Proceedings of IEEE Asia Pacific Conference on Circuits and Systems '96
November 18 ~21, 1996, Seoul, Korea

VERIFICATION STRATEGY IN PROVER

Khaled M. Elleithy and Mostafa A. Aref

College of Computer Science and Engineering
King Fahd University of Petroleum and Minerals
Dhahran 31262, Saudi Arabia
Email: Elleithy@ccse.kfupm.edu.sa
Fax: 966 3 860 3059

Abstract

In this paper, the verification strategy of PROVER environment is presented. The PROVER system (**PRO**duction system for hardware **VER**ification) is implemented using CLIPS (C Language Integrated Production System). PROVER is a rule-based framework for formal hardware verification. The environment supports verification at different levels of hardware specification. Verification strategy is illustrated in this paper using Carry select adder as a case study.

1. Introduction

With the current advances in Very Large Scale Integration (VLSI) it is impossible to produce correct designs using ad-hoc methods[1]. It is essential to detect as many errors as possible before any fabrication stage. There are two approaches for verification; simulation and formal verification. Simulation is efficient with 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 has the following characteristics[1]: there is a formal framework to describe the architecture, there is a formal technique to prove that the implementation is an isomorphism of the specification without physically construct or simulate the design, and it is possible to manipulate and study the design's performance without the physical implementation.

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. Automated Theorem Provers are efficient in proving the correctness in large scale architectures where the proof of correctness is done automatically. In this paper we are introducing a novel approach for formal verification based on a production system. **PROVER** (**PRO**duction system for hardware **VER**ification)[2-4] is a production based system for formal hardware verification implemented using CLIPS[5] (C Language Integrated Production System). In section 2, PROVER system is introduced. The verification strategy used in PROVER is discussed in section 3. Case studies using PROVER for verification are presented in section 4. Finally, section 5 offers conclusions and discussions.

2. PROVER Envionment

PROVER is a production based system for formal hardware verification[2-4]. PROVER's input has two components for the verifiable circuit: implementation description and behavioral description. The implementation description would be one or a combination of different hardware descriptions. These descriptions include transistors, gates, logical, functional, and module descriptions. PROVER has a knowledge base consists of a formal Cell Library and Rules. Cell library contains a predefined set of hardware components. It Consists of five sub libraries to represent the five levels of hardware descriptions. These sub libraries are Transistor-level Library (**TL**), Gate-level Library (**GL**), Logic-level Library (**LL**), Function-level Library (**FL**), and Module-level Library (**ML**). The block diagram of PROVER is shown in Figure 1. The incremental approach is used in developing PROVER. In this approach, a subset of the domain is considered first and a prototype is built. Then this prototype is expanded to the other subsets of the domain.

3. Formal Verification Strategy

A formal verification strategy based on transformation rules between different levels is used. Verification rules are required to prove that a given specification at level X is equivalent to specification at level X+1. The following verification rules are implemented:

- **T-to-G rules**: transform from transistor level to gate level,

- **G-to-L rules**: transform from gate level to logical level,
- **L-to-F rules**: transform from logical level to functional level, and
- **F-to-M rules**: transform from functional level to module level.

The rules define possible transformation from one level to another. Also, they reflect the semantic of each level description. Some examples of these rules are shown below.

3.1 Formal cell libraries

Any synchronous/asynchronous digital system can be verified using pre-verified constructs from the cell library. Verified components at any level are added to the cell library in the appropriate level. For ease of expandability, verified components are made modular so that they can be used for verifying modules of different word length, e.g., a definition of a Carry Look-Ahead adder can be used for verification in cases of 16, 32, and 64 bits. PROVER supports the following libraries:

A- **TL library**: the transistor level library contains specifications of components and transformation rules at the transistor level, e.g., an Inverter composed of *n/p* transistors.

B- **GL library:** the gate level library contains specifications of components and transformation rules at the gate level, e.g., a full adder composed of *and/or/not* gates.

C- **LL library**: the logical level library contains specifications of components and transformation rules at the logical level, e.g., a full adder represented using two logical functions for *sum* and *carry*.

