COMBINING TOP-DOWN AND BOTTOM-UP APPROACHES FOR ROBDD CONSTRUCTION

by

Jawahar Jain, Amit Narayan, Claudionor Coelho,
Sunil P. Khatri, Alberto Sangiovanni-Vincentelli,
Robert K. Brayton, and Masahiro Fujita

Memorandum No. UCB/ERL M95/30

10 April 1995

COMBINING TOP-DOWN AND BOTTOM-UP APPROACHES FOR ROBDD CONSTRUCTION

by

Jawahar Jain, Amit Narayan, Claudionor Coelho,
Sunil P. Khatri, Alberto Sangiovanni-Vincentelli,
Robert K. Brayton, and Masahiro Fujita

Memorandum No. UCB/ERL M95/30

10 April 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Combining Top-down and Bottom-up approaches for ROBDD Construction

Jawahar Jain (jawahar@fla.fujitsu.com) †

Amit Narayan (anarayan@ic.eecs.berkeley.edu) *

Claudionor Coelho (coelho@pegasus.stanford.edu) ‡

Sunil P. Khatri (linus@ic.eecs.berkeley.edu) *

Alberto Sangiovanni-Vincentelli (alberto@ic.eecs.berkeley.edu) *

Robert K. Brayton (brayton@ic.eecs.berkeley.edu) *

Masahiro Fujita (masahiro@fla.fujitsu.com) †

Category 1

[1]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720
tel: (510)-642-5048
fax: (510)-643-5052

[2]Fujitsu Laboratories of America, San Jose, CA 95134

[3]Department of Electrical Engineering, Stanford University, Stanford, CA 94305

Abstract

ROBDDs have traditionally been built in a bottom-up fashion, through the recursive use of Bryant's *apply* procedure [6], or the *ITE* [4] procedure. With these methods, the peak memory utilization is often larger than the final ROBDD size. Though methods like Dynamic Variable Reordering [21] have been proposed to reduce the memory utilization, such schemes have an associated time penalty.

In this paper, we show that for a large number of applications, it is more efficient to construct the ROBDD by a suitable combination of top-down (decomposition based) and bottom-up (composition based) approaches. We suitably select decomposition points during the construction of the ROBDD, and follow it by a symbolic composition to get the final ROBDD. We propose two heuristic algorithms for decomposition. One is based on a topological analysis of a given combinational netlist, while the other is purely functional, making no assumptions about the underlying topology of the circuit. We demonstrate the utility of our scheme on the standard benchmark circuits. Our results show that for a given variable ordering, our method usually has significantly better time as well as memory characteristics than existing techniques.

Our methods are easily extended to many variants of ROBDDs, and in that sense are powerful in their scope.

1 Introduction

Reduced Ordered Binary Decision Diagrams (ROBDDs) [6] are frequently used in various CAD problems such us synthesis, digital-system verification, testing etc. However, the construction of ROBDDs is often a time and memory consuming process. Accordingly, techniques that can help in a quicker construction of ROBDDs (or other efficiently manipulable BDD schemes like FDDs [16], OKFDDs [9]) are of much practical significance. In the traditional bottom-up (*apply* based) scheme of constructing ROBDDs, the peak intermediate memory requirement far exceeds the final (canonical) representation size of the given function, as is schematically indicated in figure 1. Although we are interested in obtaining the BDD of the output, the intermediate peak requirements often limit our ability to construct ROBDDs. This peak memory requirement places a limit on the complexity of circuits that can be processed using ROBDDs, and also usually dictates the time required for ROBDD construction.

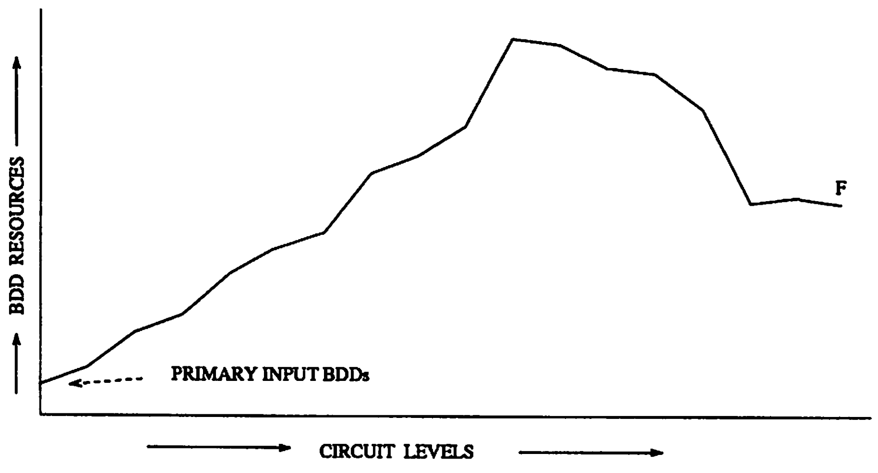

Figure 1: *Example of ROBDD Resources Required Throughout a Circuit.*

Even though the final ROBDD size is an invariant under a given variable ordering, the theoretical bounds on intermediate peak space requirements are far weaker. Accordingly, it would be very appealing to construct ROBDDs in a fashion which reduces the intermediate peak memory requirements. This has an attendant benefit of a reduction in the time required to construct the ROBDD. In our schemes, we introduce suitable decomposition points, and perform all computations on the decomposed graphs, to get a decomposed representation of the output function. Then, we compose this function, to get the canonical ROBDD for the output. In this manner, we avoid the intermediate memory explosion, since the graphs manipulated are smaller. We show, by various experiments, that the peak memory and temporal behavior of our scheme is usually better than that of a *apply* based scheme, for a given variable ordering.

