Partitioned ROBDDs - A Compact, Canonical and Efficiently Manipulable Representation for Boolean Functions

Amit Narayan[1] Jawahar Jain[2] M. Fujita[2] A. Sangiovanni-Vincentelli[1]

Abstract

We present a new representation for Boolean functions called Partitioned-ROBDDs. In this representation we divide the Boolean space into 'k' partitions and represent a function over each partition as a separate ROBDD. We show that partitioned-ROBDDs are canonical and can be efficiently manipulated. Further, they can be exponentially more compact than monolithic ROBDDs and even Free BDDs. Moreover, at any given time, only one partition needs to be manipulated which further increases the space efficiency.

In addition to showing the utility of partitioned-ROBDDs on special classes of functions, we provide automatic techniques for their construction. We show that for large circuits our techniques are more efficient in space as well as time over monolithic ROBDDs. Using these techniques, some complex industrial circuits could be verified for the first time.

1 Introduction

A large number of problems in VLSI-CAD and other areas of computer science can be formulated in terms of Boolean functions. A central issue in providing computer-aided solutions to these problems is to find a compact representation for Boolean functions on which the basic Boolean operations and equivalence check can be efficiently performed.

The requirements of compactness and manipulability are generally conflicting. Currently, Reduced Ordered Binary Decision Diagrams (henceforth, ROBDDs) serve as the most popular compromise between these conflicting requirements. ROBDDs are canonical and efficiently manipulable. For many practical functions ROBDDs are compact as well. Due to these nice properties, ROBDDs are frequently used as the Boolean representation of choice to solve various CAD problems. Unfortunately, ROBDDs are not always very compact; in a large number of cases of practical interest, any ROBDD representation of a Boolean function may require space which is exponential in the number of primary inputs (PIs). This large space requirement places a limit on the complexity of problems which can be solved using ROBDDs.

Various methods have been proposed to improve the compactness of ROBDDs. Some of the methods that achieve compactness do so by sacrificing almost all the nice properties of ROBDDs. Other methods, which maintain canonicity and manipulability, represent the function over the entire Boolean space as a single graph rooted at a unique source. In this paper, we claim that this is an unnecessary restriction.

We show that exponentially more compact representations can be obtained by partitioning the Boolean space and representing the functionality over each partition as a separate graph. This compactness in representation is achieved without sacrificing the desirable properties of the underlying graph which is used to represent each partition. This notion of *functional* partitioning is general and can be applied to other problems also. In this paper we demonstrate its utility for graph based representations of Boolean functions, and in particular, for ROBDDs.

We define partitioned-ROBDDs and show that they are canonical. We prove that partitioned-ROBDDs can give exponentially compact representations compared to monolithic ROBDDs and even Free BDDs (FBDDs). At any given time only one partition needs to be present in the memory. Different partitions can be processed independently and can employ different variable orderings. However, the compactness of representation comes not just from using different orderings for different partitions. We show that partitioned-ROBDDs can be exponentially more compact even if all partitions have the same variable order. We also show that there are functions for which even FBDDs are exponential but partitioned-ROBDDs are polynomial. Therefore, the class of functions that can be represented in polynomial space by Partitioned-ROBDD is distinct from the class of polynomially sized FBDDs. We also show that partitioned-ROBDDs are efficiently manipulable in addition to being canonical and compact. We consider some applications of partitioned-ROBDDs in formal verification of combinational and sequential circuits.

In addition to theoretically proving the properties of partitioned-ROBDDs, we demonstrate their utility on some large industrial circuits. We discuss methods of automatically generating partitioned-ROBDDs. The effectiveness of our technique is perhaps best demonstrated from the fact that we were able to verify some circuits on which all previously known techniques had failed.

2 Previous Work

In the following, we assume that the reader is familiar with the basic BDD terminology [1, 6]. A **Free BDD** (FBDD) is a BDD in which variables can appear only once in a given path from the source to the terminal; different paths can have different variable orderings. An **Ordered** BDD is an FBDD with the additional restriction that variables follow a common ordering in all paths from the source to the terminal. If no two nodes in an OBDD represent the same function then it is said to be a **Reduced** OBDD. In [6] Bryant showed that ROBDDs are canonical and can be easily manipulated. For further details on ROBDDs, and the implementation of a typical ROBDD package, please refer to [5, 6, 8].

The size of an ROBDD is strongly dependent on its ordering of variables. Many algorithms have been proposed to determine good variable orders [21, 27, 28]. Partitioned-ROBDDs enhance the effectiveness of variable ordering methods by allowing different partitions to have different orderings.

Another approach to reduce the space requirement of ROBDDs is to relax their total ordering requirement. FBDD is an example of this category in which different paths from the root to the terminal can have variables in different orders. We will show that partitioned-ROBDDs can be exponentially more compact than FBDDs. Further, the class of polynomially sized partitioned-FBDDs (defined analogously to partitioned-ROBDDs) strictly contains the class of polynomially size monolithic FBDDs.

A third approach to obtain a more compact representation for Boolean functions is to change the function decomposition associated with the nodes. Instead of using a decomposition based on the Shannon Expansion in which a function f is expressed as $\overline{x}f_{\overline{x}} + x f_x$, some other decomposition like the Reed-Muller expansion or a hybrid of different expansions is used (e.g. Functional Decision Diagrams (FDDs) [19], and Ordered Kronecker Functional Decision Diagrams (OKFDDs) [12]). These representations can also be augmented using our concept of partitioning.

All of the above mentioned methods represent a function over the entire Boolean space as a single graph (rooted at a unique source). Our approach to achieve compactness is fundamentally different. We create partitions of the Boolean space and represent the function over different partitions of the Boolean space as a separate graph. Any underlying data structure can be used to represent the function over different partitions, although in this paper we restrict our attention mainly to ROBDDs and to some extent FBDDs.

The idea of partitioning has been used in the context of digital circuit verification in the past. In [15], it was shown that many functions discussed

[1]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720

[2]Fujitsu Labs of America, Santa Clara, CA 95054

ICCAD '96
1063-6757/96 $5.00 © 1996 IEEE

in [2, 7, 13] can be represented in space polynomially bounded in the number of inputs when subfunctions are ordered independently. Circuits were verified by partitioning them into these subfunctions. Our representation uses a notion similar to the functional partitioning of [15]. However, most of the techniques suggested in [15] exploit specialized structural knowledge and cannot be automated. Also, the partitioned-ROBDD data structure for representing general Boolean functions was not adequately developed in [15].

In [9], the transition relation of a given finite state machine was expressed as either a disjunction (using an interleaved model) or a conjunction (using a synchronous model) of ROBDDs representing individual outputs and latches. The notion of partitioning was restricted to building the ROBDDs of the outputs and latches separately. Our notion of partitioning is much more general than that of building the ROBDDs of different outputs separately. In Section 4 we will discuss the application of partitioned-ROBDDs to represent the transition relation.

3 Partitioned-ROBDDs

Assume that we are given a Boolean function $f: B^n \rightarrow B$, defined over n inputs $X_n = \{x_1, \ldots, x_n\}$. We define the partitioned-ROBDD representation, χ_f, of f as follows:

Definition 1 *Given a Boolean function $f: B^n \rightarrow B$ defined over X_n, a partitioned-ROBDD representation χ_f of f is a set of k function pairs, $\chi_f = \{(w_1, \tilde{f}_1), \ldots, (w_k, \tilde{f}_k)\}$ where, $w_i : B^n \rightarrow B$ and $\tilde{f}_i : B^n \rightarrow B$, for $1 \leq i \leq k$, are also defined over X_n and satisfy the following conditions:*

- *1. w_i and $\tilde{f}_i$ are represented as ROBDDs with the variable ordering π_i, for $1 \leq i \leq k$.*
- *2. $w_1 + w_2 + \ldots + w_k = 1$*
- *3. $\tilde{f}_i = w_i \wedge f$, for $1 \leq i \leq k$*

Here, $+$ and $\wedge$ represent Boolean OR and AND respectively. The set $\{w_1, \ldots, w_k\}$ is denoted by W.

