COMBINATIONAL VERIFICATION REVISITED

by

Sunil P. Khatri, Sriram C. Krishnan, Alberto Sangiovanni-Vincentelli and Robert K. Brayton

Memorandum No. UCB/ERL M98/60

30 October 1998

COMBINATIONAL VERIFICATION REVISITED

by

Sunil P. Khatri, Sriram C. Krishnan, Alberto Sangiovanni-Vincentelli
and Robert K. Brayton

Memorandum No. UCB/ERL M98/60

30 October 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Combinational Verification Revisited

Sunil P. Khatri (linus@ic.eecs.berkeley.edu) *
Sriram C. Krishnan (krishnan@ic.eecs.berkeley.edu) *
Alberto Sangiovanni-Vincentelli (alberto@ic.eecs.berkeley.edu) *
Robert K. Brayton (brayton@ic.eecs.berkeley.edu) *

Abstract

We revisit the area of combinational circuit verification. We study the existing methods for combinational verification, and propose a new method based on the computation of a Partial Satisfiability Dont-Care (PSDC) of the networks being compared. The method involves the use of Reduced Ordered Boolean Decision Diagrams (ROBDDs) to compute the PSDC of the networks. Results based on our implementation of this scheme and some of its variants are discussed.

In addition we implement a separate method for combinational verification based on Brand's scheme [3]. This method relies on an Automatic Test Pattern Generation (ATPG) package to determine the equivalence of subnetworks within the two networks being compared.

Our results compare the two schemes, and discuss the conditions under which each scheme performs well. We also identify opportunities for potential improvement in our implementations of the two schemes.

1 Introduction and Previous Work

The problem of combinational circuit equivalence has been well researched. It is a problem frequently encountered in digital circuit design. There are many instances where a designer is given two different circuits, and would like to know if they implement the same boolean function. Alternately, a designer may like to know if a given circuit correctly implements a specification, where the specification could be described at a higher level of abstraction.

The knowledge about whether a circuit correctly implements a specification is of critical importance. Many simulations performed before an IC is manufactured are done using a higher level specification, for efficiency reasons. In order to have a correctly functioning IC, to keep manufacturing costs down, and to have a quick time to market, it is very important that we reliably know whether a circuit is equivalent to the specification, against which many of the simulations were done. If the problem of combinational verification is efficiently solved, it can have a great impact on the IC design process.

In a similar way, we may sometimes want to know if a proposed revision to an existing design would function correctly. If we know that the two revisions are equivalent, and if we know that the existing design correctly implements the specification, we can be assured that the proposed revision functions correctly.

Prior research in this area has taken three fundamental approaches. The first is to represent the two networks in a canonical form, and then prove their equivalence. This is the basis of many ROBDD (henceforth referred to as BDD) based approaches, where the BDDs for both networks are built in terms of the network primary inputs. Since BDDs are canonical, the test for whether the networks are equivalent simply involves ensuring that the BDDs are identical.

*CAD Research Group, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720. This research was funded under the Semiconductor Research Corporation Grant SRC-324-040.

This is a constant time operation. However, these schemes can require a great deal of memory. It has been shown in [4] that for commonly occurring functions, BDDs can be of exponential size. As a result, there have been extensive attempts to come up with better ordering schemes, allowing the scheme to verify somewhat larger circuits. Such improvements were made in [9], [7], and [6]. In all these schemes, the BDDs of the network are built in terms of the primary inputs of the networks.

The second approach attempts to decompose the networks into smaller subnetworks, which are easier to verify. This is the approach of [2], where nodes of the two networks that are determined to be potentially equivalent are attempted to be simultaneously decomposed. A simultaneous decomposition of the candidate nodes is one in which the decomposed circuit shares a common subfunction. In order to determine if the nodes are functionally equivalent, it is sufficient to show that the nodes in the simultaneous decomposition that are not part of the common subfunction are equivalent. Exhaustive simulation is employed to determine if this is true, if the remaining circuits are sufficiently small, otherwise further decomposition is done. In this fashion, nodes that are determined to be equivalent are added to the list of equivalent nodes, and are treated as primary inputs. The authors concede that this procedure can result in false negatives, where the method declares two networks to be different, even when they are identical. This results from the fact that nodes determined to be equivalent are treated as independent signals. As a result, this scheme is not complete.