A decomposition based approach is central to almost every technique that uses non-conventional BDD techniques for purpose of verification. This includes gBDDs[2], XBDDs [14], IBDDs [12] and probabilistic verification as in [13]. Research on decomposition is likely to impact these areas significantly. In these techniques, one is required to find decompositions which make the problem of ROBDD construction computationally less expensive than the bottom-up approach. Only when this can be reasonably assured can these alternate techniques offer an improvement over ROBDD-based symbolic calculations.

We propose two separate methods to combine the top-down and bottom-up approaches. In the first, we conduct a topological analysis to determine if an internal region of the given circuit can be classified as a circuit *waist*. A waist is comprised of circuit nodes at a given level of the circuit, such that the number of nodes is a minimum. A local alteration of the waist is done, until the ROBDD of the output in terms of the altered waist is smaller than that of other local alterations. At this point, the ROBDDs of the (best altered) waist nodes are composed into the ROBDD of the output, resulting in the final monolithic ROBDD. In the second scheme, we consider functional decomposition methods. Here, a bottom-up construction of the ROBDD is attempted; decomposition points are introduced when the ROBDD and / or the ROBDD manager grows beyond a predetermined threshold. Finally, the decomposed variables are composed back in to the output ROBDD, to get the monolithic ROBDD of the output. This approach makes no assumptions about the underlying topology of the circuit. There is no physical correspondence between decomposition points and the nodes in the circuit. As such, this scheme can be used in a general ROBDD package in a manner transparent to the user.

We show that our schemes usually have a smaller peak memory requirement than the *apply* based scheme. For this reason, we are often able to build ROBDDs that could not be built by existing schemes, since the peak memory requirement was larger than could be handled by existing ROBDD packages, even when the final ROBDD size was reasonable for the ROBDD package. Keeping the peak memory requirements low has the added advantage that the ROBDD is built in less time, since operations on large ROBDDs are avoided. Further, our scheme can take advantage of a good static as well as dynamic ordering, as is demonstrated in the results. So, for a given variable ordering, our scheme usually builds the resulting ROBDD in less time, with reduced peak memory requirements.

In Section 3, we discuss the basic ideas of decomposition for efficient ROBDD composition, and in Section 4 we discuss the problem of computing efficient decompositions. We describe our decomposition approach in Section 5, and discuss the results obtained in Section 6. We conclude with Section 7 where we discuss the possible applications for our techniques, and give directions for future research in this area.

2 Previous Work

Though BDDs have been researched for about four decades [17, 1], they found widespread use only after Bryant [6] showed that such graphs, under some restrictions, can be easily manipulated. The two restrictions imposed are that the graph is reduced (i.e. no two nodes have identical subgraphs), and that a total ordering of the variables is enforced. The resulting BDD is called a Reduced Ordered BDD (ROBDD). The (critical) symbolic manipulation procedures presented by Bryant were *apply* and *compose*; these techniques combine two identically ordered ROBDDs. *Apply* allows ROBDDs to be combined under some Boolean operation, and *compose* allows the composition of one ROBDD into another. Both algorithms require the ROBDD variables to have an identical order.

The use of *compose* for the purpose of constructing ROBDDs requires the introduction of suitable decomposition points in a function. It can also be argued that to some extent, the heuristic use of decomposition attracted lesser attention in the past because another ROBDD problem, that of computing a good variable ordering, was perceived to be more important. [21] contains an excellent bibliography of many such ordering studies. Correspondingly, composition-based procedures have not been adequately studied.

ROBDDs are typically constructed using some variant of Bryant's *apply* procedure [6]; the ROBDD for a gate g is synthesized by a symbolic manipulation of the ROBDDs of its inputs, based on the functionality of g. The gates of the circuit are processed in a depth-first manner until the ROBDDs of the desired output gate(s) are constructed.

However, this exclusively bottom-up approach does not necessarily lead to the most computationally efficient technique for constructing ROBDDs. Such a technique is oblivious to the peculiarities of a given function, such as various simplifications that may arise later in the ROBDD construction process. As an extreme example, if the output function can easily be checked to be unsatisfiable by analysis of the later portion of the circuit, any construction of the ROBDDs of the earlier portions is unnecessary. In this contrived example, we could have an exponential reduction in the total resources required, if we used a composition-based, top-down method to construct the ROBDDs. (On the other hand, one can as easily construct examples where a top-down scheme to construct the ROBDDs can spend exponentially more resources in constructing the target ROBDD over the apply-based bottom-up scheme).

Further, an exclusively bottom-up approach is inadequate in accounting for regularities in the circuit which can be used to significantly reduce the ROBDD construction time. This is largely the result of Boolean "absorption" and "cancellation" that can potentially take place at fanout reconvergent regions.

