Reducing BDD Size by Exploiting Structural Connectivity

Ronnie L. Wright
wrightr6@egr.msu.edu

Michael A. Shanblatt
mas@egr.msu.edu

Department of Electrical and Computer Engineering
Michigan State University
East Lansing, MI 48824-1027

Abstract

Computer-aided design tools have been limited by the use of the Binary Decision Diagram (BDD). The major drawback of the BDD is its abundant usage of CPU time and memory. Techniques such as BDD variable ordering and sharing have been used in the past to address the size issue. However, these techniques remain to be limited to modest-sized circuits. In this paper, we present a significant variation to the conventional BDD, the Connective Binary Decision Diagram (CBDD). The CBDD addresses the size issue concerning conventional BDD implementations by employing the use of minimized-scalable binary decision diagrams (MSBDDs) combined with the structural connectivy present in the circuit's netlist. The experimental results section will demonstrate that the proposed method reduces the BDD size by more than two orders of magnitude for large circuits.

1 Introduction

Low power consumption has emerged as a key design parameter for digital VLSI systems. The trend has been to develop methodologies and techniques which maintain a circuit's throughput and area constraints while achieving some desired level of power efficiency. The power efficiency revolution was initiated by the introduction of high-throughput portable electronic devices, such as laptop computers, portable televisions, camcorders, and wireless communications systems. Thus, power analysis and optimization have become important tasks to be addressed by computer-aided design tools.

Probability-based power analysis tools depend heavily on BDDs to determine signal activity, while historic uses of BDDs have been in the digital circuit design areas of synthesis, verification, and testing. The BDD is not a new concept. As early as 1959, Lee in [6] introduced the concept of Binary Decision Programs and a set of rules to transform these programs into switching circuits. Later, in 1978, Akers in [1], revisited the concept of BDDs by using the diagram as a means to define, analyze, and test large digital functions from and implementation-free perspective. It was in 1986, that Bryant in [4], demonstrated the advantages of BDDs as a canonical representation. Bryant demonstrated that BDDs have two very useful properties. First, BDDs are canonical: given the BDDs for two circuits, the two circuits are equivalent in behavior if the BDDs are identical. Second, BDDs are effective at representing combinatorially large sets, which is useful in FSM equivalence checking and logic minimization.

2 BDD Overview and Background

BDDs represent a switching function as a directed acyclic graph (DAG). A graph will fundamentally consist of an interconnection of nodes (vertices) and edges (arcs). There are two node types: decision nodes or terminal nodes. *Terminal nodes* are characterized by not having outgoing edges which lead to children nodes, and contain fixed values which possibly correspond to the output of a function. *Decision nodes* are characterized by having outgoing edges which lead to other decision nodes or terminal nodes. The decision node is labeled with a variable identifier and has one outgoing edge for each value this variable can assume.

Since Boolean decisions are being made, the decision node variable identifiers can only assume the values of 0 and 1. The terminal node values will be fixed at either 0 or 1, and the possibilities for edges will be either the $0-edge$ or $1-edge$. The $0-edge$ will be chosen when the decision node variable assumes the value 0, and the $1-edge$ is chosen when the decision node variable takes on the value 1. Figure 1 provides an example BDD. Typically in BDD graph representations dotted arcs represent the $0-edges$, while the solid arcs represent $1-edges$.

The BDD size is directly related to the number of nodes in the BDD's graph, which is controlled by the number of input variables and their ordering. One such BDD type,

0-7695-0104-4/99 $10.00

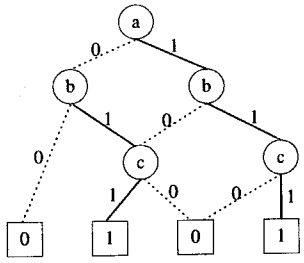

Figure 1. BDD example.

the ordered binary decision diagram (OBDD), addresses the BDD size issue by considering input variable ordering. The ordering of input variables determines the level at which each input will appear in the BDD's graph. In the OBDD the ordering will remain the same for each path taken from the root (lowest order) node to a terminal node. Different input variable orderings lead to different BDDs, with each potentially having a different size. Consider the switching function given by Equation 1. An input variable ordering of $a < b < c < d$ leads to the BDD displayed in Figure 2. While an input variable ordering of $b < c < a < d$ leads to the BDD representation displayed in Figure 3.