Most recently, [3] and [8] brought a new approach to the problem. Their approach eliminates the false negative problem of [2], by recognizing that the internal nodes of the two networks need not implement identical functions in order to be equivalent. The idea that they utilize is that under certain input minterms, the output of a node does not determine the output(s) of the entire network. For such input minterms, the nodes from the two networks are free to have different values. Brand's scheme [3] makes use of a test generator to check if two potentially equivalent nodes are indeed equivalent. The two nodes are connected to an XOR gate, and the output of this gate is tested for a stuck-at-zero fault, at the network outputs. This implicitly checks the condition mentioned above. If the XOR gate is not stuck-at-zero testable, then the two nodes are equivalent, and one of the nodes is replaced by the other. This replacement is done in order to keep the test generation process for future potentially matching nodes as simple as possible. Kunz's scheme [8] is similar to Brand's, and uses recursive static learning in addition. Learning is done based on implications for a given set of specified signals. If learning alone is not sufficient to prove equivalence, a test generator is used, which uses the learned clauses to determine if the networks are equivalent.

A generalization of this concept is found in [5]. Here, the problem was stated in terms of cross-controllability and cross-observability relations between nodes on a cut that spans both the networks. This procedure is complete.

A weakness of Brand's scheme is that it defers the difficulty of the problem of verification to the ATPG program that it calls. This routine is a structural test routine, and as such can require exponential search time before it determines whether a fault is testable. In order to keep this problem under control, test generation is abandoned if it takes more than a preset amount of time. As a result, Brand's procedure is not complete. An interesting point to be made here is that even if the test generation is abandoned for a node, it does not preclude future nodes from being declared as equivalent and replaceable.

Our approach, like Berman's [2] approach, is based on computing internal equivalences in the network. Unlike the Brand scheme, we call two nodes equivalent only if their functions are identical. We compute a Partial Satisfiability Dont-Care set (PSDC) of the networks. Our scheme is based on a theorem that states that two networks are identical if and only if their PSDC sets are identical. We build a BDD to compute the PSDC sets of the two networks, and declare the networks to be equivalent if the PSDC BDDs are identical. The BDDs we build are in terms of the network primary inputs, as well as the equivalent internal nodes. Under the condition that there is a uniform distribution of equivalent internal nodes, we expect that PSDC BDDs will be well behaved. It is observed in [3] as well as in [2] that even after significant optimizations are done on a network, there are still a large number of nodes whose function remains unaltered. Further, when one network is transformed into another, based on manual edits, this is even more likely to hold true.

2 Problem Statement

The problem of combinational circuit equivalence can be stated as follows. Given two combinational networks η_1 and η_2, we are to determine if $\eta_1 \equiv \eta_2$.

Two functions $f : B^n \rightarrow \{0,1\}$ and $g : B^n \rightarrow \{0,1\}$ are equivalent ($f \equiv g$) if

$$\forall m \in B^n (f(m) = g(m))$$

If f_i are the primary outputs of η_1, and g_i are the primary outputs of η_2, then the two networks are equivalent if the following holds.

$$\forall_i (f_i \equiv g_i)$$

3 Preliminaries and Definitions

A **Boolean Network** is a directed acyclic graph (DAG), with each node i having a logic function $f_i(x,y)$ associated with it. Here $x_j \in x$ is a primary input, and $y_k \in y$ is a node variable representing the internal node k. There is an edge (i,j) from node y_i to y_j if f_j depends explicitly on f_i.

The **transitive fanout** of node j (TFO_j) is defined as

node i s.t. i=j or $\exists$ path from j to i

The **transitive fanin** of node j (TFI_j) is defined as

node i s.t. i=j or $\exists$ path from i to j

The **Satisfiability Dont-Care set of a node** i is defined as

$$SDC_i = (y_i \oplus f_i)$$

It reflects the fact that nodes within the Boolean network are not independent, but are dependent on the primary inputs (and other nodes). SDC_i represents a set of variable values that cannot occur in the network, over the extended space of network variables (which includes primary inputs and internal nodes).

