Logic Verification Methodology for PowerPC™ Microprocessors

Charles H. Malley

Motorola Inc.
9737 Great Hills Trail
Austin, Texas 78759

Max Dieudonné

IBM Corporation
11400 Burnet Road
Austin, Texas 78758

Abstract—The PowerPC logic verification methodology is a general purpose approach suitable for a large class of chip designs that can exceed five million transistors in size. Several validation techniques are integrated into an automated logic verification strategy. The success of this methodology has been demonstrated by realizing three PowerPC microprocessor chips that were functional the first time.

I. Introduction

Logic verification is a crucial element in the success of a microprocessor chip design. The goal is to verify that the functional specification matches the switch-level implementation. Traditional techniques rely on massive amounts of event driven simulation at both the gate and switch levels. This type of simulation suffers from slow performance (number of simulation cycles per second) and poor functional coverage [1], [2], [3]. This situation can be drastically improved by applying specific point solutions to a variety of verification problems within an automated environment. This technique demonstrates the benefits received from exploiting the use of hierarchy within the microprocessor chip design to enable a divide and conquer strategy. Boolean comparison is used to verify logic models represented at the gate and block levels. For switch-level circuits, exhaustive or ATPG patterns are simulated to demonstrate equivalence.

In the following section, the design methodology will be introduced to define some of the concepts used in the overall verification flow. Next, the logic verification strategy will be presented and details provided for each of the methods employed for the different design types. Finally, elements of the automated environment will be highlighted and explored.

PowerPC is a trademark of International Business Machines Corporation.

32nd ACM/IEEE Design Automation Conference ®

II. Elements of the Methodology

The functional specification for each chip design is defined in terms of a Register Transfer Level (RTL) description and is the primary (golden) model used to determine functional accuracy. This model, which is code-compiled and verified using a cycle simulator, is state equivalent with the physical implementation; that is, each storage element found in the physical implementation must correspond to a component specified within the RTL description. Simulation vectors are generated using a Random Test Pattern Generator (RTPG) [4] and are supplemented with designer specified functional patterns. These patterns are simulated on the RTL description and the results are compared against the expected response from an architectural model which is presumed to be correct.

The RTL model is constructed to be hierarchically similar to the actual partitioning used in the chip floorplan. The logic design of the PowerPC microprocessor is partitioned within functional boundaries for which various design strategies are applied. One such partition, known as a Random Logic Macro (RLM), is a control logic structure which instantiates functional components. RLMs are implemented using techniques that range from automated logic synthesis [5], [6] to full custom design.

A fundamental concept within the PowerPC microprocessor design methodology is the use of library elements called building blocks. A building block is the lowest level element that can be referenced within the RTL model hierarchy. It is a self contained unit which has a functional specification, a logic schematic implementation, and a physical layout. The circuit design and construction of building block libraries takes place in parallel with the development of the RTL model. Building block circuits represent simple functions such as Boolean logic gates, latches, and multiplexors as well as more complex datapath elements including adders, shifters, rotators, and arrays. A building block may be used just once or it may be used a number of times to take advantage of reuse. Building blocks are assembled in the construction of RLMs and are instantiated in higher level hierarchical structures which make up the chip design. A *structure* refers to a level of hierarchy within the RTL model that represents a

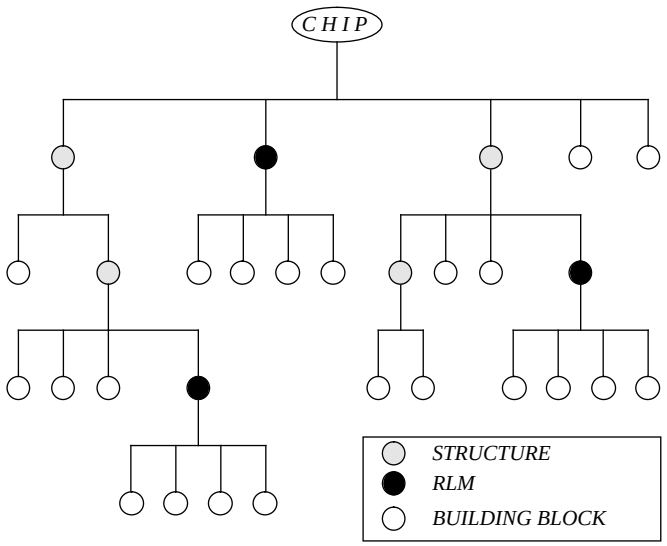

Fig. 1. Chip Logic Design Hierarchy

physical boundary in the chip floorplan which is above the building block and RLM levels. A structure may reference an RLM, building block or another structure. A structure is also used to represent the top (highest) level in the chip design hierarchy. Fig. 1 shows a representative chip design hierarchy which illustrates the relationships between the different types of levels.

