HOOVER: Hardware Object-Oriented Verification

Mostafa M. Aref
Information And Computer Science Dept.
aref@dpc.kfupm.edu.sa

Khaled M. Elleithy
Computer Engineering Department
elleithy@dpc.kfupm.edu.sa

King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia

Abstract

*In this paper a new formal hardware verification approach based on object oriented techniques is presented. The HOOVER system (**H**ardware **O**bject **O**riented **VER**ification) is described. A cell library of different hardware components has been implemented as classes. Components in the cell library are described at the transistor level, gate level, and logical level, and functional level. The verification of a CMOS inverter and 1-bit CMOS adder using HOOVER is given in the paper.*

1. Introduction

The design process is a transformation between different specifications (Figure 1). An input algorithm may be specified using a specific algorithmic specification language. Architecture may be specified using a realization specification language. The role of design can be viewed as a transformation process between the algorithm specification language and the realization specification language. The objective of any design procedure is to produce an architecture that correctly implements the required behavior subject to a given set of constraints on area and timing. It is very expensive to fabricate a design before verifying the functional correctness of the design. There are two approaches for verification; simulation and formal verification. Simulation is efficient for small size architectures where it is possible to exhaustively run the simulator. Formal verification is suitable for large size architectures.

A verification methodology is formal if it satisfies the following characteristics [1]:

- There is a formal framework to describe the architecture.
- There is a formal technique to prove that implementation and specifications are equivalent without physically construct or simulate the design.
- It is possible to manipulate and study the design's performance without the physical implementation.

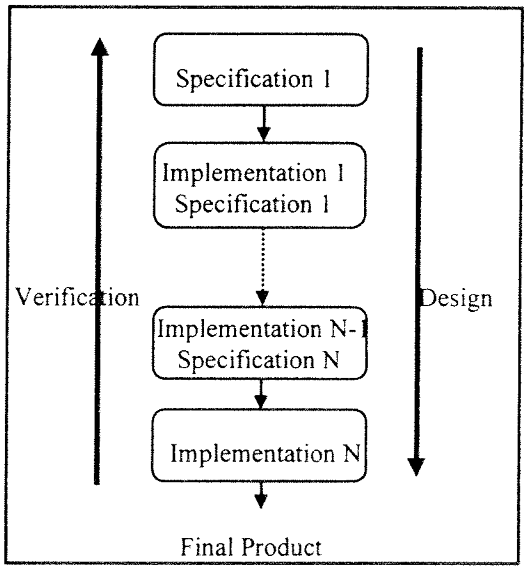


Figure 1: A Hierarchical Representation of Design and Verification

The heart of any formal verification methodology, then, is the availability of a formal specification language where formal proofs can be driven. Logic is one of the most widely used specification languages for verification. First order logic has been used in a number of systems [2-4]. Higher order logic has been used in a number of applications [5-8]. Joyce [7-8] used HOL system to verify a microprocessor. Temporal logic is an appropriate

0-8186-8409-7/98 $10.00 © 1998 IEEE

approach for specifying timing characteristics of a design. Temporal logic has been successfully used for verification in [9-10]. Productions systems have been used in formal verification [11].

Object Oriented Paradigm (OOP) have been proven successful in software engineering due to its reusability which increases design productivity. Object oriented techniques provide a number of features, discussed in section 2, which fit hardware verification. In [12], Nebel argued the suitability of object oriented techniques in hardware system design. Several proposals where introduced to adopt OOP in hardware description languages such as VHDL [13-15]. In [16], Schumacher discussed the problems in these approaches.

In this paper we are introducing a novel approach for hardware verification based on OOP. The **HOOVER** (**H**ardware **O**bject **O**riented **VER**ification) system is introduced. In section two an overview to object oriented techniques is given. In section 3 the HOOVER system is discuused. Examples using the HOOVER for verification are given in section 4. Finally, section 5 offers conclusions and future extensions.

2. Object Oriented Techniques

The characteristics of an object-oriented language are:

- **abstraction**: is a higher level, more intuitive representation for a complex concept;
- **encapsulation**: is the process whereby the implementation details of an object are masked by a well-defined external interface;
- **inheritance**: where classes may be described in terms of other classes by use of inheritance;
- **polymorphism**: is the ability of different objects to respond to the same message in a specialized manner; and
- **dynamic binding**: is the ability to defer the selection of which specific message-handlers will be called for a message until run-time.