Each w_i is called a *window function*. Intuitively, a window function w_i represents a part of the Boolean space over which f is defined. Every pair $(w_i, \tilde{f}_i)$ represents a partition of the function f. Here we are not using the term "partition" in the conventional sense where partitions have to be disjoint. If in addition to Conditions 1-3 in Definition 1, we also have that $w_i \wedge w_j = 0$ for $i \neq j$ then the partitions are said to be **orthogonal**; clearly, each $(w_i, \tilde{f}_i)$ is now a partition in the conventional sense.

Condition 1 in Definition 1 states that each partition has an associated variable ordering which may or may not be different from the variable orderings of other partitions. Condition 2 states that the w_is cover the entire Boolean space and Condition 3 states that $\tilde{f}_i$ is the same as f over the Boolean space covered by w_i. In general, each $\tilde{f}_i$ can be represented as $w_i \wedge f_i$; the value of f_i is a *don't care* for the part of the Boolean space not covered by w_i. The size of an ROBDD F is denoted by $|F|$. Then the sum of the sizes of all partitions, denoted by $|\chi_f|$, is given by $|\chi_f| = (|\tilde{f}_1| + \ldots |\tilde{f}_k| + |w_1| + \ldots + |w_k|)$. From Conditions 2 and 3, it immediately follows that:

$$f = \tilde{f}_1 + \tilde{f}_2 + \ldots + \tilde{f}_k \tag{1}$$

This type of partitioning in which f is expressed as a disjunction of $\tilde{f}_i$s is called a **disjunctive** partition. A **conjunctive** partition can be defined as the dual of the above definition. That is, the i^{th} partition is given by $(w_i, \tilde{f}_i)$, Condition 2 in Definition 1 becomes $w_1 \wedge \ldots \wedge w_k = 0$, and Condition 3 becomes $\tilde{f}_i = w_i + f$. In this case $f = (\tilde{f}_1 \wedge \ldots \wedge \tilde{f}_k)$.

3.1 Canonicity of Partitioned-ROBDDs

It is easy to see that for a given set $W = \{w_1, \ldots, w_k\}$ and a given ordering π_i for every partition i, the partitioned-ROBDD representation is canonical. In Theorem 3.1 we show that for a given function f and a given partitioned-ROBDD representation $\chi_f = \{(w_i, \tilde{f}_i) | 1 \leq i \leq k\}$ of f, $\tilde{f}_i$ is unique. Since each $\tilde{f}_i$ is represented as an ROBDD which is canonical (for a given ordering π_i), from Theorem 3.1 it follows that the partitioned-ROBDD representation is canonical.

Theorem 3.1 *Given functions $f: B^n \rightarrow B$ and $g: B^n \rightarrow B$ both defined over X_n, let $\chi_f = \{(w_i, \tilde{f}_i) | 1 \leq i \leq k\}$ and $\chi_g = \{(w_i, \tilde{g}_i) | 1 \leq i \leq k\}$ be the partitioned-ROBDD representations of f and g respectively, i.e., χ_f and χ_g satisfy Conditions 1-3 of Definition 1. Then $f = g$ iff $\tilde{f}_i = \tilde{g}_i$ for $1 \leq i \leq k$.*

3.2 Compactness of Partitioned-ROBDDs

In Section 3.2.1 we show examples where partitioned-ROBDDs are exponentially more compact than monolithic ROBDDs. We consider two cases: in the first case, the compactness in the representation is obtained by using the flexibility of independently ordering different partitions while in the second case we show an example where the partitioned-ROBDD representation is exponentially smaller than the corresponding monolithic ROBDD even though all partitions have the same order. In Section 3.2.2 we show that the partitioned-ROBDD representation can be exponentially more compact than an FBDD. In Section 3.2.3 we generalize the notion of partitioned-BDDs for other classes of BDDs (like FBDDs, typed-FBDDs [14] etc.) and make some observations about their compactness. In Section 3.2.4 we review the relationship between the VLSI complexity theory and ROBDDs [7] and discuss some reasons for the compactness of partitioned-ROBDDs.

3.2.1 Monolithic ROBDDs vs. Partitioned-ROBDDs

A straightforward way of constructing examples of Boolean functions with polynomially sized partitioned-ROBDDs and large monolithic ROBDDs is to exploit the flexibility of independently ordering different partitions available in the case of partitioned-ROBDDs. Consider the following function:

$$f(x_1, \ldots, x_n) = x_1 f_1(x_2, \ldots, x_n) + \overline{x}_1 f_2(x_2, \ldots, x_n) \tag{2}$$

Assume that f_1 and f_2 have small ROBDDs for orderings π_1 and π_2 respectively but exponentially sized ROBDDs for orderings π_2 and π_1. The ROBDD representing f will be exponential under both orderings (x_1, π_1) and (x_1, π_2). In contrast the partitioned-ROBDD representation $\chi_f = (x_1, x_1 \wedge f_1), (\overline{x}_1, \overline{x}_1 \wedge f_2)$ is polynomial (under the orderings (x_1, π_1) for partition 1 and (x_1, π_2) for partition 2). Now consider 2^k functions, where $k = O(log_2 n)$, $f_1, f_2, \ldots f_{2^k}$ having polynomially sized ROBDDs such that they do not have any good variable ordering in common, i.e. under all orderings at least one of the 2^k functions becomes exponential. If we combine these 2^k functions with a multiplexor tree of k variables, the resulting function will have exponentially sized ROBDD for any ordering. On the other hand, since the partitioned-ROBDD representation keeps $f_1, f_2, \ldots f_{2^k}$ separately and can employ different orderings in different partitions, it will be polynomial. In this way we can get a large class of functions for which the partitioned-ROBDDs are exponentially more compact than the corresponding monolithic ROBDDs; FHS function of [13] belongs to this class [15].

In the above example we have utilized the flexibility of independently ordering different partitions to achieve compactness of representation. One can show that there are functions for which partitioned-ROBDDs employ the same orderings in different partitions but are still exponentially more compact than monolithic ROBDDs. One such example is the Hidden Weighted Bit function (HWB). This function has n inputs: $X_n = \{x_1, \ldots, x_n\}$. For an input assignment $x = (x_1 x_2 \ldots x_n)$, its "weight" is defined as the number of inputs set to 1, i.e., $wt(x) = |\{x_i | 1 \leq i \leq n, \text{and } x_i = 1\}|$. The HWB function selects the ith input if $wt(x) = i$, i.e.,

$$HWB(x) = \begin{cases} 0 & \text{if } wt(x) = 0 \\ x_{wt(x)} & \text{if } wt(x) > 0 \end{cases}$$

It was shown in [7] that any ROBDD representation of the HWB function requires $\Omega(1.14^n)$ vertices. We will show that the partitioned-ROBDD representation of the HWB function is polynomial [15]. A circuit implementation for a seven input HWB function is shown in Figure 1. The circuit inputs are fed as data into both a balanced tree of adders and a balanced tree of 2-input multiplexors. The adder tree computes the binary representation of $wt(x)$. There are $k = O(log_2 n)$ wires (labeled as ψ_1, ψ_2, and ψ_3 in Figure 1) which serve as the control signal to the multiplexor tree, causing the value

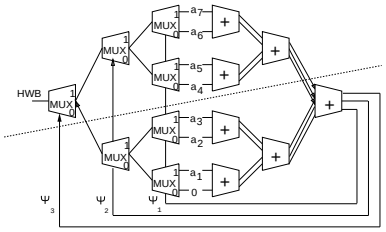

Figure 1: *A Circuit implementation for HWB(7)*

$x_{wt(x)}$ to appear at the output. Consider the following partitioned-ROBDD representation of HWB(n):

$$\chi_{HWB} = \{(w_0, 0),\ (w_1, w_1 \wedge x_1),\ \ldots,\ (w_n, w_n \wedge x_n)\} \qquad (3)$$

where $w_0 = \overline{\psi}_1\overline{\psi}_2 \ldots \overline{\psi}_k$, $w_1 = \psi_1\overline{\psi}_2 \ldots \overline{\psi}_k$, . . ., and $w_n = \psi_1\psi_2 \ldots \psi_k$.

