Advanced Verification Techniques Based on Learning*

Jawahar Jain Rajarshi Mukherjee† Masahiro Fujita
Fujitsu Laboratories of America, 77 Rio Robles, San Jose CA 95134

Abstract

Design verification poses a very practical problem during circuit synthesis. Learning based verification techniques prove to be an attractive option for verifying two circuits with internal gates having simple functional relationships. We present a verification method which employs a learning technique based on symbolic manipulation and which can more efficiently learn indirect implications. The method can also learn some useful functional implications. We also present a framework in which an indirect implication technique is integrated with an OBDD based verification tool. We present highly efficient verification results on some ISCAS circuits as well as on some very hard industrial circuits.

1 Introduction

Ordered Binary Decision Diagrams (OBDDs) [3] are now increasingly employed as one of the primary tools for solving the problem of combinational design verification [6, 9] and analyzing Boolean functions. However, owing to the intractability of typical Boolean function analysis problems, building OBDDs for all classes of circuits is not an easy task. We find that many industrial circuits cannot be verified, or require demandingly large resources, even if we order variables using a dynamic ordering method [13]. Therefore, it is necessary to augment OBDD-based design verification algorithms with a sophisticated methodology such that building of complete OBDDs is obviated.

Also, notice that the verification problems that typically arise in industry are often somewhat "simpler" since the two designs being compared have many simple internal relationships. For such cases verification has been speeded up by exploiting *some* of these relationships [1, 2, 4, 8]. However, the currently available techniques can be either inefficient for analyzing large designs or are not general enough to detect useful relations that are more involved; for example, implications that can relate a set of gates with another such set. Further, they lack the capability or a reasonable framework to successfully use logical relations if only a limited set of such implications could be derived between two circuits. Thus, the problem of verification can greatly benefit on either extremes: with *BDDs* as well as with *learning based techniques* if the above drawbacks can be ameliorated where possible and as a result the two methods integrated such that one can choose the benefits of both.

One possible solution is through *functional learning* (henceforth referred to as FL), a technique based on OBDDs proposed in [10]. Implications are discovered by constructing the OBDDs for various internal gates (functions) in terms of other internal gates. Through symbolic manipulation between such OBDDs and by application of the law of contrapositum and some simple and well known properties of OBDDs, we can easily learn indirect implications. Our theoretical and experimental results show that such techniques, apart from being more general, also typically perform better than other known learning techniques. Due to the capability of OBDDs to model and manipulate general functions, they can be easily used to derive more involved internal relationships as well. Importantly, FL can be naturally extended to verify logic circuits when analyzing internal relationships alone will not suffice in proving functional equivalence. In such cases, our method allows more efficient construction of an OBDD to model the given equivalence problem, leading to a more complete combinational verification method than by using only OBDDs or only (weaker) learning techniques.

In this paper we discuss various issues in detecting internal correspondences using learning techniques. We also briefly outline conceptual arguments as to why FL can be more efficient than other learning methods such as *recursive learning* [7] (henceforth referred to as RL). We then discuss how a viable framework can be created for the general verification problem based on integration of learning techniques and OBDDs (or other efficient representation schemes).

The organization of the paper is as follows: In Section 2 we review the application of learning techniques in verification. In Section 3 we discuss a basic framework of the FL technique and the types of learning that can be obtained by FL but are impossible or very difficult to get using other known learning techniques. In Section 4 we briefly discuss the learning algorithm, and in Section 5 we discuss how FL and OBDDs can be integrated. We present our results in Section 6 and some concluding remarks in Section 7.

2 Learning Techniques and Verification

Berman *et. al.* [1] proposed the first method of using internal equivalent points to establish the equivalence of two circuits. In [1] a decomposition found using the min/cut algorithm breaks down the problem of verifying the whole circuit into much smaller and simpler problems. Cerny and Mauras presented Mauras [4] made further observations to establish cross-relations between two appropriate cuts in the two circuits. Both of these were useful ideas but needed to be developed further to obtain useful verification tools.

A useful technique for utilizing internal equivalent points for logic verification was presented in [8] through the use of RL. However, in RL, or some other known learning techniques such

*The second author was supported in part by the Texas Advanced Technology Development and Transfer Program, Project #003658-469

†Also with Dept. of Electrical and Computer Engineering, University of Texas at Austin, Austin TX 78712

32nd ACM/IEEE Design Automation Conference ®

as *static learning* in SOCRATES [14], which are frequently used in ATPG tools, finding such relationships can become computationally very inefficient for large designs. The time complexity of RL is exponential in the number of levels of recursion. Unfortunately, here the "base" of the exponent can be extremely large. Thus, the method is often impractical for recursion levels greater than 4, making it cumbersome on large designs [5]. In contrast, the time complexity of FL does not necessarily grow exponentially with the number of "levels" of learning in most practical cases (the time complexity is exponential only if the BDD sizes grow exponentially). Hence it is reasonable to expect that FL will have a greater applicability to larger industrial designs where a relatively large number of levels of learning are needed. Another useful ATPG-based technique to carry out design verification using internal equivalences and observability don't cares has been presented in [2]. But in some cases the internal relationship between circuits cannot be captured by ATPG. Therefore, this method also has scope for further improvement.

