REACHABILITY ANALYSIS USING PARTITIONED-ROBDDs

by

Amit Narayan, Adrian J. Isles, Jawahar Jain, Robert K. Brayton,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M97/27

10 April 1997

REACHABILITY ANALYSIS USING PARTITIONED-ROBDDs

by

Amit Narayan, Adrian J. Isles, Jawahar Jain, Robert K. Brayton,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M97/27

10 April 1997

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Reachability Analysis Using Partitioned-ROBDDs

Amit Narayan Adrian J. Isles Jawahar Jain[1] Robert K. Brayton
Alberto L. Sangiovanni-Vincentelli

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

Abstract

In this paper we address the problem of finite state machine (FSM) traversal, a key step in most sequential verification and synthesis algorithms. We propose the use of partitioned-ROBDDs to reduce the memory explosion problem associated with symbolic state space exploration techniques.

In our technique, the reachable state set is represented as a partitioned-ROBDD [23]. Different partitions of the Boolean space are allowed to have different variable orderings and only one partition needs to be in memory at any given time.

We show the effectiveness of our approach on a set of ISCAS89 benchmark circuits. Our techniques result in a significant reduction in total memory utilization. For a given memory limit, partitioned-ROBDD based method can complete traversal for many circuits for which monolithic ROBDDs fail. For circuits where both partitioned-ROBDDs as well as monolithic-ROBDDs cannot complete traversal, partitioned-ROBDDs can reach a significantly larger set of states.

1 Introduction

A large number of problems in VLSI-CAD including verification and synthesis of sequential circuits require efficient techniques to perform state enumeration of finite state machines (FSMs). For a given sequential circuit, the number of reachable states can be exponential in the number of state elements present in the circuit. A popular approach to deal with this 'state explosion problem' consists of implicitly representing the set of reachable states and the transition relation of an FSM as Reduced Ordered Binary Decision Diagrams (ROBDDs) [10, 21]. In many cases, ROBDDs can represent a large number of states very compactly, thus allowing automatic verification and synthesis of systems having large state spaces. However, in many

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

other cases, the ROBDD representation is not compact. In fact, the size of an ROBDD representing the set of reachable states of a circuit can be exponential in the number of state holding elements present in the circuit. Unfortunately, circuits which exhibit this worst case complexity do frequently arise in practice. This problem, commonly known as the ROBDD 'memory explosion problem', places a limit on the complexity of circuits that can be processed using ROBDDs.

Many researchers have addressed this problem of ROBDD memory explosion and have proposed alternative representations for Boolean functions which are more compact than ROBDDs (sometimes exponentially so) on certain classes of functions. Some of these are Free BDDs [12, 1, 26], Functional Decision Diagrams (FDDs) [19], ZBDDs [22], OKFDD [11], k-OBDD [2], IBDDs [16], partitioned-ROBDDs [17, 23], $\oplus$-OBDDs [29], TBDDs [1], Canonical TBDDs [13] etc.

In spite of the theoretical advances made in the area of Boolean function representation, ROBDDs still remain the most popular representation for the problems arising in sequential circuit verification and synthesis. One important reason behind this is that the effectiveness of these alternative representations has not been adequately demonstrated on practical problems. In particular, nobody (to the best of our knowledge) has shown an application of these representations in the context of sequential circuits.

Recently, Narayan, Jain, Fujita and Sangiovanni-Vincentelli [23] proposed a representation for Boolean functions, called partitioned-ROBDDs. Partitioned-ROBDDs [17, 23], besides being canonical and efficiently manipulable, can be exponentially more compact than monolithic ROBDDs in representing certain classes of Boolean functions. Experimental results on a set of hard combinational circuits showed a significant improvement in memory requirement [23].

In this paper, we show how partitioned-ROBDDs can be applied for performing reachability analysis on sequential circuits. Reachability analysis constitutes the core computation of many sequential synthesis and verification algorithms. We present an algorithm to create the partitioned-ROBDD representation for the set of reachable states starting from a sequential netlist. The algorithm is built on top of a conventional reachability algorithm and can fully leverage other advances made in the BDD technology and reachability analysis. In this sense, it is backward compatible with other approaches. We have implemented our algorithm in the VIS [5] environment using the CUDD package from University of Colorado [27]. Experimental results

on a set of ISCAS89 and ISCAS89-addendum93 circuits show up to 2 orders of magnitude reduction in space required to represent the set of reachable states. For a given memory limit, the partitioned-ROBDD based reachability algorithm can complete traversal for many circuits on which monolithic-ROBDDs fail. For hard circuits where both monolithic-ROBDDs and partitioned-ROBDDs are unable to complete traversal, partitioned-ROBDDs are able to reach a significantly larger number of states than monolithic ROBDDs.

The rest of the paper is organized as follows. In Section 2, we briefly review partitioned-ROBDDs and show an example of a sequential circuit where partitioned-ROBDDs can be exponentially more compact than monolithic ROBDDs in representing the set of reachable states. In section 3, we briefly review the monolithic ROBDD based FSM traversal algorithm and some previous approaches to deal with the memory explosion problem. In Section 4 we present our algorithm of partitioned-ROBDD based FSM traversal and discuss some implementation issues. Experimental results are presented in Section 5 .

