15

A formal proof of absence of deadlock for any acyclic network of PCI buses

Francisco Corella
Hewlett Packard Co.
8000 Foothills Blvd., Roseville, CA 95747-5649, USA
phone: (916)785-3504
fax: (916)785-3096
email: fcorella@rosemail.rose.hp.com

Robert Shaw
Department of Computer Science, University of California
Davis, CA 95616, USA
tel: (916)752-7004
fax: (916)752-4767
email: shaw@cs.ucdavis.edu

Cui Zhang
Department of Computer Science, California State University
Sacramento, CA 95819-6021, USA
Tel: (916)278-7352
Fax: (916)278-5949
Email: zhangc@ecs.csus.edu

Abstract

The Delayed Transaction mechanism introduced in version 2.1 of the PCI protocol includes rules for deadlock avoidance which are known to be incomplete in the design community. We have formalized a more complete set of rules as fairness constraints on the behavior of PCI devices and bridges. We present a mathematical proof that these fairness constraints are sufficient to guarantee absence of deadlock for an arbitrary acyclic network of PCI buses, using a novel notion of deadlock-freedom which is generally applicable to any transaction processing system. This verification problem falls outside of the scope of decision procedures based on model checking.

Keywords

PCI protocol, deadlock freedom, formal verification

1 INTRODUCTION

The last ten years have seen the development of a wide array of formal techniques for hardware verification in research laboratories. Recently, several of these techniques have been successfully applied to practical, industrial-size hardware verification problems [1, 5, 10]. However, the use of formal techniques in industry is still more the exception than the rule. As a consequence, not enough experience has been acquired to know what techniques or combination of techniques are best suited to tackle the many different kinds of practical verification problems.

We report here on an on-going industrial project which has already produced significant results and which offers new perspectives on the applicability of certain formal techniques. Specifically, we present a proof of absence of deadlock for an arbitrary tree of PCI buses. This is an early result of a broader formal verification project concerning a computer system that uses PCI as an I/O bus.

From a practical point of view, the reported work is significant because of the importance of the PCI bus and the acute need for a formal analysis of the newest version of the protocol, Revision 2.1. PCI is the dominant I/O bus in the PC market, and it is spreading in the Unix workstation market as well. The newest version of the PCI protocol, Revision 2.1 [7] became the official production version on June 1, 1995. Revision 2.1 introduced a new transaction mechanism, *delayed transactions*, that allows much higher performance. Unfortunately, the 2.1 specification has problems concerning transaction ordering and forward progress.

The forward progress problems, which include deadlock scenarios, are known in the hardware design community, but their extent is not known and cannot be ascertained except by formal analysis. This uncertainty makes designers uneasy and has been a motivation for our work. The proof that we present here shows that certain modifications of the protocol are sufficient to guarantee absence of deadlock. We hope that this will eliminate the uncertainty and give more confidence to designers of systems that use PCI.

The ordering problem is partly documented in the PCI specification, but its consequences have been underestimated. We discovered the full extent of the ordering problem as a side effect of our work on forward progress. We also found a solution that has been submitted as an Engineering Change Request to the PCI Special Interest Group.

From a technical point of view, the work is significant for three reasons. First, we establish liveness by mathematical proof. Most published work on verification of liveness has used model checking tools. However, model checking applies to a specific transition relation, realized by a hardware implementation having a specific topology. For example, although a hierarchical bus protocol was verified in [2], this was done only for a few specific topologies with up

to three buses and eight processors. By contrast, we have verified the PCI protocol for *any* tree of PCI buses.

Model checking procedures have been devised for systems having an arbitrary number of replicated components. The problem of verifying such systems is known as the *parameterized model checking problem* (PMCP) [3]. Practical procedures for special cases of PMCP have been used successfully to verify cache coherence protocols [6, 8]. However, those procedures have only been applied to the verification of safety properties. More fundamentally, the problem of verifying liveness of the PCI protocol, where absence of deadlock hinges on the acyclicity of the network of PCI buses, seems to fall outside of the scope of the PMCP.

A second point of technical interest is the type of liveness specification that we use. The PCI protocol does not guarantee absence of starvation. Thus we can only verify absence of deadlock. But, surprisingly, there is no universally accepted formal concept of deadlock in the literature. Different authors have used different notions of deadlock, and those notions are usually specific to the problem at hand. In Section 3 we propose a general notion of deadlock which applies to any *transaction processing system.* This is the notion that we have used in our proof of absence of deadlock. We have also carried out a proof of full liveness (absence of deadlock and starvation) for a modification of the protocol that makes it possible to prevent starvation, but this will be reported elsewhere.

A third point of technical interest is that the reported work is a mathematical proof, done by hand, rather than a mechanical proof carried out with a theorem prover. It may seem paradoxical to claim this as a point of interest, since mechanical proofs are clearly superior to manual proofs. But we believe that manual proofs are an important tool in the arsenal of formal methods that has been unduly neglected.

Because mechanical proofs are still very time consuming, manual proofs make it possible to find bugs or establish correctness more quickly, thus providing guidance to the designers very early in the design process. For example, in the broader project mentioned above, we have found several deadlock scenarios at the block diagram level, before the detailed design was developed. It was then very easy to modify the design. If the same deadlocks had been found later in the development process, the required modifications would have been much more costly.

Moreover, manual proof development yields the proof as a deliverable. The proof tells why a design is correct, a much more useful result than a simple "yes" answer to the correctness question. For example, the proof of absence of deadlock that we present here provides a better understanding of the issue of forward progress in PCI, which will be useful when considering future modifications or extensions of the protocol. In contrast, verification by today's theorem provers rarely results in a proof that is readily understandable by humans. It is well known, for example, that proof scripts for the HOL theo-

rem prover [4] are difficult to understand or modify. And model checkers, of course, being based on decision procedures, can only provide a "yes" answer by their very nature.

Clearly, there are cases where a manual proof would be unfeasible due to the required amount of bookkeeping. We recognize this, and in fact the manual proof presented here is a proof at a high level of abstraction that will be transcribed into HOL and integrated in a mechanical proof carried out at a more detailed level of abstraction. Our argument is that, today, manual verification, if at all possible, should precede mechanical verification as a way of obtaining an early and readable proof; and that a manual proof may be very useful by itself in cases where a mechanical proof is too costly. On the other hand we do believe that, in the long run, advances in mechanical theorem proving will render manual proofs obsolete.

2 THE PCI PROTOCOL

2.1 Overview

(a) Transaction propagation

In Revision 2.1 of the PCI Specification there are two kinds of transactions: *posted* transactions, and *delayed* transactions. Posted transactions are write transactions, while delayed transactions can be read or write transactions. A transaction is issued by an agent, the *master* of the transaction, and specifies an address which uniquely determines another agent, the *target* of the transaction. Master and target may be on the same PCI bus, or on different buses belonging to an acyclic network of buses and bridges.

