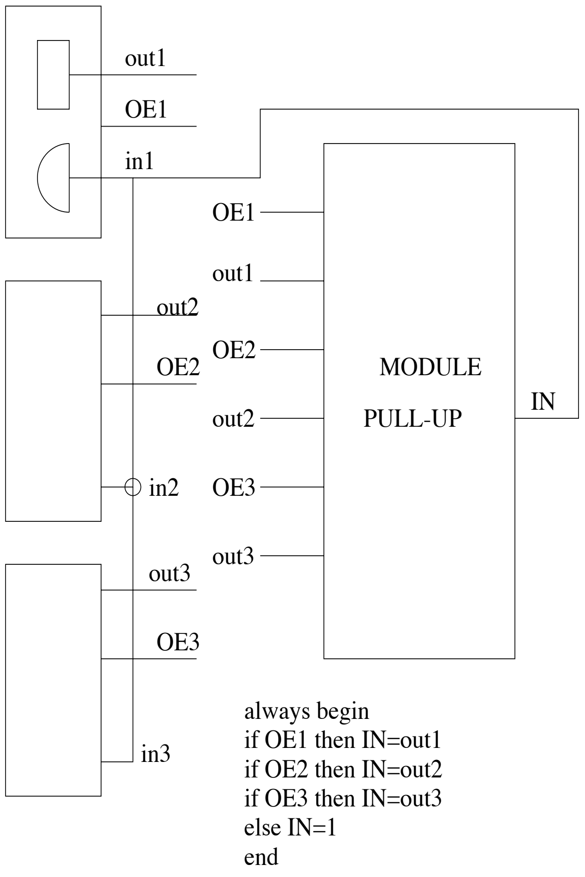
  The transactions: Interrupt, Configuration and Lock were dropped as the design for these transactions is very complex. Lock has been implemented only for the Target. The transactions: Memory Read, Memory Write, IO Read and IO Write have been implemented and the Bus Protocols verified for these transactions.

- Data Bus cut down to 3 bits

  The Data Bus width for a PCI Bus is 32/64. This width cannot be handled well by VIS. We cut down the width to 3 bits. 3 bits are sufficient to verify burst accesses, fast back to back transaction and abnormal terminations.

3.3.2 Modeling *inout* and *tri-stated* lines

Since VIS doesnot support inout and tristated lines, these lines were modelled as shown in the Figure 8. Consider a inout line which has to be pulled up to one. This line has to represented by two separate lines - IN and OUT. IN should reflect the value which is driven from any of the modules (if any module is driving it). Else, it should take the value one (when none of the modules is driving it). This is taken care of by the "Module Pull-Up".



```
always begin
if OE1 then IN=out1
if OE2 then IN=out2
if OE3 then IN=out3
else IN=1
end
```

Figure 8: Modelling *in-out* and *tristated* lines

3.4 Experimental Results

- Number of CTL Formulas: 17
- Number of Design Debug Iterations: 3
- Number of formulas passed on master alone: 5 (out of 5)
- Number of formulas passed on target alone: 2 (out of 2)
- Number of formulas passed on integrated system: (17 out of 17)
- Number of Bugs found: 4
  - Abnormal master abort was happening due to master not waiting sufficienltly long for device to respond.
  - There were some glue problems with the tristate modelling.
  - Device select was not being asserted due to register width mismatch leading to wrong initialization.

– Retry was being preceded by Disconnect.

4 Conclusion and Experience

The protocol for the PCI Bus is rich in properties. So it lends itself to Formal Verification of the protocol. But coming up with the CTL properties is not *obvious.* In fact most often than not, it was found that the formulas failed because the correct properties were not written down. It goes hand in hand with the usual Formal Verification problem of *not knowing how to ask the correct question.*

Learning VIS should not be taken as one of the big hurdles of the project, but cutting down the Verilog code to a VIS-able subset was. Added to it was problems faced with the translator *vl2mv.*

Also, during property verification, we realised with time, that sometimes a property failing can be a boon in disguise, not only it does not gives a false sense of security when a property passes *wrongly* but also it really uncover bugs in the design.

Modeling of the tri-states was done with precision and can be regarded as one of the most useful things learnt and used in the project.