2 Partitioned-ROBDDs

The idea of partitioning was used to discuss a function representation scheme called partitioned-ROBDDs in [17, 15] which was then extensively developed, both theoretically and experimentally in [23]. In this section, we first briefly review the definition of partitioned-ROBDDs and then show an example of sequential circuit verification where the set of reachable states has an exponential ROBDD representation but can be efficiently represented by partitioned-ROBDDs using only polynomial space.

2.1 Partitioned-ROBDDs: Definition

Assume that we are given a Boolean function $f : B^n \rightarrow B$, defined over n inputs $X_n = \{x_1, \ldots, x_n\}$. The partitioned-ROBDD representation, χ_f, of f is defined as follows [23]:

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, $\chi_f = \{(w_1, f_1), \ldots, (w_k, f_k)\}$ where, w_i: $B^n \rightarrow B$ and f_i: $B^n \rightarrow B$, are also defined over X_n and satisfy the following conditions:*

- *1. w_i and 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. $w_i \wedge w_j = 0, for\ i \neq j$
- 4. $f_i = w_i \wedge f, for\ 1 \leq i \leq k$

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* and represents a part of the Boolean space over which f is defined. Intuitively speaking, in partitioned-ROBDDs the Boolean space is divided into k partitions (using the window functions $w_i \in W$). The functionality of f is represented over each partition as a separate ROBDD f_i. ROBDDs for different partitions can have different variable orders.

It was shown in [23] that partitioned-ROBDDs are canonical and various Boolean operations can be efficiently performed on them just like ROBDDs. In addition, they can be exponentially more compact than ROBDDs for certain classes of functions. The practical utility of this representation was demonstrated by constructing ROBDDs for the outputs of combinational circuits. Recently, it was proved by Bollig and Wegener [3] that the class of partitioned-ROBDDs form a strict hierarchy i.e. for any given k, there are functions which have a polynomial $(k+1)$- partitioned-ROBDD representation but no polynomial k-partitioned-ROBDD representation. Therefore, by increasing the number of partitions, we can increase the class of functions that become tractable. An excellent comparison of the computational power of various BDD based representations and partitioned-ROBDDs can be found in [4].

2.2 Partitioned-ROBDD: An Example

The examples used in [23] to prove that partitioned-ROBDDs can be exponentially more compact than monolithic ROBDDs are somewhat artificial in the sense that one is unlikely to encounter these functions in real-life circuits. Here we present a more realistic example of sequential circuit verification where partitioned-ROBDD representation of the set of reachable states is exponentially more compact than the monolithic representation. This example is taken from [20] and consists of verifying whether two hardware implementations of a bounded queue are equivalent (Figure 1). The first implementation is a *shift register* in which the most recent item is always stored in location 0, and all items shift over when a new item is

inserted. The second implementation is a *ring buffer*, where a "head pointer" points to the oldest item, and the items themselves remain fixed. The two circuits can be verified by constructing the product machine and checking that the outputs of the two circuits are always equal [10]. To check this, we need to construct the set of reachable states. To get a small ROBDD representation for the reachable states, we need to put the

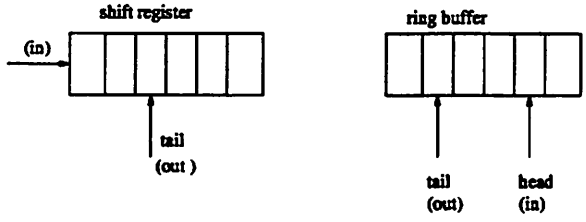

Figure 1: *Two implementation of FIFO queue*

related state variables together in the variable order of the ROBDD representing the set of reachable states. However, since the location of the head pointer is not fixed, we cannot fix an order which keeps the related variables together. Notice that although there is no order which is good for ROBDDs, if we fix the position of the head pointer then there is a one-to-one correspondence between the locations of the two queues. Therefore for a given location of the head pointer, we can find an ordering which leads to a small ROBDD for the reachable state set. Since ROBDDs are restricted to have only one order, any variable ordering leads to an exponentially sized ROBDD. Partitioned-ROBDDs, on the other hand can employ different orders for different partitions. Therefore, if we select the different locations of the head pointer as our window functions, then for each window function there is a one-to-one correspondence between the state variables of the two implementations. For each window function we can select the variable ordering which keeps these related state variables together, resulting in a small partitioned-ROBDD representation.

Notice that a free-BDD representation of the above example is also polynomial. However, using free-BDDs in sequential verification is difficult because of the inherent difficulties associated in performing existential quantification on them.

3 Reachability Analysis Using Monolithic ROBDDs

In this section, we discuss the standard monolithic algorithm for reachability analysis and some improvements that have been proposed for it. The partitioned-ROBDD based traversal algorithm uses the monolithic ROBDD based algorithm in its inner loop (to perform fixed point on individual partitions). Therefore, it is able to take advantage of all of these improvements.

