

Formal Verification of DSP VLSI Architectures: A Tutorial

Khaled M. Elleithy & Muhammad A. Hummaigani

Computer Engineering Department
King Fahd University of Petroleum and Minerals
Dhahran 31262, Saudi Arabia

Abstract—In this tutorial paper the area of formal verification of DSP VLSI architectures is presented. The paper discusses the following topics: production systems, formal logic, the equational approach, and the signal flow graph approach. Each approach is explained using one or more of the current available systems.

1. INTRODUCTION

The production of DSP Very Large Scale Integration chips is very expensive and time consuming. It is very important to design correct DSP VLSI architectures before any production phase. Simulation is a popular method that is used to prove the correctness of systems for specific sets of inputs and outputs. With the current advances in VLSI circuits, there is no testing procedure that is capable of accomplishing an exhaustive simulation of complex circuits. Formal verification is another approach to prove correctness of a DSP VLSI architecture without going through exhaustive testing.

The design process is a transformation between specifications at different levels of abstractions. An algorithm may be specified using a specific algorithmic specification language. A DSP VLSI architecture may be specified using a realization specification language. The role of formal verification is to prove using a mathematical framework that the realization is equivalent to the specification. The components of any formal verification framework are: a formal algorithmic specification language, a formal realization specification language, and a mathematical framework.

A verification methodology is formal if it is [1]:

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

In this paper an overview formal verification of DSP VLSI architectures is presented. In section 2, production systems are discussed. In section 3, two examples of higher order logic are presented. In section 4, the equational approach is discussed. In section 5, signal flow graph approach is presented. In section 6, a discussion on the current and future directions is presented.

2. PRODUCTION SYSTEMS

Production systems are efficient in proving the correctness in large scale architectures where the proof of correctness is done automatically. A novel approach for formal verification based on a production system has been presented in [2-3]. The **PROVER (PROduction system for hardware VERification)** is implemented using CLIPS (C Language Integrated Production System)[4]. CLIPS is written in C to support the goal of high portability, extensibility, low-cost and ease of integration with external systems [5]. CLIPS, as a production system [6], provides pattern-directed control of a problem-solving process.

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. As shown in Figure 1, 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 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 other subsets of the domain.

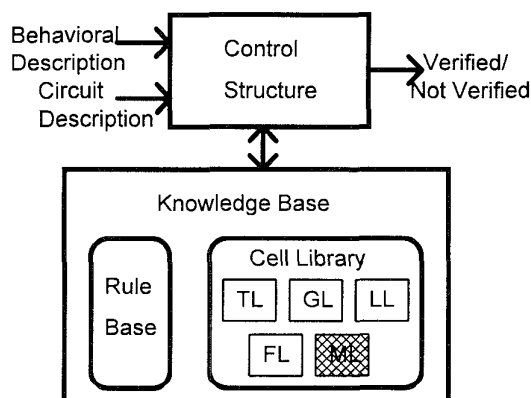


Figure 1. PROVER's Block Diagram.

A formal verification methodology 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.

Any DSP VLSI 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 Conditional Sum adder can be used for verification in cases of 16, 32, and 64 bits.

3. FORMAL LOGIC

Logicians have proposed different formalisms to represent the branch of knowledge concerned with truth and inference. The most important ones in the area of formal verification are: first order logic, higher order logic, and temporal logic.

In first order logic the following connectives are used: negation (\sim), conjunction (\cap), disjunction (\cup), and then (\Rightarrow). The existential (\exists) and universal (\forall) quantifiers are used.

In the higher order logic the domains of variables range over functions and predicates, and functions can take functions as their arguments.

Temporal logic is used to specify properties of all possible execution times that may evolve from the present state. The following operators are defined:

- Henceforth(\diamond): means that an assertion is true in the present state and all future states.
- Eventually(∇): means that an assertion will be true at some state in the future (possibly the current) state, but not necessarily remains true.
- Next(\circ): means that an assertion will be true in the next instant of time. This operator introduces the concept of discrete time and a transition that occurs between subsequent time instants.
- While(Ξ): a statement like $A \Xi B$ means that if B is true at the present, then A is true at the present and remains true as long as B remains true.