For example, $\chi_{HWB(7)}$ shown in Figure 1 has the following 8 partitions: $(\overline{\psi}_1\overline{\psi}_2\overline{\psi}_3, 0)$, $(\psi_1\overline{\psi}_2\overline{\psi}_3, \psi_1\overline{\psi}_2\overline{\psi}_3 \wedge x_1)$, . . ., and, $(\psi_1\psi_2\psi_3, \psi_1\psi_2\psi_3 \wedge x_7)$. It is easy to see that χ_{HWB} defined in Equation 3 satisfies all the conditions of Definition 1 and is indeed a partitioned-ROBDD representation. In fact, χ_{HWB} is an *orthogonally* partitioned-ROBDD.

Since the number of control wires (labeled ψ_is) are $O(log_2 n)$, the total number of partitions are $O(2^{log_2 n}) = O(n)$. Further, for the ith window function w_i, $w_i = 1$ iff $wt(x) = i$. Since every w_i for $1 \leq i \leq n$ is a totally symmetric function of its inputs, the ROBDD representing w_i is $O(n^2)$[6]. Further, $f_i^{HWB} = x_i$ and is of size $O(1)$. Therefore, $\tilde{f}_i = w_i \wedge f_i$ is also of size $O(n^2)$ and the partitioned ROBDD representation of the HWB function is $O(n^3)$, i.e., it has n partitions each of size $O(n^2)$. Since the Boolean manipulation and verification algorithms keep only 1 partition in the memory at a given time, the maximum space complexity is only $O(n^2)$.

3.2.2 Monolithic FBDDs vs. Partitioned-ROBDDs

In this section we show that partitioned-ROBDDs can be exponentially more compact than FBDDs. This demonstrates that the flexibility obtained by representing a function as multiple graphs is more than that of just independently ordering different assignments.

Consider the function $f : B^{3n} \rightarrow B$ defined on $3n$ variables: $X_n = \{x_0, \ldots, x_{n-1}\}$, $Y_n = \{y_0, \ldots, y_{n-1}\}$, and $Z_n = \{z_0, \ldots, z_{n-1}\}$. Let $xyz = (x_0 \ldots x_{n-1}y_0 \ldots y_{n-1}z_0 \ldots z_{n-1}$ be a given assignment of the inputs variables and let $wt(x) = x_0 + x_1 + \ldots + x_n$, $wt(y) = y_0 + y_1 + \ldots + y_n$, and $wt(z) = z_0 + z_1 + \ldots + z_n$. We also define $e(x)$, $e(y)$, and $e(z)$ which are equal to 1 iff $wt(x)$, $wt(y)$, and $wt(z)$ respectively are even. Similarly, $o(x)$, $o(y)$, and $o(z)$ are 1 iff $wt(x)$, $wt(y)$, and $wt(z)$ respectively are odd. f is defined as follows:

$$f(x,y,z) = \begin{cases} 0 & \text{if } e(x) \wedge e(y) \wedge e(z) = 1 \\ z_{((wt(x)+wt(y))mod\ n)} & \text{if } e(x) \wedge e(y) \wedge o(z) = 1 \\ y_{((wt(x)+wt(z))mod\ n)} & \text{if } e(x) \wedge o(y) \wedge e(z) = 1 \\ y_{((wt(x)+wt(z))mod\ n)} & \text{if } e(x) \wedge o(y) \wedge o(z) = 1 \\ x_{((wt(y)+wt(z))mod\ n)} & \text{if } o(x) \wedge e(y) \wedge e(z) = 1 \\ z_{((wt(x)+wt(y))mod\ n)} & \text{if } o(x) \wedge e(y) \wedge o(z) = 1 \\ x_{((wt(y)+wt(z))mod\ n)} & \text{if } o(x) \wedge o(y) \wedge e(z) = 1 \\ 0 & \text{if } o(x) \wedge o(y) \wedge o(z) = 1 \end{cases}$$

It has been proved that any FBDD representation of the above function is exponential [4]. We will show that the partitioned-ROBDD representation is polynomial.

Consider the partitioned-ROBDD naturally suggested by the 8 orthogonal cases in the function definition; with the right column (such as $e(x) \wedge o(y) \wedge e(z)$) representing the window function and the left column (such as $y_{((wt(x)+wt(z))mod\ n)}$) denoting the corresponding $\tilde{f}_i$. It is easy to check that this χ_f satisfies conditions 1-3 of Definition 1 and is indeed a partitioned-ROBDD representation of f. Now we will show that χ_f is of polynomial size. By symmetry of the input variables we know that the ROBDDs representing $e(x)$, $e(y)$, $e(z)$, $o(x)$, $o(y)$, and $o(z)$ are $O(n^2)$ sized. Any w_i is a conjunction of three functions of size $O(n^2)$ with disjoint supports (for example, $w_1 = e(x)e(y)e(z)$). ROBDD representing the conjunction of two functions with disjoint supports can be obtained by concatenating the ROBDDs of the two operands, i.e., by directing all the edges going to 1 terminal in the first ROBDD to the source of the second ROBDD. Therefore, the size of each w_i is $O(3n^2)$. Now let us consider the size of f_is. Consider $f_2 = z_{(\ (wt(x)+wt(y))mod\ n\ )}$. It is easy to see that this function can be represented by an ROBDD of size $O(2n^2)$. This ROBDD will have the ordering in which the z variables come after x and y variables. The information about $wt(x) + wt(y)$ can be represented in the first $2n$ levels in $(2n)(2n+1)/2$ nodes. Then one out of the n bits is chosen based on the value of $(\ (wt(x) + wt(y))mod\ n\ )$. Therefore, the size of ROBDD representing f_2 is $((2n)(2n+1)/2 + n) = O(2n^2)$. Similarly it can be shown that every f_i is of size $O(2n^2)$. Since both w_i and f_i are polynomial, it follows that $\tilde{f}_i = w_i \wedge f_i$ is also polynomial. Also, there are a constant number (eight) of partitions for any n. Therefore, the overall size of χ_f is polynomial.

3.2.3 Partitioned-FBDDs and other Partitioned-BDDs

We can define a **partitioned-FBDD** representation, χ_f^{FBDD}, for a given function f by replacing Condition 1 in Definition 1 with the condition that w_i, and $\tilde{f}_i$ are represented by FBDDs. Similarly, a partitioned typed-FBDD can be defined by associating a type τ_i with every partition i in the definition of χ_f^{FBDD}.

Now the results of the previous sections can be summarized as follows. The class of functions representable by polynomially sized monolithic ROBDDs is strictly contained in the class of functions representable by polynomially sized partitioned-ROBDDs. The containment is obvious as monolithic ROBDDs are a special case of partitioned-ROBDDs with only one partition. The containment is strict as there are functions (such as FHS, HWB) which have polynomially sized partitioned-ROBDD representations but no polynomially sized monolithic ROBDD (see Section 3.2.1). Now, since ROBDDs are a special case of FBDDs (and also typed-FBDDs), it follows that partitioned-ROBDDs are a special case of partitioned-FBDDs (and also partitioned typed-FBDDs). Therefore, the function defined in Section 3.2.2 is an example for which FBDD (and hence typed-FBDD) is exponential but partitioned-FBDD (and also partitioned typed-FBDD) is polynomial. That is, the class of functions which have a compact FBDD representations is strictly contained in the class of functions which have a compact partitioned-FBDD representation and the class of polynomially sized typed-FBDD is strictly contained in the class of polynomially sized partitioned typed-FBDDs.