3.1 Standard Algorithm

The standard reachability algorithm is based on a breadth-first traversal of finite-state machines [10, 21, 28]. The algorithm takes as inputs the set of initial states, $I(s)$, expressed in terms of the present state variables, s, and a transition relation, $T(s, s', i)$, relating the set of next states, $N(s')$, that a system can reach from a state s on an input i. The transition relation, $T(s, s', i)$, is obtained by taking a conjunction of the transition relations, $s'_k = f_k(s, i)$, of the individual state elements. Given a set of states, $R(s)$, that the system can reach, the set of next states, $N(s')$, is calculated using equation (1) (also known as *image computation* and referred to as $IMAGE(R(s), T(s, s', i))$ in the algorithms):

$$N(s') = \exists_{s,i}[T(s, s', i) \wedge R(s)] \tag{1}$$

where $T(s, s', i)$ is given by the following equation,

$$T(s, s', i) = \prod(s'_k = f_k(s, i)) \tag{2}$$

The set of reached states is computed by adding $N(s)$ (obtained by replacing variables s' with s) to $R(s)$ and iteratively performing the above image computation step until a fixed point is reached. Figure 2 shows the outline of this algorithm.

```
FSM_TRAVERSAL(I(s), T(s, s', i)) {
        R(s) = F(s) = I(s)
        while (F(s) ≠ 0)
                N(s') = IMAGE[F(s), T(s, s', i)]
                F(s) = N(s' ← s) ∧ R̄(s)
                R(s) = R(s) + F(s)
        repeat
}
```

Figure 2: Standard Monolithic ROBDD based FSM Traversal Algorithm

3.2 Enhancements to the Standard Algorithm

The FSM traversal algorithm presented in Figure 2 needs the ROBDD representations of $T(s, s', i)$ and $R(s)$. The algorithm will not be able to terminate if either of the two ROBDDs become very large. Many techniques have been proposed which can substantially reduce the memory required to represent $T(s, s', i)$ and $R(s)$ as ROBDDs.

3.2.1 Partitioned Transition Relation

To control the size of transition relations, Burch, Clarke and Long [6] propose the use of partitioned transition relations. In this method, instead of using a monolithic ROBDD representation of the transition relation, the transition relations of different latches are kept as separate ROBDDs (or clustered into small groups of latches [24]). Since ROBDDs representing the individual latch transition relations are typically much smaller than when they are combined, this method can result in substantial memory savings. In addition, it allows for early quantification of variables which are not present in the support of other transition relations [14, 7, 28]. This technique can also result in substantial savings in memory during image computation. Notice though, that the term 'partitioning' is being used here in a totally different context; in the partitioned-transition relation approach, the set of latches are being partitioned into different groups while in partitioned-ROBDDs, it is the Boolean space which is being partitioned. In fact, the two approaches are orthogonal and partitioned-transition relations are used in the inner loop of our approach.

For large circuits, even the ROBDDs of the individual latches can become very large. In these cases [8, 18] suggest the use of intermediate variables to control the size of transition relations . These intermediate

variables can be quantified out along with the present state variables during the image computation step.

3.2.2 Reducing Intermediate Memory Required to Represent $R(s)$

Most of the techniques for controlling the size of the ROBDD representation of $R(s)$ are based on the observation that the set of reachable states usually has a much smaller representation at the fixed-point as compared to the peak size observed during the intermediate stages of computation. These techniques try to reduce the intermediate peak memory requirement by changing the order in which the states are visited during traversal. One popular approach, proposed by Ravi and Somenzi [25], tries to maximize the 'density' of the ROBDD representing the reachable state set at each step of traversal. Here density is defined as the number of states represented by an ROBDD divided by the ROBDD size. They present a sub-setting heuristic which tries to reduce the size of reached state set ROBDD by performing reachability analysis only along certain paths when the size grows beyond a certain threshold. Their algorithm can be easily integrated in the algorithm presented in Figure 2 by replacing the 'frontier' set, $F(s)$, with a dense subset of it. Recently, Cabodi, Camurati and Quer [9] have presented a very interesting technique which they call λ-latch removal. In this technique, the latches are divided into subsets A and B such that the latches in B are not in the support of the next state functions of other latches. Reachability is performed by using the latches in A until a fixed point is reached. Latches in B are introduced back only in the last image computation step. Since the ROBDD of the reachable states is usually smaller at the fixed point than at the intermediate stages, this technique leads to a global reduction in the memory requirement. Another interesting technique proposed in [9] which is somewhat related to ours is of set decomposition. In this approach, the set of reachable states is decomposed into two or more sets during the intermediate stages of computation and reachability is performed on these decompositions separately. After a few steps of reachability, results from these different sets are combined to obtain a monolithic ROBDD representation of the reachable state set. The fundamental difference between the approach of [9] and ours is that it uses partitioning to reduce the intermediate memory requirement during the construction of a monolithic ROBDD representation of $R(s)$, whereas our goal is to construct a partitioned-ROBDD representation of $R(s)$. We will do a more detailed comparison with their approach in section 6.

4 Reachability Analysis Using Partitioned-ROBDDs

In this section, we describe our partitioned-ROBDD based reachability analysis algorithms in detail. This algorithms is built on top of the standard BFS algorithm and thus benefits from the improvements discussed in the previous section.

