Variable ordering for ordered binary decision diagrams by a divide-and-conquer approach

F.-M. Yeh
S.-Y. Kuo

Indexing terms: Ordered binary decision diagram, Variable ordering, Boolean function

Abstract: An efficient variable ordering strategy for ordered binary decision diagrams (OBDD) based on interleaving the compacted clusters is proposed in this paper. The novelty of this method is to apply the divide-and-conquer approach to find a good variable ordering efficiently for circuits with a large number of I/Os. First, a given circuit is partitioned into a number of clusters according to the correlations among the fan-in cones. A good ordering for each cluster is obtained and then a good global ordering is derived by interleaving the orderings of individual clusters. In this way, the time-consuming process of searching good orderings is restricted within individual clusters each with a manageable number of input variables. This divide-and-conquer approach is able to obtain a good variable ordering more efficiently than existent methods for circuits with a large number of I/Os. One notable result from the method is that we are able to build the OBDD for the cs38417 circuit within 1000 seconds on a SPARC 20 with 128M byte memory.

1 Introduction

The ordered binary decision diagram (OBDD) [1] is one of the most efficient and versatile representation of Boolean functions. Its applications on formal verification, test generation, and logic synthesis [2–6] are being extensively investigated and bearing fruitful results. To facilitate these applications, efforts have been invested on the OBDD to explore its full potential.

In the manipulations as well as the applications of OBDDs, the efficiency is dominated by the size of OBDD for representing a given function. It has been observed that the OBDD size strongly depends on the ordering of input variables. As a dramatic example, a multiplexer of n variables in its best ordering has an OBDD with a size smaller than $2n$ while in the worst case a size larger than $2^{(n+1)}/n$ [7]. Theoretically, the optimal ordering to yield the most compact OBDD for a given function can be obtained by exhausting all variable permutations. However, the best algorithm for finding the optimal ordering has a complexity $O(n^2 3^n)$ [8]. It is clear that this complexity is not realistic. Consequently, there have been intensive studies on the ordering of variables for OBDDs. Heuristic methods [9–15] for good variable ordering have been proposed. A common feature of these heuristics is that they are employed in a static way, i.e. the ordering remains intact during the OBDD building process. These heuristics, although effective for medium-scale circuits, are unable to deal with larger circuits due to their inflexibility to adjust.

Recent studies have allowed the ordering to change dynamically during the building process and have extended the method significantly to accommodate larger circuits. In [16–18] algorithms dynamically modify an initial ordering according to the encountered adverse situation to reorder the variables for maintaining the intermediate OBDD within a reasonable size as well as obtaining a compact final OBDD. In particular, the dynamic sifting algorithm proposed in [16] performs better for variable ordering of OBDD than previous works. By using efficient level exchange algorithm, it is able to explore a large search space and obtain very compact OBDD. However, in the worst case, the sifting algorithm requires $O(n^2)$ swaps of adjacent levels in the dynamic reordering process where n is the number of variables. For circuits with a large number of I/Os, it may need excessive time if reordering occurs frequently. Another algorithm for large circuits has also been proposed [19] based on interleaving individual ordering of primary outputs. By keeping separate orderings as intact as possible during the interleaving process, it successfully extended the DFS algorithm [12] from single-output circuits to multiple-output circuits. Although the resultant OBDDs are not as compact as those in [16, 18], the ordering can be determined quickly.

In many application of OBDDs, the output Boolean functions have to be manipulated after completely built such as recursively constructing reachable states and removing redundant states in a finite state machine, logic circuit resynthesis, or design error diagnosis. A good global ordering will be a dominating factor in improving the efficiency of these algorithms. The popular benchmarks for variable ordering heuristics in tackling circuits of various sizes are the circuits of ISCAS85 benchmark [20] and combinationalised ISCAS89 benchmark [21]. In these circuits, there is a 16-bit multiplier, c6288, which is already shown to have exponential OBDD size regardless of any variable ordering. Among the remaining ISCAS85 circuits, the static heu-