There has been some prior research on ROBDD construction using decomposition. Fujita et. al. briefly discussed a few interesting decomposition experiments in [10] while computing efficient variable orders to compute ROBDDs for a given circuit. However, the results obtained by decomposition were not always encouraging; further no intuitively sound decomposition procedure was outlined.

Berman [3] looked into the problem of circuit decomposition and found a relation between the register allocation problem and efficient circuit decomposition. For a bounded-width circuit of n variables, that is, a circuit whose elements can be linearly ordered such that any cut crosses at most w wires, it was shown that there exists a variable order such that the size of the ROBDD will be bounded by $n * 2^{w}$. While the decomposition problem was conceptually well analyzed, efficient and well-tested procedures to find good decompositions were not provided. McMillan [20] generalized the results of Berman and proved that if w_f bounds the number of cut wires in the forward direction, and w_r bounds the number of cut wires in the reverse direction, then the size of the resulting ROBDDs is bounded by $n2^{w_f 2^{w_r}}$ [1]

Jain et. al., in [13], discussed the possibility of obtaining decompositions for the efficient "hashing" of Boolean circuits. However no method was described to obtain such decompositions. Both McMillan [20]

[1] In a related discussion, McMillan also proved that for bounded-width tree circuits, the size of the ROBDD will be polynomially bounded (in the number of gates) if the circuit elements can be arranged such that width of the linear order is logarithmic in number of gates.

and Jain [13] discussed a new *composition* procedure based on extracting variables sequentially from a given decomposed Boolean expression. Despite the apparent usefulness of these techniques, no results on any benchmark circuits have been shown so far.

Experiments on using auxiliary (decomposed) variables were reported in [8] for creating more efficient symbolic state-space traversal techniques. General decomposition techniques, however, were not reported. The benchmarks used sequential circuits with data paths having a very regular structure.

3 Preliminaries

In this section we establish the terminology for the rest of this paper. In the sequel, we restrict the discussion to Reduced Ordered Binary Decision Diagrams (OBDDs) alone. However, the discussion is equally well applicable to other manipulable graph based representation schemes which are an improvement over ROBDDs [18, 15, 5] or fundamentally different representation methodologies such as FDDs [16], OKFDDs [9], Typed-Free BDDs [11] etc. (For details on OBDDs, please refer to [5, 6, 7].)

Assume we are given a circuit representing a boolean function $F \equiv F : B^n \rightarrow B^o$, with n primary inputs $x_1, \ldots, x_n$, and o primary outputs. To simplify the discussion, we will focus on a single output G. Let $G_d(\Psi)$ represent the ROBDD of G expressed in terms of a variable set $\Psi = \{\psi_1, \psi_2, \ldots, \psi_m\}$. Each $\psi_i \in \Psi$ has a corresponding ROBDD, $\psi_{i_{bdd}}$, in terms of primary input variables as well as (possibly) other $\psi_j \in \Psi$, where $\psi_j \neq \psi_i$. Elements of Ψ can be ordered such that $\psi_{j_{bdd}}$ depends on ψ_i only if $i < j$. These $\psi_i \in \Psi$ are called *Decomposition points* of G. $G_d(\Psi)$ is the decomposed ROBDD in terms of the decomposition points.

The composition [6] of ψ_i in $G_d(\Psi)$ is denoted by $G_d(\Psi).(\psi_i \leftarrow \psi_{i_{bdd}})$ where $G_d(\Psi).(\psi_i \leftarrow \psi_{1_{bdd}}) = \overline{\psi_{i_{bdd}}}.G(\Psi)_{\overline{\psi_i}} + \psi_{i_{bdd}}.G(\Psi)_{\psi_i}$. For an efficient implementation of the ROBDD package, [5] is an excellent reference.

$G_d(\Psi)_{\psi_i}$ represents the *restriction* of $G_d(\Psi)$ at $\psi_i = 1$, and is obtained by directing all the incoming edges to the node with variable id ψ_i to its $\psi_i = 1$ branch.

Other techniques for ROBDD composition have been proposed [20, 13]; for our purpose we will consider the approach described above.