A posted transaction propagates from the *originating bus* of the transaction, i.e. the master's bus, to the *destination bus*, i.e. the target's bus. A delayed transaction propagates from the originating bus to the destination bus, and then the *completion* of the transaction travels back from the destination bus to the originating bus. The completion carries the data, in the case of a delayed read transaction, or the termination status (normal or abnormal) in the case of a delayed write transaction. The address of the transaction uniquely determines the target of the transaction and hence the path that the transaction must follow.

As a transaction propagates it causes one or more *subtransactions* to be issued on the one or more buses that separate the master from the target of the overall transaction. (We refer to the subtransactions as *local transactions*, or *bus transactions*, and to the overall transaction as a *global transaction*.) Each of those subtransactions has a local master, which may be the master of the global transaction or a bridge acting on its behalf, and a local target, which may be the target of the global transaction or a bridge acting on its behalf.

A posted subtransaction may be either *retried* or *completed* by its local target. The term *retried* means that the local target tells the local master to retry the transaction later. When the subtransaction is retried, the local master must reissue it forever until it is completed. When the substransaction is completed, the local target latches the transaction information, creating a *P entry* (P stands for Posted)*. If the local target is a bridge, the P entry travels through the bridge from the bus of the subtransaction (the *local bus*) to the bus on the other side of the bridge (the *remote bus*).

A delayed subtransaction may also be retried or completed by its local target. If it is retried, its local master must reissue it forever until it is completed. The local target may retry the transaction in two different ways: it may ignore it altogether, or it may latch it, creating an *R entry* (R stands for Request).* The R entry specifies the address of the transation, and in the case of a delayed write, the data. The local target then tries to obtain a completion for the transaction. When the completion is obtained, a *C entry* is created.* The C entry specifies the address and the data in the case of a read, or the address and the termination status (normal or abnormal) in the case of a write. Once the C entry is ready, the subtransaction may be completed when it is issued again by the local master.

If the local target is a bridge, the R entry travels, through one of two bridge channels, from the local bus to the remote bus. Later, the R entry causes another subtransaction to be issued on the remote bus. When that subtransaction is completed, the bridge creates a C entry which travels, through the opposite bridge channel, from the remote bus to the local bus.

(b) Ordering rules

P, R and C entries travel together through the bridge channels. However, the bridge channels cannot behave as FIFO queues, because that would cause a variety of deadlock scenarios. Hence some entries must be allowed to *pass* other entries. On the other hand, some minimal ordering must be maintained to ensure predictable behavior in the absence of acknowledgements of posted write transactions.

There are thus two kinds of *passing rules* in PCI:

1. Those that *prevent passing* to provide ordering, viz.:

   (a) A P entry cannot pass a P entry.
   (b) A C entry cannot pass a P entry.
   (c) An R entry cannot pass a P entry.

*In the PCI specification, what we call a P entry is called a PMW entry.
*In the PCI specification, an R entry is called a DRR entry in the case of a delayed read transaction, or a DWR entry in the case of a delayed write transaction.
*C stands for completion. In the PCI specification, a C entry is called a DRC entry in the case of a delayed read transaction, or a DWC entry in the case of a delayed write transaction.

2. Those that *prevent* bridge channels from *preventing passing* in order to avoid deadlocks, viz.:

   (a) A P entry must be allowed to pass an R entry.
   (b) A P entry must be allowed to pass a C entry.

   A third rule is not included in the PCI specification, but is generally known to be necessary:

   (c) A C entry must be allowed to pass an R entry.

To illustrate the second kind of rule, consider the deadlock scenario shown in Figure 1, which arises in the absence of rule 2(c). The agents A_1 and A_2 are connected to the buses B_1 and B_2 respectively. These two buses are connected to a third bus B_3 by two bridges G_1 and G_2. The channels of G_1 are N_1 and N_1', those of G_2 are N_2 and N_2'.

A_1 issues a delayed transaction that targets A_2. The (global) transaction is first issued (as a local transaction) on bus B_1, and it is latched and retried by G_1. This results in an R entry being placed in channel N_1. Symmetrically and simultaneously, A_2 issues a delayed transaction that targets A_1 and is latched and retried by G_2, causing an R entry to be placed in N_2'. The resulting state of the system is shown if Figure 1(a).

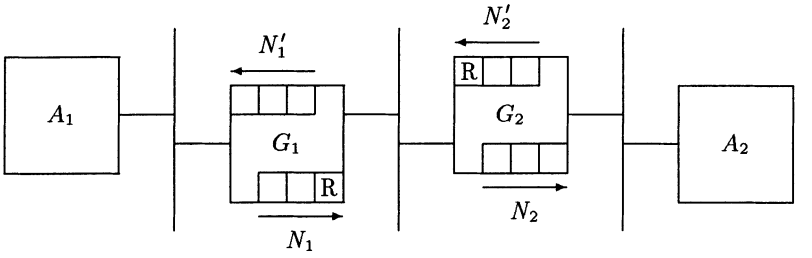
Now the R entry in N_1 triggers a local transaction on B_3, which is latched and retried by G_2, resulting on a R entry being placed in channel N_2. After that, symmetrically, the R entry in N_2' triggers a local transaction on B_3 which is latched and retried by G_1, resulting in an R entry being placed in N_1'.

Then the R entry in G_2 triggers a local transaction on B_2, which is completed by A_2, resulting on a C entry being added to channel N_2' and the R entry being removed from channel N_2. Symmetrically and simultaneously, the R entry in N_1' triggers a transaction on bus B_1 resulting in a C entry being added to N_1 and the R entry being removed from channel N_1'. The resulting state of the system is shown if Figure 1(c).

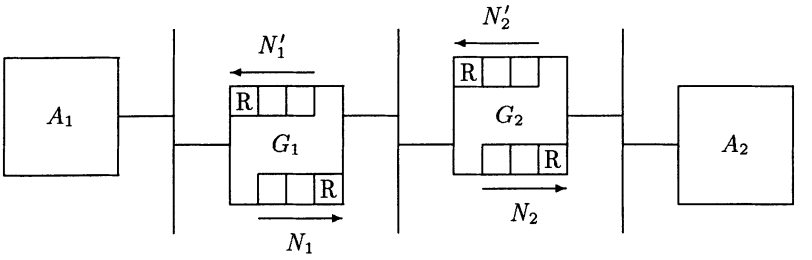
The system is now deadlocked. The R entry in N_1 repeatedly triggers transactions on B_3 that could be completed using the matching C entry in N_2', but this completion is not allowed because the C entry cannot pass the preceding R entry in N_2'. Symmetrically, the R entry in N_2' repeatedly triggers transactions on B_3 that match the C entry in N_1, but this C entry cannot pass the preceding R entry in N_1.