IEE Proceedings online no. 19971370

Paper first received 22nd October 1996 and in revised form 4th April 1997

The authors are with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan

ristics can successfully handle most of them except the c2670 and the c7552 circuits. The dynamic heuristics [16–18] extend the appliation to these two circuits and all but one ISCAS89 circuits. The remaining *hard* circuit is the cs38417. The technique of accessing secondary memory such as hard disks to manipulate very large OBDDs [22] fails to build complete OBDD for this circuit. With more than 100 million nodes allocated on two-gigabyte virtual memories, only the first 60% gates of the cs38417 circuit have been manipulated by [22] on a SPARC 10/41 using 26 hours. There had been no reports presenting techniques to efficiently construct OBDDs for the cs38417 circuit before the preliminary results in [23]. We will make significant enhancement in this paper and present more experimental results to show the effectiveness of our approach.

An efficient strategy for variable ordering of OBDDs based on interleaving the compacted clusters is proposed. The novelty of this method is to apply the divide-and-conquer approach to find a good variable ordering efficiently for circuits with a large number of I/Os. First, a given circuit is partitioned into a number of clusters according to the correlations among the fan-in cones. A good ordering for each cluster is obtained and then a good global ordering is derived by interleaving the ordering of individual clusters. In this way, the time-consuming process of searching good orderings is restricted within individual clusters each with a manageable number of input variables. This divide-and-conquer approach is able to obtain a good variable ordering more efficiently than existent methods for circuits with a large number of I/Os. One notable result from our method is that we are able to build the OBDD for the cs38417 circuit within 1000 seconds on a SPARC 20 with 128M byte memory.

2 Variable ordering by divide-and-conquer

In this Section, the divide-and-conquer approach of our variable ordering method for circuits with a large number of I/Os will be described. Our method of variable ordering is based on a divide-and-conquer approach. The method consists of three main steps.

1. In the dividing phase: the large circuit is partitioned into a number of small circuits, or clusters, such that the correlation among the clusters is sufficiently low.
2. In the conquering phase: each cluster is then extensively searched for a good variable ordering. A variable ordering is good in the sense that it yields a compact OBDD.
3. In the merging phase: based on those good orderings of clusters, a global ordering is then obtained by preserving the original orderings of clusters as much as possible.

The most time-consuming part is in the second step to search a good ordering for each cluster. However, such a cluster is in general much smaller than the original circuit and therefore, significant improvement in efficiency can be expected when the original circuit has a large number of I/Os. We will describe the first two steps in this section and the last step in the next Section.

In the first main step, we are to partition a given circuit into clusters so that a good ordering can be efficiently searched for each cluster. We will describe the cluster partitioning first. A cluster is a subcircuit consisting of some primary outputs and all the gates and primary inputs within the fan-in cone of these primary outputs. In cluster partitioning, the primary outputs of a cluster do not overlap with those of other clusters. However, a primary input may appear in several clusters depending on the circuit topology. The correlation of primary inputs among clusters has a major impact on the last step when the orderings of clusters are to be combined. The cluster partitioning procedure is shown in Fig. 1, which is regulated by a *cluster_factor* to reduce such correlation.

```
Cluster_Partition()
{
    Given a cluster_factor, cluster_number = 0;
    for each primary output O
        according to the decreasing number of gates in the fan-in cone
        { for each Cluster[i]
            Compute the input correlation with O by

            Common[i].ratio = (# PI for O included in cluster[i]) / (# PI for O),

            where # PI is the number of primary inputs.
            if ( Common[i].ratio is greater than cluster_factor )
                Select Cluster[i] with the highest ratio to merge;
                Cluster[i] = Cluster[i] ∪ O;
            else
                Increment cluster_number by 1;
                Create a new Cluster[cluster_number] = O;
        }
}
```

Fig. 1 *Cluster partitioning procedure*

The *cluster_factor* may have a value between 0 and 1. The value of *cluster_factor* is determined empirically. A high *cluster_factor* implies a high threshold for clustering, i.e., only highly correlated primary outputs are merged into one cluster. In general, high *cluster_factor* yields more clusters of small sizes.