4 Observations on Decomposition Techniques

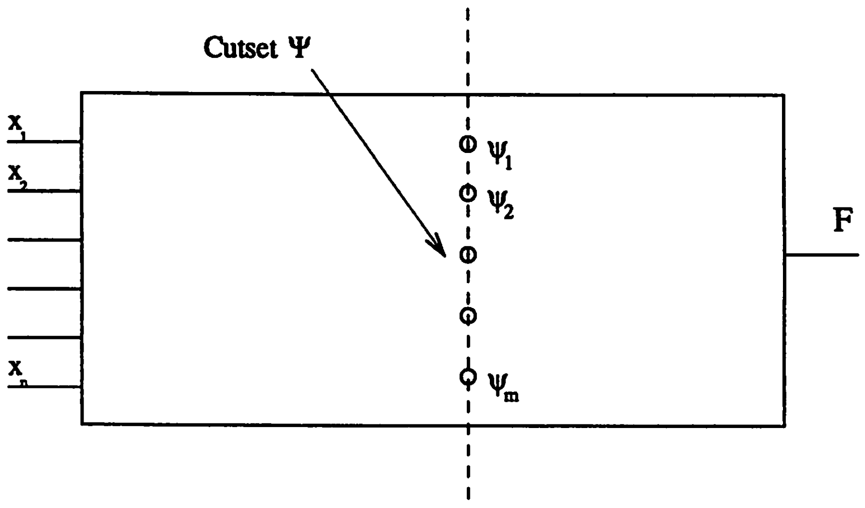

Figure 2: *A Single Level Decomposed Circuit*

As discussed in Section 2, it is easy to construct test cases where an exclusively bottom-up approach will require exponentially lesser or greater time than an exclusively top-down approach. Suitably mixing the two approaches logically provides the best possibility for capturing the stage were ROBDDs start simplifying. In deference to the intractability of such analysis problems, we note that one can work out examples where even mixing these two techniques can require exponentially more resources than either technique used in isolation. Thus we can at best hope to present heuristics which perform well in most cases.

4.1 Decomposition and Alternate BDD Schemes

To find a decomposition such that (under the same variable order) after composition, no more total time and space is required over bottom-up approach, is a difficult task. This is of critical importance for the performance of numerous non-canonical representation schemes such as gBDD [2], XBDD [14], IBDD [12] and probabilistic analysis [13].

In any alternate representation scheme, to synthesize the graph for some Boolean operation $\odot$, one must repeat a procedure such as *apply* for the ROBDDs. Now, given graphs G_1, G_2, in the worst case one must at least match every node in the first graph with each node in the other graph. For ROBDDs this bound is tight [6]. Hence, typically, one can only hope for an improvement in an alternate scheme if the graphs being manipulated are smaller. For this reason, the approach adopted by schemes such as gBDDs, XBDDs, IBDDs and probabilistic analysis techniques for gaining advantage over ROBDDs has typically been to decompose the circuit; working with smaller representations, they try to create a suitable graph which can sometimes be manipulated more efficiently. Just as these alternate schemes are sensitive to input variable ordering, bad decompositions can make ROBDD construction extremely inefficient.

4.2 Decomposition for ROBDDs

During a typical ROBDD synthesis procedure there is frequent functional simplification due to Boolean Absorption: $x \vee (x \wedge y) = x$, and Boolean Cancellation: $x \wedge \overline{x} \wedge y = 0$. This appears to be the heart of the reason why decomposition techniques can be useful. In the following discussion we will consider a circuit level representation but the principles are applicable to higher or more abstract models as well. We consider the circuit to be levelized, where a level L at gate g denotes the length of the longest path between any primary input and the gate g.

On a circuit without any reconvergence the size of ROBDDs must monotonically increase towards the primary output. Such circuits can be represented by ROBDDs using polynomial resources. In a circuit, when some signals reconverge at a gate, the input functions to the gate must have a non-null intersection of their respective support sets. This raises the possibility that the resulting function may now be "simpler" (of course reconvergence can also make the resulting functions much "harder"). Capturing each such instance where the function *simplifies* is the task of an efficient decomposition procedure.

As discussed in Section 1, the total space requirements while processing intermediate ROBDDs of a given circuit can far exceed the actual (canonical) size of the final function. Hence it is worth investigating if the memory utilization, which dictates the time required for ROBDD construction as well, can be reduced through decomposition techniques.

4.2.1 Boolean Absorption and Cancellation

Given two DNF expressions of equal size, the greater the number of variables common among different terms, the higher (on average) is the likelihood that the disjunction of two terms is simpler because of applications of boolean *absorption*. When an internal signal fans out to a large number of gates in a circuit, and is used as a decomposition point, this decomposition point is more likely to be present in many expressions. If Ψ_a and Ψ_b are two decomposition sets discovered in the same locality of the circuit and if variables of Ψ_a have a larger fanout, then typically $|\Psi_a| < |\Psi_b|$. Hence, finding a decomposition set with a smaller cardinality usually ensures that the decomposed variables will have a higher average fanout. We have experimentally observed that decomposing a circuit at its waist often leads an efficient decomposition.

If there is a decomposition point which is present in many terms of a CNF expression, in both its phases, then there is a likelihood that a large number of terms will cancel out, simplifying the function. In this fashion, boolean *cancellation* also plays an important role in the selection of decomposition points.

4.2.2 Location of Waists

If a waist is very close to primary inputs or primary outputs then almost all ROBDD resources spent in constructing G_d, the decomposed ROBDD, will be spent in either only a bottom-up or only a top-down procedure. In this case, since the growth behavior of G_d closely mirrors that of G, any contribution of decomposition is negated.

Hence we focus our attention on obtaining a waist in the circuit which is not located extremely close to either ends of the circuit.

4.2.3 Choice of Decomposition

According to Bryant [6], composing ROBDD H within a ROBDD G_d can require $O(|G_d|^2|H|)$ resources.[2] Clearly, given the option of choosing to decompose a function at any one of gates $H_1, H_2, \ldots, H_i$ for $|H_1| \approx |H_2| \approx \ldots \approx |H_i|$, we must choose to decompose at H_j $(1 \leq j \leq i)$, if $|G_{d_j}| = min(|G_{d_1}|, \ldots, |G_{d_i}|)$; where G_{d_j} represents the decomposed graph if we chose to decompose the function at gate H_j.

