Don't care based implications for faster and improved verification

Debjyoti Paul Mitrajit Chatterjee Dhiraj K. Pradhan
Dept. of Computer Science
Texas A & M Univ.
College Station, TX-77843
debjyoti@cs.tamu.edu

Contents

1 Introduction

As the design complexity of the digital logic chips keep on increasing, maintaining design quality has become very important. The recent chaos created by a design error in the floating point ALU in the `INTEL` Pentium chip [PENT] exemplified the need for design verification. In general, verification is the methodology of ascertaining whether the design satisfys the specifications. **Logic level verification** of circuits is a simpler job in which both the specification and the design are given as logic circuits. This tool is very vital to quality as a lot of design errors can potentially be introduced at the logic level design as it is manipulated repeatedly to achieve some desired characteristics like lower gate complexity, lesser power, higher speed among others. The tools used for the manipulations cannot be guaranteed to be absolutely bug free. These need to be certified by a sentinel who checks that the design before and after the manipulation have the same functionality. This sentinel, which must be an independent entity, (i.e. it shouldn't be beset with the same bugs) is the **logic level verification tool**.

1.1 Current state-of-art in logic level verification

Logic level verification has attracted a lot of attention recently as the need for greater design integrity is felt. The current logic level verification tools can be classified by their basic methodology into two categories :

1. **Functional representation :** The functionality of the logic circuits is represented in the canonical OBDD representation and then compared [BRYA] [MALI]. The main problem is the bad behavior of the OBDD representation which often requires exponential space.

2. **Structural similarity based :** These methods target those logic designs which are similar structurally. They try to exploit the similarity in structure to check if the functionality is the same. [HANN] [BRAN] [OBDD] [FUNC]

The second class of verification methodologies is much more attractive as it is less sensitive to the size of the circuit. BBD based verification is very limited due the current memory limits. Often,incremental design steps like optimization, technology mapping and engineering changes does not change the circuit structure much. The verification of these design steps is more efficiently done by the second class of algorithms. Hence there is a strong case for structure based verification. Recently, BMDs [BMD] have been suggested as a much more concise representation of arithmetic logic circuits than BDDs but functional representation of random logic is still a problem.

1.2 Comparison of Structural methods

There are two structure based verification methods which have the best performance and distinct enough to merit seperate mentions. A brief description of each of these algorithms follows :

1. **Hannibal :** It is a recursive learning [RECL] based verification tool [HANN]. Recursive learning is a procedure which can identify all necessary assignments for a given value assignment. These necessary assignments are stored as indirect implications. The two logic circuits are connected as a miter as shown in Figure 1 Another related method is a hybrid one combining implications and OBDD representations [OBDD]. Implications aare used to identify equivalent points between the two circuits and then the circuits are partitioned by these equivalent points. The partitions are now represented as OBDDs and compared. As the

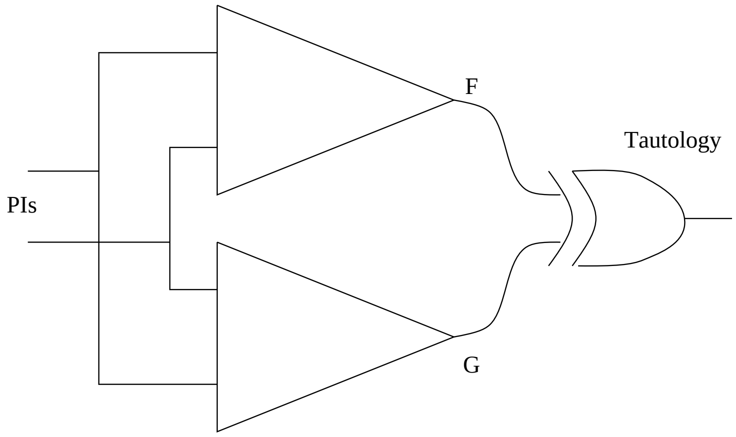

Figure 1:

two circuits are similar structurally, a value assignment in one circuit has some implications in the other circuit. The ultimate objective is to have implications between the outputs of the two circuits, to prove them functional equivalent.