A. Cathedral-II Environment

Verkest, Claesen, and Man developed a methodology for the correctness proof of parametrized module generators using Boyer-Moore logic[10,11]. The methodology is illustrated with the proof of a specific module generator for the ALU module, based on the Mead-Conway ALU[12] of the CATHEDRAL-II environment [7,8,9].

While modules' layouts are generated starting from the textual description in the module generator file, the description used for the formal verification of the module is written in Boyer-Moore logic. Therefore, these two descriptions have to be equivalent. An automatic translation of the Module Generator (MGE) text to Boyer-Moore text is not feasible because of:

1. In case of the ALU, the module generator describes the structure directly in terms of the leaf cells (general function blocks, even carry blocks, ...). This does not allow performing a proof using Boyer-Moore without first proving the behavior of the various functional bars (propagate, kill, carry and result bar). These bars form a necessary intermediate level of hierarchy for the proof which can not be found in the module generator code.
2. Even when an intermediate level of hierarchy is not needed in the proof (in simple module generators such as an adder), automatic translation would result in an unstructured description in Boyer-Moore. This does not enable the construction of proofs in a reasonable efficient way.

As a result, the structure part of the module generators is translated manually into Boyer-Moore code. The manually translated Boyer-Moore code has to be equivalent to the original code. Figure 2 shows the approach used for the verification steps.

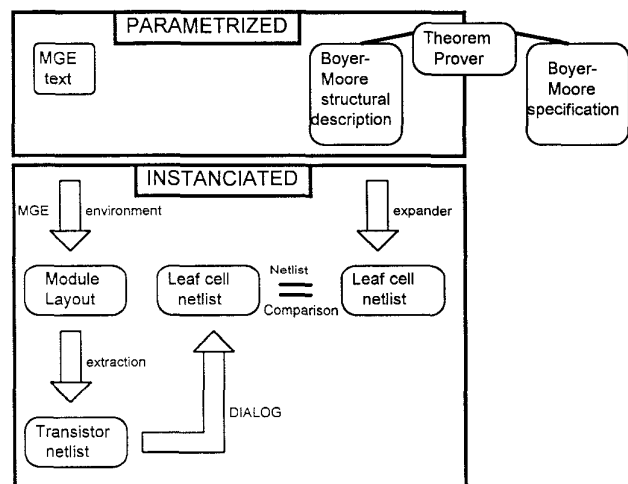


Figure 2. Verification steps in CATHEDRAL II

As shown in Figure 2, two proofs must be carried out:

1. By means of the Boyer-Moore theorem prover, the parametrized structural description in Boyer-Moore corresponds to the parametrized specification of the module in Boyer-Moore.
2. The proof that the structural part of module generator, in the MGE text which describes the module in a parametrized way, corresponds to the parametrized Boyer-Moore structural description.

While the first proof is done only once for each module, the second proof is done for every generated instance of each module. Every time a specific module instance is generated for the use in CATHEDRAL II environment, we obtain a layout from the MGE environment. From this layout, a transistor net list is extracted. Then, an expert system called DIALOG [13], which can recognize leaf cells, is used to get a net list of leaf cells from the transistor net list. On the other hand, the Boyer-Moore module text is expanded after instansiating the parameters with particular values. This expansion results in a net list of leaf cells.

This net list of leaf cells is then compared with the net list of leaf cells obtained from the layout. The comparison is done using a net list comparison program such as WOMBAT [14]. If discrepancies occur, they must have resulted from errors in the translation of the MGE description to the Boyer-Moore description.

B. PIRAMID Silicon Compiler

Many DSP devices can be specified concisely by describing their functional behavior. There is, however, a large gap between specification and silicon implementation. In the prototype silicon compiler for Digital Signal Processing PIRAMID[17], the SILAGE[9] behavior description is mapped onto an architecture of EXecution Units (EXU's), busses connecting the EXU's, and a controller for regulating the bus-traffic and making EXU's work together.