Thus, if Ψ_a and Ψ_b have the same size and the same average fanout, and if the sizes of the ROBDDs of the decomposition set members are more or less the same, the smaller decomposed ROBDD should yield the more effective of the two decompositions. This analysis is easily verified on practical circuits.

[2]In practice this bound is never observed but one does, at times, observe quadratic resources in the size of G_d and H.

4.2.4 Trading off sizes of G_d and ψ_is

Given two alternate circuit decompositions, we need to resolve whether a larger decomposed graph G_d with a smaller $\Sigma_{i=1}^{i=m}|\psi_{i_{bdd}}|$ is preferable over a smaller G_d with a larger $\Sigma_{i=1}^{i=m}|\psi_{i_{bdd}}|$.

Also, one must determine if the given function needs be decomposed at all. For either of the above two questions, one must analyze the rate with which graph G_d grows as it is constructed beginning from the primary output, proceeding towards the decomposition points, followed by successive composition. Here a decomposed graph at level $L+1$ in the circuit is composed in terms of gates (pseudo-inputs) at level L to obtain G_{d_L}, the new decomposed graph at level L. One similarly needs to analyze the total size of intermediate graphs constructed during the computation of ROBDDs $\psi_{i_{bdd}}, \forall \psi_i \in \Psi$. The above analysis can then indicate when the ROBDDs "blow up" along a certain direction, as well as the average rate at which the graphs are growing in either direction.

This allows one to heuristically decide the difficult choice posed above. If the decomposed graph G_{d_L} at some level L in the circuit is much smaller than the predicted size while graph G_{d_K} is larger than its predicted size at level K, G_{d_L} can be considered superior to decomposed graph G_{d_K} even if $|G_{d_L}| > |G_{d_K}|$. Similarly one can account for the increases in the size of graphs of decomposed points $\psi_{i_{bdd}}$.

4.2.5 Functional Decomposition

We still have not resolved how a given pair of $\psi_{i_{bdd}}$ will fare when composed in some G_d, or if such a method will be superior to constructing ROBDDs using *apply* exclusively. Also the above framework is often geared towards a netlist based representation of Boolean functions, and may not be suitable for processing other abstract representations of boolean functions.

Since the primary purpose for decomposition was to circumvent an intermediate explosion if possible, one can choose decomposition points based on the increase in the graph size during the intermediate stages of ROBDD construction. Beginning with primary input variables, as the graphs are built using *apply*, one can decompose them whenever any operation increases the total graph size by certain factor. The resulting graphs are typically multiply decomposed; a decomposed variable is usually a function of other decomposition variables. The approach does not perform a decomposition if the bottom-up technique does not result in the explosion factor being exceeded.

5 Our Approach

In this section we will outline useful circuit decomposition procedures by piecing together observations detailed in the previous sections. We will discuss a structural (topology based) approach augmented using functional information. We also discuss a purely functional approach.

5.1 A Structural Analysis Based Decomposition Approach

We observe empirically that the resources required for obtaining the ROBDDs for G_d as well as all $\psi_{i_{bdd}}$, for $1 \leq i \leq m$ are significantly less than the time spent constructing the function G. In our experience, any decomposition that is typically not located at either extreme of the circuits can be computed in a very short frame of time. Thus one can typically afford to experiment with various decompositions without paying an excessive space or time penalty.

The structural heuristic attempts to locate a suitable waist in the circuit representation. By the arguments detailed in Section 4, any waist close to the primary inputs or primary outputs is ignored. For the remaining circuit, also called a *candidate region*, there are two techniques that can be used to locate a set of candidate waists $\Psi_1, \ldots, \Psi_c$. The selection of candidate waists is made by analyzing their decomposed ROBDDs as explained in Section 4.

We construct the decomposed ROBDDs $G_d(\Psi)$ by a top-down approach, where the ROBDD at the output of a gate is progressively composed in terms of its input gate variables, and the composition proceeds recursively until we reach the given decomposition set. If during construction of some decomposed ROBDD $G_d(\Psi_i)$, for some candidate waist Ψ_i, one observes a sharp decrease in the peak graph size between two intermediate ROBDDs then there is indication of the presence of a region of functional simplification. A good decomposition will typically precede such an area of simplification.

We compute a cut set such that its members are present essentially at the same level. We begin by computing the *width profile* of the given circuit which is the size of the cutset at each level of the circuit (figure 3). The cutset size at level L is computed as $cutsize_L = jmp_L + g_L$, where jmp_L gives the number of *jump gates* at level L, that is, all gates at any level $p|p < L$ which fanout to some gate at any level $q|q > L$. g_L gives the total number of gates at level L.

We now compute the gradient of the circuit, which is computed as the rate at which the cutset size of the circuit decreases as we traverse the circuit from the level 1 to the primary output. A circuit may have a set of gradients characterizing different regions in it.

A waist is defined as the cutset in the candidate region whose width (1) is relatively small (2) decreases by a rate significantly greater than the average gradient of its surroundings. We generate a set of candidate waists and select the final waist by analyzing their decomposed ROBDDs as discussed earlier.

Simply selecting the minimum number of gates / wires at any level is not enough for computing the waist (for single outputs, the minimum number of gate / wires will always be at the primary output). For tree-like circuits, we can not obtain the "waist". Here we select the waist based on the gradient analysis.