The **Satisfiability Dont-Care set of the network** is defined as

$$SDC = \sum_{i \in I} (y_i \oplus f_i)$$

where $I \subseteq \{1, \cdots, m\}$, and m is the number of nodes in the network.

The **Partial Satisfiability Dont-Care** *PSDC* set of the network can be written as

$$PSDC = \sum_{i=1}^{k} (y_i \oplus f_i)$$

The *PSDC* set is the SDC set of a subset of nodes of the network. The nodes that do not appear in the PSDC set are eliminated. The *PSDC* set can be thought of as the *SDC* set of an equivalent transformed network, in which these nodes do not appear.

The **Global Function** of a node i of the network is the function f_i of the node i expressed in terms of the primary inputs.

A **Reduced Ordered Boolean Decision Diagram** ((RO)BDD) is a Shannon Cofactoring "tree" which has the same order of cofactoring variables for every branch. Further, it is reduced in the sense that nodes with identical BDDs are merged, and a node which has identical BDDs as its children is eliminated.

A **Stuck-At-Fault** is a model for the faulty behavior of a circuit node. In this fault model, we assume that under the action of the fault, the circuit behaves as if that node was statically stuck at a 0 or a 1.

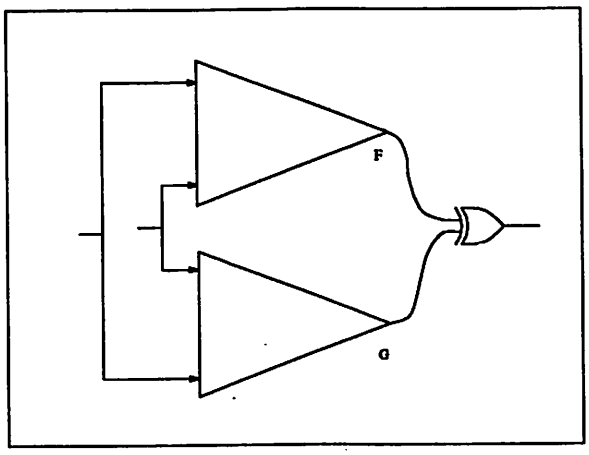

Figure 1: A miter

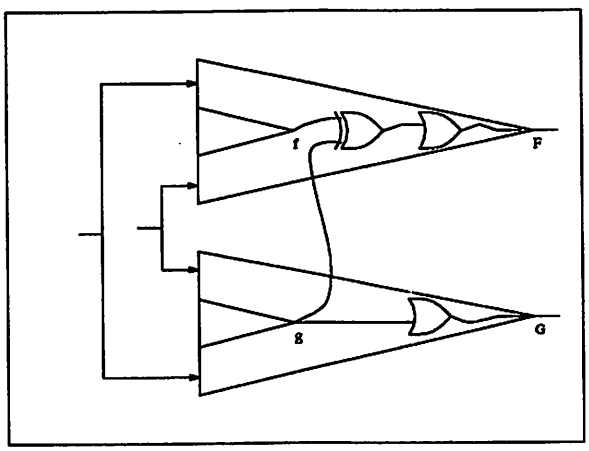

Figure 2: Testing if g can replace f

4 The Brand Scheme

Brand, in [3] proposed a test-generation based scheme to test for the equivalence of two combinational networks. Assuming that we were trying to test two functions F and G for equivalence, he observes that we could simply form the XOR of the signals F and G, and test the output of the XOR gate for a stuck-at-0. Brand refers to such a configuration of gates as a **miter**. In general a miter consists of a 2-input XOR gate, plus the symmetric set difference between the *TFIs* of the fanins of the two inputs to the XOR gate. Alternately stated, a miter starts at an XOR gate, and extends towards the primary inputs until nets that are common to both the cones of logic are encountered. Such a miter is shown in figure 1.

If we find a test for this fault, we have an input vector under which $F \neq G$, and the functions are not equivalent. If we cannot find such a test, then $F \equiv G$.

However, Brand observes, most test generators have difficulty with miters. So he proposes a scheme where the miter is always kept small, in order to keep the test generation problem manageable. Suppose f is a sub-function of F, and g is a subfunction of G. Under his scheme, he first tries to see if f and g are equivalent, modulo their surrounding logic. The means for performing this check is ATPG. An XOR gate is inserted between f and its immediate fanouts, and the other input of the XOR gate is connected to g. Hence a miter is created. This is shown in figure 2.