2. **Brand's method :** This method [BRAN] tries to identify a corresponding line **g** in one circuit **G** for a line **f** in the other circuit **F**, such that **g** can safely replace **f** without changing the functionality of the circuit, as shown in Figure 2. The subfunction **g** is called a **permissible subfunction** for subfunction **f**. This takes into consideration controllability issues as well as the observability don't care effects. The line **g** replaces **f** in circuit **F**,reducing the size of the miter. It is well known that large miters (large redundancies) tend to give a lot of trouble to an ATPG tool. Ultimately when there are sufficient interconnections between the two circuits, it becomes easier for a ATPG tool to prove that the two circuits do not have a satisfying input vector to have a ONE on the EXOR output.

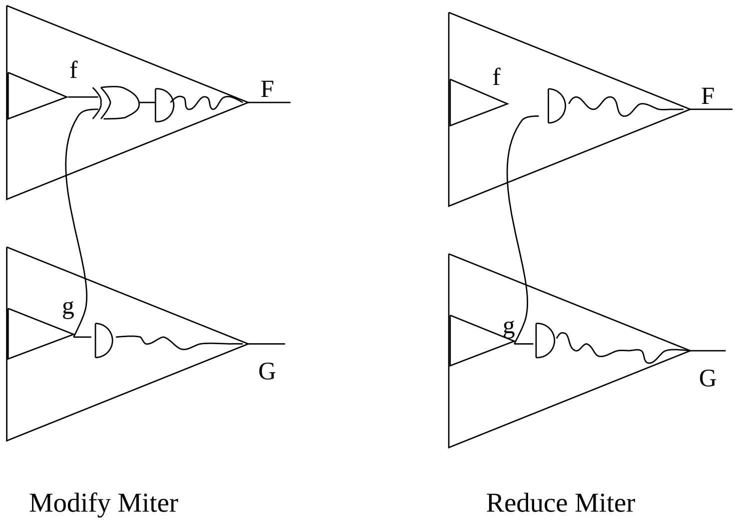

Figure 2:

Both methods are fast and complete. The only constraints being that the circuits to be compared have to be similar in structure. Importantly, circuits with the same functionality can have different structure. Both these methods require that some lines in the circuits have the similar functionality, but they use very different methods to detect this similarity.

The principal differences in these two methods are summarized below :

1. Recursive learning-based identification of the similarity of line functionality allows one way implication as well as two way implications (equivalence) to be used. Brand's method uses only equivalences. Consider two lines, A and B; if A = 1, it is found that B = 0 is always true. So B = 0 is a necessary assignment for A = 1. This is a one way implication (which does not delineate for the case when A = 0), which can be represented as:

$$(\mathrm{A} = 1) \longrightarrow (\mathrm{B} = 0)$$

Now, consider an equivalence between two nodes A and B, such that:

$$A = B$$

In actuality, this represents two one-way implications, namely:

$$(A{=}0) \longrightarrow (B{=}0)$$

$$(A{=}1) \longrightarrow (B{=}1)$$

Naturally, occurence of equivalences is much rarer than one-way implications.

2. When identifying similarities, Brand's method incorporates the concept of don't cares. When considering the functionality of two lines, if they are equal for some inputs and different for others, it is checked if the input space where the lines have different values also does not let the difference to propagate to any output. If that is the case, then the two lines can be declared equal, for all practical purposes. The equivalence of two subfunctions (lines) A and B under don't cares is denoted by:

   $$A = B \ \mathbf{mod}\ DC \quad \text{where DC denotes the Don't Care set}$$

   Recursive learning does not check for similarity under don't cares.