(c) Discarding R and C entries

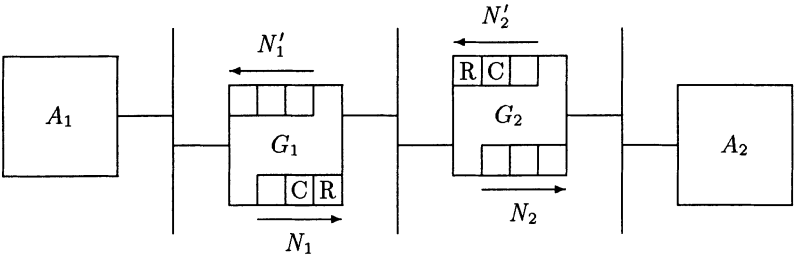
An interesting feature of the PCI protocol, from the point of view of forward progress, is that R and C entries may be discarded. The PCI specification does not completely specify the circumstances under which discarding is allowed. In our formal treatment, restrictions on discarding to guarantee absence of



(a)



(b)



(c)

Figure 1 A deadlock scenario.

deadlock are precisely specified by the transition relation of Section c and the fairness constraints of Section 4.1.

(d) Completion stealing

After the local target of a delayed subtransaction has prepared the C entry, it waits for the local master to issue the subtransaction again. However, a different master may issue a delayed subtransaction with same type (read or write) and same address before the original master does. Because there is no master ID in the PCI protocol, the comparison of a C entry to an incoming request only takes into account the transaction type and the address. Thus the local target is not able to distinguish between a retry by the original master and a new request by a different master, and it may grant the completion to the wrong master. The next retry of the original master will then propagate as if it were a new transaction. This erroneous behavior is partly documented in Section 3.11, item 6 of the PCI specification, Revision 2.1, but the impact of the error is underestimated.

We intend to propose a backwards-compatible solution to the problem, that uses four reserved pins to implement a local master ID. The bus arbiter communicates the local master ID to the local target during the address phase. The local target includes the ID in the R entry, then copies it to the C entry when the completion is obtained. The C entry is then matched only to an incoming request by the same master.

While the absence of a master ID affects the correctness of the PCI protocol, it has no bearing on the presence or absence of deadlock; in the proof of absence of deadlock given in Section 4 we use the current protocol, without master-ID lines. But the absence of a master ID makes it impossible for a local target to be fair to competing local masters, and thus makes it impossible to guarantee absence of starvation. With master-ID lines, sophisticated agents and bridges could try to avoid starvation. In fact, we have been able to formally specify a collection of fairness constraints that guarantee full liveness; this will be reported elsewhere.

2.2 Formal model of transaction propagation

We give now a formal description of the transaction propagation mechanism of the PCI protocol. Although formalization is kept to a minimum, the description could be readily translated into set theory or higher-order logic.

(a) Topology

A system of PCI buses can be viewed as a connected and acyclic hypergraph, with agents as external vertices (leaves), bridges as internal vertices, and buses as hyperedges, each internal vertex being connected to exactly two hyperedges. A *path* in the graph from vertex V_0 to vertex V_n, $n \geq 1$, is, as usual, an

alternating sequence

$$V_0, H_0, V_1, H_1, \ldots, V_{n-1}, H_{n-1}, V_n$$

where each hyperedge H_i, $i < n$, connects V_i and V_{i+1}. A path is *regular* if it "does not turn back", i.e. if $V_i \neq V_{i+1}$ for $i < n$ and $H_i < H_{i+1}$ for $i < n-1$. If there is a path from V_0 to V_n then there is also a regular path. The fact that the hypergraph is acyclic means that there exists no regular cycle

$$V_0, H_0, V_1, H_1, \ldots, V_{n-1}, H_{n-1}, V_n = V_0.$$

This implies that there is at most one regular path between any two vertices. And the fact that the hypergraph is connected means that there is at least one path between any two distinct vertices. Thus there is *exactly one regular path* between any two distinct vertices.

A bridge between two buses B and B' has two *opposite channels* corresponding to the two directions of traffic. One of the channels has B as its *in-bus* and B' as its *out-bus*, while the other has B' as its in-bus and B as its out-bus.

Similarly, every agent has two opposite channels, called the master channel and the target channel. The bus to which the agent is connected is the *out-bus* of the master channel and *in-bus* of the target channel.

The hypergraph defined above, which has bridges and agents as vertices, gives rise to a directed graph that has the channels of bridges and agents as nodes. There is an edge from a channel N_1 to a channel N_2, $N_1 \rightarrow N_2$, iff the out-bus B of N_1 is the in-bus of N_2 (in which case we say that B is *between* N_1 *and* N_2), and it is not the case that N_1 and N_2 are opposite channels of the same bridge. Note that $N_1 \rightarrow N_2$ implies $N_2' \rightarrow N_1'$, where N_1' and N_2' are the opposite channels of N_1 and N_2 respectively.

There is an obvious one-to-one correspondance between paths in the hypergraph and paths in the directed graph. The acyclicity of the hypergraph implies that the node graph is a *directed acyclic graph*. The fact that the hypergraph is connected in addition to being acyclic implies that, for every pair (V_1, V_2), where each V_i is an agent or a bridge, there exists exactly one pair (N_1, N_2), where each N_i is one of the two channels of V_i, such that there is a path from N_1 to N_2 in the directed path; if V_1 is an agent then N_1 can only be the master channel, and if V_2 is an agent, then N_2 can only be the target channel.

The configuration process maps every address to an agent, and sets up routing information in the bridges. Every transaction is issued by an agent A, the master of the transaction, and specifies an address α which is mapped to a distinct master A', the target of the transaction. In the hypergraph, there is a unique path between A and A'. In the directed graph, there is a unique path from the master channel of A to the target channel of A', and a unique return path from the master channel of A' to the target channel of A.

From now on, by *path* we shall refer to a path in the directed acyclic graph which has channels as nodes.

(b) Time and state

The PCI protocol assumes a global clock, so we use discrete time, isomorphic to the natural numbers. At any time t, each channel contains a collection of *entries* $(E_i)_{1 \le i \le n}$, $n \ge 0$. An entry is a tuple $(N, \epsilon, \tau, \alpha)$, where N is the channel containing the entry, ϵ is the entry type (P, R, or C), τ is the transaction type (posted write, delayed read, or delayed write), and α is the transaction address. We refer to the pair (τ, α) as the *parameters* of the entry. The collection $(E_i)_{1 \le i \le n}$ functions as a non-FIFO queue: entries are added at the end, but may be removed from any position. Note that this queue is an abstraction of the actual data structures used by a given implementation, which need not consist of a single queue. The relative position of an entry in the queue $(E_i)_{1 \le i \le n}$ gives the relative age of the entry, entries with lower index being older.