If the miter output is not stuck-at-0 testable, then either $f = g$, or $f \neq g$, but their difference is not observable at the output. In the latter case, the fault can be justified, but not propagated, to use testing terminology.

If two nodes f and g are found to be equivalent in the above sense (i.e. the miter output is not stuck-at-0 testable),

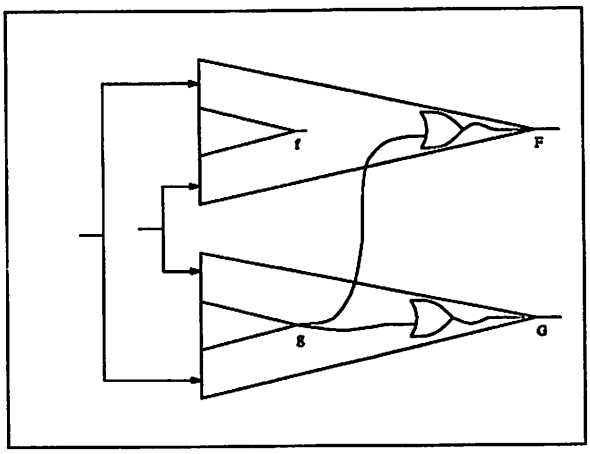

Figure 3: How g replaces f.

then the node g replaces the node f in F, as shown in figure 3. This is an important aspect of the scheme, since it ensures that in subsequent efforts to test nodes for equivalence, the miter would remain small, since the new network F shares the node g with network G. Proceeding in this fashion, if we are able to replace the outputs of F with the outputs of G, then the networks are indeed equivalent. The proof of this statement is provided in the paper, and is repeated here for clarity

Theorem 4.1 *Let $x \in B^n$ be a vector of variables, $f : B^n \rightarrow \{0,1\}$, and $g : B^n \rightarrow \{0,1\}$. Then,*

$$F(x,f) = F(x,g) \text{ iff } F(x, f \oplus g) = F(x,0) \tag{1}$$

and

$$F(x,f) = F(x,\overline{g}) \text{ iff } F(x, f \oplus g) = F(x,1) \tag{2}$$

Proof: The two equations are shown to be true for any x. For a given x, there can be four combinations of values on f and g. For examples, take $f = 0$, and $g = 1$. Then, 1 becomes

$$F(x,0) = F(x,1) \text{ iff } F(x,1) = F(x,0) \tag{3}$$

and 2 becomes

$$F(x,0) = F(x,0) \text{ iff } F(x,1) = F(x,1) \tag{4}$$

■

The RHS of 1 states that the output of the miter is untestable for stuck-at-0, and the LHS states that g can replace f.

If the miter fault cannot be proven untestable in a specified amount of time, the ATPG for that miter is abandoned. This does not preclude subsequent nodes from being declared equivalent, as discussed earlier. Of course, this implies that the scheme is not guaranteed to verify a design.

4.1 Our Implementation of the Brand Scheme

We implemented this scheme within the *SIS* [10] framework. The program consists of around 1000 lines of code written in the C programming language. It makes extensive use of the network, node and atpg packages in *SIS*.

Our scheme first simulates all the nodes of both the networks with a common randomly generated set of input vectors. The signatures of all the nodes are hashed, and nodes from different networks that share the same signature

are considered potential matches. [1] Next, for each node f in F, moving from the primary inputs to the primary outputs, we look for potential equivalences g in G. If there is a node from G that shares its name with the node being considered, it is tested for equivalence first. We create the miter node, and call ATPG on the miter node, for a stuck-at-0 fault. If the fault is untestable, then the node g replaces the node f in F.

In our case, we set a limit on the number of backtracks that the ATPG package is allowed. This is currently set at 50.

5 Our Approach

Until Brand's work [3] the de-facto standard scheme for Combinational Verification, since the advent of Bryant's Reduced Ordered Binary Decision Diagrams, has been building BDDs and testing for equivalence. Brand borrowed from methods existing for testing, exploited the multi-level structure of the circuit, and transformed the equivalence problem to that of satisfiability. Testing techniques have the advantage of being more time-intensive than space-intensive, and can succeed where BDDs for the functions cannot be built. In addition Brand's scheme works on the network structure itself, i.e. it exploits and retains the multi-level structure of the original circuit completely. Multi-level circuits are a powerful and compact means of representing logic functions and verification schemes that can retain this structure are desirable.

