International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:3, 2007

BDD Package Based on Boolean NOR Operation

M. Raseen[1], A.Assi[2], P.W. C. Prasad[1] and A. Harb[3]
[1]United Arab Emirates University, College of Information Technology, U.A.E
[2]American University of Technology, Department of Computer Engineering, Lebanon
[3]United Arab Emirates University, Department of Electrical Engineering, U.A.E

Abstract— Binary Decision Diagrams (BDDs) are useful data structures for symbolic Boolean manipulations. BDDs are used in many tasks in VLSI/CAD, such as equivalence checking, property checking, logic synthesis, and false paths. In this paper we describe a new approach for the realization of a BDD package. To perform manipulations of Boolean functions, the proposed approach does not depend on the recursive synthesis operation of the IF-Then-Else (ITE). Instead of using the ITE operation, the basic synthesis algorithm is done using Boolean NOR operation.

***Keywords*—** Binary Decision Diagram (BDD), ITE Operation, Boolean Function, NOR operation.

I. Introduction

Synthesis, verification, and testing algorithms of VLSI circuits manipulate large number of switching functions [1], [2]. Therefore it is important to have efficient methods to represent and manipulate such functions. A large class of problems in the area of VLSI CAD can be solved by adopting efficient underlying data structures. During the last decade, several methods based on *decision diagrams* (DDs) have been proposed and successfully used in many industrial applications [3], [4], [5]. Recently, Binary Decision Diagrams (BDDs) have emerged as one of the best representation methods for wide range of applications. Despite the fact that BDD is relatively old technique, its advantages as canonical representations was only recognized and made clear by Bryant in [2]. The success of this technique has attracted many researchers in the area of synthesis and verification of digital VLSI circuits [1], [4]. BDD packages are based on recursive synthesis operation of ITE. Since all binary synthesis operation can easily be described by the use of ITE operation no alternative concepts have been proposed [6].

BDDs as introduced by Bryant [2], have been traditionally used to solve the equivalence checking problem due to their canonical property. However, it is this requirement for canonicity that makes BDDs inefficient in representing certain classes of functions. For example, integer multipliers have displayed exponential memory requirements for any variable ordering [7].There has been increased interest in BDDs techniques that reduce the time and space needed to solve the equivalence checking problem [7], [8], [9]. The combinational equivalence checking needs to perform very fast due to the use of larger designs which require more comparisons to be carried out [10], [11]. In such situations no dynamic variable ordering will be considered mainly due to time complexity. A fast and efficient checking based on NAND-BDDs, instead of three operand ITE operation have been presented in [6]. In this paper we present an approach for the realization of a BDD package, which uses two operand NOR operation instead of using three operand ITE operation to perform manipulation of Boolean functions. This method does not consider some of the main features of BDD package such as dynamic variable ordering, complemented edges, etc. In the second section of this paper, background information's pertaining to the construction and implementations of BDD are given. The new Two operand NOR operation and the main difference between NOR and ITE operations are discussed in the third and fourth sections. Finally we conclude our paper with future developments.

II. Preliminaries

Basic definitions for BDDs are given in [1], [10], [11] and [12]. In the following we review some of these definitions.

Definition 1: A BDD is a directed acyclic graph (DAG). The graph has two sink nodes labeled 0 and 1 representing the Boolean functions 0 and 1. Each non-sink node is labeled with a Boolean variables v and has two out-edges labeled 1 (or *then*) and 0 (or *else*). Each non-sink node represents the Boolean function corresponding to its 1 edge if v=1, or the Boolean function corresponding to its 0 edge if v=0.

Definition 2: An OBDD is a BDD in which each variable is encountered no more than once in any path and always in the same order along each path.

Definition 3: An ROBDD is a BDD with the following properties.

-- There are no redundant nodes in which both of the two edges leaving the node point to the same next node present within the graph. If such a node exists it is removed and the incoming edges redirected to the following node.

-- If two nodes point to two identical sub-graphs (i.e. Isomorphic sub-graphs) then one sub–graph will be removed and the remaining one will be shared by the two nodes.

Variable Ordering

The size of a BDD is largely affected by the choice of the variable ordering. This is illustrated by the following example:

Example: Let $f = x_1 \cdot x_2 + ... + x_{2n-1} \cdot x_{2n}$. If the variable ordering is given by $(x_1, x_2, ..., x_n)$, i.e. $\pi(i) = x_i \forall_i$, the size of the resulting BDD is $2n$. On the other hand, if the variable ordering is chosen as $(x_1, x_3, ..., x_{2n-1}, x_2, x_4, ...x_{2n})$, the size of the BDD is $\theta(2^n)$.

Thus, the number of nodes in the graph varies from linear to exponential depending on the variable ordering. Fig. 1 shows the effect of the variable ordering on the size of BDDs.

