

Efficient Verification of the PCI Local Bus using Boolean Satisfiability

Fadi A. Aloul and Karem A. Sakallah

Department of Electrical Engineering and Computer Science

University of Michigan

AnnArbor, MI 48109-2122

{faloul, karem}@eecs.umich.edu

***Abstract:** The purpose of this paper is to study the application of Boolean Satisfiability to the verification of the PCI Local Bus. The novel feature of this approach is the generation of several propositional formulas that describe the specification of the bus system. The formulas are tested using a powerful SAT solver and the bus is verified for errors. The SAT-based approach has several important advantages over conventional BDD-based approaches such as achieving high speed testing. To demonstrate how our method works, we have modeled the PCI Local Bus and verified several properties.*

1 Introduction

Recent developments in semiconductor technology have made possible the integration of over 10 million transistors on a single chip. A related practice which is evolving to avoid designing systems from scratch is the use of predefined logic blocks referred to as Intellectual Property Components (IP cores) which are usually integrated using standard buses. While most components are usually pre-validated, standard bus protocols still need to be tested and verified.

Testing bus protocols is a major challenge. Most bus interface specifications are complex and informally specified. Although a significant amount of work is devoted to the design and validation of bus interfaces, errors produced by bus protocols can be very difficult to debug especially when system designers have restricted knowledge of the connected IP cores due to the protection of intellectual property.

This paper examines the behavior of the PCI Local Bus [12]. It was developed by Intel to accommodate the advance design of their pentium processors. The high performance bus is also used in DEC Alpha processors. We will model the PCI bus and verify several of its properties using SAT techniques.

Several methods have been proposed to verify the PCI Local Bus [5, 6, 10, 11]. Formal verification techniques, such as model checking [8], have been used by several researchers to verify the complex hardware design. Chauhan et al. [6] specified the PCI bus using CTL [7] and model checked the state machine mentioned in the PCI specification document [12]. On the other hand, Shimizu et al. [10] specified the PCI bus with monitors and model checked various monitor properties. They present a simple specification style using monitor circuits. The style encourages readability and discourages errors. The English description of the PCI specification was translated into a collection of simple properties and a model checker was used to debug the monitor specifications.

Recently, formal verification techniques, such as bounded model checking [1, 2], that uses satisfiability solvers instead of BDDs have been proposed. SAT-based approaches have several advantages over BDD-based approaches. BDDs [3] are based on canonical representations. A complete BDD has to be constructed for every problem before testing it for 1-terminals. For large functions, however, the BDD sizes might be too large to fit into available computer memory. In any case, approaches that attempt to construct a complete representation of the function before checking its satisfiability end up doing more work than necessary. Therefore, SAT-based approaches that intelligently sample the variable space are often more effective. They are based on a polynomial representation and follow a depth-first search algorithm, looking for a satisfying assignment. They are less likely to fail due to exponential space blowup. Furthermore,

BDD-based approaches require a good variable ordering to minimize the size of the BDD. However, these orderings usually need long periods of time. In contrast, SAT-based approaches can use different decision heuristics at different parts of the search tree to obtain an optimal solution in a short period of time.

In this paper, we use Boolean satisfiability to verify the PCI Local Bus. Simple rules from the PCI specification document [12] are extracted and written into a set of logical expressions in conjunctive normal form (CNF). The set of simple properties are based on the monitors introduced by Shimizu et al. [10]. Several properties represented in CTL are converted into CNF and added into a CNF formula that contains the PCI specification expressions. A counter example of length K that violates a property can exist based on the solution of the CNF formula. Several formulas are created to verify each property. The formulas are tested using a powerful SAT-solver.

The remainder of this paper is organized as follows. Section 2 presents a brief description of the CNF representation and the GRASP SAT solver. Section 3 discusses the modeling of the PCI Local Bus and Section 4 shows how it can be verified using our techniques. We also show some of the bugs found in the PCI protocol. Finally, Section 5 concludes the paper and provides some prospective on future work.