Importantly, other existing learning techniques are limited in the types of relationships that they can practically discover between a set of functions. They can learn *Constant-Value Relationships*; i.e. a constant Boolean value $b \in \{0,1\}$ at a given gate implies another constant Boolean value at another gate. However, they cannot detect more involved relationships between a set of functions and another set of functions, a limitation that can be overcome by FL due to the use of OBDDs. For example, we can very conveniently find if a gate $f_i = b$, $b \in \{0,1\}$ implies that some set of gates must assume an identical value. Or we may learn the implications that arise when a simple or complex set of clauses is always true. A simple example is learning the set of gates that maintain the same Boolean value whenever $f_i \wedge f_j = 1$. Note, f_i and f_j in the above examples need not correspond to any gate in the given circuit. For the application of learning techniques when a set of clauses must be iteratively refined, for example, as in many fixed point computation algorithms, FL provides a complete framework to learn and use such functional implications.

Our experience shows that the sizes of the OBDDs that are built during learning are quite small; hence, there is no memory explosion. Use of dynamic ordering [13] makes it practical for most combinational circuits encountered in real life. An advantage over traditional *topology* driven learning methods is that our approach is not limited by the peculiarities of a given circuit structure but only by its underlying functionality; a common advantage of symbolic manipulation methods.

In other words, FL encodes the topological structure of a circuit as an OBDD. Thus, it bridges the gap between structure (topology) based and symbolic manipulation based techniques. FL is a complete method for learning in digital circuits, i.e. given sufficient time, all the internal equivalences and unidirectional implications can be identified by FL alone. Our results show that FL based verification can be a highly potent verification technique. In this context note that although learning alone may not be sufficient for verification within reasonable time resources, the knowledge acquired in this process is still very useful. Consider verifying output pair F_1 and F_2, in circuits C_1 and C_2. To verify any such output pair, we begin by composing OBDD for $F_1 \oplus F_2$, and dynamically pruning the OBDD during its construction by the successive use of learning conditions as explained in Section 5. Thus, the method can intuitively be more effective than the traditional OBDD approaches as each time an intermediate OBDD is constructed, it can be simplified (modified) using the set of learnings derived earlier through the use of some learning technique; the final result remains unchanged but can potentially be obtained using lesser space-time resources if during this successive graph modification, we use only the modifications whose introductions prune the argument graph size. Our techniques thus extend the range of applications of OBDDs as well as their application in combinational verification even further.

3 On Functional Learning

Due to its ability to learn functional relations as well as constant-valued learnings, FL [10] proves to be a superset of the previous learning techniques, namely, SOCRATES [14], which carries out static learning and RL [7]. Learning techniques, through the temporary injection of logic values at arbitrary signals in a digital circuit and subsequent examination of its logical consequences, allow one to determine logical relations even between gates not on any common path.

3.1 The concept of Functional Learning

The concept of FL [10] is explained below with the help of an example.

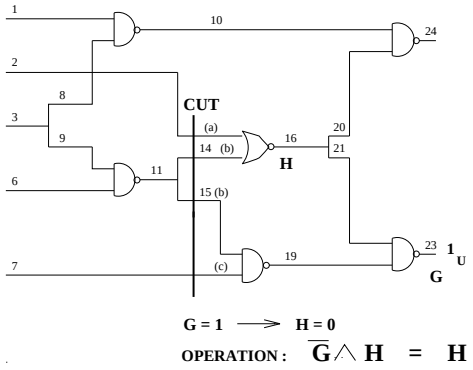

Figure 1: *Functional learning*

Consider the gate 23 in the circuit shown in Figure 1 and let it be unjustified to a 1. G and H are the OBDDs for the gates 23 and 16 respectively, built in terms of the pseudo inputs a, b and c. The two OBDDs are shown in Figure 2. Note, the result of the AND operation between OBDDs for $\overline{G}$ with H is H. This implies that when G is a Boolean 1, H is a Boolean 0. Hence, it is learned that a Boolean 1 on the gate 23 implies a Boolean 0 on the gate 16. By forward implication, it is learned that the gate 24 must carry a Boolean 1. A Boolean 0 on the gate 16 and a Boolean 1 on the gate 24 are the *necessary conditions* for a Boolean 1 on the gate 23.

As discussed in [10], if $G \Rightarrow H$ then $G \bigwedge H = G$. Conversely, if $G \bigwedge H = G$, then $G \Rightarrow H$, and thus by the Law of Contrapositum $\overline{H} \Rightarrow \overline{G}$. Identical arguments hold for $\overline{G} \bigwedge H$. Therefore, all constant-value learnings between G and H can be obtained by the two operations $G \bigwedge H$ and $\overline{G} \bigwedge H$.