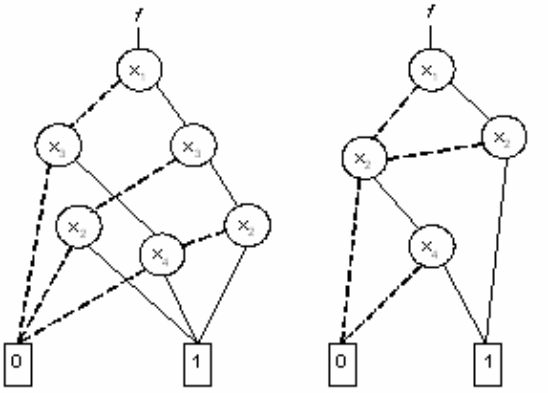

Figure 1: Effect of the variable ordering on the size of BDDs

III. ITE Operation

BDD have two outgoing edges and are labeled by 'then' and 'else', i.e. one labeled by the Boolean 1 (or then) and the other by the Boolean 0 (or else). The ITE operation is a recursive form of the Shannon decomposition theorem [1], [13] given as below:

$$ite(f, g, h) = f \cdot g + \overline{f} \cdot h \quad (1)$$

Shannon's decomposition theorem can be shortly written as:

$$f = ite(x_i, f|_{x_i=1}, f|_{x_i=0}) \quad (2)$$

Let *x* be the top variable of functions *f*, *g*, *h* and f_x and $f_{\bar{x}}$ denote function $f|_{x_i=1}$ and $f|_{x_i=0}$ respectively. Then the following recursive formula can be derived.

$$ite(f, g, h) = ite(x, ite(f_x, g_x, h_x), ite(f_{\bar{x}}, g_{\bar{x}}, h_{\bar{x}})) \quad (3)$$

Recursion proceeds until one of terminal calls occurs:

$$ite(f, 1, 0) = ite(1, f, g) = ite(0, g, f) = ite(g, f, f) = f : ite(f, 0, 1) = \overline{f}$$

The algorithm for ITE operation is given in Fig. 2.

```
Function ite (f, g, h: edge type): edge type;
begin
    execute the first step
    if terminal case
      then return result
    else
      begin
        execute the second and third step
        if result is in the computed-table
          then return result
        else
          begin
            let v be the top variable of function f,g,h;
              T := ite(f_x, g_x, h_x)
              E := ite(f_x̄, g_x̄, h_x̄)
              if T=E
                  then ite:=T;
            else
                  R:= find or add node(x,T,E);
                Insert in computed table (f,g,h,R);
                      ite :=R
              end
        end
end;
```

Figure. 2: *ITE operation*

All two variable operations can be implemented as an ITE expression as shown in Table 1 and therefore it forms a basic operation of BDD.

TABLE 1
ITE OPERATION

Table	Name	Expression	Equivalent Form
0000	0	0	0
0001	f AND g	$\overline{f \cdot g}$	$ite(f, g, 0)$
0010	$f > g$	$f \cdot \overline{g}$	$ite(f, \overline{g}, 0)$
0011	f	f	f
0100	$f < g$	$\overline{f}.g$	$ite(f, 0, g)$
0101	g	g	g
0110	f XOR g	$f \oplus g$	$ite(f, \overline{g}, g)$
0111	f OR g	$f + g$	$ite(f, 1, g)$
1000	f NOR g	$\overline{f + g}$	$ite(f, 0, \overline{g})$
1001	f XNOR g	$\overline{f \oplus g}$	$ite(f, g, \overline{g})$
1010	NOT g	$\overline{g}$	$ite(g, 0, 1)$
1011	$f \geq g$	$f + \overline{g}$	$ite(f, 1, \overline{g})$
1100	NOT f	$\overline{f}$	$ite(f, 0, 1)$
1101	$f \leq g$	$\overline{f} + g$	$ite(f, g, 1)$
1110	f NAND g	$\overline{f \cdot g}$	$ite(f, \overline{g}, 1)$
1111	1	1	1

IV. NOR Operation

This new approach work with only one operation (Boolean NOR) compared to the two operations used in ITE. The modified algorithm for NOR operation is given in Fig. 3. This algorithm is exactly the same as the ITE algorithm except the use of two operands instead of three operands required for ITE operations. All two variable operations can be implemented as an NOR expression as shown in Table 2. This method does not provide a fully fledge BDD package mainly due to the following factors:

- Combinational Equivalent checking needs very fast execution of larger designs. Variable ordering might result in an increase of the time complexity. A heuristic Ordering might solve this problem.
- Complemented edges are not considered, therefore no comparison between the variables are required. It leads to minimum use of memory for the BDD package.