The logic schematic of each building block may be hierarchical and will consist of objects called cells and transistors, the transistor being the atomic primitive element in the hierarchy. A cell is a node in the schematic hierarchy that contains transistors and/or other cells and is represented by a pictorial representation called a symbol. The logic schematic for the building block is used to represent two distinct views. One is called the gate-level model and the other is called the switch-level model. For those cells in the hierarchy that do not contain transistors, the gate-level model and switch-level model refer to the same schematic.

A separate gate-level schematic model is manually constructed for any cell within the building block schematic hierarchy that contains transistors, including cells that contain references to cells and transistors. The gate-level view is a Boolean equivalent logic representation of the stuck-at fault model for these transistor circuits. Gate-level models are used by Automatic Test Pattern Generation (ATPG) tools to create production test patterns. The ATPG tools understand a small set of primitive logic functions or test primitives (AND, OR, NAND, NOR, INVERT, TRI-STATE, LATCH, RAM). The switch-level model refers to the cells within the building block schematic hierarchy which represent the transistor network that implements the functional model. Fig. 2 shows an example building block schematic and the resulting gate-level and switch-level representations

III. Verification Strategy

Since the RTL model is used to determine functional verification, the goal of the logic verification methodology is to determine equivalence between the functional specification (RTL model) and the switch-level implementation. The switch-level implementation is represented by the combination of the structure interconnect within the chip level netlist, the building block interconnect contained within RLMs, and the logic schematic which represents the building block's circuit design. This verification is accomplished hierarchically in three steps:

1) Determine the equivalence of the three building block level models: functional, gate-level, and switch-level.

2) Verify that the RLM's logic equations match the synthesized or customized technology mapped implementation.

3) Verify that the interconnect of RLMs and building blocks within the RTL description matches the interconnect of these blocks described by the chip level netlist.

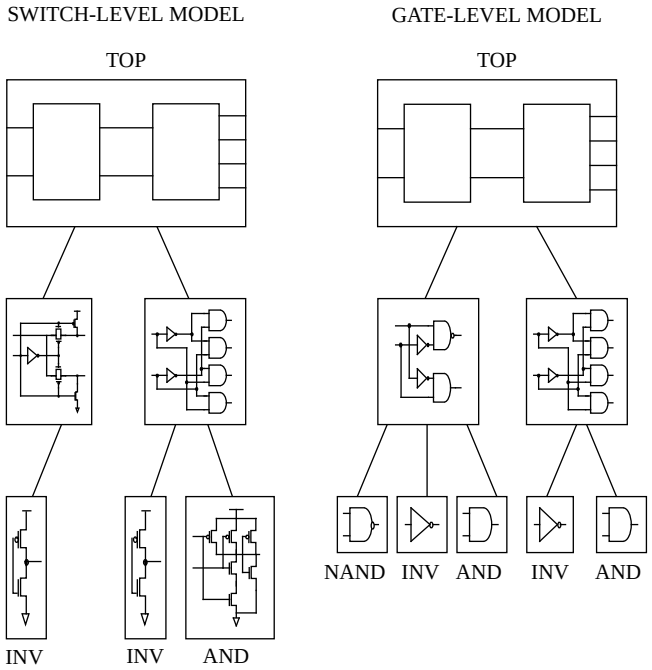

Fig. 2. Building Block Schematic Hierarchy

IV. Formal verification using boolean comparison

A Boolean Equivalence Checker known as BEC is a design verification tool which compares static Boolean networks for logical equivalence using Shannon's expansion [7] to build binary decision diagrams (BDDs) [8], [9]. In general, given enough time and space, BEC can compare any two networks of logic; but, because Boolean comparison is an NP complete problem, there is no guarantee that a comparison will be completed within the specified time and space constraints.

BEC is used to verify building blocks, RLMs, and structures. In the case of building blocks and RLMs, the simulation model (RTL) is translated into a technology independent Boolean logic network. During this translation, heuristics are applied to transform complex or high level RTL expressions into a logic structure containing low-level primitive logic functions.

The first application, known as building block BEC, verifies that the functional model and gate-level model are equivalent. The second application, RLM BEC, verifies that the functional specification for RLMs matches the technology mapped netlist created either by logic synthesis or by custom design. The third application, called structure BEC, verifies that the interconnect within the upper levels of the hierarchy in the technology mapped chip netlist matches the interconnect specified within the RTL simulation model.