The new approach we suggest borrows some ideas from network optimization theory [1], but is still BDD-based. Although our scheme is BDD-based, we exploit and retain the multi-level structure to a certain extent.

Berman and Trevillyan [2] suggested some schemes for choosing sub-functions (in the subject combinational networks) to test for equivalence. Once nodes have been detected to be equivalent in the corresponding networks they are treated as "primary" inputs. This scheme has the drawback of not being able to detect subsequent nodes to be equivalent although they actually may be.

Our means of testing functional equivalence is not checking the equivalence of the two BDD's built but checking the equivalence of a special kind of partial SDC-set. For simplicity assume that we have two single output combinational networks on the same set of PIs, which we want to test for equivalence. Along the lines of Berman and Trevillyan and other schemes, we choose internal nodes appropriately, and test if they are equivalent. Rather than build BDDs of these internal nodes in terms of the PIs (the naive way), or treat previously computed equivalent internal nodes as primary inputs, we impose the relative correlation amongst the internal nodes via a partial-SDC set.

Given two internal nodes (the subject nodes) to be tested for functional equivalence, we build the partial-SDC set (see Equation 5) for the networks and compare them to test for equivalence. Equation (5) is termed a partial-SDC set because, one can think of recursively eliminating internal nodes if these nodes have not been detected equivalent between the two subject networks; the partial-SDC set is precisely the SDC set for this "collapsed" network. In this "collapsed" network all internal nodes have their functional counterparts in the other collapsed network.

Theorem 5.1 asserts that this scheme for detecting equivalences via building the SDC set is complete.

Theorem 5.1 *Give two combinational networks η_1 and η_2 on the same set of PIs, and a syntactic correspondence (a 1-1 map) on a subset of the internal nodes, then this implies a functional correspondence amongst the chosen nodes if and only if*

$$\sum_{k=1}^{m} f_{i_k}\overline{y_{i_k}} + \overline{f_{i_k}}y_{i_k}, \tag{5}$$

the partial-SDC set is equivalent for both networks, where y_{i_k} is the variable corresponding to the i_k^{th} internal node and f_{i_k} is the (global) function realized at the node.

[1] It is possible that even if the signatures of two nodes are different, they are still equivalent. Our scheme does not account for this possibility, however.

Proof: If the identified nodes are functionally equivalent then it is clear that the equality of Equation (5) for both networks holds. On the other hand if (5) is equivalent for both networks then their complements are equal as well. Therefore in the space of B^{n+m} exactly the same minterms are present in both networks implying the functional correspondence. ■

Observe that in (5) it is sufficient to express the function realized at the i_k^{th} internal node in terms of other internal nodes present in the correspondence subset, as well as PIs. That is, it is not necessary to build the BDD for the function in terms of just PIs; it is sufficient to compose functions of internal nodes starting from the "closest" cutset of the transitive fan-in cone of the node for which the function f_i is being constructed.

In the worst case the BDD for the partial-SDC set could be exponential in the size of the network even assuming that the function of each internal node in terms of it nearest previously equivalent nodes is a constant size-BDD. However, under good variable orders we expect the size of the partial-SDC BDD to be well behaved. The scheme suggested may be viewed as a function decomposition scheme, i.e. the function representing the multi-level network is retained in multi-level form as far as possible by the retention of the intermediate variables. Therefore the problem that now arises is how best to order the variables, PIs *and* intermediate variables included, so as to exploit (retain) as much of the multi-level nature of the circuit as possible in a *small* BDD.

In our scheme the decomposition points chosen are those with equivalent nodes in the other network. As has been observed in [3] and [2], there are usually a large number of such decomposition points identifiable, even after one network is altered by various combinational optimization scripts.

5.1 Implementations

We implemented the scheme in the C-programming language, under the *SIS* [10] framework. Our program consists of about 1300 lines of code. We make extensive use of the *ntbdd* and *bdd* packages within *SIS*.