Before describing the algorithm in detail, let us first take a look at image computation step for partitioned-ROBDDs. For the case of monolithic ROBDDs, the image computation is performed according to equation (1). Now suppose we are given the partitioned-ROBDD representation, χ_R, of $R(s)$ where, $\chi_R = \{(w_j(s), R_j) | 1 \leq j \leq k\}$ satisfying conditions 1-4 in Definition 1. We want to get the partitioned-ROBDD representation, χ_N, of the set of next states. Suppose we take the image of R_j under $T(s, s', i)$ to obtain N_j for $1 \leq j \leq k$. These N_j's obtained under the image of $T(s, s', i)$ are not disjoint and hence cannot be used in the partitioned-ROBDD representation of $N(s)$ (Definition 1). There are two simple solutions to this problem. One is that we re-partition these N_j's according to the window function W and assign the min-terms in $w_i \wedge N_j$ to partition i. However, since we want to maintain different orders for different partitions, this computation can be expensive if performed at every step of image computation. Another solution is that we continue with the reachability analysis on the N_j's obtained for a few more steps and then re-partition to obtain a partitioned-ROBDD representation. The problem with this approach is that in the intermediate stages of computation, the same stages can be visited multiple times making it hard to detect when a fixed point is reached and leading to unnecessary extra computation.

We propose an alternative solution to the above based on the following observation. Let us assume that we are given a partitioned-ROBDD representation $\chi_R = \{(w_j(s), R_j) | 1 \leq j \leq k\}$. If we take the image of R_j under $T_{jj}(s, s', i)$, where $T_{jj}(s, s', i) = w_j(s) w_j(s') T(s, s', i)$, we get,

$$N_j(s') = \exists_{s,i}[w_j(s) w_j(s') T(s, s', i) R_j(s)] \tag{3}$$

Since $w_j(s')$ is independent of the variables to be quantified, it can be taken out of existential quantification,

giving us the following equation:

$$N_j(s') = w_j(s')[\,\exists_{s,i}[w_j(s)T(s,s',i)R_j(s)]\,] \tag{4}$$

Equation 4 shows that the image of R_j under $T_{jj}(s,s',i)$ lies completely within the partition j. Similarly, the image, N_l of R_j under $T_{jl}(s,s',i)$ where $T_{jl}(s,s',i) = w_j(s)w_l(s')T(s,s',i)$, will lie completely within the partition l. This simple observation forms the basis of our algorithm. We perform multiple steps of image computation on each R_j under T_{jj}. Since these steps of image computation add states only within the same partition, and since different partitions are disjoint, we are guaranteed that the same state is not being visited multiple times within different partitions. Once a fixed-point is reached within a partition j, transition relations $T_{jl}(s,s',i)$ are used to communicate the new set of states to the partition l for for $1 \leq l \leq k$ and $l \neq j$. The overall flow of the partitioned-ROBDD based algorithm is outlined in Figure 3.

The POBDD_REACH algorithm takes BDDs representing the transition relation $T(s,s',i)$, the set of initial states $I(s)$ of an FSM M and an integer k. The result of the algorithm is the set of reachable states of M represented as POBDD: $\chi_R = \{(w_j(s), R_j) | 1 \leq j \leq k\}$. Associated with each partition, j, is a $bddMgr_j$. Each BDD manager is free to choose a unique ordering that minimizes the number of nodes in its partition. In addition to $R_j(s)$, each partition utilizes k transition relations. These transition relations are computed for each j using $T_{jl}(s,s',i) = w_j(s)w_l(s')T(s,s',i)$ $(1 \leq l \leq k)$. Intuitively, T_{jl} represents the transitions from states in window j to states in window l. POBDD_REACH attempts to minimize memory usage by only keeping the reachable state set and transition relations associated with one partition in memory at a given time. The reachable state sets for partitions not being processed are saved on disk. The transition relations for partitions not being processed are freed and recomputed when needed.

Given $I(s)$, $T(s,s',i)$ and k, function COMPUTE_WINDOW_FUNCTIONS returns a set of window functions $\{w_j(s)\}$. The heuristic used to determine these window functions is discussed in Section 4.1.

The algorithm maintains an array, $eventQueue$, which keeps track of the partitions on which reachability has to be performed. This array is initialized using the function INITIALIZE_QUEUE, which computes the set of initial states for each partition. Each $I_l(s)$ is computed by taking the image of $I(s)$ under $T_{jl}(s,s',i)$