A bridge channel may contain P, R and C entries. The master channel of an agent may contain P and R entries. The target channel of an agent contains no entries.*

A subset of the R entries present in a bridge channel N at time t comprise the *retry set* of N at time t. (Formally, the retry set is a collection of *entry indices.*) The entries in the retry set are said to be *committed.* Other entries said to be committed are the oldest P entry in a bridge channel and all the entries in the master channel of an agent.*

(c) Events and transitions

At a high level of abstraction, the evolution of the system is described in terms of *events.* In a more detailed description, these abstract events can be defined in terms of more concrete hardware state transitions.

Events have enabling conditions that define the states in which they may occur. When they occur, events cause state transitions. The transition relation of the system can thus be defined in terms of events. In Section 4, fairness constraints will also be defined in terms of events.

In addition to defining the possible transitions of the system state, events also define the transitions of a more abstract view of the system, where each entry is deemed to belong to a given transaction. In this abstract view, the set of entries is partitioned into a collection of *transaction snapshots.* There are the following kinds of transaction snapshots:

1. A *P snapshot* $T = \{E\}$ consists of a single P entry E with address α in a bridge channel or master channel N from which there exists a path to

*In an implementation, a target channel may contain queued P and R entries, and a master channel may contain C entries of delayed transactions. However these entries need not be included in the analysis of forward progress.

*We shall see below that a committed entry is never discarded and "is retried forever until completed".

the target channel of the agent specified by α. If the length of the path (number of nodes) is p, we let $Left(T) = p - 1$.

2. A *D snapshot* can be of one of two kinds:

   (a) An *R snapshot* is a set T of R entries, all having same parameters (τ, α) and all contained in the master channel N_0 of an agent and in a possibly empty sequence of bridge channels $N_1, \ldots, N_n$, with at most one entry per N_i, where $N_0, N_1, \ldots, N_n$ is a proper prefix of the path from N_0 to the target channel N' of the agent specified by the address α. All the R entries in T are committed, except possibly the one in N_n if $n > 0$. Let the length of the path from N_0 to N' be p. If the R entry in N_n is not committed, let $Left(T) = p-n$. If it is committed, let $Left(T) = p-n-1$.

   (b) An *RC snapshot* T consists of: (i) a set of R entries, all having same parameters (τ, α) and all contained in the master channel of an agent N_0 and in a possibly empty sequence of bridge channels $N_1, \ldots, N_n$, with at most one entry per N_i; and (ii) a C entry in a channel N_{n+1}; where $N_0, N_1, \ldots, N_n, N_{n+1}$ is a proper prefix of the path from N_0 to A. All the R entries in an RC snapshot are committed.

   In both cases we refer to the R entry in N_n, $n \geq 0$ as the *foremost R entry* in the D snapshot.

Let now S be a state of the system. Assume that the set of entries in S can be partitioned into a collection Π of transaction snapshots. For each event Q we define a condition on S that enables Q, the state transition $S \rightarrow S'$ that Q causes on S, and the transaction-level transition $\Pi \rightarrow \Pi'$ that Q causes on Π. An event can be of one of the following kinds:

- A *P event* that *occurs on bus* B and is *triggered* by a P entry E in a bridge channel or master channel N with out-bus B. We refer to the parameters (τ, α) of the entry E (where τ indicates a posted write transaction) as the parameters of the event. E belongs to a P transaction snapshot $T = \{E\}$. Therefore, if N' is the target channel specified by the address α of E, there exists a path $N = N_0, N_1, \ldots, N_n = N'$ from N to N' in the channel graph. A P event is an abstraction of a posted bus transaction on bus B, which may be retried or completed by its local target. Consequently there are two kinds of P events:

  - A *P_retry event*, which has no effect.
  - A *P_completion event*, which deletes the entry E from N_0, and creates a P entry E' in N_1 with same parameters except in the case where $N_1 = N'$. If $N_1 = N'$ no entry is created in N_1. In that case the P_completion

event is also called a *P_end event*, because it marks the termination of the global P transaction.*
At the transaction level, $\Pi' = (\Pi \backslash \{E\}) \cup \{E'\}$, except if the P_completion event is a P_end event, in which case $\Pi' = (\Pi \setminus \{E\})$.

- A *D event* that occurs on bus B and is *triggered* by an R entry E in a channel N with out-bus B. We refer to the parameters (τ, α) of the entry E as the parameters of the event. If N is a bridge channel, such an event may happen only if there are no P or C entries older than E in N. If N is a bridge channel and E is not in the retry set of N, then E is added to the retry set of N and the event is said to be an *R_commit event*, which commits the entry E. E belongs to a D transaction snapshot. Therefore, if N' is the target channel specified by the address α, there exists a path $N = N_0, N_1, \ldots, N_n = N'$ from N to N' in the channel graph.
A D event is an abstraction of a delayed bus transaction on bus B, which may be retried or completed by its local target. Consequently there are two kinds of P events:

  - A *D_retry event*, which in turn can be:

    * A *D_noop event*, which has no effect, or
    * A *D_latch event*, which creates an R entry E' in N_1 with same parameters as E. A D_retry event may be a D_latch event only if $N_1 \neq N'$ and, in state S, N_1 contains no R or C entry with parameters (τ, α). The new entry E' is not placed in the retry set.

  - A *D_completion event*. A D event may be a D_completion event only if (i) $N_1 = N'$, or (ii) $N_1 \neq N'$, the channel N_1' opposite to N_1 contains a C entry E'' with parameters (τ, α), and there is no P entry in N_1' older than E''. Its effect on state S is to remove E from N_0, to remove E'' from N_1' in case (ii), and to create a C entry E''' with same parameters in the channel N_0' opposite to N_0 in the case where N_0 is a bridge channel. If N_0 is the master channel of an agent, no C entry is created, and the event is said to be a *D_end event*, which marks the completion of a D transaction.

  Note that in all cases where a D event has any effect (i.e. except in the case of a D_noop event), E is the foremost entry in a D transaction snapshot T. The transaction-level effects of the event can then be informally described as follows. If E' is created, it is added to T; if E''' is created, it is added to T; it E is deleted, it is removed from T; if E'' is deleted, it is removed from

*The fully formal specifications of the state transitions caused by P_completion events and other events are left to the reader.

the transaction snapshot where it belongs, which is usually T, but may be other than T as explained in Section d above.