Experimentally, selecting the decomposition points at 65% of the maximum level seems to work well. So, we try to compute the waist weighted by how close the level is to 65% of the maximum level.

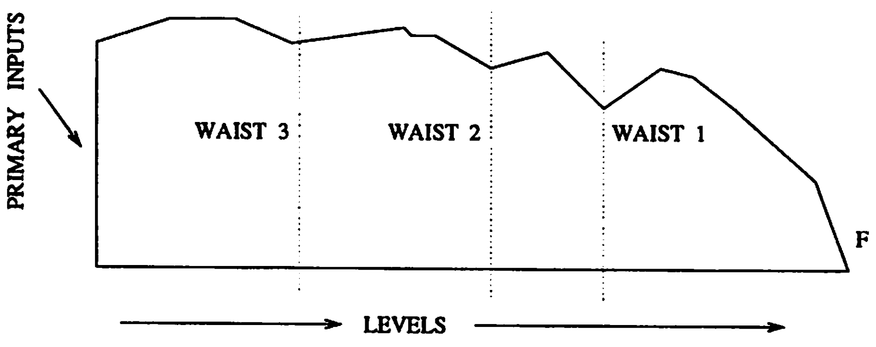

Figure 3: *Circuit width profile and "waists" for efficient decomposition*

5.2 A Functional Decomposition Approach

For certain boolean operations during the construction of the ROBDD of the output, the graph size increases by a *disproportionate measure*. We decompose the function at precisely these points. Decomposition is required when we are processing a function F which requires prohibitive computational resources.

Since the above decomposition point only captures instances of difficult functional manipulations, any simplification of graphs made on such decomposition variables will likely decrease total resource requirements as well. Also in the above scheme no decomposition is done when the function can be very easily processed without any memory explosion.

In our current implementation, we decompose a function when the total ROBDD manager size increases by more than a presettable factor due to some Boolean operation. This check is only done if the manager size is larger than a predetermined minimum. Also, decomposition points are added when the individual ROBDD grows beyond another threshold value.

In this approach, decomposition points are frequently nested inside another decomposition point. To decrease total number of compositions required, we compose all such decomposed variables which are not nested within any other similar variable. This method of composition is now recursively applied till a fully composed graph is obtained.

6 Results

In this section we discuss the implementation results of our structural and functional decomposition methods. We use the ISCAS85 benchmark circuits. Results are reported for the outputs which are considered 'hard' for building ROBDDs. The implementation is done in the SIS [23] environment. Results are obtained on DEC station5000/260 with 128MB of memory.

The "size" entry in all the following tables refers to the maximum ROBDD manager size. Table 1 and Table 2 show the results of the structural decomposition techniques. Tables 3 4 5 and 6 show the results of the functional decomposition approach. We use different variable ordering schemes to show that our approach works equally well under various schemes. Table 1 uses Malik's ordering [19](MO) without dynamic variable (DR) reordering[22]. Table 2 uses MO along with DR. Table 3 uses natural input ordering (NO) without DR. Table 4 uses MO without DR while Tables 5 6 use MO and NO respectively with DR.

In all the tables, the first column shows the peak memory utilization and the time taken by the reference algorithm. The reference algorithm builds the monolithic ROBDD of the outputs in terms of the primary inputs in a depth first order. If an ROBDD of a node is not required in subsequent computations it is freed to reduce the memory requirements.

The second column in Tables 1 and Table 2 (denoted by "XTop") gives the results for our algorithm where we do the local refinements of the best cutset to obtain a small decomposed graph (G_d). The third column presents the best results obtained after considering three different candidate cutsets and "XTop". In almost all the cases in Tables 1 and Table 2, the "XTop" algorithm provides better results than the reference algorithm. Overall memory gains are about 40% and the time gains are about 22% averaged over all examples. The reference algorithm is unable to make the ROBDD for the 13th output of C6288 (multiplier) in 500K nodes while the XTop algorithm finishes it in less than 28 seconds. In the absence of DR (Table 1), "XTop" usually gives the best results. When DR is invoked, "XTop" gives the best result less often. This is because of the fact that a small G_d selected under one variable ordering may not be small under a different variable ordering. Despite this, the "XTop" algorithm performs satisfactorily compared to the reference algorithm with substantial gains in both time and memory (22% gain in memory and 30% gain in time). The few cases where the "XTop" algorithm performs poorly compared to the reference algorithm are typically those where the intermediate memory requirements are small.