for $1 \leq j \leq k$ and adding it to $w_l(s) \wedge I(s)$. Partition l is inserted into $eventQueue$ only if $I_l(s)$ is non empty. INITIALIZE_QUEUE initially saves all partitions to disk.

```
POBDD_REACH( I(s), T(s,s',i), k ) {

    {w_j(s)} = COMPUTE_WINDOW_FUNCTIONS( T(s,s',i) )
    {bddMgr_j } = CREATE_BDD_MANAGERS( k )
    eventQueue = INITIALIZE_QUEUE( {w_j(s), I(s), T(s,s',i)} )
    {C_j(s) = φ}

    while(|eventQueue| ≠ 0 ) {
        newQueue = φ
        foreach (j ∈ eventQueue) {
            R_j(s) = LOAD_FROM_DISK( bddMgr_j, j)) + C_j(s)
            FREE( C_j(s) )
            {T_jl(s,s',i)} = CREATE_TRANSITION_RELATIONS( bddMgr_j, j )
            N_j(s) =FSM_TRAVERSAL( R_j(s), T_jj(s,s',i) )
            F_j(s) = R̄_j(s) ∧ N_j(s)
            R_j(s) = R_j(s) + F_j(s)
            if(F_j(s) ≠ 0 ) {
                foreach ( l s.t. ( (1 ≤ l ≤ k) & (l ≠ j))) {
                    N_l(s' ← s) = IMAGE(R_j, T_jl(s) )
                    C_l(s) = MGR_COMMUNICATE( N_l(s), bddMgr_j, bddMgr_l, φ) + C_l(s)
                    if(|C_l(s)| ≠ 0)  l → newQueue
                }
            }
            SAVE_TO_DISK( bddMgr_j, R_j(s) )
            FREE( R_j(s) )
            FREE( {T_jl(s,s',i)} )
        }
        eventQueue = newQueue
    }
}
```

Figure 3: POBDD_REACH Algorithm

Once the initial states of each partition are computed, the algorithm proceeds by loading $R_j(s)$ from disk and then performing a fixed point computation using T_{jj}. Next, $k-1$ image computations are performed under T_{jl} (for $l \neq j$) to communicate the information about the new states, $F_j(s)$, added in partition j. For this communication between managers, we keep a communication cache, C_l, which is a BDD in memory that keeps the states that can be reached from the new states $F_j(s)$ of other partitions j, $j \neq l$ under the

image of $T_{jl}(s, s', i)$. Note that Since C_l is defined in $bddMgr_l$, MGR_COMMUNICATE must be called using $bddMgr_j$ and $bddMgr_l$ as arguments. Each C_l is saved in memory and added to the set of reachable states in l the next time it is loaded from disk. The sets Partition l is added to the $newQueue$ if C_l is non-empty. The algorithm continues until no new partitions need to be processed.

The function MGR_COMMUNICATE (Figure 4) takes a BDD f that is defined in the BDD manager $srcMgr$, and returns a BDD g that is functionally equivalent to f but defined in the BDD manager $destMgr$. The function also takes the table $computedTable$ as an argument which maps BDD nodes in the $srcMgr$ to equivalent nodes in the $destMgr$ that have been previously computed by the function. Initially, the $computedTable$ is empty. MGR_COMMUNICATE proceeds recursively. First, f is checked to see if it is a tautology. If so, the appropriate node in the $destMgr$ is returned. Otherwise, the $computedTable$ is checked to see if a BDD equivalent to f has already been constructed in the $destMgr$. If such a BDD is found the result is returned. If that is not the case, MGR_COMMUNICATE is called on the then and else branches of f. The ite function is then used to construct g. We assume that the variables in both managers have the same id. As a final step, the pairs (f, g) and ($\overline{f}$, $\overline{g}$) are inserted into the $computedTable$ and g is returned. Note that MGR_COMMUNICATE does not require $srcMgr$ and $destMgr$ to have the same variable orderings.

MGR_COMMUNICATE(f, $srcMgr$, $destMgr$, $computedTable$){
    if($f == 0$) return $bdd\_zero(destMgr)$
    if($f == 1$) return $bdd\_one(destMgr)$

    if($g = computedTable(f)$) return g

    t = MGR_COMMUNICATE($bdd\_then(srcMgr, f)$, $srcMgr$, $destMgr$, $computedTable$)
    e = MGR_COMMUNICATE($bdd\_else(srcMgr, f)$, $srcMgr$, $destMgr$, $computedTable$)

    $g = ite(topVar(f), t, e)$

    $(f, g) \rightarrow computedTable$
    $(\overline{f}, \overline{g}) \rightarrow computedTable$

    return g
}

Figure 4: Algorithm to convert between BDD managers

4.1 Partitioning Heuristic

Currently we use a static algorithm to obtain the window functions in which the number of partitions is specified a priori. Window functions, $w(s)$'s, are cubes on the present state variables. The algorithm assigns a cost to each variable and selects the best $log_2 k$ variables (for k partitions) for partitioning. From these $log_2 k$ variables k partitions are created which correspond to all the binary assignments of these variables. Our goal is to create small and balanced partitions. For this we define the cost of partitioning a transition relation $T(s, s', i)$ on variable s as

$$cost_s(T) = \alpha[p_s(T)] + \beta[r_s(T)] \tag{5}$$

where $p_s(T)$ represents the partitioning factor and is given by,

$$p_s(T) = max(|T_s|, |T_{\bar{s}}|) \tag{6}$$

and $r_s(T)$ represents the redundancy factor and is given by,

$$r_s(T) = |T_s| + |T_{\bar{s}}| \tag{7}$$

Here, T_s and $T_{\bar{s}}$ represent the positive and the negative cofactors of T with respect to s respectively. 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.