5 Appendix A: The reachability of our model

The reachability of our model has been found out and the corresponding VIS session has been included in this Appendix.

```
vis> sift
Dynamic variable ordering forced with method sift....
vis> wo ord1
vis> flt
Deleting current network and creating new one.
Warning: No checking for complete and deterministic specification.
Warning: Do "help flatten_hierarchy" for detailed information.
vis> ro ord1
vis> part
vis> pii
Printing Information about Image method: IWLS95
        Threshold Value of Bdd Size For Creating Clusters = 1000
                (Use "set image_cluster_size value " to set this to desired value)
        Verbosity = 0
                (Use "set image_verbosity value " to set this to desired value)
        W1 =  6 W2 = 1 W3 = 1 W4 = 2
                (Use "set image_W? value " to set these to desired values)
Shared size of transition relation for forward image computation is
28862 BDD nodes (43   components)
```

```
vis> rch -s1 -v1
BFS step = 0    |states| =   262144     Bdd size =      278
BFS step = 1    |states| =   524288     Bdd size =      788
BFS step = 2    |states| =   983040     Bdd size =     1064
BFS step = 3    |states| = 1.24518e+06  Bdd size =     1963
BFS step = 4    |states| = 1.50733e+06  Bdd size =     4769
BFS step = 5    |states| = 1.76947e+06  Bdd size =    10744
BFS step = 6    |states| = 1.83501e+06  Bdd size =    16531
BFS step = 7    |states| = 1.90182e+06  Bdd size =    21308
BFS step = 8    |states| = 1.9776e+06   Bdd size =    26940
BFS step = 9    |states| = 2.05901e+06  Bdd size =    31810
BFS step = 10   |states| = 2.1632e+06   Bdd size =    35855
BFS step = 11   |states| = 2.25357e+06  Bdd size =    38744
BFS step = 12   |states| = 2.30554e+06  Bdd size =    40408
BFS step = 13   |states| = 2.32346e+06  Bdd size =    41950
BFS step = 14   |states| = 2.3383e+06   Bdd size =    43198
BFS step = 15   |states| = 2.35059e+06  Bdd size =    44415
BFS step = 16   |states| = 2.36186e+06  Bdd size =    45550
BFS step = 17   |states| = 2.37312e+06  Bdd size =    46782
BFS step = 18   |states| = 2.38438e+06  Bdd size =    47747
BFS step = 19   |states| = 2.39514e+06  Bdd size =    48695
BFS step = 20   |states| = 2.40538e+06  Bdd size =    49506
BFS step = 21   |states| = 2.41459e+06  Bdd size =    50529
BFS step = 22   |states| = 2.43046e+06  Bdd size =    51402
BFS step = 23   |states| = 2.44787e+06  Bdd size =    52745
BFS step = 24   |states| = 2.46682e+06  Bdd size =    53910
BFS step = 25   |states| = 2.48525e+06  Bdd size =    55058
BFS step = 26   |states| = 2.50368e+06  Bdd size =    56256
BFS step = 27   |states| = 2.51955e+06  Bdd size =    57847
BFS step = 28   |states| = 2.53389e+06  Bdd size =    59672
BFS step = 29   |states| = 2.55232e+06  Bdd size =    61840
BFS step = 30   |states| = 2.57536e+06  Bdd size =    63732
BFS step = 31   |states| = 2.59891e+06  Bdd size =    65595
BFS step = 32   |states| = 2.6153e+06   Bdd size =    67233
BFS step = 33   |states| = 2.62451e+06  Bdd size =    67923
BFS step = 34   |states| = 2.62912e+06  Bdd size =    68067
BFS step = 35   |states| = 2.63117e+06  Bdd size =    67860
Computing reachable states using the iwls95 image computation method.
Printing Information about Image method: IWLS95
        Threshold Value of Bdd Size For Creating Clusters = 1000
                (Use "set image_cluster_size value " to set this to desired value)
        Verbosity = 0
                (Use "set image_verbosity value " to set this to desired value)
        W1 =  6 W2 = 1 W3 = 1 W4 = 2
                (Use "set image_W? value " to set these to desired values)
Shared size of transition relation for forward image computation is
28862 BDD nodes (43   components)
********************************
Reachability analysis results:
FSM depth =                 36
reachable states =  2.63117e+06
BDD size =              67860
analysis time =             96
vis>


SECOND RUN
----------
vis> sift
Dynamic variable ordering forced with method sift....
vis> wo ord2
vis> flt