- *An R_discard event*, that discards a non-committed R entry E from a bridge channel N. Such an event is not allowed to happen if E is the only entry in N and there are no P or C entries in the channel N' opposite to N.* T being the transaction snapshot that contains E, $\Pi' = (\Pi \setminus \{T\}) \cup \{T'\}$, with $T' = T \setminus \{E\}$.
- *A C_discard event* that discards a C entry E from a bridge channel N. Such an event may only happen if E is not the oldest C entry in E. T being the transaction snapshot that contains E, $\Pi' = (\Pi \setminus \{T\}) \cup \{T'\}$, with $T' = T \setminus \{E\}$.
- *A D_begin event*, whose effect is to create an R entry E in the master channel of an agent. $\Pi' = \Pi \cup \{E\}$.
- *A P_begin event*, whose effect is to create a P entry E in the master channel of an agent. $\Pi' = \Pi \cup \{E\}$.

It is clear that, in all cases, Π' is indeed a partition into transaction snapshots of the set of entries in S'. Thus, given a sequence of events $Q_0, \ldots, Q_n$, a sequence of states $S_0, \ldots, S_{n+1}$ such that each Q_i, $i \leq n$, is enabled in state S_i and takes S_i to S_{i+1}, and a partition Π_0 of the set of entries in S_0 into transaction snapshots, we can define by induction on i a sequence $\Pi_0, \ldots, \Pi_i, \ldots, \Pi_n, \Pi_{n+1}$, where each Π_i is a partition of the set of entries in S_i into transaction snapshots, and each Q_i takes Π_i to Π_{i+1}, for $1 \leq i \leq n$.

Now let S be a state, and Σ a set of events. We say that the events of Σ are *jointly enabled* in S iff they are individually enabled as described above and they not include any pair of *conflicting* events; two events conflict iff (i) they occur on the same bus, or (ii) one of them is a D event triggered by an R entry E and the other is an R_discard event that discards E, or (iii) one of them is a D_completion event involving a C entry E'' and the other is a C_discard event that discards E''. It is easy to verify that, if the events of Σ are jointly enabled in S, and Q is one of them, then the events in Σ other than Q are jointly enabled in the state that results from S when Q occurs. Thus, the events of Σ may occur in any order. Moreover, the state S' that results from the consecutive occurrence of all the events of Σ does not depend on the ordering of occurrence. We refer to S' as the *cumulative effect* of Σ on S. Furthermore, if Π is a partition of the entries of S into transaction snapshots, the result Π' of the events of Σ on Π is also independent of the order of the events. We refer to Π' as the cumulative effect of Σ on Π, and write $\Pi \xrightarrow{\Sigma} \Pi'$.

The evolution of the system from an initial state S_0 having no entries is given by a sequence $\Sigma_0, \Sigma_1, \ldots, \Sigma_n, \ldots$ of sets of events and a sequence $S_0, S_1, \ldots, S_n, \ldots$ of states such that the events in each Σ_n are jointly enabled on S_n and their cumulative effect on S_n is S_{n+1}. We can then define the

*The absence of P and C entries from N' is relevant for implementations where the two channels of a bridge share storage.

transaction-level evolution of the system to be the sequence $\Pi_0, \Pi_1, \ldots, \Pi_n, \ldots$ where $\Pi_0 = \emptyset$, each Π_n is a partition of the set of entries in S_n into transaction snapshots, and $\Pi_n \xrightarrow{\Sigma_n} \Pi_{n+1}$.

Let *P_num*(Π) and *D_num*(Π) be, respectively, the number of P and D transaction snapshots in Π. It is then easy to see that:

Lemma 1 *If* $\Pi \xrightarrow{\{Q\}} \Pi'$, *then*

$$\begin{aligned}
&P_num(\Pi') = P_num(\Pi) + 1 \text{ if } Q \text{ is a } P_begin \text{ event,}\\
&P_num(\Pi') = P_num(\Pi) - 1 \text{ if } Q \text{ is a } P_end \text{ event, and}\\
&P_num(\Pi') = P_num(\Pi) \text{ otherwise.}
\end{aligned}$$

and

$$\begin{aligned}
&D_num(\Pi') = D_num(\Pi) + 1 \text{ if } Q \text{ is a } D_begin \text{ event,}\\
&D_num(\Pi') = D_num(\Pi) - 1 \text{ if } Q \text{ is a } D_end \text{ event, and}\\
&D_num(\Pi') = D_num(\Pi) \text{ otherwise.}
\end{aligned}$$

Let $P\_left(\Pi) = \sum_T(Left(T))$ where T ranges over the P transaction snapshots in Π, and $R\_left(\Pi) = \sum_T(Left(T))$ where T ranges over the R transaction snapshots in Π. Then:

Lemma 2 *If* $\Pi \xrightarrow{\{Q\}} \Pi'$, *then:*

$$\begin{aligned}
&\text{if } Q \text{ is a } P_completion \text{ event, } P_left(\Pi') = P_left(\Pi) - 1;\\
&\text{if } Q \text{ is a } P_begin \text{ event, } P_left(\Pi') > P_left(\Pi);\\
&\text{otherwise, } P_left(\Pi') = P_left(\Pi).
\end{aligned}$$

Lemma 3 *If* $\Pi \xrightarrow{\{Q\}} \Pi'$, *then:*

$$\begin{aligned}
&\text{if } Q \text{ is an } R_commit \text{ event, } R_left(\Pi') = R_left(\Pi) - 1;\\
&\text{if } Q \text{ is a } D_begin \text{ event, } R_left(\Pi') > R_left(\Pi);\\
&\text{otherwise, } R_left(\Pi') = R_left(\Pi).
\end{aligned}$$

3 DEADLOCK IN TRANSACTION PROCESSING SYSTEMS

While deadlock is a very basic notion, there does not seem to be a universally accepted meaning for the concept. It is easy to recognize a deadlock scenario as such, but it is difficult to agree on what it means for a transition system to be deadlock-free. There are several formal notions of absence of deadlock in the literature, and none of them is completely satisfactory:

- For some authors, absence of deadlock means absence of reachable stop states, a stop state being a state that has no successor for the transition relation. However, there are clearly systems where the transition relation

has no stop states at all, and which nevertheless can be said to have deadlocks.

- For others, absence of deadlock means the possibility of reaching a forward progress milestone, along some path, from every reachable state. This can be expressed by an EF property in CTL. This has two drawbacks: the notion of forward progress milestone is problem-specific, and the truth of an EF formula depends on the amount of non-determinism in the system, which itself depends on the simplifications that have been made.
- The notion of deadlock is sometimes defined in terms of cycles in a dependency graph, or in terms of competition for shared resources among processes, but these are problem-specific formulations.

The notion of absence of deadlock that we use is applicable to all *transaction processing systems*: we say that such a system is deadlock-free iff every transaction is guaranteed to terminate provided that only a finite number of transactions are started, i.e. provided that no more transactions are started after a certain time. This can be easily formalized: a transition system is turned into a transaction processing system by labeling each state S with a *transaction count* $\mathcal{T}(S)$, and each edge E with a *transaction-creation count* $\mathcal{C}(S)$, in such a way that, if E takes S to S', then $\mathcal{T}(S') \leq \mathcal{T}(S) + \mathcal{C}(E)$. We can then define:

> The system is *deadlock-free* iff every path that starts at a reachable state, satisfies the fairness constraints if any, and traverses no edges with positive creation counts, eventually reaches a state where the transaction count is 0.