3. No comparison of the two methods is available yet. However, it is clear each possess advantage and disadvantage as enumerated above. To add the the confusion, Brand's method uses name similarity to identify possibly similar lines, considerably pruning the search space. When creating a new circuit by maniplutaing it, the names of lines which don't change remain intact. Though this is a logical method to use if line names can be maintained as the circuit changes, they should be compared without such heuristics.

2 Results of verifying Mentor Graphics circuits

The **Hannibal** verification system was used to verify some circuits given by **Mentor Graphics**. Previously, **HANNIBAL** had been tested on structurally very similar circuits: the original ISCAS85 benchmark with the nonredundant version of the same circuits. As redundancy removal doesn't change the structure much, these circuits were very similar. It was the first time to our knowledge that **Hanniabl** was being used to verify quite dissimilaar circuits. It waas expected taht it won't be very efficient.

2.1 Results

There were two pairs of circuits : one pair are 32 bit adders and the other 27 bit multipliers. These were different type of designs for the same function. One of the adders was a ripple-carry adder and the other a carry-lookahead adder. One of the multipliers was a carry-save-adder type and the other of a different type. Results of verifying these circuits is in Table 2.1. The machine is a Sparc5.

The verification times are quite reasonable. OBDD based methods would not have been able to verify a 27 bit multiplier [BRYA]. It is difficult to predict whether Brand's method would give a better result, as no comparisons are available. The hybrid method [OBDD] mentioned before was also tried on these circuits. Though it was able to verify the adders much faster in under 0.5 seconds, it was unable to verify the multipliers. This indicates the similarites between the ciruits was not sufficient to partition the circuits enough. Hence the OBDD size exploded.

The encouraging results prove that verification using stored implications, as **Hannibal** does, is a very powerful concept and has a wider application

Circuit name:	Add_rpl/Add_cla	Mult/Mult_csa
# of lines	734/932	3483/3373
# of Inputs	32+32	13+14
# of Outputs	32	27
Learning Time	16 seconds	2 min 10 sec
ATPG Time	1 second	1 min 16 sec
Netlist Time	0	2 seconds
Total Verification Time	17 seconds	3 min 28 sec
Recursion level (Learning)	2	2
Recursion level (ATPG)	2	4

Table 1: Results for Menotr Graphics circuits

domain than thought. These circuits were quite dissimilar and the results are promising. It looks like structural similarity which is thought to be necessary is not so important. It is functionality of the lines which should have similarity and allow implications to exist. Considering that these were independent designs and they have sufficient implications between them to enable verification, the power of implication based verification had been underestimated. It is intuitive that when two logic circuits perform the same function, internal lines in the two circuits are bound to have some similarity. **Hannibal** exploits this fact to a sufficient degree to perform fast verification.

2.2 Preprocessing of circuits

The circuits provided were in VHDL(structural) description. **Hannibal** needs the input file to be in NL description. To convert the VHDL description to NL format, a two step translation process was chosen. The VHDL description would be first translated to ISCAS85 description and then to NL description.

Initially, when the translated circuits were verified by **Hannibal**, they were reported to be different. But it was was found that this was caused by a mixup in the order the outputs were listed in the netlist in the translation process. A tool had to be written to correct the output correspondence of the translated circuits. This is basically an event driven logic simulator which found the output correspondence by elimination. Initially, it is assumed that an output in one circuit can correspond to any output of the other circuit. Then the circuits were simulated with the same input vectors and the correspondence list pruned based on the fact that outputs with different values cannot correspond to each other. The simulation process repeated till all outputs have a one-to-one correspondence. If the verification still reports the circuits to be different, they are guaranteed to be different, under the assumption that the input correspondence is correct. There was some confidence i the latter fact, as the inputs occur first in the topologically ordered netlist. There is little reason for this order to change.

After correcting the output correspondence, the circuits were found to be equivalent. The whole verification process, starting from the VHDl description to proving the circuits to be equivalent is represented stepwise as :

1. Translate VHDL (structural) description to ISCAS85 description.
2. Translate ISCAS85 description to NL description
3. Correct output correspondence
4. Verify final circuits by **Hannibal**

The input vectors used for simulating the circuits to distinguish outputs were interesting. Using random input vectors can take a lot of time. So these input vectors had to be generated more intelligently for faster convergence. They would of course depend upon the function of the circuits being considered. We note that to match two outputs without doubt, they must have a value that others don't have. For example if one output in each circuit has a 1 and all other outputs are 0. Then it can be said that these two outputs correspond to each other, guaranteed. For the adder this can be done if only one bit of the input vectors has a 1. The position of the 1 is then shifted through all the bit positions. This strategy was sufficient to distinguish the outputs of the adder. For the multiplier, to get one 1 in the output, one of the multiplicands has to have integer value 1 and the other should take the integer value of powers of 2. This is achieved by making a LSB bit 1 and shifting a 1 through the other bit pos itions. This distinguished all the outputs for the multiplier.

3 Verification using don't care based implications

Though the performance of **Hannibal** has been good, there are many improvements possible. The circuits used as a benchmark are not really representative of the industrial circuits vis-a-vis their size. The size of combinational part of sequential circuits can be larger by orders of magnitude than the benchmark circuits. There is a need for a more powerful verification method. Secondly, a lot of current optimization tools (for size, power etc) use local and global don't care sets to optimize the circuits. Verification of these optimized circuits with original circuits can be inefficient using **Hannibal**. This happens because controllability implications may not exist.

Kunz et. al. in [OPTI] have defined the D-implication, an extended definition of the normal controllability based implications. These implications are not strict controllability based implications but are based on don't cares and called D-implications. To clarify the difference between a implication and a D-implication, consider the implication :

$$(\mathrm{X} = 0) \longrightarrow (\mathrm{Y} = 1)$$

For an implication, if X = 0, it is guaranteed Y = 1. These are based strictly on controllability and also called controllability implications. Now consider a D-implication which is denoted as :

$$(\mathrm{A} = 0) \xrightarrow{D} (\mathrm{B} = 1)$$

When A = 0, it is not guaranteed that B = 1. But if B = 0, it is guaranteed that it is a Don't care condition and hence doesn't affect the output. One example of don't care condition is a observability don't care condition. In that case, if A = 0, then a B s-a-1 fault is not detectable. Hence it is safe to assign B as 1. D-implications are based on observability as well as controllability. Input don't care conditions can also be include trivially.

3.1 Example

To show that D-implications can be useful and lead to faster verification in some cases, an example is considered. The two circuits shown in Figure 3 have been extracted from the circuits c432.isc which is a part of the ISCAS85 benchmark suite. Both outputs Y1 and Y2 have the same functionality. There are three lines marked by a cross and labeled "EQ." These lines are equivalent to the corresponding lines in the lower circuit, based on controllability implications found in one level of recursion. The arrow shows a D-implication from line **f** to line **g**. The D-implication is :

$$(\mathrm{f} = 1) \xrightarrow{D} (\mathrm{g} = 1)$$

Hannibal needs two levels of recursion to prove the outputs Y1 and Y2 equivalent. If D-implications is also used in conjunction with the controllability implications, the outputs can be proved to be equivalent in one level of recursion. This has been shown in Figure 4. This example shows how D-implications can help to reduce verification time. Note that given enough resources (time in case of **Hannibal** and memory in case of BDD based methods), all verification algorithms will do the job. But using D-implications, verification can be done faster as now more implications can be found. For large circuits, if the verification can be done by using fewer levels of recursion, it can lead to big saving in time. Though finding D-implications takes time, it is expected to be much lesser than an extra level of recursion. For heavily optimized circuits based on don't cares, using D-implications will be greatly beneficial. Addition of D-implications will make implication based verification a lot more powerful.

3.2 Redefining D-implications for verification

The definition in [OPTI] is not suited for understanding how D-implications are used in verification. The new definition of D-implications is given here which makes it easier to comprehend in the context of verification. The definition is :

If the value assignment **u** *to line* **A** *makes a fault* **stuck-at-v** *at line* **B** *undetectable, the assignment,* $\mathbf{B} = \mathbf{v}$ *can safely be made under the condition* $\mathbf{A} = \mathbf{u}$*. This is a D-implication denoted as :*

$$(\mathrm{A} = \mathrm{u}) \xrightarrow{D} (\mathrm{B} = \mathrm{v})$$

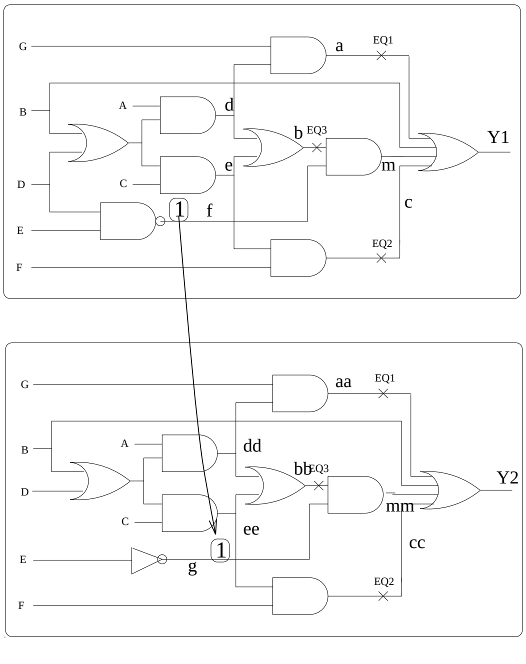

Figure 3: Example circuits for verification

ATPG phase of verification using 3 equivalences and one D-implication

Y1 = 1 (Assingment)

Justifications :

1. a=1

$\longrightarrow$ aa=1 (Equivalent)

$\longrightarrow$ Y2 = 1

2. B = 1

$\longrightarrow$ Y2 = 1 (Primary input)

3. m = 1

$\longrightarrow$ b = 1, f= 1 (Justification for AND = 1)

$\longrightarrow$ bb = 1 (Equivalent)

$\longrightarrow$ g = 1 (D-implication)

$\longrightarrow$ mm = 1

$\longrightarrow$ Y2 = 1

4. c = 1

$\longrightarrow$ cc = 1 (Equivalent)

$\longrightarrow$ Y2 = 1

Y2 = 1 $\longleftarrow$

$\boxed{\text{Y1 = 1} \longrightarrow \text{Y2 = 1}}$

Similarly Y2 =1 $\longrightarrow$ Y1 = 1 can be proved

Y1 = Y2, proved

Figure 4: ATPG phase of verification for the example

To understand why the assignment $\mathbf{B} = \mathbf{v}$ is safe to make if $\mathbf{A} = \mathbf{u}$, let us consider the two actual possibilities that either $\mathbf{B} = \mathbf{v}$ or $\mathbf{B} = \bar{\mathbf{v}}$. In the former case, there is no problem. In the latter case, it appears as if the line **B** has a **s-a-v** fault. This fault is excited as the actual circuit value is $\bar{\mathbf{v}}$ and the value injected is **v**. But by definition, this fault is undetectable, which means the difference is unobservable.

3.3 Example (Continued)

In the example, the D-implication $(\mathrm{f} = 1) \xrightarrow{D} (\mathrm{g} = 1)$ can be derived in the following process. As shown in Figure 5, the assignment $\mathbf{f} = \mathbf{1}$ is made. This assignment has three justifications. For two of the justifications, line g in the other circuit takes value 1. So it is guessed that a D-implication may exist. Figure 6 shows the one justification for $\mathbf{f} = \mathbf{1}$ which makes $\mathbf{g} = \mathbf{0}$. Trying to prove the D-implication a **g s-a-1** fault is introduced. Figure 7 shows the attempt to detect this fault. The values in the circuits in slanted font are the necessary fault propagation assignments and those in straight font are necessary excitation assignments. The OR gate inside the dotted oval has a conflict between excitation and propagation assignments. Hence this fault is undetectable. By definition, this is a D-implication:

$$(\mathrm{f} = 1) \xrightarrow{D} (\mathrm{g} = 1)$$

This is shown in Figure 8.

4 Algorithm for verification using D-implications

Motivated by the potential gains for better verification using D-implications, a tool is being developed. It will be based on the concepts of extending implications based verification presented above. However, it is not trivial to

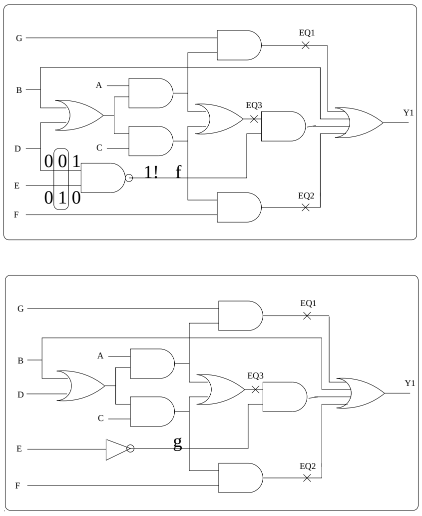

Figure 5: Justifications for a justified gate for line (f=1)

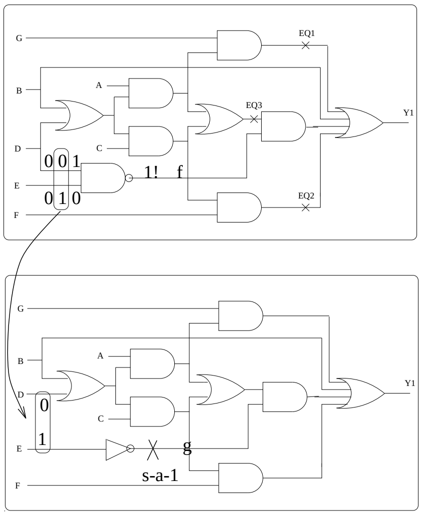

Figure 6: The conflicting justification where line g has a different value (0) than for other two justifications modeled as a stuck-at-fault

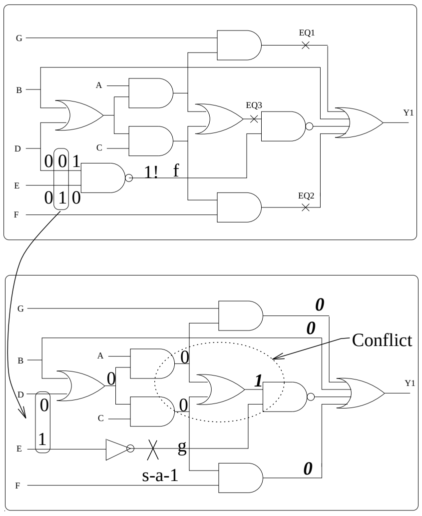

Figure 7: Trying to generate a test for this s-a-1 fault : unsuccessful

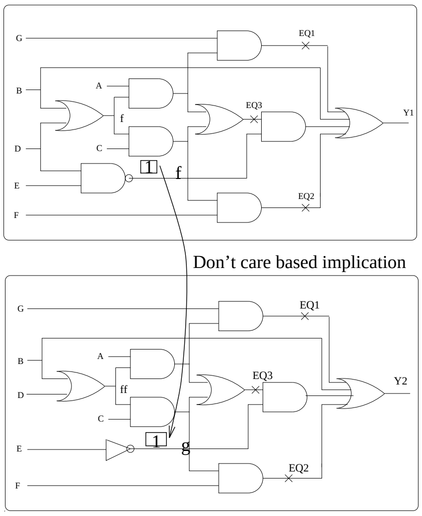

Figure 8: Don't Care based implication from line f to line g : (f=1) $\longrightarrow$ (g=1)

use D-implications and many contentious issues need to be sorted out. Some some issues are presented in this section.