Similarly, we can generalize the notion of partitioning the Boolean space for any class of BDD representations (e.g. partitioned-OKFDD, partitioned-FDD, partitioned-ZBDD etc.). We conjecture that the compactness result of partitioned-ROBDDs holds for any class of canonical BDD representation, i.e., the class of functions having polynomial monolithic *-DD representation (where *-DD can be FDD, OKFDD, ZBDD [22], etc.) is strictly contained in the class of functions having polynomial **partitioned *-DD** representation.

3.2.4 ROBDDs, Partitioned-ROBDDs and VLSI Complexity Theory

In [7] an important relation was established between the size of ROBDD representations and the complexity of VLSI implementations. In classical VLSI complexity theory [29], the complexity of a Boolean function is expressed in terms of the product of AT^2, where A is a measure of the chip area needed to implement the function and T is a measure of the computation time. In [7] Bryant showed that if the area-time complexity of a function is quadratic then any ROBDD representing it will have exponential size. However, the converse is not always true. There can be functions with low area-time complexity ($AT^2 = O(n^{1+\epsilon})$, for $\epsilon > 0$) which have exponential ROBDDs under any possible ordering. The reason for this is that VLSI implementations can exploit two-way communication between variables while ROBDDs

cannot; in ROBDDs the information can flow only in one direction.

To understand this intuitively, let us again consider the implementation of the HWB function shown in Figure 1. For this implementation, the area-time complexity is $AT^2 = O(nlog^6 n) = O(n^{1+\epsilon})$ [7]. Now if we consider an imaginary line across the chip dividing the inputs into two equal halves, around $log_2 n$ bits of information crosses it in both directions. The VLSI implementation of HWB is efficient because of this two-way communication. If we restrict the information to flow only in one direction, then we will need to transfer $\Omega(n)$ bits of data across the partition; otherwise there would be no way to accurately determine which bit is required at the output. This is exactly what happens in the ROBDD representation of the HWB function. Since the same inputs serve as both the control and the data, $\Omega(n)$ bits of information needs to be represented (as distinct nodes of an ROBDD) before a decision can be made about the value of the function. Hence, the ROBDD is exponential.

The power of partitioned-ROBDDs lies in the fact that that they allow two way communication between variables without sacrificing canonicity or manipulability. In the case of HWB, this is achieved by partitioning the relevant information needed about the control signals (i.e. ψ_is) into disjoint cases (w_i's) and representing the value of the resulting functions (f_is) as separate ROBDDs. Since the number of disjoint cases is $O(n)$ and the relevant information, $wt(x)$, can be easily represented using ROBDDs in $O(n^2)$ space (due to symmetry), the partitioned-ROBDD representation takes only $O(n^3)$ space to represent all partitions.

This discussion leads to some interesting theoretical questions which are unanswered at present: 1) What techniques can be used to prove lower bounds for the size of partitioned-ROBDDs? Communication complexity based arguments used in the case of ROBDDs are not valid for partitioned-ROBDDs; 2) Are there functions with small area-time complexity (i.e. $AT^2 = O(n^{1+\epsilon})$ but exponential partitioned-ROBDDs? 3) Are there functions with quadratic area-time complexity (i.e. $AT^2 = O(n^2)$) but small partitioned-ROBDDs?

3.3 Boolean Manipulation Using Partitioned-ROBDDs

Given a partition of the Boolean space, $W = \{w_1, \ldots, w_k\}$, we will show that the asymptotic complexity of performing basic Boolean operations (e.g. NOT, AND, OR) on the partitioned-ROBDD representations is polynomial in the sizes of the operands; the same as ROBDDs. Therefore, the compactness of representation doesn't cost anything in terms of the efficiency of manipulation. In fact, since partitioned-ROBDDs are in general smaller than monolithic ROBDDs and each partition can be manipulated independently, their manipulation is also more efficient.

Theorem 3.2 *Let f and g be two Boolean functions and let $\chi_f = \{(w_i, \tilde{f}_i)|1 \leq i \leq k\}$ and $\chi_g = \{(w_i, \tilde{g}_i)|1 \leq i \leq k\}$ be their respective partitioned-ROBDDs satisfying Conditions 1-3 in Definition 1. Further assume that the ith partitions in both χ_f and χ_g have the same variable ordering π_i. Then, (a) $\chi_{\overline{f}} = \{(w_i, w_i \wedge \overline{\tilde{f}}_i)|1 \leq i \leq k\}$ is the partitioned-ROBDD representing $\overline{f}$ (i.e. NOT of f); and, (b) $\chi_{f \odot g} = \{(w_i, w_i \wedge (\tilde{f}_i \odot \tilde{g}_i))|1 \leq i \leq k\}$ is the partitioned-ROBDD representation of $f \odot g$ where $\odot$ represents any binary operation between f and g.*

3.3.1 Complexity of Operations

Given two ROBDDs F and G, the operation $F \odot G$ can be performed in $O(|F||G|)$ space and time. In partitioned-ROBDDs, different partitions are manipulated independently and the worst case *time* complexity of $f \odot g$ is $\Sigma_{i=1}^{k}(|\tilde{f}_i||\tilde{g}_i|)$ which is $O(|\chi_f||\chi_g|)$. Since only one partition needs to be in the memory at any time, the worst case space complexity is given by $\max_i(|\tilde{f}_i||\tilde{g}_i|)$ which is in general $\ll |\chi_f||\chi_g|$. Also, similar to ROBDDs, the size of the satisfying set of a function f can be computed in $O(|\chi_f|)$ for orthogonally partitioned-ROBDDs.

3.3.2 Existential Quantification

Besides the basic Boolean operations, another useful operation which is extensively used in formal verification of sequential circuits is the existential quantification ($\exists_x f$) operation. The existential quantification of variable x from the function f ($\exists_x f$) is given by $\exists_x f = f_x + f_{\overline{x}}$ where f_x and $f_{\overline{x}}$ are the positive and negative cofactors of f respectively. In the partitioned-ROBDD representation, the cofactors can be obtained easily by cofactoring each w_i and $\tilde{f}_i$ with respect to x, i.e., $\chi_{f_x} = \{(w_{i_x}, \tilde{f}_{i_x})|1 \leq i \leq k$, *and* $(w_i, \tilde{f}_i) \in \chi_f\}$ and $\chi_{f_{\overline{x}}} = \{(w_{i_{\overline{x}}}, \tilde{f}_{i_{\overline{x}}})|1 \leq i \leq k$, *and* $(w_i, \tilde{f}_i) \in \chi_f\}$. But after performing the cofactoring operation, the positive and negative cofactors have different window functions (given by w_{i_x} and $w_{i_{\overline{x}}}$ respectively) and the disjunction cannot be performed directly on the partitions. This problem doesn't arise if we choose window functions which do not depend on the variables that have to be quantified. We state the following theorem without proof:

Theorem 3.3 *Let $\chi_f = \{(w_i, \tilde{f}_i)|1 \leq i \leq k\}$ be a partitioned-ROBDD representation of f such that $\exists_x w_i = w_i$, for $1 \leq i \leq k$. Then $\chi_{\exists_x f} = \{(w_i, \exists_x \tilde{f}_i)|1 \leq i \leq k\}$ is the partitioned-ROBDD representation of $\exists_x f$.*

In many applications we do not need to explicitly represent the window functions. The only property that a partitioned representation is required to have is that f can be represented as the disjunction of the $\tilde{f}_i$'s. In such cases our partitioned representation, χ_f, of f is just a set of $\tilde{f}_i$ such that $f = \tilde{f}_1 + \ldots + \tilde{f}_k$. Since existential quantification distributes over $+$, it implies that $\exists_x f = \exists_x \tilde{f}_{1_x} + \ldots + \exists_x \tilde{f}_{k_x}$. So from the partitioned representation of f, we can directly get the partitioned representation of $\exists_x f$ by existentially quantifying x over each $\tilde{f}_i$; though the underlying window functions have changed. One important application of partitioned-ROBDDs which doesn't need information regarding the underlying window functions is the use of partitioned transition relations in sequential circuit verification [9]. We will discuss this application in more detail in Section 4.2.