3.2 Precise marking of potential learning area

FL proceeds by choosing an appropriate *cut* λ and building the OBDD for the unjustified gate g in terms of the *cut*

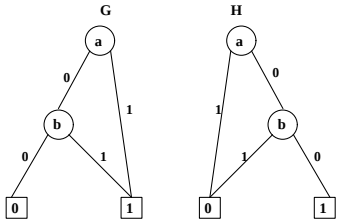

Figure 2: *OBDDs for gate 23 and gate 16*

variables, i.e., gates on the cut λ. Once the OBDD is built, appropriate AND operations, as explained in the previous section, must be performed in order to learn indirect implications. But, the gates where learning will be possible under the given situation of value assignments in the circuit are not known before hand. In order to precisely demarcate the *potential learning areas* (PLAR) in the circuit, a simple preprocessing of the OBDD is carried out.

Definition 3.1 *A justification vector in an OBDD is a path from the root variable in the OBDD to that terminal node whose value equals the value (v) of the unjustified gate (g).*

During the preprocessing of the OBDD for event $(g = v)$, that is, the unjustified gate g with an unjustified value v, a constant k number of justification vectors are extracted. These vectors are applied to the circuit and a complete implication is carried out. Assume $c(g, v, k)$ is the set of all the gates such that any gate in this set carries the same consistent Boolean value v for all k justification vectors. The gates in $c(g, v, k)$ are marked as the PLAR. A set of mutually equivalent points $e(g, v, k)$ can also be defined where every member of the set is a tuple of gates (h_1, h_2) such that the identical value is produced at gates h_1 and h_2 for each of k justification vector. Similarly, a set of mutually complement points $i(g, v, k)$ can be defined where value of h_1 is always a complement of the value of h_2.

Theorem 3.1 *With respect to the event $(g = v)$, through analysis of any possible BDD made from variables on cut λ, following assertions hold good:*

(1) A constant-valued-learning is possible only at the gates in $c(g, v, k)$.

(2) Mutually equivalent or complement point learning is possible only among gate tuples in $e(g, v, k)$ and $i(g, v, k)$ respectively.

Proof Sketch: For a constant-valued relationship to hold between g and some gate h, it must hold at every vector that satisfies $(g = v)$. Hence every candidate gate h in *assertion* 1 must be a member of $c(g, v, k)$. Similarly we prove the *assertion* 2.

Mutually equivalent or complement relationships can be also thought of as "second order" relationships. These are easily computed using FL by simply comparing in constant time the BDDs produced at the above candidate gates under condition $(g = v)$.

As a consequence of the above theorem, only the OBDDs at gates in the sets: $c(g, v, k)$, $e(g, v, k)$, and $i(g, v, k)$ are subjected to the analysis required for learning. The number of justification vectors that need to be applied in order to demarcate the PLAR with high amount of precision can be heuristically determined.

For OBDDs with a very small number of justification paths, our experiments indicate that this preprocessing is sufficient to complete the learning operations. Where large OBDDs are required, such preprocessing of the OBDD drastically reduces the search space for the PLAR; often as much as by 2 orders of magnitude. In general, we have found that extracting around 100 justification vectors is quite sufficient for the purpose of demarcating the PLAR.

In case of a larger OBDD, for which a complete enumeration of all the justification vectors is not feasible, the procedure outlined above gives the gates that comprise the PLAR. Once the PLAR has been marked, learning operations based on the theorem stated above are carried out at these gates.

Assume $S_f(L)$ and $S_r(L)$ are the set of conditions learned respectively by the FL technique when the cut λ is selected at distance L from an unjustified gate g (or "operating at level L" as we will use in the following), and the RL technique at recursion level L. It can be shown that:

Theorem 3.2 *[11, 12]* $S_f(L) \supseteq S_f(L-1) \cup \ldots \cup S_f(1)$

Note that the RL technique learns implications through a recursive analysis of a given logic function/topology. Justification cubes are recursively extracted from and simulated on the given topology. To learn at the i^{th} level of recursion, one must learn at each of the recursion levels from 1 through $i-1$. However the above theorem suggests that in the FL technique, one can directly proceed to a given (structural) level, and begin the learning there.

As shown in [10], for some functions, FL takes time polynomial in number of gates analyzed, but RL requires exponential time resources. Ignoring the arguments about OBDD variable ordering, the reverse does not seem true and an RL method apparently has a higher computational complexity. Interestingly, our experiments show that FL performs equally well as RL based verification even on c6288, a multiplier, having an exponential representation using OBDDs.