In HOOVER, a Cell Class Library is build based on the description of hardware components. These components are organized in a hierarchy that allows **inheritance** of common attributes between different components. The behavior description is the only accessible attribute of these components (i.e. **encapsulation**). The hierarchy structure of the Cell Class Library allows the component of common attributes to be on the top level (i.e. **abstraction**). Dealing with these components description would be only through messages. The same message may be passed to two different components that results two different responses (i.e. **polymorphism**).

3. HOOVER System

HOOVER is a hardware object oriented verification system. The circuit structural and behavioral descriptions are HOOVER inputs. The circuit description would be one or a combination of different hardware descriptions. These descriptions include transistors, gates, logical, functional, and module descriptions. HOOVER has a class hierarchy contains Cell Class Library. The Cell Class Library contains a predefined set of hardware components. It consists of five subclass libraries represent the five level of hardware descriptions. These subclass libraries are Transistor-level Class Library (**TCL**), Gate-level Class Library (**GCL**), Logic-level Class Library (**LCL**), Function-level Class Library (**FCL**), Module-level Class Library (**MCL**). The block diagram of HOOVER is shown in Figure 2.

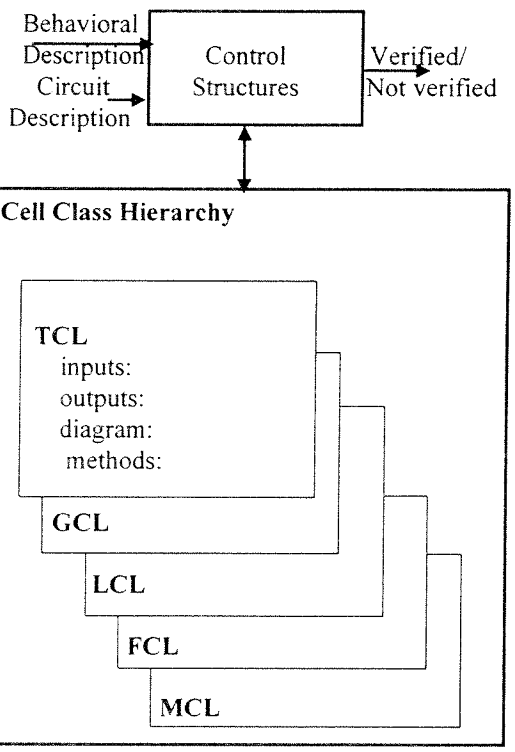


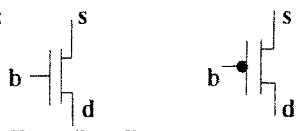
Figure 2: The Block Diagram of HOOVER

Each cell class contains several properties describe its inputs, outputs and behavior. The behavior of the cells is described through methods that may be inherited to (or override by) the subcells. Some examples of these classes are shown below.

Examples of Classes

Connection class
inputs: i **outputs:** o
diagram:

methods: Comm(i,o)
send o = i

Transistor class
inputs: b, s **outputs: d**
diagram:


methods: Trans(b,s,d)
send **d** depends on the logical level of the base **b** and **s**

An n-type transistor is an instance of the transistor class. The method is overridden by:

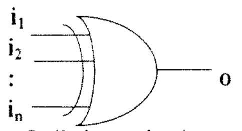
NTrans(b,s,d)
send **d** = **s** if logical level of the base **b** is 0

An p-type transistor is an instance of the transistor class. The method is overridden by:

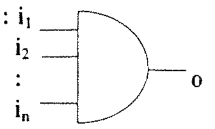
PTrans(b,s,d)
send **d** = **s** if logical level of the base **b** is 1

Inverter class
inputs: i **outputs:** o
diagram:

methods: Inv(**i**,**o**)
send **o** = not **i**

OR class
inputs: $i_1, i_2, \dots i_n$ **outputs:** o
diagram:


methods: Or($i_1, i_2, \dots i_n$,o)
send **o** = $i_1 \vee i_2 \vee \dots \vee i_n$

AND class
inputs: $i_1, i_2, \dots i_n$ **output:** o
diagram:


methods: And($i_1, i_2, \dots i_n$,o)
send **o** = $i_1 \wedge i_2 \wedge \dots \wedge i_n$

The cell class library is a hierarchy structure. The top class represents a simple hardware component; connection class. In the same cell library, there exist different transistor level components. As we move down the hierarchy, more specific hardware components are described. These components may inherit some characteristic from the upper ones.