2 Basic Definitions & Notations

A *conjunctive normal form* (CNF) formula ϕ on n binary variables x_1, \dots, x_n is the conjunction (AND) of m clauses $\omega_1, \dots, \omega_m$ each of which is the disjunction (OR) of one or more literals, where a literal is the occurrence of a variable or its complement. The size of a clause is the number of its literals. A formula ϕ denotes a unique n -variable Boolean function $f(x_1, \dots, x_n)$ and each of its clauses corresponds to an implicate (clause) of f . Clearly, a function f can be represented by many equivalent CNF formulas. We will refer to a CNF formula as a *clause database* and use “formula,” “CNF formula,” and “clause database” interchangeably. The satisfiability problem (SAT) is concerned with finding an assignment to the arguments of $f(x_1, \dots, x_n)$ that makes the function equal to 1 or proving that the function is equal to the constant 0.

In our experiment, we used the GRASP SAT solver [13] which can be classified as an enhancement to the basic Davis-Putnam backtrack search procedure [9]. The solver performs a depth-first search and can be viewed as consisting of 3 main engines: *decision*, *deduction*, and *diagnosis*. GRASP's diagnosis engine analyzes the causes of conflicts and generates adequate information to prevent the conflicts from occurring at different parts of the search space. Furthermore, it enhances the diagnosis engine with a non-chronological backtracking scheme. A detailed description of the solver can be found in [13].

3 PCI Description

This section examines the PCI Local Bus, a high performance synchronous bus standard developed by Intel [12]. The bus has a data width of 32 or 64 bits. It is mainly used to connect processors, SCSI controllers, LAN networks, motion video cards, and ISA bridge controllers.

The bus supports two main phases: *Arbitration* and *Address/Data*. Both phases are fully synchronous. The *arbitration* phase involves a centralized algorithm. Each device on the bus has a unique pair of request (REQ) and grant (GNT) lines hardwired to a central arbiter. If a device desires the use of the bus it asserts the REQ line. The arbiter views the incoming requests and makes a decision on the next clock cycle. The decision is based on a *Fixed-Priority* or *Round-Robin* policy. The central arbiter is totally fair and serves every master. If the device is granted the bus, the arbiter will assert the device's GNT line. The new bus master can't control the bus until the old bus master has transferred all its data. The new master will have access to the bus as soon as the bus becomes idle. The bus is idle whenever the signals (FRAME) and initiator ready (IRDY) are deasserted.

A device can start transmitting data as soon as its granted the bus and the bus is idle. This is referred to as the *Address/Data* phase. The transaction begins by asserting the FRAME signal. The address and the transaction type are then placed on the address/data multiplexed lines and the command line respectively. All target devices listen to the address and assert their device select (DEVSEL) line if it maps to their

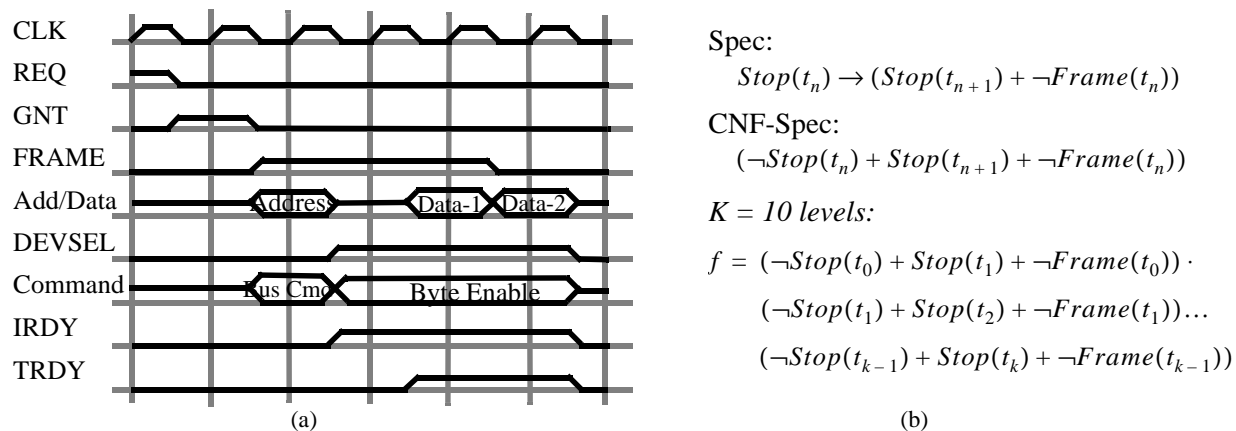


Figure 1: (a) PCI Transaction Example (b) Example of the PCI specification mode

address space. This indicates to the master that the target is present on the bus. The master then asserts the signal IRDY, declaring to the target that it is ready to transfer data. Data transfer starts when both IRDY and TRDY are asserted. Wait signals can be inserted between the data transfer phases by deasserting the IRDY or TRDY signals. The FRAME signal is deasserted one clock cycle before the end of the data phase. On the next clock cycle, both TRDY and IRDY signals are deasserted indicating the end of the *Address/Data* phase and making the bus return to the idle state. Furthermore, transactions can be terminated by the master or target in various situations. These are discussed in more detail in the next section.

4 PCI Verification

The key issues in verifying the PCI Local Bus are the *modeling* and *debugging* of the PCI specification. The PCI specification document is written in English and contains an informal overview of how the protocol works. We start by modeling the PCI protocol in CNF. For example, the PCI specification document lists a set of rules describing the target termination procedures. A simple rule such as:

“Once a target asserts STOP#, it must keep STOP# asserted until FRAME# is deasserted, whereupon it must deassert STOP#.” P.42 Sec 3.3.3.2.1

can be easily translated into a set of logical expressions:

$$Stop(t_n) \rightarrow (Stop(t_{n+1}) + \neg Frame(t_n))$$

$$(Frame(t_n) \& \neg Frame(t_{n+1})) \rightarrow \neg Stop(t_{n+2})$$

The above logical expressions can be represented in CNF as follows:

$$(\neg Stop(t_n) + Stop(t_{n+1}) + \neg Frame(t_n))$$

$$(\neg Frame(t_n) + Frame(t_{n+1}) + \neg Stop(t_{n+2}))$$

The PCI specification document is completely translated to CNF. Note that variables declared in CNF correspond to signals at specific time instances. Therefore, testing up to K time stages requires defining K unique variables for each signal. This is very useful in obtaining an accurate value for each signal at any time instance. The PCI specification is modeled in CNF for a *single* time unit t_0 using the signal variables for t_0 . The single model is then replicated up to K levels by replacing every variable by the variable corresponding to the given time unit. An example of that is shown in Fig. 1(b). Finally, in order to avoid invalid states, signal values are initialized at time unit t_0 to the idle state.

The second key issue has to do with debugging the protocol. Properties have to be added to check that no device ever observes an illegal combination of bus signals or an unexpected transaction. CTL formulas [7] can be used for dead state (existing reachable states with no transitions to other states) and property checking (incorrect behavior). CTL formulas are built from three components: *atomic propositions*, *Boolean connectives*, and *temporal operators*. Atomic propositions represent values of individual state vari-

$$f = \Phi_{model} \cdot \Phi_{debug} \quad \omega_k := \text{CTL property at time } k$$

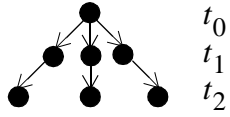
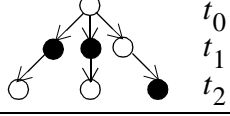
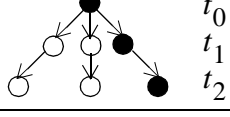
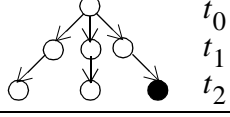
		CTL formula in CNF	Valid property if
AG		$\Phi_{debug} = (\neg\omega_0 + \neg\omega_1 + \dots + \neg\omega_{k-1})$	$f = UNS$
AF		$\Phi_{debug} = (\neg\omega_0 \cdot \neg\omega_1 \cdot \dots \cdot \neg\omega_{k-1})$	$f = UNS$
EG		$\Phi_{debug} = (\omega_0 \cdot \omega_1 \cdot \dots \cdot \omega_{k-1})$	$f = SAT$
EF		$\Phi_{debug} = (\omega_0 + \omega_1 + \dots + \omega_{k-1})$	$f = SAT$

Figure 2: Converting CTL formulas to CNF

ables. Boolean connectives include the basic AND (\wedge), OR (\vee), and NOT (\neg) operations. Finally, temporal operators include path quantifiers (A:all, E:some) and temporal modality (F:eventually, G:globally, X:next, and U:until). Path quantifiers indicate whether a property has to hold for “all” execution paths or for “some” paths. Properties expressed in CTL can be easily converted to CNF according to Fig. 2. For example, to check whether two signals are never valid together for *all* paths, we use the following:

$$AG \neg (\text{signalA} \ \& \ \text{signalB})$$

Since SAT solvers return after identifying a single valid path, it would be impossible to test if a property holds for all paths. Therefore, the inverse of the property is tested. An unsatisfiable solution indicates that the inverse of the property doesn't appear in any path, hence the original property is valid for all paths. On the other hand, a satisfiable solution indicates a correct path with a false property. Converting the CTL formula to CNF yields the following boolean function:

$$f = (\Phi_{model}) \cdot (\text{signalA}(t_0) \cdot \text{signalB}(t_0) + \text{signalA}(t_1) \cdot \text{signalB}(t_1) + \dots + \text{signalA}(t_{k-1}) \cdot \text{signalB}(t_{k-1}))$$

Proving the function unsatisfiable indicates that the CTL property holds. Note that a separate CNF formula is tested for each time unit. The function is unsatisfiable if all K formulas are proven unsatisfiable.

The PCI specification was modeled in CNF up to 10 levels. The CNF model consisted of 1500 variables and 11,000 clauses. The model included a single target with two masters but can be expanded to several devices. In order to make the verification simple, we didn't include bus bridges, 64-bit extensions, locks, or interrupts. Several properties were verified including bus arbitration, read/write transactions, back-to-back transactions, and master/target terminations. The CNF formulas were tested using GRASP. The experiments were conducted on a 333 MHz Pentium Pro running Linux and equipped with 512 MByte of RAM. In order to measure the performance difference between SAT-based and BDD-based approaches, we modeled the PCI specification in SMV [4] and tested the same properties using CTL model checking. We used Carnegie Mellon's SMV [4] as the model checker. The SAT-based approach was able to verify the protocol within 2 minutes using up to 3.5 MByte of memory. The BDD-based approach required 112 minutes of CPU and up to 27MByte of RAM.

The above experiment uncovered several errors in the PCI specification that were previously identified by Chauhan et al. [6] and Shimizu et al. [10]. Some of these errors were based on a misinterpretation of the English description in the PCI specification document. Others were actual errors in the PCI protocol. One of the main errors detected deal with target-initiated terminations. Three types of target-initiated termina-