In short, time resources required by structural procedures such as RL will be acceptable only if the total number of cubes required to analyze a given problem is not unmanageably large. However, in our experience, for every functional analysis problem, this may not always be the case. Even when learning is performed in FL through path extraction and simulation, it may be more efficient because the functional information is symbolically consolidated during any efficient BDD construction procedure, and the total cubes (paths) extracted are often minimized.

3.3 On obtaining more than just constant-valued learning

Assume that functions F and G are the outputs of gates f and g respectively. As customary in FL, in the following discussion all functions are represented in terms of an OBDD constructed using some common cut λ between the gates and (up to) the primary inputs. A precise marking of the PLAR is done to identify gates f and g (functions F and G) between which AND has to be carried out to learn if any unique constant-value learning can be obtained. When no such learning can be obtained, consider the scenario when $|H| \ll min(|F|, |G|)$ where $H = F \wedge G$. Then, the graph H can be processed completely by path extraction and simulation.

Let S be the set of gates $\{p_1, \ldots, p_k\}$ which assume a unique value by simulating the circuit for each vector satisfying H. Or, implications can be learned by using symbolic manipulation procedures between H and the BDDs produced at other gates. Unlike, traditional learning methods, S is an *extra* learning that FL provides. Note, $H = F \wedge G$ had already been computed to check if *constant-value* learning are possible. Such learnings can be used in multiple ways. In ATPG, while justifying any value assignment v in the circuit, if gates f, g assume Boolean 1, no optional assignment should be employed that contradicts assignments in S. In verification, the above technique can be used as follows. For any small H if the size of set S is large, we can write any function as $F = (\overline{H} \wedge F_{\overline{H}}) \vee (H \wedge F_H)$. Each cofactor can now be simplified as described in Section 5. Importantly, since $H = F \wedge G$ has a small OBDD representation, $\overline{H} = \overline{F \wedge H}$ must have a small OBDD too. (Note, OBDDs can be complemented in constant time). Thus, through H, and $\overline{H}$, we are covering learning in an orthogonal part of the truth table without any extra computation. Such features can be of critical help in difficult verification problems.

Similarly, any other Boolean operation such as 'XOR' or 'OR' or other more complex operations can also be carried out. In general, if for any Boolean operation performed, the resulting OBDD is small, an extra set of learnings can be easily obtained. This technique enhances the ability to obtain new learnings which constitute some of the 2^N combinations of values that can exits on N gates in a circuit. For example, in Figure 1, if a cut is assumed on primary input variables and each of the gates 11 and 19 assumes a Boolean value 1, we can learn that gates 7, 16 and 23 assume unique values, namely 0, 0 and 1 respectively.

4 Algorithm for Learning

The two circuits to be verified are joined at their primary inputs and their corresponding primary outputs are fed in pairs to 2-input XOR gates to construct a *composite circuit*. Thus, if a given output pair in the circuit is to be proven inequivalent, we need to prove the satisfiability of the output of the corresponding XOR gate.

Verification is carried out in two phases : first, a *learning phase* followed by a *checking phase*. In the learning phase [10, 11] Boolean values are injected at the gates in the composite circuit such that the gates become unjustified. For example, a Boolean 0 is injected at the output of an AND gate and a Boolean 1 at the output of an OR gate. For an XOR/XNOR gate, both a Boolean 0 and a Boolean 1 are injected at its output. Typically, this phase is started using a certain initial level of learning L_l. The initial learning level can be 1, which means for learning at the gates in the composite circuit, cuts are taken at a structural distance of 1 from the gates. Additional learning can be obtained by successively increasing the level of learning in steps of k, where k need not be equal to 1. All the indirect implications learned during this phase are stored along with the data structure of the gate from which it was learned. Before learning at a gate G for learning level L_l, learning is carried out at all the gates in the transitive fan-in of G. This helps speed up the learning process by using the pre-stored indirect implications in the fan-in cone of G. Gates in the circuit which have equivalence or inverse relations with other gates are also identified and stored during this phase. Note that this phase by itself is a complete algorithm for design verification. Given sufficient time, verification can be completed by this phase alone.

5 Augmenting Verification Using ATPG or Canonical OBDDs

To verify the two circuits we want to check if our composite output represents an unsatisfiable function. As discussed in [11], we can use an ATPG tool that tries to generate a test for the s-a-0 fault at the composite output of the appropriate XOR gate. However such an approach still has limitations if the ATPG tool lacks the capability to effectively use symbolic manipulation to exploit the learning related information, and also analyze the given circuits. One can also use OBDD based *canonicity oriented* verification. Before we discuss how a canonicity oriented verification can be usefully augmented using FL, we must discuss relevant characteristics of a (functional) learning method which are critical in augmenting canonicity driven verification.