D- **FL library:** the function level library contains specifications of components and transformation rules at the function level, e.g., an n-bit full adder composed of *n* full adders.

E- **ML library:** the module level library contains specifications of components and transformation rules at the module level where larger components are constructed from smaller components, e.g., ALUs and microprocessors.

3.2 Input Format

Components in the circuit level library are described as follows:

(Component-name {connections*})

where,

Component-name: name of the component

Connections: connections of the component, e.g., (ntrans a P4 P5) means that the component ntrans is connected to a, P4 and P5 terminals.

3.3 Example

A CMOS Inverter consists of power, ground, p-transistor and n-transistor components is shown in Figure 4. The circuit description of the Inverter is as follows:

```
(power p1)
(ptrans input p1 output)
(ntrans input output  p2)
(ground p2)
```

The behavioral description of the Inverter is as follows:

```
(Inverter       INPUT i
                OUTPUT O).
```

To formally verify that the given circuit description implies the functional description of an Inverter PROVER activate the TL library in the rule based system to transform the circuit description to:

```
(power p1)
(ground p2)
~input   ==> p1 = output
input    ==> output = p2
```

Then PROVER uses the connection rules to transform the circuit description to:

```
~input   ==> output = 1
input    ==> output = 0
```

Using the Inverter rule, PROVER indicates that the behavioral description is implied by the given circuit description.

4. Case Study

4.1 Carry Look Ahead (CLA) Adder

A carry lookahead (CLA) technique is used to speed up the carry propagation in a ripple carry adder. A carry look ahead adder consists of carry-generate-propagate unit (CGP), summation unit, and carry-look-ahead (CLA) unit. The PROVER's input consists of the modules functional description and interconnections of the modules. It is required to prove that the given circuit realizes the following function:

Si = sum (Ai, Bi, Ci-1)
Ci = carry (Ai, Bi, Ci-1)

Proof Strategy

The Function-level Library (**FL**) is activated to prove the correctness of the look ahead adder. The following rules are fired.

(a) Carry Generate and Propagate Rule

If there is a CGP unit of size n
then delete the CGP unit description and express the carry generate and carry propagate in terms of the inputs.
This rule is fired n times generating 2n logical expressions (G1 .. Gn and P1 .. Pn).

(b) Carry Look Ahead Unit Rule

If there is a CLA unit of size n
then delete the CLA unit description and express the carry outs in terms of the carry in, carry generate and carry propagate.
This rule is fired n times generating n logical expressions which include C1 .. Cn.

(c) Summation Unit Rule

If there is a summation unit of size n
then delete the summation unit description and express the sums in terms of the carry out and carry propagate.
This rule is fired n times generating n logical expressions which include S1 .. Sn.

(d) Full Adder Rule

If there is a set of logical expressions representing a full adder
then delete these logical expression and express the sum and carry in terms of the inputs as full adder.
This rule is fired n times generating n full adder expressions.

Analysis

The previous proof can be easily extended to n-bit carry look ahead adder through the general definitions of the used modules. Table 1 shows PROVER results for verifying 8, 16, 24 and 32 bits carry look ahead adders. The run time of 4-bit carry look ahead adder is considered the time unit.

4.2 Carry Look Ahead Adder based on 4-bits blocks

A two level carry look ahead adder can be implemented using 4-bit block of carry look ahead blocks. The PROVER's input consists of the modules functional description and the modules' interconnections. It is required to prove that the given circuit realizes the following function:

Si = sum (Ai, Bi, Ci-1)
Ci = carry (Ai, Bi, Ci-1)

Proof Strategy

The Function-level Library (**FL**) is activated to prove the correctness of a carry look ahead adder based on 4-bit blocks. In addition to using carry look ahead adder rules, PROVER uses the following rules:

(a) Block Carry Look Ahead Rule

If there is a BCLA unit of size n
then delete the CGP unit description, express the carry out in terms of the carry generate and carry propagate inputs, and add a block carry generate and propagate unit.
This rule is fired 3 times generating n logical expressions which include Ci .. Ci+2

(b) Block Carry Generate and Propagate Rule

If there is a BGP unit of size n
then delete the CGP unit description
and express the carry out in terms of the carry-generate-propagate functions.
This rule is fired n/4 times generating n/4 logical expressions which include Ci+3.