4. EXAMPLES

Several circuit examples are used as input to HOOVER. These examples include transistor/gate circuit (e.g. inverter), transistor/logic circuit (e.g. 1-bit full adder) and logic/function circuit (n-bit full adder). Here, two examples are presented. The first one is a CMOS inverter. The second one is a 1-bit full adder.

Example 1:
A CMOS inverter consists of power, ground, p-transistor and n-transistor components as shown in Figure 3.

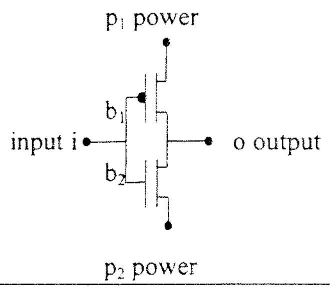


Figure 3: A CMOS Inverter

The circuit description of the inverter is described as a set of instances from the exiting classes as follows.

$\text{Ntransistor}_1(b_1,p_1,o)$	$\text{Ptransistor}_1(b_2,p_2,o)$
$\text{Connection}_1(i,b_1)$	$\text{Connection}_2(i,b_2)$

The behavioral description of the inverter is as follows.
output = invert(input)

For i = 0, HOOVER sends 0 to Connection_1 and Connection_2. Their methods $\text{Comm}(i,b_1)$ and $\text{Comm}(i,b_2)$ send b_1 and b_2 equal 0 to Ntransistor_1 and Ptransistor_1. The method of Ntransistor_1, $\text{NTrans}(b_1,p_1,o)$, sends $o = p_1$. That means the output equals 1.

Similarly, for i = 1, HOOVER sends 1 to Connection_1 and Connection_2. Their methods $\text{Comm}(i,b_1)$ and $\text{Comm}(i,b_2)$ send b_1 and b_2 equal 1 to Ntransistor_1 and Ptransistor_1. The method of Ptransistor_1, $\text{PTrans}(b_2,p_2,o)$, send $o = p_2$. That means the output equals 0.This show that the circuit description verifies the behavioral description.

Example 2:

A 1-bit CMOS full adder consists of power, ground, 12 p-type transistors and 12 n-type transistors, as shown in Figure 4.