5.1 Functional Learning Based Canonicity Driven Verification

Note, a central issue in integrating learning based verification with traditional OBDD based techniques is recognition of the cases when a learning technique may not suffice. Let $\lambda = \{\psi_1, \ldots, \psi_m\}$, be a cutset in our composite circuit C, where the output D_i of C represents difference function $F_i \oplus G_i$, the XOR between corresponding outputs of circuit C_1 and C_2; $\{\psi_1, \ldots, \psi_m\}$ are internal gates in circuit C. We can introduce a pseudo variable for each gate in λ; let $D_i(\lambda), F_i(\lambda), G_i(\lambda)$ respectively represent the difference function, and the output functions for circuits C_1, C_2, each expressed in terms of cutset (henceforth referred to as just a *cut*) λ. Some limitations on the learning techniques can be now stated:

Theorem 5.1 *A learning technique will fail to prove equivalence of F_i, G_i by examining $\mathsf{D}_{\mathsf{I}}(\lambda)$ if following conditions hold.*

1. *For some point ψ_i in cone of F_i, $\psi_i \in \lambda$, no indirect implication can be discovered between ψ_i and any other gate on or ahead of the cut λ in the cone of G_i.*
2. *ψ_i is not a redundant variable in $F_i(\lambda)$*

Proof Sketch: We simply make the observation that $F_i(\lambda) \oplus G_i(\lambda) \neq 0$ if the active (nonredundant) variables in their support set are not the same.

Thus, by any implication based analysis solely in the circuit between λ and the output D_i, we cannot determine if the circuits are equivalent. Through similar reasoning as used above, the above theorem can be augmented as in the following.

Theorem 5.2 *A learning technique will fail to prove equivalence of F_i, G_i by examining $\mathsf{D}_{\mathsf{I}}(\lambda)$ if following conditions hold.*

1. *For some gate ψ_i in cone of F_i, $\psi_i \in \lambda$, an indirect implication can be discovered only for $\psi_i = b$, for some unate value $b \in \{0, 1\}$, with respect to any gate between λ and the output $D_i(\lambda)$ in the cone of G_i.*
2. *ψ_i is a binate variable in $F_i(\lambda)$.*

Thus, for a learning based analysis to verify a circuit by itself, one must first establish that some cut exists that overcomes the restrictions imposed by Theorems 5.1 and 5.2. For example, the hardest outputs of c3540 can be verified with trivial space and time resources after locating the cut suggested in Theorems 5.1 and 5.2. In such cases, we compute OBDD for composite output $D_i(\lambda)$; If the BDD reduces to 0 then F_i and G_i are clearly equivalent. If such a cut cannot be initially located one can resort to a canonicity driven technique as described below.[1]

If no complete cut exists, we locate a good "incomplete" cut; a good heuristic is to maximize the number of learning points as well as minimize the distance of the given cut from the primary output.

5.2 BDD simplification through learning information

Sometimes learning techniques cannot prove functional equivalence by examining the circuit partition between λ and D_i due to the the restrictions explained by Theorem 5.1 and 5.2, or because it cannot be decided if $D_i(\lambda)$ is satisfiable. In such cases we can revert to computing OBDDs for composite output D_i in terms of primary input variables, or another cut λ_j between λ and primary inputs. If D_i, or some $D_i(\lambda_j)$, reduces to 0 then F_i and G_i are clearly equivalent.[2]

Note that learning condition $\overline{a} \Rightarrow b$ can be written as a tautology expression $(\overline{a} \wedge b) \equiv \overline{a}$. The above tautology expression or *invariants* can prove very handy as discussed below. Let $I = \{\tau_1, \ldots, \tau_k\}$ be k invariants discovered between gates that are on or ahead of the cut λ. It can be proved:

Theorem 5.3 *$F_i \equiv G_i$ if $D_i(\lambda) \wedge \tau_i$, $\tau_i \in I$ is not satisfiable.*

The above theorem simply states that any invariant conjuncted with $D_i(\lambda)$ cannot change the functionality of D_i. Obviously we can replace τ_i with a conjunction of any plurality of restriction conditions in I. This can prove to be of critical help in simplifying a BDD for verification as well as many other problems.

In this context note that one may also examine BDD $D_i(\lambda)$ for unsatisfiability by explicitly checking if each 1-terminal path in the graph is unsatisfiable. In such case we can prune the total number of paths that need be examined if we (breadth-first) enumerate all paths from the root leading up to some cutset ω within the given BDD; now F_i and G_i are equivalent if there is a conflict on each such path. The conflict can be tested using an ATPG tool or by a direct simulation.

As noted above, to analyze the difference graph $D_i(\lambda)$ we can first simplify it by using the learning conditions (invariants). By reducing the size of $D_i(\lambda)$, we can make the composition of $D_i(\lambda)$, and thereby the verification too, more efficient.