Kalker introduced an approach to improve the specification language SILAGE using (HOL) system in order to obtain correct designs in PIRAMID.

The time instances in the SILAGE level consists of cycles in the implementation level. The number of cycles needed to implement one SILAGE time-step depends critically on the number of busses and EXU's chosen for the realization. So, the speed requirements can not be solved solely on the SILAGE level. The cleverness of the PIRAMID scheduler, the building blocks of the layout generating back-end, and the technology used also have their large influence in this respect.

The syntactic form of a SILAGE program determines, to a large extent, the architecture realized on silicon. Also,

meeting the speed and / or area restrictions and requirements sometimes can not be solved by the PIRAMID system only. Therefore, transformations on the SILAGE program are needed, provided that these transformations have to be behavior preserving. This can be achieved by utilizing HOL system for the purpose of producing a better (with respect to specification/verification) version of SILAGE, called SILAGE+.

4. EQUATIONAL APPROACH

Narendran and Stillman described a method that implements an equational approach for theorem proving of first-order predicate calculus [18] to verify an image processing chip, called the Sobel chip.

The equational approach involves taking a set of first-order formulae, performing necessary operations to remove existential quantifiers, then translating the resulting formulae into equivalent set of polynomials over a Boolean ring whose operators are "exclusive or" and "and", in which the atoms appearing in the original formulae make up the intermediates of the polynomial equations. The original set of formulae is satisfiable if and only if the resulting system of polynomials has a solution.

Both the structural and behavioral definitions are specified in a first-order predicate calculus with equality. The approach is refutational in nature. The statements describing the structure are assumed to hold and the statements describing the behavior are assumed not to hold and an attempt is made to reach a contradiction.

The behavioral description of the chip is obtained from the high-level statements in the chip document [19]. The structural description of the chip supplied in the form of a VHDL code. The first-order statements used in the theorem prover are then extracted from the VHDL code.

5. SIGNAL FLOW GRAPH APPROACH

Genoe, Claesen, Verlind, Proesmans, and Man developed a methodology for the correctness proof of CATHEDRAL-II [7] circuits that is based on Signal Flow Graph (SFG)-Tracing[20]

SFG-Tracing is a general multi-level design verification methodology that aims at bridging the gap between higher level specifications down to lower level implementations. It uses the concept of Ordered Binary Decision Diagrams (OBDD's)[21].

Any design, starting from a behavioral description, passes through several levels of abstraction with growing amount of information [20], before obtaining a final layout. As shown in Figure 3, these levels are the Signal Flow Graph (SFG) level, the behavioral Register Transfer (bRT) level, the structural Register Transfer (sRT) level, and the transistor switch (SWITCH) level.

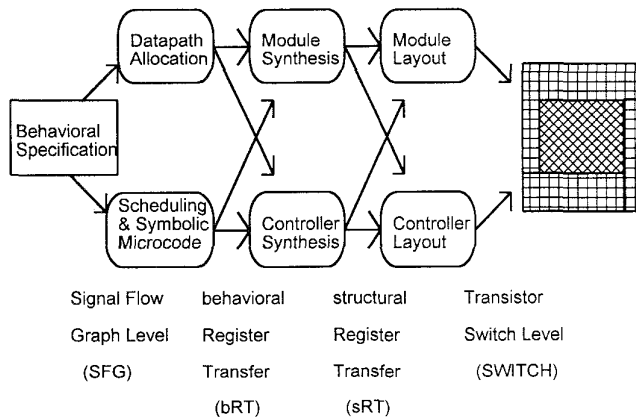


Figure 3. SFG Verification Approach Levels

The goal is to prove the correctness of an implementation (a transistor switch net list extracted from final layout) at the SWITCH level and a specification (behavioral SILAGE language [9] description) at SFG level. Intermediate levels, however, are used to get additional information. Furthermore, in case of bugs or errors, correctness between other levels has to be checked.