Of particular interest are the *closed* transaction processing systems. A system is closed if non-determinism is limited to transaction creation, i.e. if there is at most one edge with null creation count leaving any given state. Then the universal path quantifier in the above definition can be replaced with an existential path quantifier without changing the meaning of the definition.

4 PROOF OF ABSENCE OF DEADLOCK

In this section we consider a given acyclic network of PCI buses, evolving from an initial state in which there are no entries in agents or bridges. We define by induction the partition of each state S into transaction snapshots as explained in Section c, and we call $\Pi(t)$ the partition into transaction snapshots of the set of entries present in the system at time t.

4.1 Fairness constraints

The following fairness constraints are a complete set of rules that guarantee absence of deadlock. They make precise the deadlock-avoidance rules of the PCI Specification as well as some additional rules that are known to be necessary.

The specification stipulates that, once a transaction has been locally issued on a bus, it must be retried indefinitely until it is accepted. For posted transaction, this is expressed by the following fairness constraint.

Axiom 1 *Let N be a channel with out-bus B. If N contains a committed P entry with parameters (τ, α) at every time $t' > t$, then there is an infinite number of P events with parameters (τ, α) triggered by such an entry on bus B.*

Recall that a P entry present in a bridge channel N is committed, by definition, iff it is the oldest P entry in N. There may be older R or C entries in N, but such entries do not matter. This is consistent with the informal rule 2(a) mentioned above in Section b, which asserts that P entries must be allowed to pass R and C entries.

For delayed transactions, the stipulation that the bus transaction be retried again is made conditional on the availability of storage space in the opposite channel for the C entry that would result from a D_completion event. This is expressed by the following fairness constraint.

Axiom 2 *Let N be a channel with out-bus B and let N' be the channel opposite to N. If N' contains no P or C entries at any time $t' \geq t$, and N contains a committed R entry with parameters (τ, α) at every time $t' > t$, then there is an infinite number of D events with parameters (τ, α) triggered by such an R entry on bus B.*

Note that the presence of R entries in N' does not interfere with the requirement that delayed bus transactions be reissued forever on bus B. This is because, according to rule 2(b) of Section b, C entries must be allowed to pass R entries. We interpreting this as meaning that the creation of a C entry in N' as a result of a D event on bus B must not be prevented by the presence of R entries in N'.

The PCI specification stipulates that a local master (agent or bridge) must accept posted transactions under certain circumstances. The following two fairness constraints spell this out precisely.

Axiom 3 *Let N be the target channel of an agent A on bus B. If the set of events on bus B that target A contains an infinite number of P events, then it contains an infinite number of P_completion events.*

Axiom 4 *Let N be a channel of a bridge G, and let B be the in-bus of N. If the set of events on bus B that target G contains an infinite number of P events, then N contains P entries infinitely often.*

The specification should also have included a statement that a delayed transaction request must be accepted under some circumstances, but this was omitted inadvertently [9]. The following fairness constraints require that R entries be allowed to move forward if there is no interference by P and C entries.

Axiom 5 *Let N be the target channel of an agent A on bus B. If the set of events on bus B that target A contains a finite number of P events and an infinite number of D events, then it contains an infinite number of D_completion events.*

Axiom 6 *Let N and N' be the two channels of a bridge G, and let B be the in-bus of N. If N and N' are free of P and C entries after a certain time,* and the set of events on bus B that target G contains an infinite number of D events, then N contains R entries infinitely often.*

Axiom 7 *Let N be a channel of a bridge G, and let B be the out-bus of N. If, at every time $t' \geq t$, N contains R entries and is free of P and C entries, then at some time $t' \geq t$ there is a D event on bus B triggered by an R entry in N.*

The final fairness constraint captures the rule that C entries must be allowed to pass R entries.

Axiom 8 *Let N be a channel of a bridge G, with out-bus B. Then the following three statements cannot all be true: (i) there is an infinite number of D_retry events on bus B that target G, all having the same parameters (τ, α); (ii) at every time t' after some time t, N contains a C entry with parameters (τ, α) and this entry is the oldest C entry in N; and (iii) at every time t' after some time t, N contains no P entries.*

4.2 Absence of deadlock

Summary. Using the notion of absence of deadlock proposed in Section 3, we prove that, if only finitely many (global) transactions are started, then they all terminate. The proof proceeds in three stages. First, we show that after all

*The assumption that N' is free of P and C entries is relevant for implementations where the two channels of a bridge share storage.

the transactions have started, there is a time t_0 when all posted transactions have terminated. From then on there are only R and C entries in the system. In a second stage, we show that, if there is a C entry in the system at any time $t > t_0$, then a transaction terminates at some time $t' \geq t$. Finally, we show that, if the system contains R entries but no C entries at any time $t > t_0$, then, at some time $t' \geq t$, either a transaction terminates or a C entry appears. Thus all transactions terminate eventually.

Lemma 4 *If there is a P entry in the system at time t, then there is a P_completion event at some time $t' \geq t$*

PROOF. Assume that there is a P entry at time t, and consider the restriction of the channel graph to the set of channels that contain P entries at time t. The restricted graph is non-empty, finite, and acyclic, and must therefore have a leaf node N_0. Let E be a committed P entry, with parameters (τ, α), present in N_0 at time t. (If N_0 is a bridge channel, E is the oldest entry in N_0 at time t. If N_0 is the master channel of an agent, E is any P entry in N_0.)

Reasoning by contradiction, assume that there are no P_completion events at any time $t' \geq t$. Then N_0 contains a committed P entry with parameters (τ, α) at every time $t' > t$, and by Axiom 1 there is an infinite number of P events on the out-bus B of N_0. Let $N_0, N_1, \ldots N_n$, $n \geq 1$, be the path from N_0 to the target channel N_n of the target agent A specified by α.

Assume that $n = 1$, so that N_1 is the target channel of A. The set of events on bus B that target A contains an infinite number of P events but only a finite number of P_completion events. This contradicts Axiom 3.

Assume that N_1 is a channel of a bridge G. The set of events on bus B that target G contains an infinite number of P events. But since N_0 is a leaf node in the restricted graph, N_1 contains no P entries at time t. And since there are no P_completion events at any time $t' \geq t$, there are no P entries at any time $t' \geq t$. This contradicts Axiom 4.