3.3.3 Universal Quantification

Another important operation that is frequently used is the universal quantification of x from f (denoted by $\forall_x f$). A sufficient condition for universal quantification is that the window functions are *orthogonal* in addition to being independent of the variables to be quantified. We state the following theorem without proof:

Theorem 3.4 *Let $\chi_f = \{(w_i, \tilde{f}_i)|1 \leq i \leq k\}$ be a partitioned-ROBDD representation of f such that $\forall_x w_i = w_i$ and $w_i \wedge w_j = 0$ for $1 \leq i, j \leq k$ and $i \neq j$. Then $\chi_{\forall_x f} = \{(w_i, \forall_x \tilde{f}_i)|1 \leq i \leq k\}$ is the partitioned-ROBDD representation of $\forall_x f$.*

The additional restriction of orthogonality of partitions is not a very serious one in reality. In Section 5 we give effective heuristics to generate orthogonal partitions. For applications which do not need any information about the window functions, we can use conjunctive partitions (which were defined as the dual of disjunctive partitions in Section 3), and represent f as $f = \tilde{f}_1 \wedge \ldots \wedge \tilde{f}_k$. In this case we get $\forall_x f = \forall_x \tilde{f}_1 \wedge \ldots \wedge \forall_x \tilde{f}_k$.

4 Some Applications of Partitioned-ROBDDs

4.1 Combinational Verification

Partitioned ROBDDs can be directly applied to check the equivalence of two combinational circuits. Given two circuits f and g, we combine their respective outputs by an XOR gate to get a single circuit. Then we use partitioned ROBDDs to check whether the resulting circuit is satisfiable. For this we simply check whether $\tilde{f}_i \oplus \tilde{g}_i = 0$ for all partitions. In practice, this technique can be easily used as a back end to most implication based combinational verification methods [17, 26] which employ ROBDDs. The verification can be terminated even without processing all the partitions if in any window w_i the function $\tilde{f}_i \oplus \tilde{g}_i$ is found to be satisfiable.

Another way of verifying two circuits is to probabilistically check their equivalence [3, 16]. In probabilistic verification, every minterm of a function f is converted into an integer value under some random integer assignment ρ to the input variables. All the integer values are then arithmetically added to get the hash code $H_\rho(f)$ for f. One can assert, with a negligible probability of error, that $f \equiv g$ *iff* $H_\rho(f) = H_\rho(g)$. In the case of orthogonal partitions,

no two partitions share any common minterm. Hence, we can hash each partition separately, and just add their hash codes to obtain $H_\rho(f)$ [16]. This implies that to check if $H_\rho(f) = H_\rho(g)$, we can partition and hash both f and g independently. We do not need to keep both $\tilde{f}_i$ and $\tilde{g}_i$ in the memory at the same time. Further, it is not necessary that both f and g have the same window functions.

4.2 Sequential Verification

A key step in sequential circuit verification using ROBDDs is *reachability analysis* [10, 11] which consists of computing the set of states that a system can reach starting from the initial states. Given the present set of reached states, $R(s)$, and the transition relation for the system, $T(s, s^{'})$, relating present state variables, s, with the next state variables, $s^{'}$, the set of next states, $N(s')$, is evaluated using Equation 4

$$N(s') = \exists_s [T(s, s') \wedge R(s)] \tag{4}$$

The set of next states is added to the set of present states and the above computation is repeated until a fixed point is reached. This fixed point represents the set of all reachable states of the system.

In many cases, the ROBDDs representing the transition relation $T(s, s')$ become very large. To handle these cases, in [9] the notion of partitioned transition relations was proposed in which the transition relations of individual latches, $T_i(s, s^{'})$s, are represented separately (with some possible clustering of T_is). Two types of partitioned transition relations were discussed: conjunctive and disjunctive. In the conjunctive partitioning of [9], the transition relation is given by $T(s, s') = T_1(s, s^{'}) \wedge \ldots \wedge T_m(s, s^{'})$ where each T_i is represented as a separate ROBDD. This type of partitioning is a special case of our conjunctively partitioned-ROBDDs. Our notion of partitioning is more general since in our case the T_is need not always correspond to individual latches. The usefulness of the conjunctively partitioned transition relations of [9] is limited because existential quantification doesn't distribute over conjunctions. In the worst case, if all the T_i's depend on all the present state variables then the conjunctive partitions of [9] cannot be used at all.

A more interesting case is that of disjunctive partitions in which existential quantification distributes over the partitions. In the method of [9], the only way to get the disjunctive partitions is by using an interleaving model, in which only one wire is allowed to transition at a time. In general, this gives different results than a non-interleaving model and thus can be used only in some cases. In our approach, we can disjunctively partition the transition relation without having to place any restrictions on the underlying model of transition for a given system. In our case any set of f_is such that $T(s, s^{'}) = \tilde{f}_1 + \ldots + \tilde{f}_k$ can be used to represent the transition relation. The set of next states can be evaluated using the following equation:

$$N(s') = \exists_s (R(s) \wedge \tilde{f}_1) + \ldots + \exists_s (R(s) \wedge \tilde{f}_k) \tag{5}$$

This calculation can be performed by keeping only one $\tilde{f}_i$ for $1 \leq i \leq k$ in the memory. Notice that in the above calculation we do not need the window functions which correspond to f_is. (Please refer to the discussion in Section 3.3.2.)

4.3 Parallel Implementation of an ROBDD Package

Parallel algorithms for constructing ROBDDs were investigated in [20, 25]. Large communication requirements between different partitions limits their applicability. In [25] the ROBDD nodes are distributed among machines in a breadth-first manner. This method can at best give linear (in the number of partitions) reduction in space and time. In addition, dynamic reordering is limited only to the variables present in one partition. In contrast, our method of partitioning can give a superlinear reduction (even exponential) in the resources required to build ROBDDs. Further, each partition is independent and can be scheduled on a different processor with minimal communication overhead. Each partition can also be ordered independently and can exploit full power of dynamic reordering.

5 Heuristics for Constructing Partitioned-ROBDDs

The performance of partitioned-ROBDDs depends critically on our ability to generate good partitions of the Boolean space over which the function can be compactly represented. The issue of finding good partitions of the Boolean space is as central to the partitioned-ROBDD representation as the issue of finding good variable orderings is to monolithic ROBDDs. In this section we will discuss some simple heuristics which were found to be very effective in generating compact orthogonally partitioned-ROBDDs. Though we use a Boolean netlist model in the following discussion, our techniques are general and can be applied to any arbitrary sequence of Boolean operations.

In our approach we first decompose the given function F, and then obtain the window functions for creating its partitioned-ROBDD by analyzing the decomposed BDD for F. The number of windows is decided either *a priori* or dynamically. After a window w_i is decided, a partitioned-ROBDD corresponding to it is obtained by composing F in the Boolean space corresponding to the window w_i.