The final verification between different abstraction levels is done by symbolic evaluation and Boolean comparison. Two Boolean expressions are identical if their OBDD representations are identical for the same variable ordering.

Figure 4 shows the integration of the SFG-Tracing verification methodology in the CATHEDRAL II CAD environment.

Starting from a SILAGE description, the basic SFG is derived. This basic SFG is then partitioned into several pSFG's. This partitioning is caused by the fact that real designs are, in general, too complex to verify. This means that the specification (the basic SFG) has to be split up into several partitions. The input and output signals of each partition are called reference signals.

In order to verify the behavior of a single partition, mapping functions are generated during the high level synthesis to express the correspondence of the reference signals in space and time on the implementation. The optimal partitioning depends on the abstraction level.

The interface signals between all these pSFG's (the reference signals) and the corresponding mapping functions are generated by the high level synthesis tools.

In addition, a script has to be generated for the successive verification of each partition. The script generator collects the pSFG's, reference signals, and mapping functions and prepares the evaluation and verification process in a systematic way.

A transistor net list of the circuit is extracted from the final layout using CAD tools. This transistor net list can then be modeled by switch-level analysis.

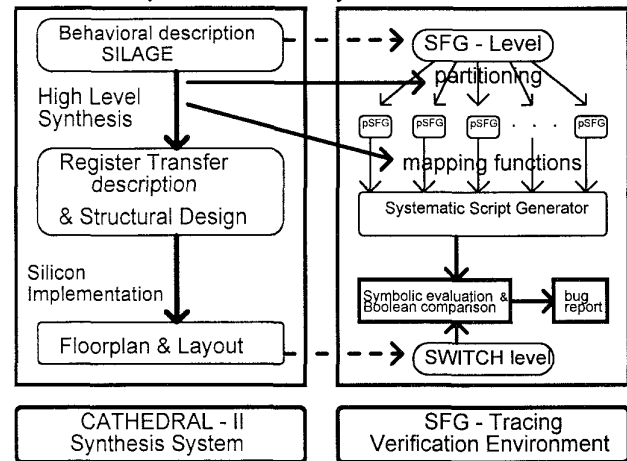


Figure 4. SFG Verification Integrated with CATHEDRAL-II

In case of inconsistencies during the verification of specific pSFG's on the analyzed circuit, a bug report is generated during the symbolic evaluation and Boolean comparison process, by which it is possible to indicate precisely where the eventual errors took place.

6. DISCUSSION

Currently, the area of formal hardware verification is of intensive research and development. Although major activities are still in academia, a number of industries are entering this domain. It is interesting to note that some industries have incorporated formal verification tools into their Computer Aided Design(CAD) systems.

The Royal Signals and Radar Establishment (RSRE) of the British Ministry of Defense applied formal verification techniques to design a microprocessor suitable for real-time control of safety-critical systems. The Viper microprocessor has been specified, designed and verified by RSRE using automated theorem provers[15]. The formal methods developed by RSRE have been endorsed by the industrial manufacturers of the VIPER.

Regularly held conferences devote part of their programs to the active research in this area. We mention here: Symposium on Computer Hardware Description Languages (CHDL), International Conference on Computer Design (ICCD), International Conference on Computer Aided Design (ICCAD), Design Automation Conference (DAC), and European Design Automation Conference (EDAC). In addition, the major part of a number of international workshops is devoted to formal hardware verification research.

Although research in the area of formal hardware verification is very promising, academia and industry still have a long way to go before formal methods of hardware specification and verification become common and widely used[18].