□

Lemma 5 *If there is a finite number of P_begin events, then there is a time after which there are no P entries in the system.*

PROOF. If there is a finite number of P_begin events, then there is a time t_0 after which there are no more such events. By Lemma 2, for every $t > t_0$, $P\_left(\Pi(t+1)) \leq P\_left(\Pi(t))$, with $P\_left(\Pi(t+1)) < P\_left(\Pi(t))$ if there is a P_completion event at time t. Hence there can only be a finite number of P_completion events, and, by Lemma 4, there can only be a finite number of times t at which there are P entries in the system. Hence there is a time after which there are no P entries.

□

Lemma 6 *If there is a C entry in the system at time t, and there are no P entries at any time $t' \geq t$, then there is a D_end event at some time $t' \geq t$.*

PROOF. Reasoning by contradiction, assume that there exists a C entry in a bridge channel N at time t and there are no P entries and no D_end events at any time $t' \geq t$. We construct then an infinite sequence of pairs (N_i, t_i), $i \geq 0$, with $N_0 = N$ and $t_0 = t$, where $t_i \geq t$, N_i is a bridge channel that contains a C entry at time t_i, and $N_i \rightarrow N_{i+1}$ in the channel graph.

Suppose the sequence has been constructed up to (N_i, t_i). Let G be the bridge containing the channel N_i and B the out-bus of N_i. Let E_C be the oldest C entry present at time t_i in N_i, and let (τ, α) be the parameters of E_C. E_C belongs to an RC transaction snapshot $T \in \Pi(t_i)$ in which there is a committed R entry E_R in a channel N_R whose out-bus is B. E_R has the same parameters (τ, α) as E_C. Let N_i' and N_R' be the opposite channels to N_i and N_R respectively. We have $N_R \rightarrow N_i'$, and therefore $N_i \rightarrow N_R'$. We distinguish two cases.

Suppose first that N_R' contains a C entry at some time $t' \geq t_i$. Then let $N_{i+1} = N_R'$ and $t_{i+1} = t'$.

Suppose now that N_R' contains no C entry at any time $t' \geq t_i$. Note that N_R' contains no P entry either at any time $t' \geq t_i$, since there are no P entries in the system at any time $t' \geq t$. We prove by contradiction that there is a D_completion event on bus B with parameters (τ, α) at some time $t' \geq t_i$.

Suppose that this is not the case. Then:

(i) Given that there is a committed R entry in N_R at time t_i with paremeters (τ, α), viz. E_R, given that there are no D_completion events at any time $t' \geq t_i$, and given that a committed R entry cannot be discarded, there is a committed R entry in N_R with parameters (τ, α) at every time $t' \geq t_i$. Hence, since there are no P or C entries in N_R' at any time $t' \geq t_i$, by Axiom 2, there is an infinite number of D events on bus B with parameters (τ, α). And since there are no D_completion events at any time $t' \geq t_i$, an infinite number of such events are D_retry events.

(ii) Given that N_i contains E_C at time t_i as its oldest C entry, given that there are no D_completion events at any time $t' \geq t_i$, and given that the oldest C entry in a bridge channel cannot be discarded, there is a C entry in N_i with parameters (τ, α) at every time $t' \geq t_i$, and that entry is the oldest C entry at time t'.

(iii) There are no P entries in N_i at any time $t' \geq t_i \geq t$.

Together, these three facts contradict Axiom 8.

Let then Q be a D_completion event on B with parameters (τ, α) at time $t' \geq t_i$, triggered by an R entry in a channel N_*. (In the absence of master ID lines, N_* may be other than N_R.) N_* cannot be the master channel of an agent, because then Q would be a D_end event; hence it must be a channel of

a bridge G_*. Let N'_* be the channel other than N_* in G_*. We have $N_* \rightarrow N'_i$ and hence $N_i \rightarrow N'_*$. And N'_* contains a C entry at time $t' + 1$. We can thus let $N_{i+1} = N'_*$, and $t_{i+1} = t' + 1$.

The sequence N_i, $i \geq 0$, that we have constructed is an infinite path in the channel graph, whose existence is a contradiction since the channel graph is non-empty, finite and acyclic.
□

Lemma 7 *If there are one or more R entries but no C entries at time t, and there are no P entries and no D_begin events at any time $t' \geq t$, then a D_completion event occurs at some time $t' \geq t$.*

PROOF. Assume that there are one or more R entries but no C entries at time t, and there are no P entries and no D_begin events at any time $t' \geq t$. Reasoning by contradiction, assume that there are no D_completion events at any time $t' \geq t$. Then, since there are no C entries at time t, there are no C entries at any time $t' \geq t$. Also, since there is an R entry at time t, there is a D transaction snapshot $T \in \Pi(t)$, and T contains a (committed) R entry in the master channel N of an agent (the agent that originated the global transaction of which T is a snapshot). In the absence of D_completion events, N contains a (committed) R entry at every time $t' \geq t$. Thus, at every time $t' \geq t$ the system contains a committed R entry but no C entries or P entries.

Since there are no D_begin events at any time $t' \geq t$, by Lemma 3, $R\_left(\Pi(t'))$ is a non-increasing function of t' for $t' > t$, with $R\_left(\Pi(t'+1)) < R\_left(\Pi(t'))$ if there is an R_commit event at time t'. To obtain a contradiction it suffices to show that there is an infinite number of R_commit events. Let us show that, for every $t' > t$, there exists an R_commit event at some time $t'' \geq t'$.

Let $t' > t$, and consider the restriction of the channel graph to the set of channels that contain a committed R entry at time t'. The restricted graph is non-empty, finite, and acyclic, and must therefore have a leaf node N_0. Let E_0 be a committed R entry present in N_0 at time t, and let (τ, α) be the parameters of E_0. Since there are no D_completion events at any time $t'' \geq t' > t$ and a committed R entry cannot be discarded, N_0 contains a committed R entry at every time $t'' \geq t'$. Moreover, the channel N'_0 opposite to N_0 contains no P or C entries at any time $t'' \geq t'$. Therefore, by Axiom 2, there is an infinite number of D events with parameters (τ, α) on the out-bus B of N_0. Let $N_0, N_1, \ldots N_n$, $n \geq 1$, be the path from N_0 to the target channel N_n of the target agent A specified by α.

It cannot be the case that $n = 1$, i.e. that N_1 is the target channel of A, for then the set of events on bus B that target A would contain a finite number of P events, an infinite number of D events, but only a finite number of D_completion events. This would contradict Axiom 5.

Hence N_1 is a channel of a bridge G, and since N_0 is a leaf of the restricted graph, N_1 contains no committed R entries at time t'. Reasoning by contra-

diction, assume that there exists no R_commit event at any time $t'' \geq t'$. Then N_1 contains no committed R entries at any time $t'' \geq t'$.