5.1 Creating a Decomposed Representation

Given a circuit representing a Boolean function $f: B^n \rightarrow B$, defined over $X_n = \{x_1 \ldots x_n\}$, our decomposition strategy consists of introducing new variables based on the increase in the ROBDD size during a sequence of ROBDD operations. We introduce a new variable whenever the total number of nodes in a ROBDD manager increases by a disproportionate measure due to some operation. For example, if while performing the operation $R = R_1 + R_2$ on ROBDDs R_1 and R_2 we find that R has become very large, we undo the operation. We introduce new variables ψ_1 and ψ_2 and express R as $\psi_1 + \psi_2$ We maintain a separate array which contains the ROBDDs corresponding to the decomposition points. We add R_1 and R_2 corresponding to the ψ_1 and ψ_2 to this array. In this way we postpone the instances of difficult functional manipulations to a later stage. Due to Boolean simplification many of these cases may never occur in the final result, especially if the final memory requirement is much less than the peak intermediate requirement [18].

In our current implementation, the check for memory explosion is done only if the manager size is larger than a predetermined threshold. Also, decomposition points are added when the ROBDD grows beyond another threshold value. This ensures that the decomposition points themselves do not have very large ROBDDs. We find that even a simple *size*-based decomposition scheme works quite effectively for demonstrating the potential of partitioned-OBDDs.

At the end of the decomposition phase we obtain a decomposed representation, $f_d(\Psi, X)$, of f where $\Psi = \{\psi_1, \ldots, \psi_k\}$ is called a *decomposition set* of the circuit and each $\psi_i \in \Psi$ is a *decomposition point*. Let $\Psi_{bdd} = \{\psi_{1_{bdd}}, \ldots, \psi_{k_{bdd}}\}$ represent the array containing the ROBDDs of the decomposition points, i.e., each $\psi_i \in \Psi$ has a corresponding ROBDD, $\psi_{i_{bdd}} \in \Psi_{bdd}$, in terms of primary input variables as well as (possibly) other $\psi_j \in \Psi$, where $\psi_j \neq \psi_i$. Similarly we represent the array of $\psi_{i_{bdd_{w_i}}}$ by $\Psi_{bdd_{w_i}}$. The composition [6] of ψ_i in $f_d(\Psi, X)$ is denoted by $f_d(\Psi, X).(\psi_i \leftarrow \psi_{i_{bdd}})$ where,

$$f_d(\Psi, X).(\psi_i \leftarrow \psi_{i_{bdd}}) = \overline{\psi_{i_{bdd}}}.f_{d_{\overline{\psi_i}}} + \psi_{i_{bdd}}.f_{d_{\psi_i}} \tag{6}$$

The vector composition of the Ψ in $f_d(\Psi, X)$ is denoted as $f_d(\Psi, X).(\Psi \leftarrow \Psi_{bdd})$ and represents successive composition of ψ_i's into f_d.

5.2 Partitioning a Decomposed Representation

5.2.1 Creating $\tilde{f}_i$ for a given w_i

Given a window function w_i, a decomposed representation $f_d(\Psi, X)$, and the ROBDD array Ψ_{bdd} of f, we want to find f_i such that the ROBDD representing $\tilde{f}_i = w_i \wedge f_i$ is smaller than f. Here we make the following observation:

Observation 1: Let $f_i = f_{d_{w_i}}(\Psi, X)(\Psi \leftarrow \Psi_{bdd_{w_i}})$ and $f = f_d(\Psi, X)(\Psi \leftarrow \Psi_{bdd})$. If w_i is a cube on PIs then $|f_i| \leq |f|$ for any given variable order for f and f_i.

Proof: We are given $f_i = f_{d_{w_i}}(\Psi, X)(\Psi \leftarrow \Psi_{bdd_{w_i}})$. If w_i depends only on PIs then the order of cofactoring and composition can be changed. Hence, $f_i = [f_d(\Psi, X)(\Psi \leftarrow \Psi_{bdd})]_{w_i}$. This gives, $f_i = f_{w_i}$. If w_i is a cube, then $|f_{w_i}| \leq |f|$ and hence $|f_i| \leq |f|$.

Now, given f_d, Ψ_{bdd} and w_is, we create the cofactors Ψ_{w_i} and $f_{d_{w_i}}$. Then by composing $\Psi_{bdd_{w_i}}$ in $f_{d_{w_i}}$, we get partition function $f_i = f_{w_i}$.

So given a set of window functions w_i, the partitioned-ROBDD χ_f of f is given by $\chi_f = \{(w_i, w_i \wedge f_{w_i}) | 1 \leq i \leq k\}$. It is easy to check that the above definition satisfies all the conditions of Definition 1

If w_i is a cube, f_i is guaranteed to have a smaller size than the ROBDD for f. Also, the ROBDD representing w_i has k internal nodes where k is the number of literals in w_i. Since w_i and f_{w_i} have disjoint support, $|\tilde{f}_i| = |w_i \wedge f_i| = (k + |f_i|) \approx |f_i|$. Also, as each intermediate result of building f_i will be smaller than that of building f, the intermediate peak memory requirement is also reduced.

Note that observation 1 doesn't hold in the presence of dynamic variable reordering when f and f_i can have different variable orderings. However, in practice since dynamic variable reordering is working on smaller graphs in the case of partitioning it is perhaps even more effective.

Even when the window function is a more complex function of PIs than a cube, we use $f_i = f_{w_i}$. Here f_{w_i} is the generalized cofactor of f on w_i. The generalized cofactor of f on w_i is generally much smaller than f. But in this case the size of the i^{th} partitioned-ROBDD $|\tilde{f}_i|$ can be $O(|w_i||f_i|)$ in the worst case. To avoid this, while using general window functions we use w_is which are small.

5.2.2 Selection of Window Functions

After deciding how to construct the partition function from a given window function we examine methods to obtain good window functions. These methods can be divided into two categories: a priori selection and "explosion" based selection.

A priori Partitioning: In this method we select a predetermined number of PIs to partition. If we decide to partition on 'k' PIs then we create 2^k partitions corresponding to all the binary assignments of these variables. For example, if we decide to partition on say x_1 and x_2 then we create four partitions x_1x_2, $x_1\overline{x_2}$, $\overline{x_1}x_2$ and $\overline{x_1}\,\overline{x_2}$. From the observation made in the previous section, we know that given window functions of this kind we can create partitioned-ROBDDs which are *guaranteed* to be smaller than the monolithic ROBDD. Since only one partition needs to be in the memory at a given time we will always win in space. In the results section we will see that the reduction in memory is large and is accompanied by an overall reduction in the time taken to process all partitions as well.

We want to select those variables which maximize the partitioning achieved while minimizing the redundancy that may arise in creating different partitions independently; a fundamental principle of any divide and conquer approach. For this we define the cost of partitioning a function f on variable x as

$$cost_x(f) = \alpha[p_x(f)] + \beta[r_x(f)] \tag{7}$$

where $p_x(f)$ represents the partitioning factor and is given by,

$$p_x(f) = max(\frac{|f_x|}{|f|}, \frac{|f_{\overline{x}}|}{|f|}) \tag{8}$$

and $r_x(f)$ represents the redundancy factor and is given by,

$$r_x(f) = \frac{|f_x| + |f_{\overline{x}}|}{|f|} \tag{9}$$

Notice that a lower partitioning factor is good as it implies that the worst of the two partitions is small and similarly a lower redundancy factor is good since it implies that the total work involved in creating the two partitions is less. The variable x which has the lower overall cost is chosen for partitioning.