Ckt	Out	Reference		"XTop"		Best Waist		% gain, XTop		% gain, B. Waist	
	#	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size
C880	26	3.36	23818	2.53	21246	2.44	21194	24.70	10.84	27.38	11.05
C1355	32	1.82	25482	1.14	13223	1.14	13223	37.36	48.11	37.36	48.11
C1908	24	5.80	34205	3.38	18102	3.38	18102	41.72	47.02	41.72	47.02
C1908	25	4.29	18288	4.36	17189	4.20	17185	01.63	6.01	2.1	6.03
C2670	140	-	-	-	-	-	-	-	-	-	-
C3540	22	-	-	-	-	-	-	-	-	-	-
C5315	123	28.80	300035	17.47	174632	17.47	174632	39.34	41.80	39.34	41.80
C6288	13	-	-	27.18	108538	27.18	108538	inf	inf	inf	inf
C6288	12	39.78	289344	12.73	65978	12.27	65978	68.00	77.20	69.16	77.20
Total		83.85	691172	68.79	418908	68.08	418852	21.89	39.39	23.16	39.40

Table 1: Structural Methods, Malik Ordering without dyn. reorder, 500K limit

In Tables 3, 4, 5 and 6, the first column is again for the reference algorithm while the second column represents the time and memory requirements for the functional decomposition approach. Once again, excellent time and space improvements are observed in almost all cases. Average improvements in time and memory are between 40%-70%. For 2 cases in Table 3, 1 case in Table 4 and 1 case in Table 5, our

Ckt	Out	Reference		"XTop"		Best Waist		% gain, XTop		% gain, B. Waist	
	#	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size
C880	26	7.15	4598	8.29	4530	7.12	4764	-15.94	1.48	0.42	-3.61
C1355	32	137.37	32600	139.02	29360	137.53	32600	-1.20	9.94	-0.12	0.00
C1908	24	31.0	8863	15.24	8876	14.82	8866	50.84	-0.15	52.19	-0.03
C1908	25	14.59	8870	14.93	8887	14.72	8873	-2.33	-0.19	-0.89	-0.03
C2670	140	127.53	12609	434.72	111907	130.39	13065	-240.88	-787.52	-2.24	-3.62
C3540	22	2446.89	311658	2392.94	285800	2333.73	270873	2.20	8.30	4.62	13.09
C5315	123	25.99	9934	6.87	9601	6.25	9530	73.57	3.35	75.95	4.07
C6288	12	1781.4	286992	155.10	63904	142.92	62524	91.29	77.73	91.98	78.21
Total		4571.92	676124	3167.11	522865	2787.48	411095	30.73	22.67	39.03	39.20

Table 2: Structural Methods, Malik Ordering with dyn. reorder, 500K limit

algorithm builds the output where the reference algorithm fails (for a 1 million node limit). Note that the % improvement figures do not reflect these outputs. Therefore the actual gains are even better than what these numbers suggest. Interestingly, space and time improvements are more prominent for bigger circuits and circuits which are known to be difficult for ROBDDs (like C6288).

Ckt	Out #	Reference Algo.		Functional		%gain, Functional	
		Time	Size	Time	Size	Time	Size
C880	26	9.56	120236	2.44	49405	74.48	58.91
C1908	24	3.58	25546	4.77	44082	-33.24	-72.56
C1908	25	5.57	30587	3.78	29139	32.14	4.73
C2670	140	–	>1000000	–	>1000000	-	-
C3540	21	–	>1000000	173.11	775137	inf	inf
C3540	22	129.23	647893	87.28	301731	32.46	53.43
C6288	12	39.55	261156	11.16	140414	71.78	46.23
C6288	13	110.36	761008	30.86	324514	72.04	57.36
C6288	14	–	>1000000	85.06	764686	inf	inf
Total		298.15	1546426	140.29	889285	52.95	42.49

Table 3: Functional Methods, Natural Ordering without dyn. reorder, 1 Million node limit

Ckt	Out #	Reference Algo.		Functional.		%gain, Functional	
		Time	Size	Time	Size	Time	Size
C880	26	0.20	6467	0.19	6468	5.00	-0.02
C1908	24	0.86	11307	0.63	12276	26.74	-8.57
C1908	25	1.01	13904	0.72	12059	28.71	13.27
C2670	140	1.40	29282	1.33	15590	5.00	46.76
C3540	22	–	>1000000	–	>1000000	-	-
C6288	12	30.59	268076	13.90	160498	54.56	40.13
C6288	13	64.57	633423	37.28	378995	42.56	40.17
C6288	14	–	>1000000	109.66	878639	inf	inf
Total		98.63	962459	54.05	585886	45.20	39.13

Table 4: Functional Methods, Malik Ordering without dyn. reorder, 1 Million node limit

Ckt	Out #	Reference Algo.		Functional.		%gain, Functional	
		Time	Size	Time	Size	Time	Size
C880	26	6.45	9871	2.53	10002	60.78	-1.33
C1908	24	6.22	12111	8.99	9635	-44.53	20.44
C1908	25	3.94	9879	3.88	11025	1.52	-11.60
C2670	140	121.72	12864	377.22	23141	-209.91	-79.89
C3540	22	269.48	83760	496.93	102904	-84.40	-22.86
C6288	12	1224.77	248453	218.75	76881	82.14	69.06
C6288	13	3109.29	727006	969.46	135894	68.82	81.31
C6288	14	–	>1000000	2734.45	217759	inf	inf
Total		4741.87	1103944	2077.76	375782	56.18	65.96

Table 5: Functional Methods, Natural Ordering with dyn. reorder, 1 Million node limit