V. Building block level verification

The verification of the three views: functional model, gate-level model, and switch-level model, is accomplished using one of three methods depending on the type of building block. Method 1 makes use of the transitive property: if A=B and B=C, then A=C. In this application, the verification methodology states that if the switch-level model can be proven equivalent to the gate-level model using exhaustive pattern simulation (A=B), and the gate-level model can be proven to be equivalent to the functional model using BEC (B=C), then the switch-level model is equivalent to the functional model (A=C); that is, all three are equivalent. This method is limited to combinatorial building blocks whose transistor cells have sixteen or fewer inputs.

Method 2 is used for building blocks that contain either sequential logic or transistor cells with more than sixteen inputs. This method uses test and functional patterns to verify the switch-level and gate-level models. BEC is still applied to verify that the gate-level and functional models match.

One type of building block called an *array* represents functional units which read and write memory structures containing multiple data locations. Circuits classified as arrays include caches, registers, block address translators, and content addressable memories. This type of building block does not lend itself to verification methods 1 or 2 because it is typically very difficult to describe the actual logic inside of arrays in terms of a functional and gate level model. The cycle simulator, ATPG fault simulator, and BEC support a high level behavioral construct called a RAM primitive which models the address decoding, read/write clocking, and output data latching. When this primitive is used in the functional and gate-level models of the array, the real logic implemented in the switch-level schematic is not represented. The actual switch-level implementation is usually very complex in terms of clock timing and data presented at the outputs, whereas the RAM primitive behaves in a very simplistic manner. To solve this problem, some of the analog logic and internal timing details are not modeled at the gate-level or functional levels. For these reasons, method 3 is used to verify arrays. This method simulates a set of verification vectors on all three models (functional, gate-level, switch-level). These patterns are a collection of test patterns, functional patterns, and patterns derived from RTPG.

A. Switch-level model vs. Gate-level model

The switch-level model is compared to the gate-level model by applying patterns to the transistor network in an event driven switch-level simulation. Three types of patterns may be applied. Exhaustive patterns are used to compare the switch-level model to the gate-level model for transistor circuits which have a relatively small number of inputs. For this purpose, an exhaustive set of input patterns (2^n, where n = number of inputs) is simulated on the gate-level model in *good machine mode* to obtain the expected outputs. These patterns are then simulated on the switch-level model and the results compared to the expected outputs.

When exhaustive simulation is not suitable for a cell, two other types of patterns may be applied. Test patterns are created by executing an ATPG tool which uses the gate-level model as input. Since the gate-level model represents the actual stuck-at fault model of the circuit implementation, a concerted effort is devoted to achieve 100 percent fault coverage in order to insure testability and to obtain the most complete set of verification patterns possible. It has been shown [10], [11] that by simulating ATPG patterns on two representations of a design and comparing the results, a large class of design errors can be detected. Using ATPG to verify the switch-level and the gate-level models insures that a correct set of production test patterns will be created for the circuit. To achieve a higher degree of verification assurance, a set of functional patterns maybe simulated on the gate-level and switch-level models in addition to the test patterns.

B. Gate-level model vs. Functional model

The verification of the gate-level model is determined by comparing it to the functional model using BEC. The logic represented by the gate-level model defines the building block's Boolean relationship in terms of its inputs and outputs. The functional model is compiled and transformed into a Boolean logic network and compared to the gate-level representation. Non-Boolean functions such as tri-state drivers and latches are converted to Boolean equivalents that allow networks containing these components to be compared.

For sequential circuits, only the combinational control logic is verified, state equivalence is not determined during logic verification. Storage elements are converted to functional gates without the feedback or data hold capability. To accomplish this, feedback loops around latches are detected and automatically cut before verification. Clock signals are then identified and forced to values that allow system data or scanned data to be flushed through the latch. Tri-state nets are matched and compared independently and nodes that are driven by multiple tri-state devices are converted into a logical OR for the CMOS technology.

For array type building blocks, BEC cannot always be applied. Correct functional or fault simulation behavior sometimes requires that a different RAM primitive implementation be used in the different models. For these cases, where the functional model and the gate-level model are structurally dissimilar, logic equivalence is determined by comparing the results of pattern simulation.