Given a predetermined *cluster_factor*, in *Cluster_Partition* procedure, the correlation of a primary output, *O*, with existent *cluster* [*i*] is computed. If one of the computed correlation, *Common_ratio*, is greater than the given *cluster_factor*, then the primary output is merged into the cluster with the highest *Common_ratio*. Otherwise, it becomes a new cluster. The sequence of primary outputs for merging is determined by sorting them according to the decreasing number of gates in the fan-in cone. An additional step for the procedure is to merge those primary outputs with less than 16 primary inputs into a single cluster to avoid an ineffective compaction due to numerous clusters. The ordering of this cluster is assigned the lowest priority.

In obtaining variable ordering of compact OBDD size for a cluster, the dynamic sifting algorithm in the CMU package is modified for this purpose. The trigger condition of reordering is modified to be adaptive to the reordering effectiveness. When reordering is less effective, we allow more increase of the number of nodes before next reordering takes place. The purpose is to reduce ineffective and time-consuming reordering when there are many intermediate gates in the same level. The termination condition of sifting is when the

number of nodes increases by 20% than the original size due to moving a variable up and down for an optimal position.

3 Cluster merging

Given good ordering for individual clusters, the last step is to obtain a good global ordering for the entire function or circuit. In the following, we will justify the interleaving method with a theorem and describe our method of interleaving variable orderings from individual clusters.

A variable ordering of an OBDD is a sequence of primary input variables for a given function. When two OBDDs for two clusters are to be combined into a compact one, it is desirable to retain as much as possible of the good ordering of each cluster. The interleaving of two ordering serves such purpose. There may be common variables between two sequences and in reality it is possible to have common variables with different orders in two sequences. In this case, some compromise on the original ordering must be made as will be discussed later. Here we will justify the interleaving method for cases without such conflict. In other words, the interleaving of two variable orderings is complete in the sense that the variables in the resultant sequence preserve the orders of the two original sequences. For example, let the two original ordering be Ordering $(A) = \{x_0, x_1, x_2, x_3, x_4, x_5\}$ and Ordering $(B) = \{y_0, y_1, x_2, y_3, x_3, y_4\}$. Then the Ordering $(C) = \{x_0, x_1, y_0, y_1, x_2, y_3, x_3, x_4, x_5, y_4\}$ is a complete interleaving of Ordering (A) and Ordering (B).

Theorem 1: Let two OBDDs have Ordering (A) and Ordering (B), respectively, without any conflict. Then the resultant OBDD with the interleaved variable ordering has a size no greater than the sum of the two original OBDDs.

[Proof] Since the OBDD is unique for a given function, it will not change when a new variable ordering is inserted into the ordering regardless of the position. Therefore, when two OBDDs without any conflict in the original orderings are combined, the resultant size can only be equal to the sum of the two original sizes before the merging and deletion rules are applied. After applying these two rules, the resultant size of OBDD can be no greater than the sum of the two original sizes.

The above theorem justifies the interleaving method to preserve the original order of variables as much as possible when combining two OBDDs. Another implication is that we may estimate the upper bound of the resultant OBDD size given individual sizes of all clusters when the variable orderings of all clusters have no conflicts. If complete interleaving of variable ordering is possible, then from Theorem 1, the resultant size can be no greater than the sum of individual sizes. Even when complete interleaving is not possible, the resultant OBDD size is generally strongly correlated to the sum of the individual sizes as will be shown in the experimental results.

Fig. 2 shows an example of resultant OBDD by interleaving orderings. In Fig. 2, OBDD A and OBDD B have 6 nodes each in their individual ordering. Obviously, the final size, 11 nodes, is less than the sum of the two OBDD sizes. This profile results from the merging rule of OBDDs so that the two $x4$ nodes are shared as one node.