Invariant BDDs can be successively arranged in increasing order of distance from the chosen cut λ and successively ANDed with $D_i(\lambda)$ in succession. If after any AND operation, the size of the BDD decreases than the resulting BDD replaces $D_i(\lambda)$ else we AND the next BDD in the above given order. Thus we are *consolidating* learning obtained in a large domain of circuit at some cut preceding it. Proceeding in the above manner, after all conditions are examined for ANDing (a reasonably efficient series of operations; note Section 6.2) the graph sizes for learning BDDs can be very small) we are guaranteedly left with a BDD $D_i(\lambda)$ which is no larger than the graph we started with, and most likely considerably smaller. Clearly, composing such an OBDD is more efficient than building OBDDs for output functions without accounting for any learning conditions. Note, using the remaining set of (unsuccessful) invariants, one can again repeat the above described series of AND operations. This iteration through the set of remaining invariant ANDing is terminated when the graph remains unchanged through every AND operation. Similarly we can use the implications discovered between some members ψ_g, ψ_h of cut λ. For example, if $\psi_g \Rightarrow \psi_h$, then we can simplify $D_i(\lambda)$ by cofactoring its BDD on $\psi_g = 1$ and making restriction $\psi_h = 1$. Note all implications between $\psi_g = 1$ and other members of λ can be introduced simultaneously in the BDD for $D_i(\lambda)_{\psi_g}$; the graph $D_i(\lambda)_{\overline{\psi_g}}$ can be similarly simplified if there are implications between $\psi_g = 0$ and another member of cut λ. Let H_{ψ_g} and $H_{\overline{\psi_g}}$ be the two reduced graphs. The resulting graph for $D_i(\lambda)$ is now $(\psi_g \wedge H_{\psi_g}) \vee (\overline{\psi_g} \wedge H_{\overline{\psi_g}})$. We maintain the resulting graph only if the cofactoring has decreased the graph size.

The process of ANDing and cofactoring can be repeated successively on each intermediate cut, progressively chosen between the old cut and the primary input variables; each successive cuts is chosen according to the same criterion as we discussed for the first cut. If at a given cut the graph sizes have decreased significantly, we compose it directly in terms of primary input variables.

Note for many invariants in I we need not spend resources constructing their OBDDs as the graphs will ultimately reduce to a Boolean 1; obviously, pursuing a direct conjunction of Boolean 1 with $D_i(\lambda)$ is not required. Specifically, we usually need not examine learning condition invariants that were discovered using (intermediate) cuts that are covered by the present cut through our composite circuit. More formally, let S_c be the set of all learning conditions that were discovered using BDD operations between graphs encoded through a set of cuts $\kappa_{cut} = \{\kappa_1, \ldots, \kappa_m\}$, where each κ_i is a cut for some intermediate gates. Also let each $\kappa_i \in \kappa_{cut}$ be such that κ_i is covered by λ. We define an intermediate cut κ_i to be covered by another cut λ if every path from primary inputs to any gate in κ_i must pass through a gate in λ. Importantly, we assume that *if* shortest path distance of every gate in κ_i from any gate in cut λ is at least d then during the learning phase the learning levels were incremented at most in increment of d. By applying Theorem 3.2, we can now prove that [12]:

Theorem 5.4 *No learning condition invariant made from the learning relationships in set S_c can help minimize the BDD size of $D_i(\lambda)$ through a Boolean operation between the learning condition invariant and $D_i(\lambda)$.*

[1] Note, in such cases one can also spend a greater computational effort in discovering learning at points where no learning was so far discovered. However such an approach has not yet been implemented and is thus not discussed in this paper.

[2] Note that whenever the goal is only to check whether some function D_i is satisfiable, by periodically checking the partially composed graph, further compositions can be immediately aborted if a path on primary inputs leading to a 1-terminal is obtained.

6 Results

At Fujitsu, we were not able to verify some difficult in-house circuits using OBDDs despite the application of numerous ordering tricks including application of dynamic variable ordering.

In this paper we focus on verifying such circuits as well as on the verification of the combinational portion of S38417 which has not been verified before using OBDDs. Discounting integer multiplier c6288, for all ISCAS circuits except c7552 the verification results using dynamic ordering appear quite efficient [13]. Identical observations hold for functional learning even when integrated with a rudimentary ATPG tool [11]. (Note, using learning information, we can verify c6288 against c6288nr in just 25 seconds [11]. Similar results hold for many other ISCAS circuits, but such results are not very instructive for purpose of discussing integration of canonicity driven and learning methods since in these particular cases there appear exceptionally many equivalent points between the given pair of circuits. Thus, in the checking phase, the BDD construction is not critical, or is not even invoked, and the verification is largely over after the learning phase.)

The above difficult circuits serve as good and fair test cases for showing that OBDD based canonicity driven verification can be usefully *augmented* with learning based verification methods. Importantly, in our implementation results we have still not employed functional implications as discussed in Section 3.3. Also, it appears to us that there are numerous implementational issues which can greatly speed up our current verification tool. Such issues will be focus of our future work, and thus the results presented in this section are preliminary in nature. The results are presented in Table 1. The times are in seconds unless otherwise mentioned.