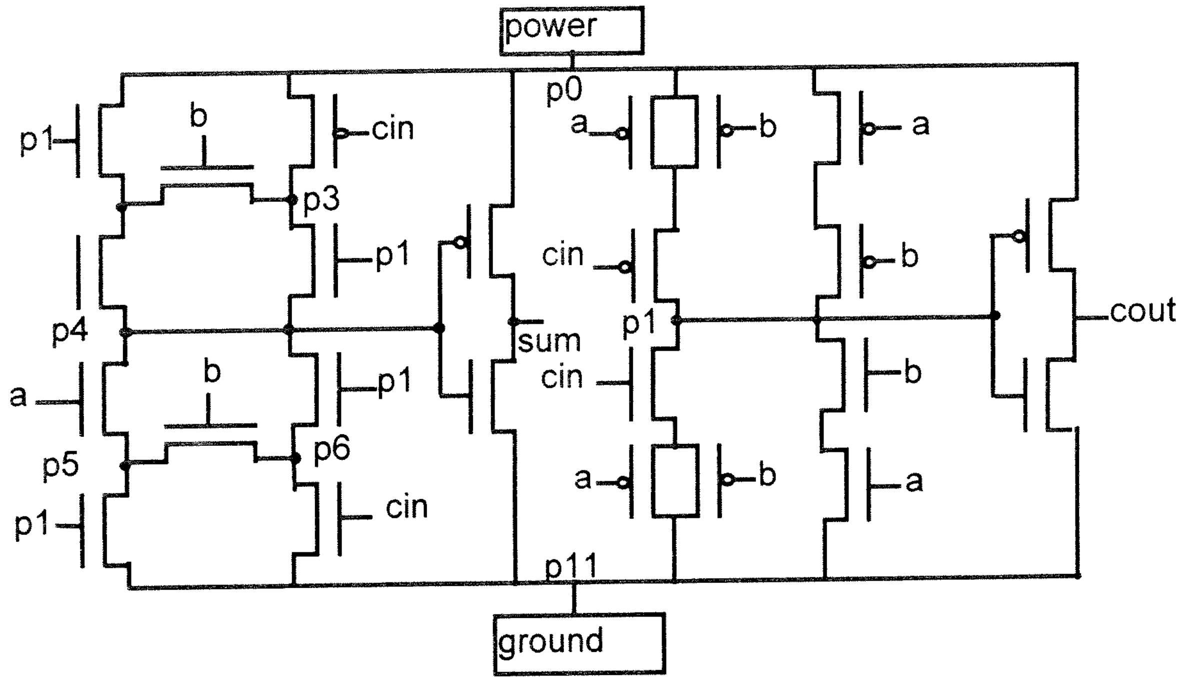


Figure 4: A 1-bit CMOS Full Adder

The circuit description of the adder circuit is as follows.

$\text{Ptransistor}_1(p_1,p_0,p_2)$	$\text{Ptransistor}_2(c_{in},p_0,p_3)$
$\text{Ptransistor}_3(b,p_2,p_3)$	$\text{Ptransistor}_4(a,p_2,p_4)$
$\text{Ptransistor}_5(p_1,p_3,p_4)$	$\text{Ptransistor}_6(a,p_0,p_7)$
$\text{Ptransistor}_7(b,p_0,p_7)$	$\text{Ptransistor}_8(p_4,p_0,\text{sum})$
$\text{Ptransistor}_9(a,p_0,p_8)$	$\text{Ptransistor}_{10}(c_{in},p_7,p_1)$
$\text{Ptransistor}_{11}(p_1,p_0,c_{out})$	$\text{Ptransistor}_{12}(b,p_8,p_1)$
$\text{Ntransistor}_1(p_1,p_4,p_6)$	$\text{Ntransistor}_2(b,p_5,p_6)$
$\text{Ntransistor}_3(p_1,c_{out},p_{11})$	$\text{Ntransistor}_4(p_1,p_5,p_{11})$
$\text{Ntransistor}_5(c_{in},p_6,p_{11})$	$\text{Ntransistor}_6(c_{in},p_1,p_9)$
$\text{Ntransistor}_7(a,p_4,p_5)$	$\text{Ntransistor}_8(b,p_1,p_{10})$
$\text{Ntransistor}_9(p_4,\text{sum},p_{11})$	$\text{Ntransistor}_{10}(a,p_9,p_{11})$
$\text{Ntransistor}_{11}(b,p_9,p_{11})$	$\text{Ntransistor}_{12}(a,p_{10},p_{11})$

The behavioral description of the adder circuit is given in a logical level as follows.

$$sum = a \oplus b \oplus c_{in}$$

$$c_{out} = (a \cap b) \cup (a \cap c_{in}) \cup (b \cap c_{in})$$

For a = 0, b = 0, c_{in} = 0

The following classes receive inputs:

$\text{Ntransistor}_2(b,p_5,p_6)$, $\text{Ntransistor}_5(c_{in},p_6,p_{11})$, $\text{Ntransistor}_6(c_{in},p_1,p_9)$, $\text{Ntransistor}_7(a,p_4,p_5)$, $\text{Ntransistor}_8(b,p_1,p_{10})$, $\text{Ntransistor}_{10}(a,p_9,p_{11})$, $\text{Ntransistor}_{11}(b,p_9,p_{11})$ and $\text{Ntransistor}_{12}(a,p_{10},p_{11})$

They send the following:

$p_5 = p_6$, $p_6 = p_{11}$, $p_1 = p_9$, $p_4 = p_5$, $p_1 = p_{10}$, $p_9 = p_{11}$, $p_9 = p_{11}$, and $p_{10} = p_{11}$

which means that $p_4 = 0$, $p_1 = 0$, $p_9 = 0$. Then the class $\text{Ntransistor}_3(p_1,c_{out},p_{11})$ send $c_{out} = p_{11}$ which means c_{out} equals 0. The final HOOVER's output indicates that the behavioral description is equivalent to the circuit description.