Our scheme first simulates all the nodes of both the networks with a common randomly generated set of input vectors. This portion of the code is identical between this scheme and our implementation of Brand's scheme. The signatures of all the nodes are hashed, and nodes from different networks that share the same signature are considered potential matches.

In the *lazy PSDC* scheme, we visit each node from one of the networks, and try to find a matching node from the other network. If there are multiple potential matches, then a match with the same name as the node being matched is tried first. The BDDs of both nodes are built, [2] and if they are identical, then the nodes are considered to be identical. Subsequent BDDs are built in terms of primary inputs, variables corresponding to previously determined equivalences, and also the variable corresponding to the nodes just determined to be equivalent. The position of this variable in the variable ordering is determined based on a depth first search of the network. We proceed in this fashion, and see if all outputs can be matched. If so, then we do not compute the PSDC BDD at all. If all outputs cannot be matched, we repeat the process of visiting matched nodes, but this time we start to build the PSDC BDD of the network, in terms of the PIs and the variables corresponding to the previously determined equivalences.

In another variant of the basic scheme calle the *incremental PSDC scheme*, we determine if function are equivalent based on whether their PSDC BDDs are identical. Unlike the lazy scheme, the PSDC BDD is built incrementally. If two candidate nodes are being tested for equivalence, then their node SDCs are ORed into the latest PSDC, and if the resulting BDDs are identical, then the nodes are declared equivalent. Just as in the lazy PSDC scheme, subsequent BDDs are built in terms of primary inputs, variables corresponding to previously determined equivalences, and the variable corresponding to the nodes just determined to be equivalent. In this case, the new variable is inserted into the variable ordering in the same position as in the lazy scheme.

Since each primary output must be included in the PSDC set, we force the SDC terms of each primary output to be ORed into the PSDC set if the primary output was not declared equivalent in the first pass. This routine, called *psdc_last_gasp*, ORs in the SDCs of unmatched primary outputs into the PSDC BDD.

[2] in terms of primary inputs and variables corresponding to previously determined equivalences

If the resulting PSDC BDDs are identical, then we declare the functions equivalent.

In all our schemes, including the Brand scheme, we tried to compare a network with an altered version of the network, derived by running the combinational optimization script *script.rugged* on the original network.

6 Experiments and Results

We ran a series of benchmark circuits on both the PSDC methods, as well as on the Brand method. The results of our simulations are tabulated in Tables 1 and 2. We determined the execution times, number of matched nodes, and, in the PSDC methods, the size of the PSDC BDD.

We notice that there are examples under which the incremental PSDC scheme performs very well compared to other schemes. However, there are also examples where it takes much more time, probably because the PSDC BDD grew too large while the scheme attempted to determine if two nodes are replaceable. Since the heuristic for choosing the two candidate nodes is simply to see if their simulation signatures match over a small number of input vectors, we may accidentally choose two nodes that cause the PSDC BDD to grow out of control. This can probably be controlled somewhat if we ensured that the two nodes that are chosen for equivalence checking have about the same level in their respective networks. We are able to prove equivalence for two examples (vda and e64) on which the Brand scheme fails.

The lazy PSDC scheme showed some interesting results as well. In 45 cases, it was able to determine circuit equivalence without requiring to build the PSDC BDD at all, which is a pretty reasonable fraction of the entire set of circuits tested. It was able to verify vda and e64 as well, and in addition, verified some circuits that the incremental scheme failed on. In many examples, it had a significantly smaller execution time than the incremental scheme, likely because it did not need to compute the PSDC set at all.

The Brand scheme completed on the most number of examples. It failed on some, but was the most robust of the schemes. We believe that the PSDC schemes will get a significant increase in their speed and their ability to verify large circuits when we implement a interleaved BDD variable order. Currently we use the *order_dfs* routine of *SIS*, and since most of our functions have a large number of outputs, we expect interleaving to play a big role in improving the PSDC schemes.

Interestingly, the Brand scheme, in almost all cases, finds the exact same number of internal equivalences as does the incremental PSDC method. This is good news for the incremental PSDC scheme. The lazy PSDC scheme finds less equivalences in general; in some cases, it finds half as many equivalences as the other schemes.