Analysis

The previous proof can be easily extended to n-bit carry look ahead adder based on 4-bit blocks through the general definitions of the used modules. Table 2 shows PROVER results for verifying 8, 16, 24 and 32 bits carry look ahead adders based on 4-bit blocks. The run time of 4-bit BCLA is used as the time unit.

5. Conclusion

Verifying large scale systems is no more a straight forward process that can be completely achieved using traditional approaches of simulation. In this paper the verification strategy used in PROVER has been discussed. A Carry lookahead adder has of 8, 16 and 32 bits has been used to demonstrate the strategy. Results show that a 32 bit Carry lookahead adder can be verified functionally in few seconds.

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, "Formal Hardware Verification: of VLSI Architectures: Current Status and Future directions" in

Proc. 5th International Conference on Microelectronics. Dhahran, Saudi Arabia. pp. 197-201, Dec. 1993
(2) Elleithy, K. M. and Aref, M. A., "A Production Based System for Formal Verification of Digital Signal Processing Architectures" in Proc. Twenty-Seventh Annual Asilomar Conference on Signals, Systems and Computers. Pacific Grove, California. Nov. 1-3, 1993.
(3) Aref, M. A. and Elleithy, K. M., "PROVER: A Production System for Formal Hardware Verification" in Proc. Fifth International Conference on Microelectronics. Dhahran, Saudi Arabia. pp. 210-213. Dec. 1993
(4) Elleithy, K. M. and Aref, M. A., "A Rule-based Approach for High Speed Adders Design Verification" in Proc. 37th Midwest Symposium on Circuits and Systems. Lafaytte, Louisiana. August 1994.
(5) CLIPS Reference Manual, Version 5.1. Software Technology Branch, Lyndon B. Johnson Space Center. September 1991.

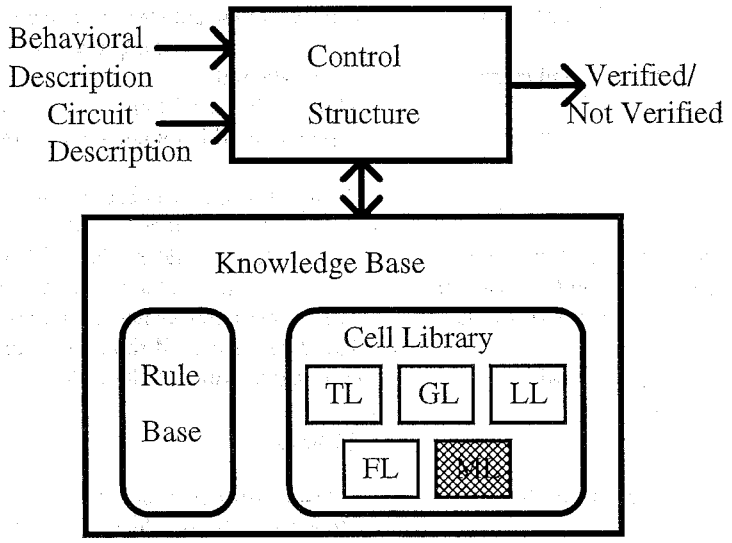

Figure 1: PROVER's Block Diagram.

Size (bits)	max # of facts	# of rules fired	run time
4-bit (unit)	23	16	1
8	43	32	2
16	83	64	5
24	123	96	10.5
32	163	128	18

Table 1. Results for 8,16, 24 and 32 bit Carry Look Ahead Adders

Size (bits)	max # of facts	# of rules fired	run time
4-bit	25	15	1
8	43	30	1.5
16	83	60	5
24	123	90	9.5
32	163	120	16

Table 2. Results for 8,16, 24 and 32 bit Carry Look Ahead Adders 4-Bit Blocks