TABLE 2.
NOR OPERATION

Table	Name	Exp.	Equivalent Form
0000	0	0	0
0001	$f\ AND\ g$	$\overline{f \cdot g}$	$NOR(\overline{f},\overline{g})$
0010	$f > g$	$f \cdot \overline{g}$	$NOR(\overline{f},g)$
0011	f	f	f
0100	$f < g$	$\overline{f}.g$	$NOR(f,\overline{g})$
0101	g	g	g
0110	$f\ XOR\ g$	$f \oplus g$	$NOR(NOR(NOR(f,\overline{g}), NOR(\overline{f},g)),1)$
0111	$f\ OR\ g$	$f + g$	$NOR(NOR(f,g),1)$
1000	$f\ NOR\ g$	$\overline{f+g}$	$NOR(f,g)$
1001	$f\ XNOR\ g$	$\overline{f \oplus g}$	$NOR(NOR(f,\overline{g}),NOR(\overline{f},g))$
1010	$NOT\ g$	$\overline{g}$	$\overline{g}$
1011	$f \geq g$	$f + \overline{g}$	$NOR(NOR(f,\overline{g}),1)$
1100	$NOT\ f$	$\overline{f}$	$\overline{f}$
1101	$f \leq g$	$\overline{f} + g$	$NOR(NOR(\overline{f},g),1)$
1110	$f\ NAND\ g$	$\overline{f \cdot g}$	$NOR(\overline{f},\overline{g}),1)$
1111	1	1	1

Advantages of using this system can be classified as follows:

- Easy to implement compared to any other logic function mainly due to the fact that NOR is a universal gate.
- This method works with 2 operand compare to ITE operation. So the execution of the BDD operations will be faster.
- No complemented edges are considered. So the memory requirement for this method will be less then other operations.
- This will fit for combinational Equivalence checking of larger circuits with less time complexity, since numbers of nodes are not counted.
- Variable ordering techniques are not required.

```
Function NOR(f,g: edge type): edge type;
begin
  execute the first step
  if terminal case
    then return result
  else
    begin
      execute the second and third step
      if result is in the computed-table
        then return result
      else
        begin
        let v be the top variable of function f,g;
          T := NOR(f_x, g_x)
          E := NOR(f_x̄, g_x̄)
          if T=E
            then NOR:=T;
            else
        R:= find or add node(x,T,E);
            Insert in computed table ((f, g), R);
              NOR :=R
          end
      end
end;
```

Figure 3: *NOR Operation*

V. Conclusion

In this paper, a new approach for the realization of a BDD package has been presented and discussed. The proposed approach is based on the Boolean NOR operation and does not depend on some of the main features used with ITE operation. The use of Boolean NOR operation will be much more appropriate than ITE in combinational equivalence checking, especially for large VLSI circuits. Since NAND logic is used more frequently in designs and testing than NOR logic, still we believe that this method will provide better results than the ITE operation and it is worth to be considered and discussed. In addition, a modification to the NAND as well as the NOR- method to work as a fully fledge BDD package with less space and time complexity is our future research work.

VI. Acknowledgement

The authors would like to thank Prof. Rolf Drechsler from the Institute of Computer Science – University of Bremen for his

constructive comments that helped to improve quality of this paper and American University of Technology (AUT). For their financial support of this paper.

REFERENCES

[1] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," IEEE Trans. Computers, Vol. 40, pp. 203–213, 1991

.[2] R. E. Bryant, "Graph–Based Algorithm for Boolean Function Manipulation," IEEE Trans. Computers, Vol. 35, pp. 677-691, 1986.

[3] H. Andreas, R. Drechsler and B. Becker, "MORE: An Alternative Implementation of BDD-Packages by Multi-Operand Synthesis," IEEE European Design Automation Conference, pp. 164-169, 1996.

[4] A. Gupta and P. Ashar, "Integrating a Boolean satisfability checker and BDDs for combinational equivalence checking," Proc. of the International Conference on VLSI Design, 1998.

[5] B. Bollig and I. Wegener, "Improving the Variable Ordering of OBDDs is NP–Complete," IEEE Trans. Computers, Vol. 45, pp. 993-1002, 1996.

[6] R. Drechsler and M. Thornton, "Fast and Efficient Equivalence Checking based on NAND-BDDs," Proceedings of IFIP International Conference on Very Large Scale Integration, pp. 401-405, 2001.

[7] J. Marques-Silva and T. Glass, "Combinational Equivalence Checking Using Satisfability and Recursive Learning," International Conference on Design, Automation and Test in Europe, pp.145-164, 1999.

[8] A. Kuehlmann, M. Ganai and B. Paruthi, "Circuit-based Boolean Reasoning," International Design Automation Conf., pp. 232-237, 2001.

[9] H. Hulgaard, P. Williams and H. Andersen, "Equivalence checking of combinational circuits using Boolean expression diagrams," IEEE Transaction on Computer Aided Design, Vol. 18, 1999.

[10] R. Drechsler and B. Becker, "Binary Decision Diagrams–Theory and Implementation," Kluwer Academic Publishers, 1998.

[11] S. Christoph and R. Drechsler, "BDD minimization using Symmetric," IEEE transaction on CAD of IC and systems, Vol. 18, no. 2, pp. 81-100, 1999.

[12] S. B. