Formal Checking of Properties in Complex Systems Using Abstractions *

Dinos Moundanos Jacob A. Abraham
Computer Engineering Research Center
The University of Texas at Austin
ENS 424
Austin, TX 78712

Abstract

Only very small designs can be verified currently using property checking due to state-space explosion. Abstractions have been developed to simplify the design in an attempt to address this problem. However, the properties themselves may involve large state spaces, and practical property checking is generally confined to the control behavior. This paper describes an elegant technique for verifying properties of complex designs where the abstraction is applied to both the property and the design, thereby allowing us to verify properties which may deal with the data space. We demonstrate the technique on a processor by checking properties which are intractable using existing model checking techniques.

1 Introduction

Advances in semiconductor technology are allowing the number of transistors integrated on a chip to increase exponentially. As the resulting designs become more and more complex, verifying their correctness is becoming more and more difficult. The current practice in industry is to use simulation to attempt to detect bugs in the design. Since simulation is able to check increasingly smaller portions of designs, there has been a growing interest in using formal techniques to verify designs [5]. Formal approaches are becoming common in checking the correctness of implementations against specifications (*implementation verification*), particularly at the Boolean logic level, and commercial tools for Boolean equivalence checking are becoming common in the design flow. The other aspect of formal verification involves proving either that a design is correct with respect to a set of properties specified by the designer, or that the design possesses the functionality intended by the designer (*design verification*) [6, 1].

*This research was supported in part by the Semiconductor Research Corporation under contract 97-DJ-483 and in part by the Texas Advanced Technology Development and Transfer Program under Project 003658-433 at the University of Texas at Austin. Dinos Moundanos is currently with Fujitsu Laboratories of America, Sunnyvale, California.

1.1 Dealing with State Space Explosion

The number of states in real designs is so large that the basic techniques for verifying properties cannot be applied directly to them. The most important technique used to handle the verification of realistic circuits is to control the complexity by *abstracting* the design into a reduced model. Even if we were able to abstract the design, the properties themselves may involve large state spaces, for example, when specifying properties on data registers. Consequently, property checking is generally confined to the control behavior, and cannot deal with properties dealing with data-dependent control.

This paper describes an elegant technique for verifying properties of complex designs where the abstraction is applied to both the property as well as to the design, allowing us to verify properties which may deal with the data space. We specify properties as state machines, and develop an abstraction which is applicable to both the property and the design. We will describe the technique and the decision procedure that gives the final answer as to whether a property holds or not on the original machine. We will demonstrate the effectiveness of this approach on a processor by checking properties which are intractable using existing model checking techniques.

1.2 The Extracted Control Flow Machine (ECFM) Model

In most applications, including design verification and test generation, the flow of control is of prime interest. However, extracting the control machine from a circuit description is not an easy task because the control circuitry cannot be distinguished easily from the data path without relying on designer annotations to the circuit description.

0-7695-0104-4/99 $10.00 © 1999 IEEE

The ECFM of a sequential circuit is a model of the control flow in the design and is extracted by looking at the finite state machine representing the complete circuit [2, 3]. The difficulty in identifying the control circuitry often lies in defining the interface of the control unit with the rest of the circuit, and not in differentiating the control registers from registers holding pure data. In the ECFM methodology, the designer chooses the registers which are to be considered as contributing to the control state space and which make up the data. The key issue here is that we are only interested in the data part of the state space to the extent that it affects the flow of control in the circuit. Consequently, we abstract the data registers from the circuit and group the data into "equivalence" classes with respect to their effect on the control. This abstraction is done by making the data registers completely non-deterministic, essentially primary inputs. This extended input space is then grouped into equivalence classes.

2 Formal Verification of Properties Using Abstraction

We have developed a novel technique for verifying certain kinds of properties for circuits with wide datapaths. The key point of our approach is the fact that we apply the same type of abstraction techniques to both the state machine describing the design and to the state machine that describes the property to be verified. We specify properties as labeled finite-state machines (FSMs) written in a hardware description language (HDL). Both liveness and safety properties can be described in this framework. Labeled FSMs are being utilized to account for the non–determinism that may be present in the design. At least one path through a labeled state and all paths through non–labeled states must satisfy the desired behavior for the labeled state machine to model the property correctly. This means that labeled states are used to describe non–deterministic behavior and non–labeled states account for deterministic behavior. Additionally, a labeled state machine may be partially specified both in terms of primary outputs and next states. A sink state is used to absorb all transitions with unspecified next states. The verification algorithm computes all the states in the design that are compatible with the specification from its start state. Both pre–image and inverse image computations are required because of the two types of states present in the specification [3].

Our verification system is depicted in Figure 1. The design is assumed to be described at the Register Transfer Level (RTL) in either Verilog or VHDL. The property is also expressed as a state machine in either Verilog or VHDL. Based on designer input, we extract the ECFM of the design. To continue we apply the same abstraction technique on the property machine, assuming that it involves data reg-

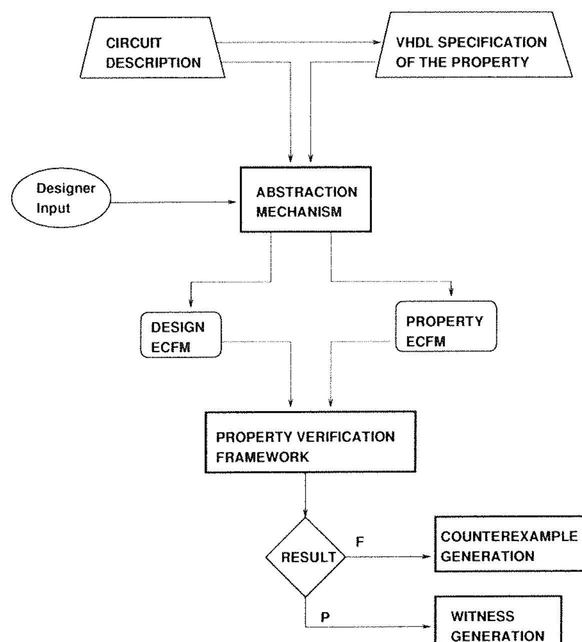

Figure 1. Property Verification System

isters. Otherwise, if it is only concerned with control behavior, the property machine is used as is.

The verification problem is set up as a containment check (see Figure 2). We check to see whether the property machine is contained in the state machine describing the ECFM of the design. This is in contrast to the approach in [4], where the language of the system is shown to be or not be contained in the language of the property. This approach ensures that the behavior of the design is consistent with that of the specification (property). In our approach we make sure that the design can perform all the behavior defined in the specification which is a more realistic paradigm. Furthermore, we do not require knowledge of the initial state(s) for the design. So our approach guarantees, in the case of success, that for every possible starting state in the specification machine, there exists a corresponding state in the design machine, such that, for each such pair of states, the language of the design machine (viewed as an automaton) is contained in the language of the property machine (also viewed as an automaton). Additionally, there are several properties involving existential quantification that are not expressible in the framework of [4]. For example, the property that there exists a path to a reset state from any design state cannot be expressed.

Abstraction is the process of reducing the proof of a property on a large state space system to a proof on a smaller state space (abstract model). Abstractions can be exact, conservative or aggressive. An abstraction is exact when properties hold on the abstract model if and only if they hold on the original model. An abstraction is conservative

when a property that holds on the abstract model holds on the original model too. Finally, an abstraction is aggressive when a property that does not hold on the abstract model does not hold on the actual model either.

In our case the abstraction falls in the second category, the category of conservative abstractions. Our abstraction eliminates portions of the datapath and retains the control behavior of the design. However, the control space of the abstracted model (the ECFM) is a conservative but close approximation of the actual control space. This is the case because, for the most part, datapath registers can assume any value at any clock cycle.

The question now is what it means for the abstracted property to be contained in the abstracted machine. The answer to that question depends on the type of properties with which we are dealing. In general we have pure control properties, pure data properties and control–data properties. Control properties have to do with the control behavior of the machine and do not involve any data dependencies. Data properties are those that the datapath must satisfy. Both Model Checking and Theorem Proving can be utilized for properties like these. Data properties are beyond the scope of this paper. Control–Data properties involve both control and datapath signals. An example of a property of this type would be that "if an *xor* command is issued, and no exceptions occur, then in 4 time steps, a location will contain the result of the *xor* operation on the correct arguments".

In our framework we are concerned with control properties and control–data properties. However, in the latter case we are not interested in verifying correct function implementation, rather we make sure that the decoding and data transfer takes place in the correct fashion.

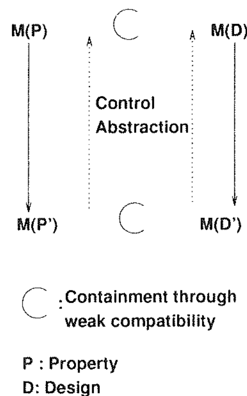

Figure 2. Theoretical Aspects of the Verification Process

Since ours is a conservative abstraction it suffers from the problem of false negatives. This means that a property may fail on the abstracted model while in reality it holds on the original machine. However, we do not have the problem of false positives, which means that we cannot have properties holding on the ECFM that do not hold on the actual machine. For universal type control properties (for example, properties 1, 2, 3, 4 in Table 1) the inference is direct. This means that if the property holds on the ECFM it is guaranteed to hold on the original machine. However, in the case of failure we need to investigate the counterexample space. For control–data properties we need to do further processing in both the case of failure and the case of success. The main question is whether the witness (in the case of success) or counterexample (in the case of failure) can be mapped back to the original machine or not, that is, whether the abstract sequence is possible on the original design. The problem of mapping back is a difficult one, and there are no simple solutions. We have had some success using manufacturing test generation (ATPG) techniques. For witnesses the processing stops when a witness is successfully mapped to an execution path in the original machine. For counterexamples, the examination must be exhaustive. This can be seen as a process of eliminating false negatives.

3 Results

We applied our techniques to the Viper microprocessor. The Viper is a processor designed for safety-critical applications, and has 33 primary inputs, 53 primary outputs, 251 D–flip flops and approximately 40 instructions. Full reachability analysis is not possible on a circuit of such complexity. This means that traditional formal methods such as model checking, language containment or explicit enumeration based techniques are no longer applicable. The power of abstraction and decomposition has to be exploited.

In Table 1 we present verification results for some control properties of the Viper microprocessor. The list of properties verified is given below.

1. Reset state not reachable from all states.
2. Reset state reachable from all states after a reset line is inserted.
3. Execution always returns to fetch state within 6 cycles. This property is not true if an exception is raised.
4. The above property is true if an exception is not raised.
5. There exist instructions for which the machine goes in the halt state and stays there. A witness was generated in 3.45s and mapped back to a sequence applicable to the original machine in 4.07s– an illegal instruction was found to be the cause of halting.

The first column specifies the index of the property, while the second column provides the extraction time for the ECFM of the Viper. Since the properties being verified are control properties, no abstraction is necessary on the state machines specifying those control properties. Columns 3 and 4 give the maximum number of BDD nodes and the

Table 1. Property Verification on the ECFM of the Viper

Property	Extraction Time	# BDD Nodes	Time to build BDDs	Time to Find		Total Time
				WC_1	WC	
1	3.85s	852	0.37s	0.0s	0.0s	4.46s
2	9.07s	1432	0.23s	0.0s	0.20s	9.51s
3	3.72s	1170	0.37s	0.0s	0.70s	4.40s
4	3.77s	1268	0.38s	0.0s	0.70s	4.46s
5	3.78s	1338	0.39s	0.0s	0.80s	4.50s

Table 2. Instruction Set verification on the ECFM of the Viper

Property	Extraction Time		# BDD Nodes	Time to build BDDs	Time to Find		Total Time
	Design	Prop.			WC_1	WC	
1	9.05s	1.91s	3479	0.64s	0.02s	0.21s	12.02s
2	10.78s	2.31s	3120	0.87s	0.05s	0.39s	14.93s
3	3.66s	0.31s	1527	0.40s	0.0s	0.12s	4.73s
4	3.96s	0.39s	1369	0.52s	0.0s	0.11s	5.22s

time to build the BDDs, while columns 5 and 6 give the time needed to obtain the sets of weakly 1–compatible states as well as the set of weakly compatible states (fixed point). Finally, column 7 gives the overall time for the verification of each property.

In Table 2 we present results on instruction set verification for the Viper. In performing instruction set verification our intent is to verify the decoding logic and the data transfer operations rather than the correct function implementation. The following instructions are verified.
1. XOR Instruction with No–op loop.
2. ADD instruction with No–op loop.
3. Comparison Instruction.
4. WRITE Instruction.
In these cases our abstraction techniques are also applied on the property machine, and so the time for this process is given in column 3. As an additional experiment we inserted a bug in the decoder of the Viper (causing an XOR instruction to be decoded as an AND instruction). We generated a counterexample for this case in 3.85 seconds, and mapped it back to the original machine in 3.2 seconds.

Note that verification of these properties would not have been possible without the utilization of the abstraction techniques. Additionally, the verification is over all possible data values, all possible initial states and for all possible input sequences. This is very critical for the validation of circuits where the flow of control is more complicated, as is the case, for example, in pipelined processors with interrupts, exceptions, dynamic scheduling, etc.

4 Conclusions

We have presented a novel abstraction technique which applies the same abstraction to both the property and the model being checked. This allows designers to verify properties involving data as well as control for complex designs, hitherto impossible to verify. In some cases, if a property is true in the abstract model, it is guaranteed to be true in the original design; in other cases, the abstract witness or counterexample needs to be mapped back to the original design. We are now studying techniques for the mapping back problem which will be able to handle more complex designs.

References

[1] Y. Hoskote, J. Abraham, and D. Fussell. Automatic verification of temporal properties written as state machines in vhdl. *Proc. of the Sixth Great Lakes VLSI Symposium*, 1995.

[2] Y. V. Hoskote. Formal techniques for verification of synchronous sequential circuits. *Ph.D. Dissertation, UT Austin ECE Dept.*, 1995.

[3] Y. V. Hoskote, D. Moundanos, and J. Abraham. Automatic extraction of the control flow machine and application to evaluating coverage of verification vectors. *Proc. ICCD*, pages 532–537, 1995.

[4] R. Kurshan. Computer–aided verification of coordinated processes–an automata theoretic approach. *Princeton University Press*, 1994.

[5] M. McFarland. Formal verification of sequential hardware: A tutorial. *IEEE Transactions on Computer-Aided Design*, 12, No. 5:633–654, May 1993.

[6] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.