If $T(s, s', i) = \prod_{k=1}^{m} T_k(s, s', i)$ is a 'partitioned-Transition Relation' (see Section 3.2.1), then the cost of partitioning for a variable s is defined as:

$$cost_s(T) = \sum_{k=1}^{m} \frac{cost_s(T_k)}{T_k} \tag{8}$$

Notice that although our current heuristic for window function selection only gives window functions which are cubes on the present state variables, in the algorithm or implementation there is nothing which restricts us

from using more general window functions (which can be arbitrary functions of s). Also, our current heuristic for window function selection is based on only the transition relation. We are currently implementing another heuristic which dynamically increases the number of partitions if the partitions become very large.

4.2 Use of $\overline{w}_j$ as Don't Cares

In our algorithm, the part of the Boolean space covered by $\overline{w}_j(s)$ is used as don't cares while performing traversal in partition j. Since we use only T_{jj} while traversing partition j, it ensures that no extra states are added to this partition if we use the space $\overline{w}_j(s)$ as don't cares. This don't care set, $\overline{w}_j(s)$, is used to minimize the BDDs of $R(s)$,$N(s)$ and $F(s)$ inside the FSM_TRAVERSAL routine. A good heuristic to minimize BDDs f in the presence of don't cares $\overline{w}_j(s)$, is to take the cofactor of f with respect to $w_j(s)$. Since, in the present implementation $w_j(s)$ is a cube, taking the cofactor of f with respect to it is *guaranteed* to reduce the size of the BDDs in the intermediate steps. Before communicating the new states to other partitions under $T_{jl}(s, s', i)$, we take the conjunction of R_j returned by the FSM_TRAVERSAL algorithm (which uses don't cares) with $w_j(s)$ to ensure that no extra states are added to partition l.

4.3 Guiding Reachability by Using Size and Depth Threshold

The POBDD_REACH algorithm presented in Figure 3 provides us with considerable flexibility in terms of changing the order in which the state space is traversed. This can be done by choosing different schedules for processing the $eventQueue$, imposing a 'size' threshold such that BDDs of any one partition do not become very large or placing a depth threshold for the number of steps of reachability that are performed for each partition.

5 Experimental Results

We have implemented our algorithm of Section 4 in the VIS [5] environment using the BDD package from the University of Colorado at Boulder [27] which incorporates state-of-the-art algorithms for dynamic reordering. All results are reported using DEC Alpha architecture with 250MHz clock. The data-size limit was set to 128Mb (except for s1423 in Table 2 for which we used a DEC Alpha with 300MHz clock and a

data-size limit of 256Mb was set).

In Table 1 and Table 2 we compare the results of reachability using monolithic ROBDDs and partitioned-ROBDDs. We use the algorithm proposed by Ranjan, Aziz, Brayton, Plessier and Pixley [24] as the reference algorithm for monolithic ROBDDs. We use the same algorithm with exactly the same parameter settings in the inner-loop of partitioned-ROBDDs to perform fixed-point computation on each partition. We present two sets of result with different values of the parameter image_cluster_size [24] to test the robustness of our results. Dynamic reordering was enabled at all times with the default settings of the parameters. Column 3 (labeled ROBDD) reports the memory and time resource used in generating the monolithic ROBDD representation. Column 4 (labeled Partitioned-ROBDD) reports the results for the partitioned-ROBDD based reachability analysis algorithm. The columns labeled |S| represent the number of states that could be reached using the respective approach before the memory limit was exceeded and the columns labeled Size report the memory required in representing the state set and is measured in terms of the number of nodes in the BDD representation. For partitioned-ROBDDs, the size refers to the size of the largest partition since only one partition needs to be active in the memory at a given time. The column labeled 'N' represents the number of partitions constructed for partitioned-ROBDDs. The column labeled 'Time' reports the total time taken for cases where traversal could be completed.

As the tables clearly indicate, partitioned-ROBDD based traversal is always able to reach more states and the representation in terms of BDDs is always smaller than monolithic ROBDDs. Given the memory-limit of 128Mb, partitioned-ROBDDs are able to complete traversal for s1269, s1512 and s3330 while monolithic ROBDDs are able to complete traversal only for one of these circuits, namely s1512. Even in this case, partitioned-ROBDDs are able to finish traversal in significantly less time. The gains in memory are even more impressive. Partitioned-ROBDDs are almost always an order of magnitude more compact than monolithic ROBDDs (even while representing a larger state space). In particular, for s3330, monolithic ROBDD based representation requires more than 400000 nodes to represent 1.17×10^{17} states (for image_cluster_size = 100) while partitioned-ROBDDs can complete traversal and represent all 7.2778×10^{17} states in about 4000 nodes; a factor of 100 reduction. This is surprising because the reduction was achieved by creating only 4 partitions.

We have plotted the BDD size profile, as a function of the number of states reached for s3330 to check whether the intermediate memory requirement of partitioned-ROBDDs is also small or not. Again, partitioned-ROBDDs have a much better behavior compared to monolithic ROBDDs as shown in Figure 5. Similar results were obtained for other circuits. In particular, for s1423 we observed that by making only 4 partitions, the partitioned-ROBDD algorithm was able to reach a significantly more number of states. In fact, the partitioned-ROBDD based algorithm was able to reach more states (8.0554×10^{10}) than monolithic ROBDD based algorithm in just a few seconds and needed only 14151 nodes to represent them. Once again over a factor of 100 improvement in representing a state set of comparable size.