Deleting current network and creating new one.
Warning: No checking for complete and deterministic specification.
Warning: Do "help flatten_hierarchy" for detailed information.
vis> ro ord2
vis> part
vis> pii
Printing Information about Image method: IWLS95
        Threshold Value of Bdd Size For Creating Clusters = 1000
                (Use "set image_cluster_size value " to set this to desired value)
        Verbosity = 0
                (Use "set image_verbosity value " to set this to desired value)
        W1 =  6 W2 = 1 W3 = 1 W4 = 2
                (Use "set image_W? value " to set these to desired values)
Shared size of transition relation for forward image computation is
26791 BDD nodes (41   components)
vis> rch -s 1 -v 1
BFS step = 0    |states| =   262144     Bdd size =      278
BFS step = 1    |states| =   524288     Bdd size =      681
BFS step = 2    |states| =   983040     Bdd size =      843
BFS step = 3    |states| = 1.24518e+06  Bdd size =     1351
BFS step = 4    |states| = 1.50733e+06  Bdd size =     2221
BFS step = 5    |states| = 1.76947e+06  Bdd size =     3497
BFS step = 6    |states| = 1.83501e+06  Bdd size =     4763
BFS step = 7    |states| = 1.90182e+06  Bdd size =     5691
BFS step = 8    |states| = 1.9776e+06   Bdd size =     7450
BFS step = 9    |states| = 2.05901e+06  Bdd size =     9021
BFS step = 10   |states| = 2.1632e+06   Bdd size =    10273
BFS step = 11   |states| = 2.25357e+06  Bdd size =    10931
BFS step = 12   |states| = 2.30554e+06  Bdd size =    11253
BFS step = 13   |states| = 2.32346e+06  Bdd size =    11620
BFS step = 14   |states| = 2.3383e+06   Bdd size =    11839
BFS step = 15   |states| = 2.35059e+06  Bdd size =    12079
BFS step = 16   |states| = 2.36186e+06  Bdd size =    12215
BFS step = 17   |states| = 2.37312e+06  Bdd size =    12307
BFS step = 18   |states| = 2.38438e+06  Bdd size =    12485
BFS step = 19   |states| = 2.39514e+06  Bdd size =    12681
BFS step = 20   |states| = 2.40538e+06  Bdd size =    12803
BFS step = 21   |states| = 2.41459e+06  Bdd size =    13017
BFS step = 22   |states| = 2.43046e+06  Bdd size =    13368
BFS step = 23   |states| = 2.44787e+06  Bdd size =    13852
BFS step = 24   |states| = 2.46682e+06  Bdd size =    14334
BFS step = 25   |states| = 2.48525e+06  Bdd size =    14699
BFS step = 26   |states| = 2.50368e+06  Bdd size =    14952
BFS step = 27   |states| = 2.51955e+06  Bdd size =    15385
BFS step = 28   |states| = 2.53389e+06  Bdd size =    15738
BFS step = 29   |states| = 2.55232e+06  Bdd size =    16082
BFS step = 30   |states| = 2.57536e+06  Bdd size =    16408
BFS step = 31   |states| = 2.59891e+06  Bdd size =    16555
BFS step = 32   |states| = 2.6153e+06   Bdd size =    16759
BFS step = 33   |states| = 2.62451e+06  Bdd size =    16893
BFS step = 34   |states| = 2.62912e+06  Bdd size =    16896
BFS step = 35   |states| = 2.63117e+06  Bdd size =    16869
Computing reachable states using the iwls95 image computation method.
Printing Information about Image method: IWLS95
        Threshold Value of Bdd Size For Creating Clusters = 1000
                (Use "set image_cluster_size value " to set this to desired value)
        Verbosity = 0
                (Use "set image_verbosity value " to set this to desired value)
        W1 =  6 W2 = 1 W3 = 1 W4 = 2
                (Use "set image_W? value " to set these to desired values)
Shared size of transition relation for forward image computation is
26791 BDD nodes (41   components)
********************************
Reachability analysis results:
FSM depth =                36
reachable states =  2.63117e+06
BDD size =               16869
```

```
analysis time =             75
vis>

THIRD RUN
---------
vis> sift
Dynamic variable ordering forced with method sift....
vis> wo ord3
vis> flt
Deleting current network and creating new one.
Warning: No checking for complete and deterministic specification.
Warning: Do "help flatten_hierarchy" for detailed information.
vis> ro ord3
vis> part
vis> pii
Printing Information about Image method: IWLS95