6.1 Successful verification of a difficult industrial circuit

To corroborate the efficacy of our method, we will first describe successful verification of some test cases from Fujitsu which we will refer to as FJ1 and FJ2 which needed to be compared against their respective redesigned versions, $FJ1_{new}$, $FJ2_{new}$. (Note, FJ1 and FJ2 are different outputs of the same circuit.) The second copy of the circuit was obtained as the specification of some outputs in the original circuit needed to be changed. Some parts of the original circuit were modified and reused. However, the designer was now not sure if many of the outputs in the original circuit retained the intended functionality in the new circuit. The circuits have around 2000 gates, 200 inputs and 9 outputs. The experiments were run on a Sparc 10 with 128 MBytes. The times reported are in seconds except where specified. The OBDD sizes reported for our technique are the maximum graph sizes required in the checking phase.

The above two functions are notoriously difficult to verify, and compared to the ISCAS circuit pairs [8, 11] have relatively fewer internal equivalences with their redesigned version. FJ2 (not a multiplier) cannot be verified by any existing technique known to us. Traditional OBDD schemes also fail even when dynamic ordering is used *separately* for each output! We found that for FJ1 a composite circuit output graph of originally 210 nodes was produced. After a series of successful AND (and cofactor and restrict) operations this size could be reduced to 110 nodes before the first successive compose was begun. This graph could be very easily composed and FJ1 output was verified in less than 10 minutes. However, using dynamic ordering alone without the AND-ing of learning conditions, verifying FJ1 required close to 2.5 hours. Similarly, FJ2 could not be verified earlier using only OBDDs even by using dynamic ordering. It could now verified in less than 7 hours. A graph of about 7000 nodes was reduced to close to 1450 nodes by the process of ANDing the learning condition BDDs, and through the cofactoring operations based on the implication relations. This reduced graph was composed later in presence of dynamic ordering and the circuit design was verified. (Even in the absence of dynamic ordering, AND-ing of invariants proved to be extremely effective and reduced a 325,000 node decomposed OBDD for FJ2 to around 16,500 nodes.)

Table 1: *Verification of some difficult circuits.*

Circuit	OBDDs With Learning			Only OBDDs	
	Lrng. time	Total time	OBDD nodes	Time	OBDD nodes
FJ1	310	590	5079	9,176	479,064
FJ2	1065	6.5 h	425,300	Abort	Abort
c7552	9	22	8303	53	26,877
s38417 (a)	1.23 h	1.26 h	5	Abort	Abort
s38417 (b)	505	5 h	65,390	Abort	Abort
s38417 (c)	1892	1976	5	6,915	739,600

We report 3 different results on s38417 distinguished by (a), (b), (c). The second (and optimized) copy of s38417 had many equivalent points with the original circuit. Hence the FL based verification was largely completed in the learning phase itself in experiment (a). But this does not prove that our *checking phase* framework is useful.

In order to test and compare the benefits of incorporating learning conditions during checking phase we also report two more experiments. In (b) we report the time & space required for verification if far lesser number of learning conditions are learned; a relatively large BDD construction in checking phase was now required. Now, it can be seen that our checking phase framework did suffice in controlling the complexity of OBDD construction and verification. Finally, to compare the benefit of our framework with the OBDD-only framework, in (c) we report and compare results on a large subset of outputs (the last 500 outputs) for which OBDDs are indeed able to complete the verification. We also compare results for verifying the hardest output of c7552.

The profile of the XOR OBDD sizes in the checking phase for the verification of FJ1 is shown in Fig. 3. The commencement of successive compose and AND-ing of learning conditions are shown on the graph with 'c' and 'a' respectively. Note that the use of learning conditions during successive compose caused appreciable reductions in the graph size, making the technique viable for this difficult circuit.

6.2 Reducing the complexity of verification by aborting unsuccessful operations

An interesting feature of *learning* or *learning based verification* is that since the goal is only to decrease the resources required, we can abort all operations that can be determined to not lead to a learning or a reduction in the size of BDD in which they are being incorporated. This technique has not

yet been implemented but we believe it can provide significant performance gains.

For example, suppose in learning between two BDDs G and H, we construct the BDD $F \wedge G$. Clearly, the given AND operation can be aborted immediately if the intermediate product is larger than $max(|G|, |H|)$. Thus the complexity of learning between two functions using AND operation can be bounded by $max(|G|, |H|)$ rather than complexity of *apply* operation bounded by $|G| * |H|$.

In a similar vein, whenever we AND any learning condition invariant with the composite output BDD, we can abort the operation as soon as the resulting graph exceeds the original graph size. If we use only relatively small invariants, as we have found to often suffice, the time required for incorporating the learning conditions in a given BDD can be drastically reduced. Identical observations hold for the use of cofactoring in exploiting implications.