7 Conclusions and Future Work

We implemented different schemes to determine combinational circuit equivalence. We implemented two types of PSDC schemes, as well as the Brand scheme.

We find that there are a set of functions under which each one of the three schemes implemented proved to be the best choice. We feel that there is still room for improvement in all the schemes.

Currently, for both the PSDC schemes, we use a variable ordering which is based on a depth-first-search of the network. We feel that incorporating an interleaved variable ordering scheme as reported in [6] could play a significant role in improving the capabilities of the PSDC schemes.

Another scheme is to use the Brand and PSDC schemes in tandem. This idea seems to have merit, and we think it is an interesting follow-on study to the current one.

Further, as mentioned earlier, we could improve the efficiency of all the schemes by choosing candidate matching nodes that are closer together in level. This could supplement our current scheme, which simply chooses nodes that have same names across the two networks, if there exist such nodes.

Circuit	Number of outputs	Brand scheme		Lazy PSDC			Incremental PSDC		
		time	matches	time	matches	BDD size	time	matches	BDD size
ex	1	0.043	1	0.035	1	*	0.039	1	8
ex2	2	0.050	2	0.055	2	14	0.035	3	14
C17	2	0.106	3	0.050	3	*	0.047	3	30
ttt2	21	2.957	37	6.304	27	29274	3.668	39	29850
x4	71	16.233	91	1.387	88	*	–	–	–
z4ml	4	0.317	9	0.129	7	*	0.149	9	221
term1	10	1.929	22	7.574	17	46427	3.656	23	51219
vda	39	–	–	58.336	59	11810	125.676	113	16043
t481	1	0.383	11	0.125	9	*	0.140	11	175
unreg	16	1.149	18	0.433	17	*	0.738	18	1049
x1	35	5.735	50	–	–	–	–	–	–
x2	7	0.328	9	0.172	7	*	0.168	9	176
tcon	16	0.293	8	0.305	8	1786	0.211	8	1786
pcler8	17	1.129	22	0.316	19	*	0.976	22	8938
pcle	9	0.602	14	0.191	14	*	0.527	14	4954
parity	1	0.566	15	0.165	10	*	0.176	15	160
sct	15	0.969	19	0.547	14	441	0.305	19	441
t	2	0.067	3	0.043	3	*	0.047	3	31
pm1	13	0.656	17	0.175	16	*	0.207	17	399
mux	1	0.379	9	0.191	8	*	0.285	9	2638
majority	1	0.028	1	0.027	1	*	0.039	1	10
lal	19	1.332	23	10.469	18	90431	8.597	23	90431
my_adder	17	6.304	63	0.699	47	*	–	–	–
i5	66	9.562	66	0.883	66	*	77.186	66	114547
i3	6	6.875	62	0.770	62	*	–	–	–
f51m	8	0.754	16	0.222	14	*	0.328	16	1155
i1	16	0.543	13	0.344	13	261	0.180	13	261
frg1	3	1.282	13	1.371	10	5741	1.410	13	11302
example2	66	12.077	82	–	–	–	–	–	–
comp	3	1.340	18	0.355	17	*	0.445	18	2252
decod	16	0.836	23	0.192	23	*	0.203	23	261
cordic	2	0.511	10	0.180	9	*	0.215	10	970
count	16	6.882	31	0.504	31	*	0.742	31	1158
cu	11	0.488	13	0.429	12	1130	0.242	13	1130
cm152a	1	0.089	1	0.063	1	*	0.070	1	17
cm163a	5	0.496	12	0.152	9	*	0.218	12	664
cm162a	5	0.453	12	0.172	9	*	0.203	12	277
cm42a	10	0.383	13	0.137	13	*	0.195	13	106
cm85a	3	0.281	6	0.132	6	*	0.144	6	107
cmb	4	0.289	8	0.179	8	*	0.152	8	259
cm151a	2	0.137	4	0.082	4	*	0.082	4	305
cm150a	1	0.395	9	0.191	8	*	0.289	9	2638
cm82a	3	0.156	6	0.074	4	*	0.067	6	37
cc	20	0.817	18	0.695	18	3322	0.441	18	3322
cm138a	8	0.297	10	0.129	10	*	0.144	10	94
cht	36	3.594	44	0.602	41	*	0.836	44	1161
b9	21	1.828	28	0.437	27	*	18.315	30	69093
apex7	37	5.796	56	–	–	–	–	–	–
c8	18	2.008	34	9.323	34	69414	9.668	34	69414
b1	4	0.086	3	0.082	3	19	0.040	3	19
alu2	6	8.492	45	1592.879	23	4250	8.609	44	7457