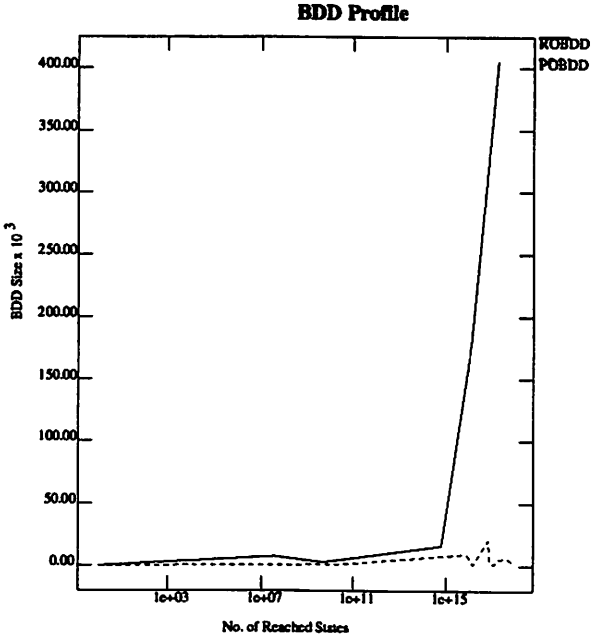

Figure 5: *BDD Profile of s3330*

6 Comment on Related Work

As previously mentioned, Cabodi, Camurati and Quer [9] use the notion of decomposition to represent the set of reachable states. At the first glance, their approach appears to be somewhat similar to ours. However, there are significant differences between the two works. In their approach, when the set of reached states becomes larger than a certain threshold, it is split into two or more partitions. Reachability is then performed on these partitions separately after which the sets are combined to obtain a monolithic representation of the

Ckt	#FF	ROBDD			Partitioned-ROBDD			
		|S|	Size	Time	|S|	Size	Time	N
s1269	37	1.3077×10^{7}	20862	N/A	1.1313×10^{9}	550	906	2
s1512	57	1.6537×10^{12}	687	7226.5	1.6537×10^{12}	649	3238.6	2
s3271	116	4.1292×10^{22}	193593	N/A	9.3374×10^{24}	80172	N/A	4
s3330	132	5.9573×10^{14}	125600	N/A	7.2778×10^{17}	6411	1031.6	8
s4863	104	3.9746×10^{15}	13270	N/A	3.1460×10^{17}	2156	N/A	8
s1423	74	7.99116×10^{9}	469271	N/A	1.4787×10^{15}	394136	N/A	4

Table 1: ISCAS-89 Circuits: Image_Cluster_Size = 5000

Ckt	#FF	ROBDD			Partitioned-ROBDD			
		|S|	Size	Time	|S|	Size	Time	N
s1269	37	1.3077×10^{7}	15886	N/AA	7.300×10^{8}	499	N/A	4
s1512	57	1.6537×10^{12}	687	2066	1.6537×10^{12}	581	1426	4
s3271	116	3.5143×10^{23}	159146	N/A	1.0096×10^{12}	39381	N/A	4
s3330	132	1.1728×10^{17}	404213	N/A	7.2778×10^{17}	4202	672.3	4
s4863	104	3.9746×10^{15}	14273	N/A	3.1460×10^{17}	2156	N/A	8

Table 2: ISCAS-89 Circuits: Image_Cluster_Size = 100

reachable state set. The goal here is to reduce the intermediate memory requirement instead of creating a partitioned-ROBDD representation. Since, reachability is performed on the different partitions using $T(s, s', i)$ (unlike our approach where we use $T_{jj}(s, s', i)$ for the jth partition), it suffers from the same problems as were discussed in the beginning of section 4. Further, since the partitions have to be combined later on, their approach is not able to use (at least in the present form) different orders for different partitions. As most of the gains in cube based partitioning come from the fact that different partitions can have different orders, this poses a serious restriction on the effectiveness of their approach. It can be proved, in fact, that for input variable based partitioning, without allowing different partitions to have different variable orderings, partitioning can achieve only linear reduction in space as opposed to partitioned-ROBDDs where the gains can be exponential. An example of such a case is the buffer-queue verification example of section 2. In addition to these, differences exist in the details of the algorithm. These include our use of an event queue, a different partitioning heuristic, our use of don't cares, communication between managers with different orderings and control on the scheduling of the partitions. Unfortunately, a direct comparison of results could

not be made since we did not have a copy of their package.

7 Conclusions

In this paper, the use of partitioned-ROBDDs for performing reachability analysis on sequential circuits has been proposed. We have shown, by means of an example, that compared to monolithic ROBDDs, partitioned-ROBDDs can represent the reachable state set of some circuits in exponentially less space. In addition, we have presented an algorithm to construct the partitioned-ROBDD representation of the set of reachable states starting from the set of initial states and the transition relation of a system. Our algorithm allows different partitions to have different variable orderings and only one partition needs to be present in memory at any given time. Further, multiple steps of reachability can be performed independently before any communication between partitions is needed.