C. Switch-level model vs. Functional model

As a final verification step, the functional model can be equated to the switch-level implementation directly by comparing the results of pattern simulation at the switch-level and functional level. This is only necessary when exhaustive verification of the switch-level vs. gate-level or gate-level vs. functional models is not possible. Fig. 3 shows the logic verification methodology for building blocks.

VI. RLM VERIFICATION

An RLM is a functional partition constructed out of building blocks. The functional specification is defined in terms of equations and/or specific building block instances. The RTL model and technology mapped implementation are both flattened into gate-level logic primitives as described earlier. BDDs are then built using these two flattened representations. All latch components are treated as boundaries that allow the RLM to be partitioned into multiple *segments* for verification. In this application, the latch component can be thought of as a *black box* where the verification of logic segments starts and stops. This implies that the two models are state equivalent and must demonstrate signal correspondence at all black box boundaries. The segments created by these black box boundaries are cones of logic that exist between latches or primary input ports which drive signals feeding a latch or a primary output port. Each segment in turn, defines the cone of logic for which BEC builds its BDD's for verification.

When performing a network comparison, BEC uses signal names to derive correspondence points between the two models at boundaries denoted by black boxes and the primary I/O ports. Due to the method for specifying correspondence, it is helpful if the two models being compared are algorithmically and structurally similar. The advantage of this can be seen for the case when a BEC comparison will not complete while attempting to verify large cones of logic. When this occurs, it is often possible to break a large cone of logic into several smaller cones which can be individually compared. If all of the subcones successfully compare, the equivalence of the original cones can be inferred. The nodes within the circuit where these subcones are formed are called *cutpoints*. The ability to specify cutpoints is dependent on the presence of internal equivalent points in the two larger cones of logic being compared. These points can be located in two ways. The first is for the user to specify them within the source model through the use of an attribute. The second, is to allow BEC to search for the best possible nodes to be used as cutpoints. This is best accomplished, if the two pieces of logic are either algorithmically or structurally similar. This similarity is convenient when cutpoints must be generated to complete the comparison of particularly large cones of logic.

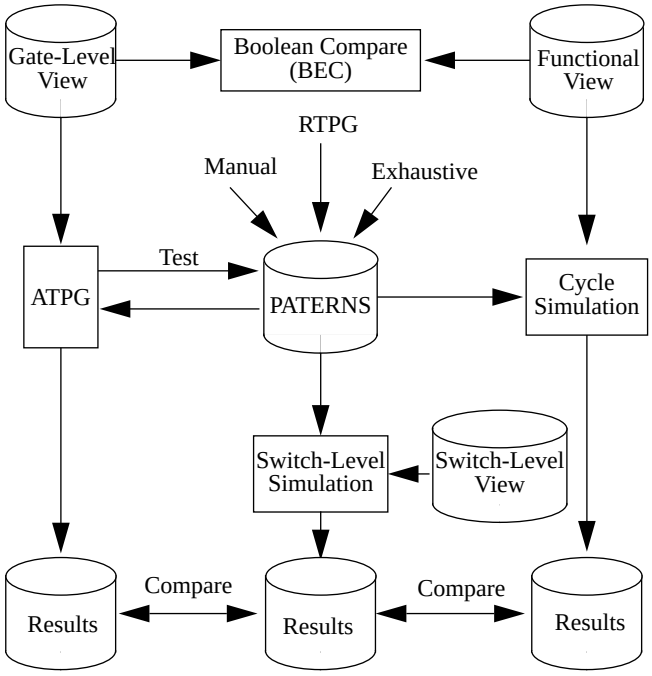

Fig. 3. Building Block Logic Verification Methodology

VII. Structure verification

BEC performs a structure verification by using the notion of hierarchy. When the technology mapped netlist for a structure is being analyzed, all structures, RLMs, and building blocks that have already been verified will be replaced by a black box. The remaining instances are expanded down to the primitive gate-level. The interface ports of each black box are preserved but the contents are not included since it is no longer necessary to expand beyond these verified boundaries. Each black box must be present in both models and their inputs and outputs must match in order to determine equivalence.

The purpose of structure BEC is to minimize the amount of time required to verify those levels of hierarchy where some or all of the instances have previously been verified. This application is similar to performing a structural LVS type function which compares a layout to a schematic, except in this case, the comparison takes place between the simulation model and the technology mapped implementation model.

VIII. Automated design environment

Significant productivity benefits are achieved by automating all the verification steps. A fundamental aspect of the design methodology schema is that all logic design entry takes place in only two places: text entry into the RTL specification and graphical entry into the schematic database. All other design data used in the logic verification flow is automatically derived using tools within the PowerPC design methodology CAD system.