Table 1: Experimental Data

Legend: – : method fails * : Did not need to compute PSDCs

Circuit	Number of outputs	Brand scheme		Lazy PSDC			Incremental PSDC		
		time	matches	time	matches	BDD size	time	matches	BDD size
9symml	1	1.329	11	2.394	9	1337	0.899	11	1592
xor5	1	0.094	4	0.040	2	*	0.051	4	61
vg2	8	0.605	10	0.199	10	*	0.320	10	2195
squar5	8	0.360	9	0.336	5	191	0.164	9	191
rd84	4	1.289	16	1.285	12	803	0.531	16	1024
rd73	3	0.668	16	0.207	13	*	0.253	16	379
sao2	4	1.332	17	2.355	13	2727	0.695	17	2727
rd53	3	0.183	5	0.226	3	55	0.113	6	55
misex1	7	0.360	10	0.164	9	*	0.172	10	118
misex2	18	1.516	25	0.301	23	*	0.434	25	1098
bw	27	3.535	–	1.418	16	716	0.718	34	716
e64	65	–	–	19.178	107	5543	30.525	107	5543
con1	2	0.101	3	0.058	3	*	0.070	3	46
clip	5	1.235	16	0.383	14	*	0.496	16	1438
ex5	63	12.644	86	1.367	79	*	4.316	86	5147
b12	9	0.680	16	0.274	14	*	0.355	16	736
apex2	3	–	–	–	–	–	–	–	–
5xp1	10	1.156	18	0.320	17	*	0.566	18	1645
9sym	1	1.016	9	0.879	8	889	0.477	9	889
o64	130	–	–	–	–	–	–	–	–
duke2	29	37.611	70	201.448	31	81611	174.7	72	107029
ex4	28	7.144	51	723.294	45	2719619			
misex3	14	120.410	81	1872.607	59	175517	426.239	82	207060
C1908	25	–	–	–	–	–	–	–	–
C1355	32	60.328	162	563.448	104	*	–	–	–
C499	32	52.118	162	561.699	104	*	–	–	–
C432	3	23.991	–	1636.403	30	655974	494.883	45	678124
C880	26	10.000	70	1.722	59	*	–	–	–
alu4	8	–	–	–	–	–	–	–	–
frg2	144	–	–	–	–	–	–	–	–
i4	6	–	–	–	–	–	–	–	–
i7	67	17.878	77	2.422	75	*	8.14	77	2735
i6	67	12.902	76	1.735	74	*	2.949	76	2045
i9	63	15.343	78	2.746	76	*	–	–	–
rot	93	73.281	–	–	–	–	–	–	–
ttt2	21	2.785	37	6.687	27	29274	3.621	39	29850
x3	99	37.099	144	–	–	–	–	–	–

Table 2: Experimental Data, continued

Legend: –: method fails * : Did not need to compute PSDCs

References

[1] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. L. Sangiovanni-Vincentelli, and A. R. Wang. Multilevel Logic Minimization Using Implicit Don't Cares. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 7(6):723–740, June 1988.

[2] C. L. Berman and L. H. Trevillyan. Functional Comparison of Logic Designs for VLSI Circuits. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 456–459, November 1989.

[3] D. Brand. Verification of Large Synthesized Designs. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 534–537, November 1993.

[4] R. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35:677–691, August 1986.

[5] E. Cerny and C. Mauras. Tautology Checking Using Cross-Controllability and Cross-Observability Relations. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 34–37, November 1990.

[6] H. Fujii, G. Ootomo, and C. Hori. Interleaving Based Variable Ordering Methods for Ordered Binary Decision Diagrams. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 38–41, November 1993.

[7] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and Improvements of Boolean Comparison Method Based on Boolean Decision Diagrams. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 2–5, November 1988.

[8] W. Kunz. HANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 538–543, November 1993.

[9] S. Malik, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 6–9, November 1988.

[10] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.