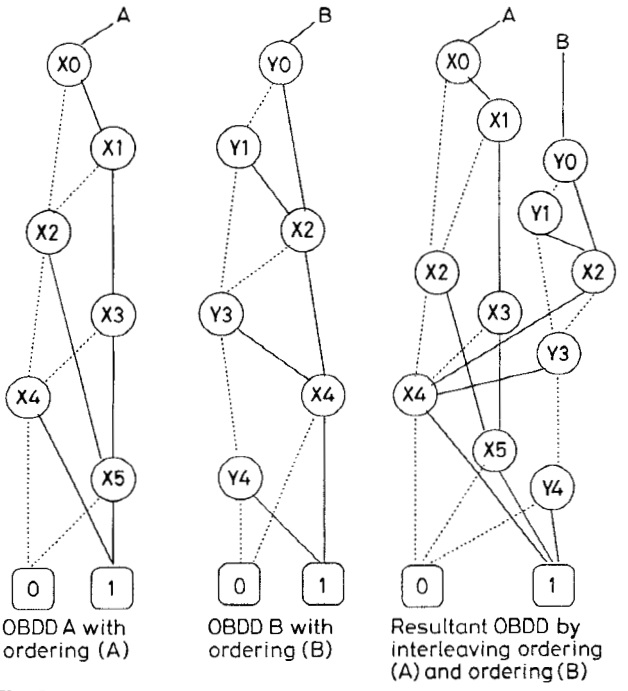

Fig.2 *Resultant OBDD by interleaving*

As discussed before, the interleaving of two given orderings is often incomplete, i.e. there are common variables with conflicting orders. In this case, some compromise must be made. The ordering of the cluster with higher priority determines the global ordering. The priority of each clusters is defined by the CPU time for OBDD compaction because large compaction time is a good indicator of a hard cluster. The interleaving procedure is therefore shown in Fig. 3.

```
Interleaving(G_ordering, L_ordering)
{   for each variable v in L_ordering from top
    {   if ( v does not belong to G_ordering )
        {
            add v to queue();
            if ( next_variable(v) belongs to G_ordering )
                while ( queue() != EMPTY )
                    w = dequeue();
                    insert w to G_ordering just before next_variable(v);
        }
    }
    if ( queue() != EMPTY )
        append all variables of queue() to G_ordering;
}
```

Fig.3 *Interleaving procedure*

The overall algorithm of our method as shown in Fig. 4 can then be described as follows. Given a net list description, first the whole circuit is partitioned into a number of clusters according to the *Cluster_Partition* procedure. The initial ordering, *Local_ordering*, of each cluster is determined by traditional DFS_appending [10] method which traverses the fan-in cone of each primary output of the cluster to obtain a sequence of variable order with depth-first-search. Then build compact OBDD for each cluster and thereby obtain a good ordering for each cluster by dynamic reordering. Since a cluster generally has far less input variables than the

entire circuit, the compaction process can be accomplished much more efficiently. After that, the orderings of clusters are merged into an effective global ordering according to the *Interleaving* procedure. Finally, we rebuild OBDD for the entire circuit with the global ordering.

```
Main()
{   Global_ordering = NULL;
    Cluster_Partition();
    for each cluster
    {
        Local_ordering = DFS_append of clusters;
        Modify Local_ordering by dynamic reordering;
    }
    Global_ordering = interleaving all Local_orderings
        according to decreasing compaction process time;
    Rebuild OBDD using Global_ordering;
}
```

Fig.4 *Algorithm for interleaving compacted clusters*

A possible alternative to the last step is to adopt a greedy approach by iteratively combining a new cluster with the partially constructed resultant OBDD. The variable ordering of the new cluster is interleaved with the current global ordering by keeping the global variable order intact. After that the OBDD is optimised only for the newly inserted variables of the new cluster. Intuitively, this greedy approach may avoid the duplicated construction of OBDD while keeping a compact size. It is strongly dependent on the first chosen cluster, but there is no sufficient information to determine the cluster priority of merging sequence. One cluster with a large number of gates, inputs, or outputs cannot be exactly recognized as a hard cluster. In general, it gives inferior results in both the size and the CPU time.

Although the principle of order interleaving is similar to that in [19], we apply it in a unique way with the divide-and-conquer approach. The correlation among clusters is reduced sufficiently low in cluster partitioning to maximise the probability of interleaving. Thereby, we are able to take a better advantage of the good orderings of clusters to obtain a more compact resultant OBDD.