The building block logic verification methodology is implemented using three design verification tools: a switch-level simulator, an ATPG tool, and BEC. These tools are integrated into the CAD system such that all data entry and verification steps can be performed through a common user interface. Fig. 4 shows the form used to perform the automated verification steps.

A schematic database is used to store the source data for the gate-level and switch-level models. Software programs access this data through a procedural interface [12] which traverses the schematic database and extracts the necessary netlist information for each of the design verification tools. These data extraction programs have the capability to traverse either the gate-level or switch-level models as required. Source data may be annotated with properties which describe specific modeling behaviors for each point tool. These properties allow: the removal of redundant faults, the annotation of timing delays, and the identification of clock signals and capacitive nets. Netlist data is placed into Unix directory structures where it is accessed during verification. During the extraction process, *metadata* files are created that contain information about the source data that allows the data management system to determine when the netlist information is out of date and must be regenerated.

The first step is to run the ATPG tool on each building block. All control files containing information about clock signals and scan timing are automatically created. The ATPG tool is executed from within the CAD system via the common user interface. The results are saved in Unix files with a synopsis written back to a user visible window. The resulting ATPG patterns are automatically translated into a format that can be applied to the switch level simulator which is also invoked from the user interface. The final step is to run Bec to determine Boolean equivalence. For this process, the gate-level and functional model representations for each building block design are instantiated into a top level design. This allows for any user defined pin constraints or *don't care* conditions to be specified. These constraints identify certain input combinations that will never occur and hence are not verified. The correct logic is automatically placed within the top level design to restrict the inputs to the building blocks in the manner requested. Bec can then be executed from the user interface with the results written both to a log file and to a user visible window. The integration of all the data creation, data extraction, and tool execution into one CAD environment through a common user interface allows for an simple and easy approach to execute the verification methodology.

Another benefit of the building block logic verification process is the notion of *audit signatures.* Audit signatures are text records that contain *metadata* pertinent to the verification of a building block. This metadata information includes return codes from the verification tools, fully qualified Unix file names, and checksums for all the design data used in the verification process. When the audit option is turned on, and the

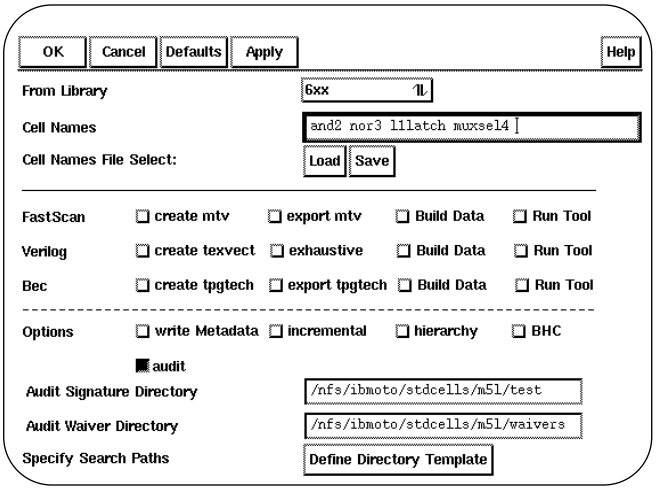

Fig. 4. User Interface for Logic Verification

verification test passes, a signature file is created. Data management utilities can take advantage of this feature by examining a collection of signatures and design data files to determine if a building block design has reached a certain quality level, and to insure that all verified data is current.

IX. Results

The actual design data from one of the PowerPC microprocessor chips was used to measure the execution times for performing the entire chip logic verification. Table I shows the time to verify the two building block libraries used on the chip. Table II contains data relating to execution time for all the RLMs on the chip. Table III shows the results from running a structure BEC at the chip level. All tests were executed on a IBM RISC System/6000™ model 550 workstation running at a clock rate of 41.6 Mhz, with an 8Kb instruction cache, 64Kb data cache, 128 Mb real and 256 Mb virtual memory. This system is rated at SPECint92™ of 36.2 and SPECfp92™ of 81.8. All times are expressed in terms of user CPU time.

X. Summary

A comprehensive suite of logic verification techniques has been presented as well as a strategy which achieves a high degree of verification assurance for microprocessor designs.