Let N_1' be the channel opposite to N_1. Neither N_1 nor N_1' contain any P or C entries at any time $t'' \geq t'$. Since there is an infinite number of D events with parameters (τ, α) on bus B, by Axiom 6, N_1 contains R entries infinitely often. Thus N_1 contains an R entry at some time $t'' \geq t'$, and this entry is not committed. In the absence of P and C entries in N_1 and N_1', and of committed R entries in N_1, a non-committed R entry can only be discarded from N_1 if it is not the only one in the channel. In the absence of R_commit events and D_completion events, this implies that there is at least one non-committed R entry at every time $t''' \geq t''$. But then, by Axiom 7, there must be a D event on the out-bus B' of N_1 triggered by such an entry at some time $t''' \geq t''$. Such a D event is an R_commit event, a contradiction.
□

Theorem 1 *If there is a finite number of P_begin and D_begin events, then there is a time after which there are no entries in the system.*

PROOF. Assume that there is a finite number of P_begin and D_begin events. Then, using Lemma 5, there is a time t_0 such that, at every time $t \geq t_0$ there are no P entries in the system and no D_begin events take place. Let us show that there is a time after which there are no D transaction snapshots, and hence no R or C entries. By Lemma 1 it suffices to show that, for every time $t > t_0$ at which there is a D transaction snapshot, there exists a time $t' \geq t$ at which there is a D_end event. So let $t > t_0$, and assume that there is a D transaction snapshot, and hence an R entry, at time t. If there is a C entry at time t then by Lemma 6 there is indeed a time $t' \geq t$ at which a D_end event occurs. Otherwise by Lemma 7 there exists a time $t^* \geq t$ at which a D_completion event occurs. If this D_completion event is a D_end event, then t^* can serve as t'. If not, the D_completion event results in the existence of a C entry at time $t^* + 1$, and then, by Lemma 6, there exists a time $t' \geq t^* + 1$ at which a D_end event occurs.
□

5 CONCLUSION

We have presented a proof of absence of deadlock for an arbitrary acyclic network of PCI buses where transactions follow Revision 2.1 of the PCI protocol, with certain modifications. Specifically, we have shown that every transaction terminates provided that only a finite number of transactions are initiated. We propose this as a notion of absence of deadlock suitable for any transaction processing system.

In order to carry out the formal proof we have formalized the protocol, transforming the informal *passing rules* of the protocol into fully precise fair-

ness constraints. In our formalization we have incorporated improvements to the protocol known to be necessary in the VLSI design community. Our proof shows that those improvements are sufficient to guarantee absence of deadlock, and hence that there are no additional deadlock scenarios lurking in the specification. This should allow developers of PCI controllers and bridges to conceive aggressive designs with confidence. Our proof also provides insight into the protocol which will be helpful when contemplating modifications or extensions of the specification.

The proof of absence of deadlock in PCI is part of a broader ongoing verification project concerning a computer system that uses PCI as an I/O bus. In this broader project we have been able to find deadlock scenarios very early in the design process. Details cannot be given at this time due to the confidential nature of the design being verified. However the proof presented here for the public domain PCI protocol is an example of a manual proof that can be very useful in the early stages of the design of a computer system.

While developing the proof presented here we found a solution to an ordering problem which is partially documented in the PCI specification. The solution involves the addition of a local master ID to the protocol, in a backwards-compatible manner. We have identified fairness constraints that take advantage of the existence of a local master ID to avoid starvation, and proved full liveness (absence of deadlock and absence of starvation) under those constraints. We plan to present those results in the near future.

REFERENCES

[1] S. Bainbridge, A. Camilleri, and R. Fleming. Theorem proving as an industrial tool for system level design. In V. Stavridou, T. F. Melham, and R. T. Boute, editors, *Theorem Provers in Circuit Design.* North-Holland, 1992.

[2] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the futurebus+ cache coherence protocol. In D. Agnew, L. Claesen, and R. Camposano, editors, *Proceedings of the 11th Int. Conf. on Computer Hardware Description Languages and their Applications.* North-Holland, 1993.

[3] E. A. Emerson and K. S. Namjoshi. Automatic verification of parameterized synchronous systems. In *Computer Aided Verification, 8th International Conference, CAV'96,* pages 87–98. Springer-Verlag LNCS 1102, 1996.

[4] M. J. C. Gordon and T. F. Melham. *An Introduction to HOL.* Cambridge University Press, 1993.

[5] P. H. Ho, Y. Hoskote, T. Kam, M. Khaira, J. O'Leary, X. Zhao, Y. A. Chen, and E. Clarke. Verification of a complete floating-point unit using word-level model checking. In M. Srivas and A. Camilleri, editors, *Proceedings of the Int'l Conf. on Formal Methods in Computer-Aided*

Design, FMCAD'96. Springer-Verlag, November 1996. LNCS 1166.

[6] C. Ip and D. Dill. Better verification through symmetry. In D. Agnew, L. Claesen, and R. Camposano, editors, *Proceedings of the 11th Int. Conf. on Computer Hardware Description Languages and their Applications*. North-Holland, 1993.

[7] PCI Special Interest Group. *PCI Local Bus Specification, Revision 2.1*, June 1995.

[8] F. Pong and M. Dubois. A new approach for the verification of cache coherence protocols. *IEEE Transactions on Parallel and Distributed Systems*, 6(8):773–787, August 1995.

[9] Norm Rasmussen. Private communication.

[10] Mandayam K. Srivas and Steven P. Miller. Applying formal verification to the AAMP5 microprocessor: A case study in the industrial use of formal methods. *Formal Methods in Systems Design*, 8(2):153–188, March 1996.

BIOGRAPHIES

Dr. Francisco Corella is a Member of the Technical Staff of Hewlett-Packard, where he currently specializes on the formal verification of computer systems. He received his PhD from the University of Cambridge, England, in 1990. Prior to joining Hewlett Packard, he was a Reserch Staff Member at the IBM T. J. Watson research center. Dr. Corella's interests include formal methods, computer architecture, database and information retrieval systems, decision support systems, and cryptography.

Rob Shaw is a fifth-year Ph.D. candidate at U.C. Davis. Before returning to graduate school, he was a programmer in the Language Group at Lawrence Livermore Laboratory. Shaw holds a Master's Degree in Computer Science from U.C. Davis, and a Bachelor's Degree in Mathematics with Computer Science from MIT. His interests include programming languages and formal reasoning techniques for hardware and software.

Dr. Cui Zhang is an Associate Professor at the Department of Computer Science, California State Unviersity at Sacramento (CSUS). She received a Ph.D. in Computer Science from Nanjing University, China, December 1986. Before joining CSUS, Dr. Zhang was on the faculty of Computer Science at Fudan University, Shanghai, and later a Professional Researcher at the Department of Computer Science, University of California at Davis. Dr. Zhang's research interests include formal methods, computer-aided verification, software engineering, and programming languages.