Experimental results have been very encouraging. Our algorithm has been implemented in the VIS environment and we have observed up to two orders of magnitude reduction in memory usage. For a given memory limit, the partitioned-ROBDD based algorithm was able to complete traversal for many circuits on which monolithic ROBDDs failed. For circuits where both monolithic ROBDDs and partitioned-ROBDDs could not complete traversal, partitioned-ROBDDs were able to cover a significantly larger state space.

Future research is directed towards improving the efficiency of our algorithm. In particular, we are experimenting with different partitioning heuristics. We are currently implementing a dynamic partitioning algorithm, which increases the number of partitions whenever the size of a particular partition crosses a certain threshold. Finally, since reachability can be performed on each partition separately, with only minimal communication between partitions, we feel that our algorithm is particularly suitable for parallel implementation.

8 Acknowledgements

The authors would like to thank Amit Mehrotra and Ravi Gunturi for their critical review of the draft. This work was supported in part by Semiconductor Research Corporation contract #97-DC-324.

References

[1] J. Bern, C. Meinel, and A. Slobodova. Efficient OBDD-Based Boolean Manipulation in CAD Beyond Current Limits. In *DAC*, pages 408–413, June 1995.

[2] B. Bollig, M. Sauerhoff, D. Sieling, and I. Wegener. Hierarchy Theorems for kOBDDs and kIBDDs. *Accepted for publication in Theoretical Computer Science*, 1997.

[3] Beate Bollig and Ingo Wegener. manuscript, Personal communication, 1997.

[4] Beate Bollig and Ingo Wegener. Partitioned-BDDs vs. Other BDD Models. In *To be published in IWLS 97*, Tahoe City, CA, May 1997.

[5] R.K. Brayton, G.D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R.K. Ranjan, S. Sarwary, T.R. Shiple, G. Swamy, and T. Villa. VIS: A System for Verification and Synthesis. In *Proc. of the 8th International Conference on Computer-Aided Verification*, volume 1102 of *LNCS*, pages 428–432. Springer-Verlag, 1996.

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

[7] G. Cabodi and P. Camurati. Symbolic fsm traversals based on the transition relation. *Manuscript Submitted to Transaction on Computer-Aided Design*, 1994.

[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] G. Cabodi, P. Camurati, and Stefano Quer. Improved reachability analysis of large finite state machines. *ICCAD*, pages 354–360, 1996.

[10] O. Coudert, C. Berthet, and J. C. Madre. Verification of Sequential Machines Based on Symbolic Execution. In *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, 1989.

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

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

[13] E. Goldberg, Y. Kukimoto, and R. K. Brayton. Canonical TBDD's - A New Data Structure for Boolean Functions. In *Proc. of the Intl. Workshop on Logic Synthesis*, May 1997.

[14] R. Hojati, S.C. Krishnan, and R. K. Brayton. Heuristic Algorithms for Early Quantification and Partial Product Minimization. Technical Report UCB/ERL M93/58, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, July 1993.

[15] J. Jain. On analysis of boolean functions. *Ph.D Dissertation, Dept. of Electrical and Computer Engineering, The University of Texas at Austin*, 1993.

[16] J. Jain, J. Bitner, M. Abadir, J. A. Abraham, and D. S. Fussell. Indexed BDDs: Algorithmic advances in techniques to represent and verify Boolean functions. *IEEE Transactions on Computers (to appear).*

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

[18] J. Jain, A. Narayan, C. Coelho, S. Khatri, A. Sangiovanni-Vincentelli, R. Brayton, and M. Fujita. 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] K. L. McMillan. A conjunctively decomposed boolean representation for symbolic model checking. In *Computer-Aided Verification*, pages 13–25, 1996.

[21] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[22] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *DAC*, June 1993.

[23] A. Narayan, J. Jain, M. Fujita, and A. L. Sangiovanni-Vincentelli. Partitioned-ROBDDs - A Compact, Canonical and Efficiently Manipulable Representation for Boolean Functions. *ICCAD*, November 1996.

[24] R. K. Ranjan, A. Aziz, R. K. Brayton, B. Plessier, and C. Pixley. Efficient BDD Algorithms for FSM Synthesis and Verification. In *Proc. of the Intl. Workshop on Logic Synthesis*, Tahoe City, NV, May 1995.

[25] K. Ravi and F. Somenzi. High-density reachability analysis. *ICCAD*, 1995.

[26] D. Sieling and Wegener I. Graph-driven OBDDs - a new data structure for Boolean functions. *Theoretical Computer Science*, 1995.

[27] F. Somenzi. CUDD: CU decision diagram package, release 2.1.1. *Department of Electrical and Computer Engineering, University of Colorado at Boulder*, February 1997.

[28] H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines using BDD's. In *ICCAD*, pages 130–133, November 1990.

[29] S. Waack. On the descriptive and algorithmic power of Parity Ordered Binary Decision Diagrams. In *Proc. of STACS*, LNCS. Springer-Verlag, 1997.