$$f(a,b,c) = abc + \overline{b}d + \overline{c}d \qquad (1)$$

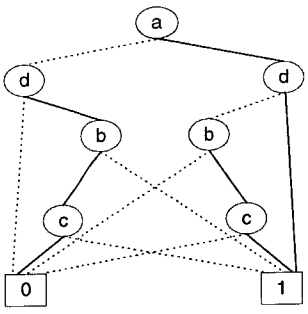

Figure 2. BDD with ordering 1.

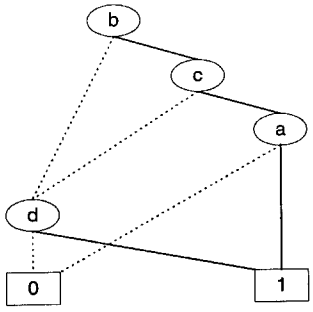

Figure 3. BDD with ordering 2.

Clearly the second ordering provides the more compact BDD representation, it has smaller node count. Hence, a good input variable ordering will yield a more compact BDD representation with reasonable memory usage [5].

A modification to the OBDD is the reduced-ordered BDD (ROBDD). In the ROBDD an initial ordering is given and the iterative identification and removal of isomorphic subgraphs and redundant nodes takes place. The removal results in a BDD which is minimal for the given input variable ordering, and canonical in form.

Once such ROBDD implementation was developed by Brace, *et al.* [3]. This ROBDD implementation made improvements in the if_then_else (ITE) operator, hashing technique, and memory garbage collection. The results reported that an amortized cost of 22 bytes per node was achieved. Additionally, it was reported that the improvements yielded a faster, more memory-efficient ROBDD implementation than the original implementation presented in [4].

Shen, *et al.* [7], proposed a data structure called the Free Boolean Diagram (FBD), which improved the ROBDD representation by trading off canonicity. One distinction between the ROBDD and the FBD is that the FBD allows different input variable orderings along different paths from the root node to a terminal node. Additionally, the nodes in the graph of the FBD may be of type function, which is further discussed in [7]. The main contribution of the Shen work was the development of a probabilistic algorithm that was based on the work of Blum, *et al.* [2]. It was reported that the FBD implementation resulted in an amortized memory cost of 32 bytes per node, and for certain cases the FBD size was significantly reduced.

3 The Connective BDD

The Connective BDD (CBDD) is a directed acyclic graph (DAG), a collection of nodes and edges. An advantage is its ability to maintain the circuit's structural input/output relationships and internal connectivity. The CBDD can be viewed as an interconnection of MSBDDs. The definition of the CBDD is somewhat based on the definitions of a DAG and conventional BDD. Structural and connective relationships are achieved by altering the definition of the conventional BDD to support additional node (vertex) and edge (arc) types and properties. A formal definition of the Connective BDD is given below.

Definition: *A Connective Binary Decision Diagram (CBDD) is a directed acyclic graph which is composed of a MSBDD set M, vertex set V and edge set E.*

1. $\forall v \in V$ may be of type:
   - 1.1. *Input*
     - 1.1.1. proceeded by children via $0-$ and $1-$ *edges*
     - 1.1.2. assumes variable primary input value
     - 1.1.3. $value(v) \in \{0,1\}$
   - 1.2. *Internal*
     - 1.2.1. proceeded by children via $0-$, $1-$ and $v-$ *edges*
     - 1.2.2. assumes variable internal output value
     - 1.2.3. $value(v) \in \{0,1\}$
   - 1.3. *Terminal*
     - 1.3.1. value termination for internal and output nodes
     - 1.3.2. proceeded by children via $v-$ *edges*
     - 1.3.3. value is fixed, $value(v) \in \{0,1\}$