4.1 Using Multiple D-implications in conjuction

Using one D-implication at a time is fine, as seen in the example and because the derivation procedure ensures that the don't care condition is such that under no condition will a faulty value propagate to the primary outputs. But when using multiple D-implications (as will have to be used to get any real improvement in verification), care should be taken. This follows from the well known result in combination ATPG that even if two stuck-at-faults may not be detectable individually, they can be detectable if present on conjuction. So, if two D-implications are derived individually, they may not exist in conjunction. By existence of D-implication, it is meant that the faulty value(s) are not detectable at the primary outputs. Therefore, a D-implication being derived needs to be checked whether it can exist in conjuction with existing D-implications. Hence, the order of choosing D-implications is important as a decision has to be made.

To look from another perspective, from the set of all possible D-implications, a subset of D-implicaitons has to be chosen which exist together. The subset depends upon the D-implications chosen. Each additional D-implication serves to contract the size of the valid subset of D-implcations. Though there is no known correlation between the size of the subset and the quality (leading to faster verification), it is important that the this set should not become exhausted.

4.2 Algorithm

The algorithm is as follows :

1. Read circuit description and create miter
2. Recursion_level_for_learning = 1
3. Derive Controllalbility implications
4. Try to verify using ATPG
5. If circuits proved equivalent or different, STOP
6. Find D-implications
7. Try to verify using ATPG
8. If ENOUGH, increment Recusion_level_for_learning
9. GOTO step 3

5 Conclusions

Modification of the implication procedure implies that all the equivalences detected by Brand's method will also be detected by this procedure, as will all one way implications. Therefore, the modified verification algorithm subsumes Brand's method. The peformance of the tool is expected to be better than existing tools like **Hannibal** and Brands' tool. Especially, for heavily optimized circuits, the performance should be better.

Acknowledgements

We are grateful to Dr. Sudhir Kadkade, Mentor Graphics, for providing the circuits.

References

[RECL] W.Kunz and D.k.Pradhan,"Recursive Learning: A New Implication Technique for Efficient solutions to CAD Problems-Test, Verification and Optimization",Transaction of CAD of Integrated Circuits and Systems, Vol13, No. 9, September 1994.

[HANN] W.Kunz,"HANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning", Proceedings of Intl. Conf. of CAD, pp 538, 1993.

[BRAN] D. Brand, "Verification of Large Synthesized Designs", Proceedings of Intl. Conf. of CAD, pp 534, 1993.

[OPTI] W.Kunz and P.R.Menon, "Multi-Level Logic Optimization by Implication Analysis", Proceedings of Intl. Conf. of CAD, pp 6, 1994.

[OBDD] S.M.Reddy, W. Kunz, and D. Pradhan, "Novel verification Framework Combining Structural and OBDD Methods in a Synthesis Environment", 32nd Design Automation Conf., pp 414, 1995.

[FUNC]
R.Mukherjee, J.Jain, M.Fujita, J.Abraham and D.Fussell,"Observations on Verification Techniques Based on Learning",Intl. Workshop on Logic Synthesis, pp 6-31, 1995.

[BRYA] R.E.Bryant, "Graph Based algorithms for Boolean Function Manipulation", IEEE Transaction on Computers, vol C-35, no. 8, Aug. 1986, pp.667-691.

[MALI] S.Malik, et. al., "Logic verification using Binary Decision Diagrams in a Logic Synthesis Environment", ICCAD, 1988, pp. 6-9.

[BMD] R.E.Bryant and Y.A.Chen, "Verification of Arithmetic Circuits with Binary Moment Diagrams", 32nd Design Automation Conf., pp 535, 1995.

[PENT] J.Markoff,"Flaw Undermines Accuracy of Pentium Chip", New York Times, November 23, 1994.