        Threshold Value of Bdd Size For Creating Clusters = 1000
                (Use "set image_cluster_size value " to set this to desired value)
        Verbosity = 0
                (Use "set image_verbosity value " to set this to desired value)
        W1 =  6 W2 = 1 W3 = 1 W4 = 2
                (Use "set image_W? value " to set these to desired values)
Shared size of transition relation for forward image computation is
22274 BDD nodes (35   components)
vis> rch -s 1 -v 1
BFS step = 0    |states| =   262144     Bdd size =      278
BFS step = 1    |states| =   524288     Bdd size =      545
BFS step = 2    |states| =   983040     Bdd size =      644
BFS step = 3    |states| = 1.24518e+06  Bdd size =     1097
BFS step = 4    |states| = 1.50733e+06  Bdd size =     1877
BFS step = 5    |states| = 1.76947e+06  Bdd size =     3060
BFS step = 6    |states| = 1.83501e+06  Bdd size =     4134
BFS step = 7    |states| = 1.90182e+06  Bdd size =     4991
BFS step = 8    |states| = 1.9776e+06   Bdd size =     6486
BFS step = 9    |states| = 2.05901e+06  Bdd size =     7909
BFS step = 10   |states| = 2.1632e+06   Bdd size =     9065
BFS step = 11   |states| = 2.25357e+06  Bdd size =     9646
BFS step = 12   |states| = 2.30554e+06  Bdd size =     9895
BFS step = 13   |states| = 2.32346e+06  Bdd size =    10212
BFS step = 14   |states| = 2.3383e+06   Bdd size =    10430
BFS step = 15   |states| = 2.35059e+06  Bdd size =    10654
BFS step = 16   |states| = 2.36186e+06  Bdd size =    10805
BFS step = 17   |states| = 2.37312e+06  Bdd size =    10856
BFS step = 18   |states| = 2.38438e+06  Bdd size =    10914
BFS step = 19   |states| = 2.39514e+06  Bdd size =    11072
BFS step = 20   |states| = 2.40538e+06  Bdd size =    11195
BFS step = 21   |states| = 2.41459e+06  Bdd size =    11358
BFS step = 22   |states| = 2.43046e+06  Bdd size =    11623
BFS step = 23   |states| = 2.44787e+06  Bdd size =    12009
BFS step = 24   |states| = 2.46682e+06  Bdd size =    12387
BFS step = 25   |states| = 2.48525e+06  Bdd size =    12613
BFS step = 26   |states| = 2.50368e+06  Bdd size =    12804
BFS step = 27   |states| = 2.51955e+06  Bdd size =    13120
BFS step = 28   |states| = 2.53389e+06  Bdd size =    13510
BFS step = 29   |states| = 2.55232e+06  Bdd size =    13777
BFS step = 30   |states| = 2.57536e+06  Bdd size =    13949
BFS step = 31   |states| = 2.59891e+06  Bdd size =    14024
BFS step = 32   |states| = 2.6153e+06   Bdd size =    14195
BFS step = 33   |states| = 2.62451e+06  Bdd size =    14236
BFS step = 34   |states| = 2.62912e+06  Bdd size =    14224
BFS step = 35   |states| = 2.63117e+06  Bdd size =    14178
Computing reachable states using the iwls95 image computation method.
Printing Information about Image method: IWLS95
        Threshold Value of Bdd Size For Creating Clusters = 1000
                (Use "set image_cluster_size value " to set this to desired value)
        Verbosity = 0
                (Use "set image_verbosity value " to set this to desired value)
        W1 =  6 W2 = 1 W3 = 1 W4 = 2
                (Use "set image_W? value " to set these to desired values)
Shared size of transition relation for forward image computation is
22274 BDD nodes (35   components)
********************************
Reachability analysis results:
FSM depth =                 36
reachable states =  2.63117e+06
BDD size =               14178
analysis time =             60
vis>
```

6 Acknowledgement

We would like to extend our sincere thanks to Mr. Prakash Chauhan, (Dell Computers, Austin, TX) for providing extensive cooperation. Without his help it would have been a much tougher job. We would like to thank Dr. Aziz for his helpful comments and advice.

References

[1] Steven E. Schulz, "Ready for Prime Time", *Integrated Systems Design Magazine.* April 1995.

[2] *PCI Local Bus Specification* Production Version, Revision 2.1, June 1995.

[3] *PCI-to-PCI Bridge Specification*, Revision 1.0.

[4] S. Campos, E. Clarke, W. Marrero and M. Minea, "Verifying the Performance of the PCI Local Bus using Symbolic Techniques", ICCAD, 1995.