1.4. *Output*
   1.4.1. absolute termination, not proceeded by children
   1.4.2. assumes variable primary output value
   1.4.3. value is fixed, $value(v) \in \{0, 1\}$
2. $\forall e \in E$ may be of type:
   2.1. $0 - edge$ $(1 - edge)$
      2.1.1. traversed when $value(v) = 0(1)$
      2.1.2. outgoing to input, internal, and terminal vertices
   2.2. $v - edge$
      2.2.1. propagates $value(v)$ to internal or output vertices
      2.2.2. incoming only to terminal vertices
3. MSBDD set M elements:
   3.1. represent minimized-scalable ROBDDs for specified logic elements
   3.2. $\forall m \in M, vertex(m) \in V$
   3.3. ROBDD terminals are connected to internal or output nodes via $v - edges$
   3.4. $\forall$ ROBDDs represent a function on vertex v, f_v
   3.5. $f_v = \overline{x_i} \cdot f_{low(v)} + x_i \cdot f_{high(v)}$, where x_i is a decision variable

A few differences exist between the conventional BDD implementation and the CBDD. The Internal and Input nodes of the CBDD are basically decision nodes in terms of the conventional BDD. The concept of a Output node is new, it represents absolute termination. Terminal nodes differ in the fact that they only represent intermediate termination for internal or output nodes. Terminal nodes still carry a constant value of 0 or 1, but are proceeded by an outgoing edge, the $v - edge$. There is the addition of the $v - edge$, which is used to propagate the value of incoming vertex v to internal or output vertices.

In spite of the differences between the conventional BDD implementation and the CBDD there exist very positive benefits when using the CBDD. First, the CBDD is not affected by variable input ordering, because the internal functional units of a circuit can be mapped to predefined MSBDDs, which are already minimal in size. Second, CBDD size or number of nodes is directly related to the number of functional units present in the circuit, so exponential growth will not occur. Third, CBDDs do not contain redundant behavior which is caused by the placement of an input variable and its potential binomial legacies/generation at each level in the BDD's graph. This is replaced by internal nodes and $v - edges$ which increase the sparseness of the CBDD's graph and better represents the structure. Fourth, CBDDs maintain the structural and connective relationships present in the actual circuit, preserving structural correlations, and allowing more accurate estimates of switching activity and power.

The drawbacks of the CBDD include loss of canonicity, however, equivalence of two circuits can be determined by manipulating their CBDDs. Additionally, CBDDs do not have a single input path from the root to a terminal node, like conventional BDDs. This is not good because several paths must be considered when determining a primary output's value. The positive side of this drawback is that CBDDs are more compact, represented by far fewer nodes and edges than conventional CBDDs, thus traversing of additional paths in the CBDD may not be that costly. Lastly, CBDDs are very compact for large circuits, but may give a larger than normal or poor representation for small circuits.

4 Implementation

The CBDD implementation reads a BLIF netlist file as input. The basic logic gates such as N/AND, N/OR, XOR, and NOT, when encountered during the input phase, are converted to minimized-scalable BDDs (MSBDDs). These MSBDDs represent the most reduced BDDs, in terms of total node count, for the given functional unit (gate) and size.

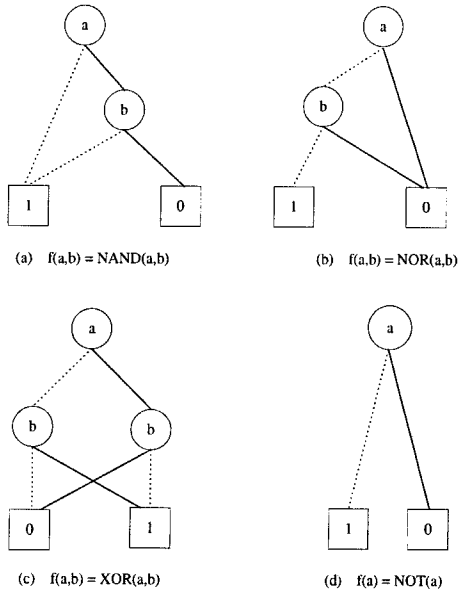

Figure 4. Minimized-scalable BDDs.