4 Experimental results

Our variable ordering method for OBDD has been implemented and evaluated on a SPARC 20 workstation with 128 M byte memory. In the evaluation, our method for OBDD has been employed to build OBDDs for the larger ISCAS85 and ISCAS89 benchmark circuits. The maximum number of nodes is limited to 1500 K in terms of sharing OBDDs with inverted edges [9]. The result is compared with previous works [18, 19] and the dependency on the *cluster_factor* is presented. The remaining smaller circuits in the benchmark can be easily built with previous published heuristics.

In Table 1, we compare our results with the reported data [18, 19, 22]. The approach in [18] is an extension of the dynamic sifting method in [16] by incorporating symmetry check for contiguous variables. The symmetry check for contiguous variables is not implemented in our implementation. Their results are in general more compact than those in [16] with negligible overhead due to the sifting algorithm [18]. The results of [16] are not shown here because no CPU time has been provided, which is essential in the discussion. The results of the CMU package is obtained by applying the dynamic sifting-sifting algorithm on our machine. The result of [19] is based on interleaving the ordering for each primary output obtained from a DFS heuristic. In this comparison result, the value of *cluster_factor* is 0.6 which is determined empirically to deliver better overall results. In this table, *nodes* indicates the number of OBDD nodes and *time* is the CPU time.

From this Table we can see that performance of our method generally falls between the fully dynamic sifting algorithm and the fully interleaving-based algorithm. The approach in [19], although faster, produces OBDDs significantly less compact than the other three. The reason is that too many clusters of less compact size lead to more conflicts and even larger resultant size. On the other hand, the fully dynamic sifting algorithm generally has the most compact result. The main disadvantage of fully dynamic sifting algorithm is the dramatic growth of processing time for circuits with a large number of I/Os such as cs15850 [18] and cs38417. In comparison our method generates OBDDs of the same order as those in [18]. However, for difficult cir-

Table 1: Comparison results

Circuit	[19]		Sift-sift [18]		Sift-sift (unpublished)		Ours	
	nodes	time	nodes	time	nodes	time	nodes	time
c7552	33k	12	6k	242	8k	139	9k	122
cs5378	5k	4	2k	170	3k	8	6k	3
cs9234	66k	14	4k	336	5k	39	6k	28
cs13207	15k	11	–	–	3k	27	6k	10
cs15850	62k	22	37k	5509	18k	174	18k	75
cs30532	6k	28	–	–	5k	46	5k	46
cs38584	35k	41	–	–	27k	697	32k	123
cs38417	>> 2M	–	–	–	511k	26354	696k	1023

[19]: On SPARC 470 with 96 M byte
[18]: On Dec 5000/200
[22]: On SPARC 20 with 128 M byte
Ours: On SPARC 20 with 128 M byte
– : no data
nodes: number of nodes
time: in seconds

Table 2: Results with various cluster-factors

Circuit	1.0				0.8				0.6			
	C	R	nodes	time	C	R	nodes	time	C	R	nodes	time
c7552	4	0.25	94k	68	2	1.0	9k	119	2	1.0	9k	122
cs5378	48	0.9	6k	8	17	0.9	6k	4	10	0.9	6k	3
cs9234	55	2.7	15k	37	7	1.1	6k	29	5	1.0	6k	28
cs13207	14	1.3	6k	12	9	1.3	6k	10	8	1.2	6k	10
cs15850	87	2.1	27k	79	11	0.5	37k	76	11	1.0	18k	75
cs30532	1	1	5k	46	1	1	5k	46	1	1	5k	46
cs38584	129	0.7	41k	502	54	1.0	30k	165	28	0.9	32k	123
cs38417	129	0.9	857k	2360	41	0.9	607k	996	24	1.0	696k	1023

C: number of clusters
R (ratio): number of all cluster nodes over number of nodes in final OBDD
nodes: number of nodes
time: in seconds