Ckt	Out #	Reference Algo.		Functional.		%gain, Functional	
		Time	Size	Time	Size	Time	Size
C880	26	0.29	6467	0.44	6468	-51.72	-0.02
C1908	24	1.71	11307	1.37	12276	19.88	-8.57
C1908	25	11.12	10679	5.64	9795	49.28	8.28
C2670	140	49.76	9566	52.37	9578	-5.25	-0.13
C3540	22	2856.73	331653	1572.94	178436	44.94	46.20
C6288	12	1821.21	280845	726.07	77775	60.13	72.31
C6288	13	7237.81	735401	1378.26	133426	80.96	81.86
C6288	14	–	>1000000	–	>1000000	-	-
Total		11978.63	1385918	3737.09	427754	68.80	69.14

Table 6: Functional Methods, Malik Ordering with dyn. reorder, 1 Million node limit

7 Conclusion

In this paper we have studied the problem of efficient ROBDD construction using decomposition based techniques. The problem was to construct the ROBDD of a target function in terms of the primary inputs so that the intermediate peak memory requirement and the time taken for the ROBDD construction is small. In our approach, we first suitably decompose the given function and construct the ROBDD of the target function in terms of the decomposed variables. Next, we compose the decomposed variables to get the final ROBDD. The main conclusions of our work are as follows:

- We have suitably combined the bottom-up and top-down approaches of building ROBDDs to circumvent the memory explosion problem commonly encountered while constructing ROBDDs.
- Two new schemes for determining decompositions, one based on structural information, and another on functional information have been presented.
- The structural approach performs a topological analysis of the circuit to find good decomposition points.
- Functional approach makes no underlying assumption about the topology of the circuit in finding decomposition points. This approach is applicable to any arbitrary set of ROBDD operations and can be easily incorporated into any existing general purpose ROBDD package.
- Results on ISCAS85 benchmark circuits show excellent improvements over the conventional bottom-up procedures of building ROBDDS.
- Results are reported for different variable ordering schemes including dynamic ordering and are consistently better than the conventional method.
- In many cases, our method was able to build ROBDDs where the existing schemes failed.
- We show impressive reductions in both memory and time utilization in many cases where existing schemes are able to build the ROBDD. This is starkly in contrast to dynamic variable ordering where a significant time penalty is associated with the memory reduction.

- Gains in time and memory are more prominent for examples which are particularly bad for ROBDDs, like multipliers.

Future research is directed towards identifying better decomposition techniques and studying various computational issues in efficient function composition. Also, we plan to use this approach to reduce the resources required in computing transition relations, reachability, and also for alternate BDD approaches.

References

[1] Sheldon B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27:509–516, June 1978.

[2] P. Ashar, A. Ghosh, and S. Devadas. Boolean satisfiability and equivalence checking using general binary decision diagrams. *ICCD*, pages 259-264, October 1991.

[3] C. L. Berman. Circuit width, register allocation, and ordered binary decision diagrams. *IEEE Transactions on Computer-Aided Design*, pages 1059–1066, August 1991.

[4] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *Proc. of the Design Automation Conf.*, pages 40–45, June 1990.

[5] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. *27th Design Automation Conference*, pages 40–45, 1990.

[6] R. E. Bryant. Graph based algorithms for Boolean function representation. *IEEE Transactions on Computers*, C-35:677–690, August 1986.

[7] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24:293–318, September 1992.

[8] G. Cabodi, P. Camurati, and Stefano Quer. Auxillary variables for extending symbolic traversal techniques to data paths. *31st Design Automation Conference*, pages 289-293, 1994.

[9] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski. Efficient representation and manipulation of switching functions based on ordered kronecker functional decision diagrams. *31st Design Automation Conference*, pages 415–419, 1994.

[10] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvements of Boolean comparison method based on binary decision diagrams. *ICCAD*, pages 2–5, 1988.

[11] J Gergov and Ch. Meinel. Efficient analysis and manipulation of typed free bdds can be extended to read-once-only branching programs. *Tech. Report 92-10, University of Trier. Also to appear in IEEE Transaction on Computers, June 1995*, 1992.

[12] J. Jain, M. Abadir, J. Bitner, D. S. Fussell, and J. A. Abraham. IBDDs: An efficient functional representation for digital circuits. *European Design Automation Conference*, March 1992.

[13] J. Jain, J. Bitner, D. S. Fussell, and J. A. Abraham. Probabilistic verification of Boolean functions. *Formal Methods in System Design*, 1, 1992.

[14] S.-W. Jeong, B. Plessier, G. Hachtel, and F. Somenzi. Structural BDDs: Trading canonicity for structure in verification algorithms. *ICCAD*, 1991.

[15] Kevin Karplus. Using if–then–else dag's for multi–level minimization. *Decennial Caltech Conference on VLSI*, May 1989.

[16] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. *European Design Automation Conference*, pages 43–47, 1992.

[17] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Syst. Tech. J.*, 38:985–999, 1959.

[18] J. C. Madre and J. P. Billon. Proving circuit correctness using formal comparison between expected and extracted behavior. *25th Design Automation Conference*, pages 205–210, 1988.

[19] 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.

[20] K. L. McMillan. Symbolic model checking: An approach to the state explosion problem. *Ph.D Thesis, Dept. of Computer Sciences, Carnegie Mellon University*, 1992.

[21] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. *ICCAD*, pages 42–47, 1993.

[22] R. L. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams . In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 42–47, November 1993.

[23] 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.