Table I
Building block logic verification

Name of library	Number of building blocks	Number of cells	Netlist create time	ATPG CPU run time	Switch-Level simulation CPU run time	BEC CPU run time
standard cell library	90	373	7 min - 11.76 sec	4 min - 17.24 sec	3 min - 59.68 sec	45 sec
datapath library	90	401	25 min - 54.06 sec	6 min - 49 sec	7 min - 4.33 sec	262 sec

Table II
RLM BEC logic verification

Number of RLMs	Number of Segments	CPU run time
29	18942	1hr - 59min - 56 sec

Table III
Structure (chip) BEC logic verification

Number of segments	CPU run time
55,055	17 h - 53 min - 34 sec

IBM and RISC System / 6000 are registered trademarks of International Business Machines Corporation.

SPECint92 and SPECfp92 are trademarks of Standard Performance Evaluation Corporation.

Using an automated verification system that employs divide and conquer methods combined with formal verification and pattern simulation results in a realistic deterministic approach to measuring equivalence between a functional specification and a switch-level implementation. Three building block verification methods were detailed which cover a variety of library element types. Boolean comparison was shown to be effective for validating RLMs and structures. As of this writing, three PowerPC microprocessor chips using the logic verification methodology described here have been fabricated. All building block circuits that have passed logic verification using the method 1 or method 2 approach (using Boolean comparison combined with exhaustive and ATPG pattern simulation) have been free of defects.

Future work needs to focus on better methods for verifying array type building blocks. Method 3 is not as robust as the other methods because it relies on pattern simulation at the top level of the design. One possible approach to solve this problem is to provide an option that will compile the functional array model into separate *sub-arrays*, where each sub-array consists of control logic and a block of RAM bit cells. Since these sub-array RAM blocks can be easily identified within the schematic hierarchy, it should be possible to correspond these memory blocks and verify that the decoded wordlines and bit lines are equivalent.

Another improvement to the methodology currently under development is the introduction of a formal verification tool called Verity [13] which symbolically proves the equivalence between a switch-level circuit and either a functional or gate-level model. This tool offers enormous potential since it eliminates the need for switch-level simulation and offers an exhaustive prove of correctness.

References

[1] R. E. Bryant, “MOSSIM: A switch level simulator for MOS VLSI,” 18th Design Automation Conference, ACM/IEEE, July 1981, pp. 786-790.

[2] R. E. Bryant, D. Beatty, K. Brace, K. Cho, and T. Sheffler, “COSMOS: A compiled simulator for MOS circuits,” *24th Design Automation Conference, ACM/IEEE*, July1987, pp. 9-16.

[3] R. E. Bryant, “A survey of switch level algorithms,” *IEEE Design and Test of Computers*, vol 4, no. 4, August 1987, pp. 26-40.

[4] A. Aharon, A. Bar-David, B. Dorfman, E. Gofman, M. Leibowitz, V. Schwartzburd, “Verification of the IBM RISC System/6000 by a dynamic biased pseudo-random

test program generator", *IBM Systems Journal*, Vol 30, No 4, 1991.

[5] J. Darringer, D. Brand, J. Gerbi, W. Joyner, L. Trevillyan, "LSS: A System for Production Logic Synthesis," *IBM Journal of Research and Development*, Vol. 28, No. 5, pp. 537-545.

[6] D. Kung, R. Damiano, T. Nix, "BDDMAP: A Technology mapper on a new converging algorithm," *29th Design Automation Conference, ACM/IEEE*, June 1992.

[7] G. L. Smith, R. J. Bahnsen, and H. Halliwell, "Boolean comparison of hardware and flowcharts," *IBM Journal of Research and Development*, Vol. 26 no 1, January 1982, pp. 106-116.

[8] S. J. Friedman and K. J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams," *24th Design Automation Conference, ACM/IEEE,* June 1987, pp. 348-355.

[9] C. Berman, L. Trevillyan, "Functional Comparison of Logic Designs for VLSI Circuits", *ICCAD*, 1989, pp 456-459

[10] M. Abadir, J. Ferguson, T. Kirkland, "Logic Design Verification via Test Generation", *IEEE Transactions on Computer-Aided Design*, Vol 7 No 1, January 1988

[11] T. Sasaki, S. Kato, N. Nomizu, and H. Tanaka, "Logic Design Verification Using Automated Test Generation," *Proc. 1984 International Test Conference,* pp. 99-94, 1984.

[12] T. J. Barnes, "SKILL: A CAD System Extension Language," *27th Design Automation Conference* 1990.

[13] A. Kuehlmann, A. Srinivasan, D. Lapotin, "Verity - a formal verification program for custom CMOS circuits" *IBM Journal of Research and Development, Vol 39*, No 1/2, pp. 149-166.