Figure 4 displays a small selection of the minimized-scalable BDDs. Once the MSBDDs are generated, their outputs are connected to the input nodes of other MSBDD variables according to the structural properties found within the netlist. The circuit diagram displayed in Figure 5 is represented as a CBDD in Figure 6.

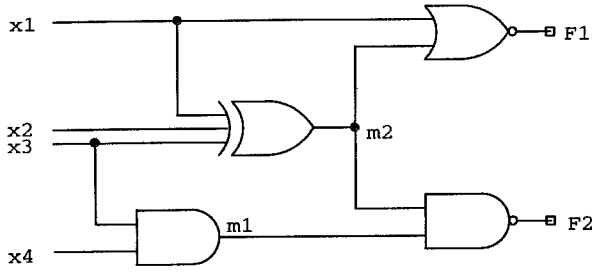

Figure 5. Circuit diagram example.

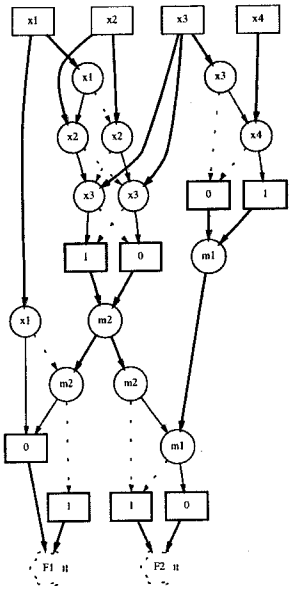

Figure 6. Connective BDD example.

5 Experimental Results

The ISCAS85 benchmark circuits were chosen for the experiment. The results after applying the CBDD implementation to the benchmark circuits are summarized in Table 1. Additionally, as a means of comparison, Table 1 provides the results of implementations used by Brace and Shen in [3, 7] for the same benchmarks. The main entity used in comparing the BDD implementations was the BDD node count. The BDD node count is used as a measure of performance for both CPU and memory utilization. For almost every circuit, a significant savings in the BDD node count was observed. This savings was due to sharing of intermediate internal MSBDD node outputs, which were used as subsequent MSBDD inputs at deeper levels within the circuit structure. The small size of the CBDD is due to the fact that CBDDs grow with respect to the number of functional units present in the circuit's structure, not the number of inputs.

Circuit	*#In*	*#Out*	*ROBDD [3] #Nodes*	*FBD [7] #Nodes*	*CBDD #Nodes*
c432	36	7	30200	31195	2017
c499	41	32	49786	33214	2899
c880	60	26	7655	7761	4276
c1355	41	32	39858	33214	6411
c1908	33	25	12463	12734	9387
c2670	233	140	**Unable**	57767	12886
c3540	50	22	208947	88652	17874
c5315	178	123	32193	26129	26078
c6288	32	32	**Unable**	115607	29361
c7552	207	108	**Unable**	19187	37774

Table 1. Benchmark Results.

6 Conclusion

This paper has described the Connective BDD (CBDD) which was defined as a DAG interconnection of Minimized-Scalable BDDs. CBDD's are very economical in representing large circuits and maintain the circuit's structural and connective relationships. CBDDs represent circuits with far fewer nodes than previous BDD implementations. CBDDs have a few drawbacks including loss of canonicity, multiple paths from the root to terminal nodes, and poor representation of small circuits. The main advantage of the CBDD is that it does not suffer from exponential growth, when the number of inputs and logical interconnections grow.

References

[1] S. B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, c-27(6):509–516, June 1978.

[2] M. Blum, A. K. Chandra, and M. N. Wegman. Equivalence of Free Boolean Graphs Can Be Decided Probabilistically in Polynomial Time. *Information Processing Letters*, 10(2):80–82, March 1980.

[3] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 40–45, June 1990.

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

[5] R. E. Bryant. Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification. *International Conference on Computer-Aided Design*, pages 236–243, November 1995.

[6] C. Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell System Technology J*, 38:985–999, July 1959.

[7] A. Shen, S. Devadas, and A. Ghosh. Probabilistic Manipulation of Boolean Functions Using Free Boolean Diagrams. *IEEE Transactions on Computer-Aided Design*, 14(1):87–95, January 1995.