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 Architectures: Current Status and Future directions," *5th International Conference on Microelectronics*, Dhahran, Saudi Arabia, 1993, pp. 197-201.
- [2] Aref, M. A. and Elleithy, K. M., "PROVER: A Production System for Formal Hardware Verification," *Fifth International Conf. on Microelectronics*, Dhahran, Dec. 1993, pp. 210-213.
- [3] Elleithy, K. M. and Mostafa Aref, M. A., "A Production Based System for Formal Verification of Digital Signal Processing Architectures," *Twenty-Seventh Asilomar Conf. on Signals, Systems & Computers*, Pacific Grove, California, Nov. 1-3, 1993.
- [4] *CLIPS Reference Manual*, Version 6, Software Technology Branch, Lyndon B. Johnson Space Center, June 1993.
- [5] Mettrey, "A Comparative Evaluation of Expert System Tools," *IEEE Computer*, Vol. 24, No. 2, Feb. 1991, pp. 19-31.
- [6] George F. Luger, William A. Stubblefield, *Artificial Intelligence and the design of Expert System*, The Benjamin/Cummings publishing Company, 1989.
- [7] Man, H., Rabaey, J., Six, P., and Claesen, L., "Cathedral-II: A Silicon Compiler for Digital Signal Processing," *IEEE Design and Test of Computers*, December 1986, vol. 3, no. 6, pp. 73-85.
- [8] Man, H., "Evolution of CAD tools Towards Third Generation Custom VLSI Design," *Revue Phys. Appl.*, vol. 22, January 1988, pp. 31-45.
- [9] Hilfinger, P., "SILAGE: a High Level Language and Silicon Compiler for Digital Signal Processing," *Proceedings IEEE 1985 Custom Integrated Circuits Conference*, Portland, May 1985, pp. 213-216.
- [10] Boyer, R. and Moore, J., *A Computational Logic*. Academic Press, 1979.
- [11] Hunt, W. and Brock, B., "FM8501: A verified Microprocessor," *Technical Report 47*, Inst. for Comp. Science, Univ. of Texas, Austin, Feb. 1986.
- [12] Mead, C. and Conway, L. *Introduction to VLSI Systems*. Addison-Wesley, 1980, pp. 150-154.
- [13] Man, H., Bolsens, I., Meersch, E., and Cleynenbreughel, J., "DIALOG: An Expert Debugging System for MOS VLSI Designs," *IEEE Transactions on Computer Aided Design*, CAD-4, no. 3, June 1985, pp. 303-311.
- [14] Spickelmier, R. and Newton, A., "WOMBAT: A New Netlist Comparison Program," *Digest of Technical Papers ICCAD-83*, pp. 170-171.
- [15] Cohn, A., "A Proof of Correctness of the Viper Microprocessor: The First Level," *Technical Report no. 104*, University of Cambridge Computer Laboratory, 1987.
- [16] Gordon, M., "A Proof Generating System for Higher Order Logic," *Technical Report no. 103*, University of Cambridge Computer Laboratory, 1987.
- [17] Meerbergen, J. and Man, H., "A Real Silicon Compiler for the Design of Complex Integrated Circuits for Digital Signal Processing," *Philips Technical Journal* 44, no. 7, February 1989.
- [18] Kapur, D. and Narendran, P., "An Equational Approach to Theorem Proving in First-Order Predicate Calculus," *In Proceedings of the 9th Intl. Joint Conference on Artificial Intelligence*, Los Angeles, California, 1985.
- [19] Vasanthavada, N., Baker, R., and Kanopoulos, N., "A Monolithic Image Edge Detection Filter," *In Proceedings of the IEEE Custom Integrated Circuits Conference*, 1987.
- [20] Claesen, L., Genoe, M., Proesmans, F., Verlind, E., and Man, H., "SFG-Tracing: a Methodology for the Automatic Verification of MOS Transistor Level Implementations from High Level Behavioral Specifications,," *In Lecture Notes in Computer Science*, ed. Springer Verlag, 1991.
- [21] Bryant, R., "Graph Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, August 1986, pp. 667-691.
- [22] A D&T Roundtable, "Formal Verification is it Practical for Real-World Design," *IEEE Design and Test of Computers*, Dec. 1989, pp. 50-58.