For a given vector of functions $\mathcal{F}$ and a variable x, the cost of partitioning is defined as:

$$cost_x(\mathcal{F}) = \sum_{i=1}^{k} cost_x(f_i) \tag{10}$$

We order all the PIs in increasing order of their cost of partitioning f_d and Ψ and select the best 'k' (where 'k' is a predetermined number specified by the user). Note that using a similar cost function we can select not only PI variables but also psuedo-variables, such as a $\psi_{i_{bdd}}$ expressed in terms of PIs, to create partitioned-ROBDDs. In this case the cofactor operations become generalized cofactor operations for window functions which are non-cubes. This type of selection, where all the PIs are ranked according to their cost of partitioning f_d and Ψ, is called a $static$ partition selection. On the other hand, we can have a $dynamic$ partitioning strategy in which the best PI (say x) is selected based on f_d and Ψ and then the subsequent PIs are recursively selected based on f_{d_x} and Ψ_x in one partition and in $f_{d_{\overline{x}}}$ and $\Psi_{\overline{x}}$ in the other partition. The dynamic partitioning method will require an exponential number of cofactors and can be expensive. This cost can be somewhat reduced by exploiting the fact that the only values that we are interested in are the sizes of the cofactors of f_d and $\psi_{i_{bdd}}$s. An upper bound on the value of $|f_{d_x}|$ can be calculated by traversing the ROBDD of f_d and taking the $x = 1$ branch whenever the node with variable id corresponding to x is encountered. This method doesn't give the exact count as the BDD obtained by traversing the ROBDD in this manner is not reduced. The advantage is that no new nodes need to be created and the traversal is fast.

Explosion Based Partitioning: In this method we successively compose the $\psi_{i_{bdd}}$s in f_d. If the graph size increases drastically for some composition (say ψ_j), we select a window function, w, based on the current f_d and $\psi_{j_{bdd}}$. (The window functions are either a PI and its complement or some $\psi_{k_{bdd}}$ and its complement which is expressed in terms of PIs only and has a very small size.) Once the window function w, is obtained, we create two partitions $(w \wedge f_{d_w}, \Psi_w)$ and $(\overline{w} \wedge f_{d_{\overline{w}}}, \Psi_{\overline{w}})$ and recursively call the routine on each of the partitions.

5.3 Order of Composition

After we have selected a window function and created the decomposed representation for the i^{th} partition given by $f_{d_{w_i}}$ and Ψ_{w_i}, the final step is to compose Ψ_{w_i} in $f_{d_{w_i}}$, i.e., $f_{d_{w_i}}(\Psi, X)(\Psi \leftarrow \Psi_{bdd_{w_i}})$. Although, the final ROBDD size is constant for a given variable ordering, we observe that the intermediate memory requirement and the time for composition is a strong function of the order in which the decomposition points are composed.

For every candidate variable that can be composed into f_d, we assign a cost which estimates the size of the resulting composed ROBDD. The variable with the lowest cost estimate is composed. Once again a simple cost function based on the support set size was found to perform well in practice. Accordingly, we choose that decomposition variable which leads to the smallest increase in the size of the support set of the ROBDD after composition. At each step, we restrict the candidate ψs for composition to those decomposition points which are not present in any of the other ψ_{bdd}s. This guarantees that a decomposition variable needs to be composed only once in f_d. Further details can be found in [24]

6 Experimental Results

In the following we compare monolithic and orthogonally partitioned-ROBDD representations for some hard industrial circuits. In our experience, partitioned-ROBDDs turn out to be far superior to ROBDDs only when the ROBDD graph sizes grow quite large. We consider an ROBDD of around 100,000 nodes as large. Since, in the presence of dynamic ordering, large ROBDDs are not encountered in the ISCAS85 benchmark (except for multiplier), we devote the following space mainly to our experiments with industrial circuits.

Test Circuit Details: Our test circuits include various designs from industry such as data transfer buffers, data transfer controllers, hard-wired models for logic/fault simulation, and crossbar switch controllers. The sizes of the circuits are indicated through a triplet of the form {*# of PI, # of PO, # of gates*}.

Experimental Setup and Results: The partitioning method of Section 5 was implemented in a C program and the experiments were performed on Sparc-20 machines. For ROBDDs which were proving intractable on regular machines, we employed a Sparc-20 server machine with a 512MB RAM, and more than 2GB swap space. However, we were able to run the partitioned-ROBDD examples on regular workstations with 128 MB memory.

For smaller circuits (such as ISCAS85 circuits), decomposition points were introduced at graph sizes of 100 node or more and for the larger industrial circuits the graph decomposition size was set at 500 nodes. In our implementation, the program automatically controls the decomposition threshold as the size of the ROBDD manager increases. In the current implementation, number of partitions are decided a priori. The industrial experiments were

performed with either 8 partitions (OPU2, RC, RBC) or 32 partitions (OPU1, NINPTB).[1]

Since only one partition is needed at a time, the maximum memory required for partitioned-ROBDDs is given by the size of the largest partition. In the following, we demonstrate that we can construct the partitioned-ROBDD representation for many difficult industrial circuits for which building ROBDDs was much harder or not feasible at all - even on our Sparc-20 server. Some of these circuits are not sufficiently similar and hence the traditional verification techniques which exploit circuit similarity were found to fail. Using partitioned-ROBDDs we were able to verify these circuits for the first time.

In Table 1 we compare the time taken by ROBDDs against the total time in constructing *all* partitions. (The time is reported in seconds, and the space is reported in the number of ROBDD nodes.) Since at any given time we need only one partition, the ROBDD size in column 5 is compared with the size of the largest partition-ROBDD in column 6. Column 7 gives the sum of the sizes of all partitioned-ROBDDs. However, this is only an upper bound; if all the partitions are represented at the same time, then the resulting graph size may be smaller due to the sharing of identical graphs between different partitions.

Partial Verification Using Partitioning: For some hard cases we were not able to compactly represent the entire function even by partitioned-ROBDDs. In these cases we were able to construct a significant fraction of the function. For example, in NINPTB[C], we could construct 132 out of 256 partitions before we aborted the program execution due to time resource constraints. In this way we could analyze about 52% of the truth table. In contrast, monolithic ROBDDs aborted without giving any meaningful partial information. Needless to say, a simulation technique would also have been grossly inadequate in covering the given function. Similar results were obtained for MSWCN; a circuit too difficult for ROBDDs as well as partitioned-ROBDDs. Due to the difficult nature of this circuit, we tried constructing 5096 partitions and aborted the computation after one week. Though only 32 partitions could be constructed, at least some partial information about the function was obtained. Such partial function coverage is indicated by annotating the space and time entries by "(p)". When a design is erroneous, there is a high likelihood that the erroneous minterms are distributed in more than one partition and can be detected by processing only a few partitions. Our experience with erroneous circuits suggests that in almost all cases the errors can be detected by constructing only one or two partitions.

Control on the Success of Verification Experiments: Partitioned-ROBDDs allow a remarkable control on the space/time resources and functional-coverage. Thus, the success of a verification experiment can be ensured by changing the parameters of decomposition and the number of partitions that need to be created. While at present such control can be exercised only manually in our programs, this is to be contrasted with no or a very minimal control available in the case of ROBDDs.

Tautology Checking Using Partitioning: We have also implemented an ROBDD based tautology checker in the SIS environment. Given design and specification circuits F, G, the program checks the tautology for each corresponding output pair by analyzing the BDD $F_i \oplus G_i$. The results of our experiments can be found in [23].

Deficiencies and Possible Improvements: In most of our experiments, including the results reported here, the primary input based partitioning was found to perform the best. Although theoretically more general window functions can give up to an exponentially better performance, our current heuristics are not able to consistently discover them. Only OPU1[A] in Table 1 (marked as *) used a generalized window function. We are currently in the process of developing better heuristics which can exploit the full power of partitioned-ROBDDs.

We have observed that many times different partitions have significant similarities. In the present implementation, for a given set of window functions, we generate all partitions separately. The process of constructing Partitioned-ROBDDs can be made more efficient by identifying these partitions and combining them into one.

[1] For the intractable output NINPTB[C] we had to use 256 partitions, and for MSWCN 5096 partitions.