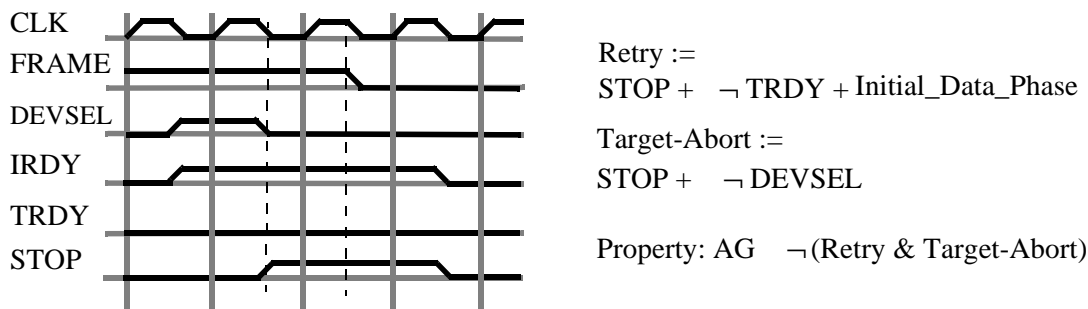


Figure 3: Error detected in the PCI specification

tions are defined: *Retry*, *Disconnect*, and *Target-Abort*. Masters must be capable of uniquely identifying each termination type in order to respond to the target. Hence, termination types have to be unique for each transaction. GRASP was able to detect a scenario where both *retry* and *target-abort* were valid at the same time. An example of the transaction is shown in Fig. 3.

5 Conclusions & Future Work

This paper provides a new method of verifying busses based on Boolean satisfiability. The idea is motivated by studies that show SAT-based approaches to be more efficient than BDD-based approaches. A BDD-based approach used in verifying large hardware systems is likely to run out of memory and as a result it is usually very difficult to test such systems. However, by using SAT-based approaches such systems can often be easily solved using less memory and time requirements. We have attempted to model the PCI Local Bus specification as a set of logical statements which are converted to conjunctive normal form. We verified several properties and confirmed the detection of several errors reported by other researchers. We also model checked the PCI using SMV and compared the experimental results obtained by both techniques. The experimental results validate the effectiveness of our approach. A natural progression of this work is to apply the technique to other types of busses and hardware systems.

6 Acknowledgments

We would like to thank D. Dill, K. Shimizu, and A. Hu for providing us with the PCI SMV model used in this study.

7 References

- [1] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu, "Symbolic Model Checking using SAT procedures instead of BDDs," in Proc. of the Design Automation Conference, 1999.
- [2] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," in TACAS, 1999.
- [3] R. Bryant, "Graph-based algorithms for boolean function manipulation," in Proc. of IEEE Transactions on Computers, 1986.
- [4] J. Burch, E. Clarke, K. McMillan, and D. Dill, "Sequential circuit verification using symbolic model checking," in Proc. of the 27th ACM/IEEE Design Automation Conference, 1990.
- [5] S. Campos, E. Clarke, W. Marrero, M. Minea, "Verifying the performance of the PCI Local Bus using Symbolic Techniques," in Proc. of the IEEE International Conference in Computer Design, October 1995.
- [6] P. Chauhan, E. Clarke, Y. Lu, and D. Wang, "Verifying IP-Core based System-On-Chip Designs," in Proc. of the IEEE ASIC Conference, September 1999.
- [7] E. Clarke and E. Emerson, "Synthesis of synchronization skeletons for branching time temporal logic," in Logic of Programs: Workshop, Yorktown heights, NY, May 1981.
- [8] E. Clarke, E. Emerson, and A. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic," in Proc. of ACM Transactions on Programming Languages and Systems, 8(2):244-263, April 1986.
- [9] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," Journal of the Association for Computing Machinery, vol. 7, pp. 201-215, July 1960.
- [10] K. Shimizu, D. Dill, A. Hu, "Monitor Based Formal Specification of PCI," unpublished manuscript, 2000.
- [11] A. Mokkedem, R. Hosabetu, and G. Gopalakrishnan, "Formalization and Proof of a Solution to the PCI 2.1 Bus Transaction Ordering Problem," in Proc. of the Second International Conference, Formal Methods in Computer-Aided Design, 1998.
- [12] PCI Special Interest group, "PCI Local Bus Specification," Revision 2.2, December 1995.
- [13] J. Silva and K. Sakallah, "GRASP-A New Search Algorithm for Satisfiability," in Proc. of the ICCAD, November 1996.