cuits with a large number of I/Os, our method is significantly faster as in cs15850 [18]. Moreover, for this circuit, our divide-and-conquer approach actually yields a more compact OBDD than the fully dynamic sifting algorithm [18]. More significantly, we are able to construct the OBDD of cs38417 more efficiently than existent methods. Its resultant OBDD has a size of about 696k nodes and is completed in 1023 seconds. The combination of huge size and a large number of I/Os of cs38417 is the major difficulty for OBDD construction. Our divide-and-conquer approach is especially effective for such circuits. With the problem for cs38417 been resolved, all the circuits in ISCAS85 and ISCAS89 benchmarks, except the multiplier c6288, have now been shown to have an OBDD of manageable size. There seems no reason that the building of cs38417 can not be accomplished with dynamic sifting algorithm alone. We directly employed the CMU package to the circuit cs38417 on a machine with a larger swapped area. After about 7 hours, the OBDD is completely built with 511k nodes. Although the OBDD size is about 20% smaller than the result of our approach, the processing time is 20 times longer. In this case, our divide-and-conquer strategy is able to efficiently obtain a sufficiently good ordering for circuits with a large number of I/Os.

The dependency of the result on the *cluster_factor* is shown in Table 2 for various *cluster_factors* in the *Cluster_Partition* procedure. Three *cluster_factors*, 1.0, 0.8, and 0.6, are evaluated. A higher *cluster_factor* allows only primary outputs with higher correlation to be put into a cluster. As a result, more clusters are generated with smaller size. With *cluster_factor* equal to one, then only the primary outputs with completely overlapped fan-in cones are merged into one cluster. It is similar but not identical to have each primary output in a separate cluster. The number of clusters for these three *cluster_factors* are shown in column *C* of Table 2. Also shown in this table are the number of OBDD nodes, the CPU time, and *R* which is the ratio of the total number of nodes of all clusters over the resultant node number.

The value of *R* is an indication of success of order interleaving. If the interleaving is completed without any conflicts, then *R* should be no less than 1 by Theorem 1. realistically, there will be conflicts during interleaving and *R* can be less than or greater than 1. From Table 2, a *cluster_factor* of either 0.8 or 0.6 yields results better than a *cluster_factor* of 1.0 both in the OBDD size and the CPU time. This is due to that there are too many clusters for *cluster_factor* 1.0 and hence more conflicts during interleaving. The phenomenon can be clearly seen in the c7552 circuit for which the OBDD size with *cluster_factor* 1.0 is ten times larger than the other two. When the *cluster_factor* is below 0.5, there will be less clusters with larger size and the compaction time for a cluster will be increased significantly. Another interesting observation is that the ratio *R* for both *cluster_factors* of 0.8 and 0.6 are close to one which implies that the interleaving process for these given *cluster_factors* does not produce many extra nodes due to conflicts. In addition, the total number of nodes of all clusters can serve as an estimate of the number of nodes in the resultant OBDD.

5 Conclusions

An efficient strategy for variable ordering of OBDDs based on interleaving the compacted clusters has been proposed in this paper. The novelty of this strategy is to apply the divide-and-conquer approach to find a good variable ordering efficiently for circuits with a large number of I/Os. In this way, the time-consuming process of searching good orderings is restricted within each individual cluster which has a manageable number of input variables. This divide-and-conquer approach is able to obtain a good variable ordering more efficiently than existing methods for circuits with a large number of I/Os. One notable result from our method is that we are able to build the OBDD for the cs38417 within 1000 seconds on a SPARC 20 workstation with 128 M byte memory.

Acknowledgment

This research was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC 85-2221-E002-018.

7 References

1 BRYANT, R.E.: 'Graph-based algorithms for boolean function manipulation', *IEEE Trans. Comput.*, Aug. 1986, **C–35**, (8), pp. 677–691
2 TOUATI, H.J., SAVOJ, H., LIN, B., BRAYTON, R.K., and SANGIOVANNI-VINCENTELLI, A.: 'Implicit state enumeration of finite state machines using OBDDs'. Proceedings of international conference on *Computer-aided design*, Nov. 1990, pp. 130–133

3 CHO, H., HACHTEL, G.D., JEONG, S.W., PLESSIER, B., SHWARZ, E., and SOMENZI, E.: 'ATPG aspect of FSM verification'. Proceedings of international conference on *Computer-aided design*, Nov. 1990, pp. 134-137

4 CHO, H., HACHTEL, G.D., and SOMENZI, F.: 'Fast sequential ATPG based on implicit state enumeration'. Proceedings of international *Test* conference, Sept. 1991, pp. 67–74