7 Conclusions

In this paper we have introduced an efficient data structure, partitioned-ROBDDs, to represent Boolean functions. We have shown that partitioned-ROBDDs are canonical as well as efficiently manipulable. We have also shown that they can be exponentially more compact than not only monolithic ROBDDs but even FBDDs. More over, at any given time, only one partition needs to be manipulated which further increases its space efficiency. Efficient parallel construction and a capability of earlier detection of inequivalence of two given functions are some additional advantages of Partitioned-ROBDDs.

In addition to theoretically showing the utility of partitioned-ROBDDs on special classes of circuits, we have given effective and automatic heuristics to construct them. We have performed extensive experiments on ISCAS85 circuits as well as some very hard industrial circuits and have shown that partitioned-ROBDDs can give orders of magnitude reduction in memory requirement over monolithic ROBDDs. This reduction in memory is also accompanied by a large reduction in the time required to construct partitioned-ROBDDs. Using our technique, some very hard real life circuits were verified for the first time. For circuits which were so hard that even our techniques could not fully verify them, we were able to do partial verification and achieve a significant vector coverage.

We believe that along with the issues of node decomposition, and variable ordering, partitioning should be considered as another dimension central to the representation of Boolean functions. Future research is directed towards solving some of the theoretical issues relating to partitioned-ROBDDs which were raised in this paper and developing more powerful heuristics for constructing them.

8 Acknowledgement

We would like to thank Ingo Wegener and Beatte Bollig for providing us the example in Section 3.2.2, and Claudionor Coelho for his assistance with this project. The first author was supported by CA State MICRO program grant #94-110 and SRC 95-DC-324.

References

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

[2] P. Ashar *et. al.* Boolean satisfiability and equivalence checking using general binary decision diagrams. *ICCD*, 1991.

[3] M. Blum *et. al.* Equivalence of free Boolean graphs can be decided probabilistically in polynomial time. *Inf. Proc. Letters*, 10, March 1980.

[4] B. Bollig. Personal communication, manuscript, February 1996.

[5] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *DAC*, pages 40–45, June 1990.

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

[7] R. E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. on Comp.*, C-40:206–213, Feb. 1991.

[8] R. E. Bryant. Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification. In *ICCAD*, November 1995.

[9] J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic Model Checking with Partitioned Transition Relations. In *DAC*, pages 403–407, 1991.

[10] J. R. Burch *et. al.* Symbolic Model Checking: 10^{20} States and Beyond. *Inf. and Comp.*, 98(2):142–170, 1992.

[11] O. Coudert *et. al.* Verification of Sequential Machines Based on Symbolic Execution. In *Proc. of the Workshop on Automatic Verif. Methods for Finite State Systems*, Grenoble, France, 1989.

[12] R. Drechsler *et. al.* Efficient representation and manipulation of switching functions based on Ordered Kronecker Functional Decision Diagrams. In *DAC*, pages 415–419, 1994.

Ckt	TOTAL TIME			SPACE			
	ROBDDs	POBDDs	Gain Factor	ROBDDs	Largest POBDD	Gain Factor	Sum of POBDD sizes
C3540(22)	315	47	6.70	6266	6071	1.03	15741
C6288(12)	352	171	2.05	20840	8892	2.34	35446
C6288(13)	820	291	2.82	49924	19542	2.55	75986
C6288(14)	5368	2351	2.28	100156	23941	4.18	717472
C6288(15)	5519	2547	2.17	241865	28801	8.39	1.35M
C6288(16)	24986	7228	3.45	581153	57418	10.12	5.29M
Total	**37360**	**12635**	**2.96**	**1M**	**125123**	**8**	**7.49M**
OPU1[A]*	45515	6683	6.81	1.7M	156652	10.85	214138
OPU1[B]	9278	3031	3.06	245812	28554	8.61	157047
OPU1[C]	18090	4831	3.74	139481	15682	8.89	73356
OPU1[D]	11399	3222	3.54	201573	49096	4.10	114505
Total	**84K**	**17.76K**	**4.73**	**2.23M**	**0.25M**	**8.96**	**0.56M**
OPU2[A]	10716	6168	1.73	654412	61561	10.63	147535
OPU2[B]	7371	2187	3.37	272518	19576	13.92	79651
OPU2[C]	6527	2795	2.33	131377	53150	2.47	169418
OPU2[D]	15739	6961	2.26	369824	113066	3.27	255568
Total	**40K**	**18K**	**2.22**	**1.43M**	**0.25M**	**5.72**	**0.65M**
NINPTB[A]	-	20686	INF	spaceout	41413	INF	481973
NINPTB[B]	-	26110	INF	spaceout	62405	INF	575951
NINPTB[C]	-	>100K (p)	INF	spaceout	228437	INF	17.2M (p)
MSWCN	-	>500K (p)	INF	spaceout	5.46M	INF	17.3 M (p)
RC[1]	-	3778	INF	spaceout	135265	INF	263513
RCB[1]	-	4374	INF	spaceout	93857	INF	262477
Total	-	**>600K**	**INF**	**spaceout**	**8.07M**	**INF**	**36.02M**

Table 1: ISCAS85 Circuits C3540 and C6288. Industrial Circuits OPU1 (Size: $\{317, 232, 17076\}$), OPU2 (Size: $\{317, 232, 17148\}$), NINPTB (Size: $\{226, 392, 15308\}$), MSWCN (Size: $\{205, 131, 1939\}$), RC (Size: $\{203, 8, 1320\}$), and RCB (Size: $\{203, 8, 1175\}$).

[13] L. Fortune, J. Hopcroft, and E. M. Schmidt. The complexity of equivalence and containment for free single variable program schemes. *Goos, Hartmanis, Ausiello and Bohm, Eds., Lecture Notes in Computer Science 62, Springer-Verlag*, pages 227–240, 1978.

[14] J. Gergov and C. Meinel. Efficient Boolean Manipulation With OBDD's can be Extended to FBDD's. *IEEE Transaction on Computers*, 43(10):1197–1209, 1994.

[15] J. Jain, J. Bitner, D. S. Fussell, and J. A. Abraham. Functional partitioning for verification and related problems. *Brown/MIT VLSI Conference*, March 1992.

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

[17] J. Jain, R. Mukherjee, and M. Fujita. Advanced Verification Techniques Based on Learning. In *DAC*, pages 420–426, June 1995.

[18] J. Jain *et. al.* Decomposition Techniques for Efficient ROBDD Construction. In *Formal Methods in CAD 96*, LNCS. Springer-Verlag, 1996.

[19] U. Kebschull *et. al.* Multilevel logic synthesis based on Functional Decision Diagrams. *European DAC*, pages 43–47, 1992.

[20] S. Kimura and E. M. Clarke. A parallel algorithm for constructing binary decision diagrams. *ICCD90*.

[21] S. Malik *et. al.* Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment. In *ICCAD*, pages 6–9, November 1988.

[22] S. Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. *30th DAC*, 1993.

[23] A. Narayan *et. al.* Overcoming Memory Constraints in ROBDD Construction By Functional Decomposition and Partitioning. Technical Report UCB/ERL M95/91, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, 1995.

[24] A. Narayan *et. al.* A Study of Composition Schemes for Mixed Apply/Compose Based Construction of ROBDDs. In *Proc. of the Intl. Conf. on VLSI Design*, January 1996.

[25] M. Rebaudengo, S. Gai, and M. Sonza Reorda. An improved data parallel algorithm for Boolean function manipulation using BDDs. *Euromicro Workshop on Parallel and Distributed Processing 1995*.

[26] S. Reddy *et. al.* Novel verification framework combining structural and OBDD methods in a synthesis environment. *DAC*, pages 414–419, 1995.

[27] R. L. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams . In *ICCAD*, pages 42–47, November 1993.

[28] F. Somenzi, S. Panda, and B. Plessier. Symmetry Detection and Dynamic Variable Ordering of Decision Diagrams. In *ICCAD*, November 1994.

[29] C. D. Thompson. Area-time complexity for VLSI. In *ACM STOC*, 1979.