5. Conclusions

The verification of large-scale systems is no more a straightforward process that can be completely achieved using traditional approaches of simulation. In this paper we are describing a novel formal verification approach based on object-oriented paradigm. A cell class library that supports the HOOVER has been analyzed at different specification levels. Examples of the transistor, gate, logical, functional cell class libraries have been implemented in HOOVER 1.0 using Java. To illustrate the idea, a number of small size examples have been presented in this paper.

Currently, we are working to complete the cell class library in HOOVER to be able to test complex examples. A module level class library will be implemented. The new class library will support specifications at the module level. Further work will be done for supporting timing verification.

Acknowledgments

The authors wish to acknowledge King Fahd University of Petroleum and Minerals for utilizing the various facilities in preparation and presentation of this paper.

References

1. Elleithy, K. M. "Formal Hardware Verification of VLSI Architecture Current Status and Future Directions," Fifth International Conference on Microelectronics, Dhahran, pp. 197-201, Dec. 1993.
2. Uehara, T., et al., "DDL Verifier and Temporal Logic," Proc. CHDL 83: IFIP 6th Int'l Symp. Computer Hardware Description Languages and their Applications, Pittsburgh, May 1983, pp. 91.
3. Eveking, "Formal Verification of Synchronous Systems," Formal Aspects of VLSI Design: Proc. 1985 Edinburgh Conf. VLSI, G. J. Milne and P. A. Subrahmanyam, eds., North Holland Publishing, Amsterdam, 1986, pp. 137-151.
4. Hunt, W. A., "FM8501: A verified Microprocessor," IFIP WG 10.2 Workshop, From HDL Descriptions to Guaranteed Correct Circuits Design, North Holland Publishing, Amsterdam, Sept. 1986, pp. 85-114.
5. Hanna, F. K. and Daeche, "Specification and Verification of Digital Systems Using Higher order Logic," IEE proc., Vol. 133, Pt. E, No. 5, Sept. 1986, pp. 242-254.
6. Gordon, M. J. C., "Why High-Order Logic is a Good Formalism for Specifying and Verifying Hardware," Formal Aspects of VLSI Design: Proc. 1985 Edinburgh Conf. VLSI, G. J. Milne and P. A. Subrahmanyam, eds., North Holland Publishing, Amsterdam, 1986, pp. 153-177.
7. Joyce, J., Birtwistle, and Gordon, M. "Proving a Computer Correct in Higher Order Logic," Tech. Rept. No. 100, Computer Laboratory, The Univ. of Cambridge, Cambridge, England, 1986.
8. Joyce, J., "Formal Verification and Implementation of a Microprocessor," VLSI Specification, Verification, and Synthesis, Birtwistle, G. and Subrahmanyam, P.A., eds., North Holland, Amsterdam, The Netherlands, 1988, pp. 371-378.
9. Bochmann, G. V., "Hardware Specification with Temporal Logic: An Example," IEEE Trans. Computers, Mar. 1982, pp. 223-231.
10. Fujita, M., et al., "Logic Design Assistance with Temporal Logic," Proc. CHDL 85: IFIP 7th Int'l Symp. Computer Hardware Description Languages and their Applications, Aug. 1985, pp. 129-137.
11. Elleithy, K. M. and Aref, M., "A Production Based System for Formal Verification of Digital Signal Processing Architectures," Twenty-Seventh Annual Asilomar Conference on Signals, Systems and Computers, Pacific Grove, California, pp. 1618-1622, Nov. 1-3, 1993.
12. Nebel, W. and Schumacher, G., "Object-Oriented Hardware Modeling - Where to Apply and what are the objects?," Proc. of the Euro-Dac 1996 with Euro-VHDL 96, IEEE Computer Society Press 1996.
13. Glunz, W., et al., "System Level Synthesis " in Michel, P. and et al. (eds): The Synthesis Approach to Digital System Design, Kluwer, pp. 221-260, 1992.
14. Zippelius, R. et al. "An Object Oriented Extension to VHDL," Proceedings of the VHDL-Forum, Spring'92 Meeting, 1992.
15. Willis, J., et al."A Proposal for Minimally Extending VHDL to Achieve Data Encapsulation and Multiple Inheritance. " Proceedings of the VHDL International User's Forum, 1994.
16. Schumacher, G. and Nebel, W., "Inheritance Concept For Signals in Object-Oriented Extensions to VHDL." Proc. of the Euro-Dac 1995 with Euro-VHDL 95, IEEE Computer Society Press 1995.