5 COUDERT, O., and MADRE, J.C.: 'Implicit and incremental computation of primes and essential primes of Boolean function'. Proceedings of 29th *Design automation* conference, June 1992, pp. 36–39

6 CHEN, K.-C., and FUJITA, M.: 'Efficient sum-to-one subsets algorithm for logic optimization'. Proceedings of 29th *Design automation* conference, June 1992, pp. 443–448

7 LIAW, H.-T., and LIN, C.-S.: 'On the OBDD representation of general Boolean functions', *IEEE Trans. Comput.*, July 1992, **41**, (6), pp. 661–664

8 FRIEDMAN, S.J., and SUPOWIT, K.J.: 'Finding optimal variable ordering for binary decision diagrams', *IEEE Trans. Comput.*, May 1990, **39**, (5), pp. 710–713

9 MINATO, S.-I., ISHIURA, N., and YAJIMA, S.: 'Shared binary decision diagrams with attribute edges for efficient Boolean function manipulation'. Proceedings of 27th *Design automation* conference, June 1990, pp. 52–57

10 MALIK, S., WANG, A.R., BRAYTON, R.K., and SANGIOVANNI-VINCENTELLI, A.: 'Logic verification using binary decision diagrams in a logic synthesis environment'. Proceedings of international conference on *Computer-aided design*, Nov. 1988, pp. 6–9

11 CALAZANS, N., ZHANG, O., YERNAUX, B., and TRULLEMANS, A.-M.: 'Advanced ordering and manipulation techniques for binary decision diagrams'. Proceedings of European *Design automation* conference, Sept. 1992, pp. 452–457

12 BRACE, K.S., RUDELL, R.L., and BRYANT, R.E.: 'Efficient implementation of a OBDD package'. Proceedings of 27th *Design automation* conference, June 1990, pp. 40–45

13 ISHIURA, N., SAWADA, H., and YAJIMA, S.: 'Minimization of binary decision diagrams based on exchanges of variables'. Proceedings of international conference on *Computer-aided design*, Nov. 1991, pp. 472–475

14 MERCER, M.R., KAPUR, R., and ROSS, D.E.: 'Functional approaches to generating ordering for efficient symbolic representations'. Proceedings of 29th *Design automation* conference, June 1992, pp. 624–627

15 BUTLER, K.M., ROSS, D.E., KAPUR, R., and MERCER, M.R.: 'Heuristic to compute variable orderings for efficient manipulation of ordered binary decision diagrams'. Proceedings of 28th *Design automation* conference, June 1991, pp. 417–420

16 RUDELL, R.: 'Dynamic variable ordering for ordered binary decision diagrams'. Proceedings of international conference on *Computer-aided design*, Nov. 1993, pp. 41–47

17 YEH, F.-M., and LIN, C.-S.: 'Building OBDDs with ordering-reshuffle strategy', *Electron. Lett.*, Aug. 1993, **29**, (17), pp. 1540–1541

18 PANDA, S., SOMENZI, F., and PLESSIER, B.F.: 'Symmetry detection and dynamic variable ordering of decision diagrams'. Proceedings of international conference on *Computer-aided design*, Nov. 1994, pp. 628–631

19 FUJII, H., OOTOMO, G., and HORI, C.: 'Interleaving based variable methods for ordered binary decision diagrams'. Proceedings of international conference on *Computer-aided design*, Nov. 1993, pp. 38–41

20 BRGLEZ, F., and FUJIIWARA, H.: 'A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran'. International symposium on *Circuits and systems*, June 1985,

21 BRGLEZ, F., BRYAN, D., and KOZMINSKI, K.: 'Combinational profiles of sequential benchmark circuits'. International symposium on *Circuits and systems*, June 1989, pp. 1924–1934

22 ARSHAR, P., and CHEONG, M.: 'Efficient breadth-first manipulation of binary decision diagrams'. Proceedings of international conference on *Computer-aided design*, Nov. 1994, pp. 622–627

23 YEH, F.-M., and LIN, C.-S.: 'OBDD variable ordering by interleaving compacted clusters', *Electron. Lett.*, Sept. 1995, **31**, (20), pp. 1724–1725