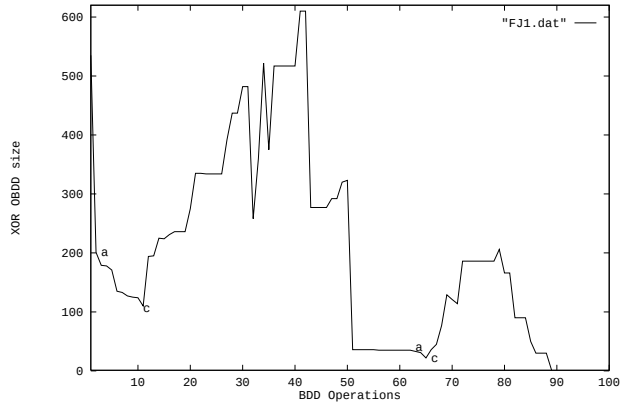

Figure 3: *Profile of XOR OBDD sizes in checking phase for FJ1 verification*

7 Conclusions and Future Work

In this paper we have addressed the problem of combinational circuit verification. We have presented an approach to merge the FL technique with the conventional OBDD based methods to improve the space and time complexity.

FL can learn constant-valued learning more efficiently then existing learning techniques when a large block of circuit needs be analyzed. It is also capable to learn relationships between different sets of gates where each set is allowed to have a plurality of logic gates in it. As appears from our theoretical, empirical and intuitive analysis, the FL technique is more powerful and general than other known learning techniques. We show how learning techniques can be integrated with traditional OBDD based verification methods to give significant space and time improvement for some very difficult industrial circuits over methods that conduct verification by comparing canonical OBDDs for each circuit. Efficient verification results were obtained for other benchmark circuits also for which canonicity driven verification appeared somewhat ineffective or was intractable.

The methods can naturally be extended for sequential circuit verification in the cases where one expects to find many indirect implications between a given pair of circuits. Our subsequent research will be directed towards the application of FL to the optimization of combinational and sequential circuits. The current research was focussed on developing a conceptually sound as well a practically viable framework for obtaining learning and checking functional equivalence. We can also incorporate techniques presented in [15] and minimize the size of BDDs using a don't care set derived from learning relationships. We plan to integrate the current programs with any advanced ATPG tool. We also plan to use observability don't care information such as successfully exploited in [2]. We will also focus on implementational issues which we believe can significantly enhance the performance of the verification framework presented in this paper.

8 Acknowledgements

We would like to thank Jacob Abraham and Don Fussell for the financial support of the second author during this research. We would also like to thank David Long for prompt help on many inquiries about his dynamic ordering based BDD package and Amit Narayan for a critical reading and helpful comments on the draft.

References

[1] Berman C. L., Trevyllian L. H., "Functional Comparison of Logic Designs for VLSI Circuits", ICCAD, 1989, pp. 456-459.

[2] Brand D., "Verification of Large Synthesized Designs", ICCAD, 1993, pp. 534-537.

[3] Bryant R. E., "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Trans. on Computers, vol. C-35, no. 8, Aug. 1986.

[4] Cerny E., Mauras C., "Tautology Checking Using Cross-Controllability and Cross- Observability Relations", ICCAD, 1990, pp. 34-38.

[5] Entrena L., Cheng K-T., "Sequential Logic Optimization By Redundancy Addition And Removal", ICCAD, 1993, pp. 310-315.

[6] Fujita M., Fujisawa H., Kawato N., "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams", ICCAD, 1988, pp. 2-5.

[7] Kunz W., Pradhan D. K., "Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation in Digital Circuits", Int. Test Conf., pp. 816-825, 1992.

[8] Kunz W., "HANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning", ICCAD 1993, pp. 538-543.

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

[10] Mukherjee R., Jain J., Pradhan D. K., "Functional Learning: A new approach to learning in digital circuits", IEEE VLSI Test Symp., pp. 122-127, April 1994.

[11] Mukherjee R., Jain J., Fujita M., "VERIFUL : VERIfication using FUnctional Learning", European Design and Test Conf., March 1995, pp. 444-448.

[12] Jain J., Mukherjee R., Fujita M., "Verification Techniques Based on Functional Learning", Technical Report No. FLA-CPS95-01, Fujitsu Laboratories of America, 1995.

[13] Rudell R., "Dynamic Variable Ordering for Ordered Binary Decision Diagrams", ICCAD, 1993, pp. 42-47.

[14] Schulz M., Trischler E., Safert T., "SOCRATES: A highly efficient automatic test pattern generation system", IEEE Trans. on CAD, vol. 7, Jan. 1988, pp. 126-137.

[15] Shiple T. R., Hojati R., Sangiovanni-Vincentelli A., Brayton R. K., "Heuristic minimization of BDDs using don't cares", DAC, 1994, pp. 225-231.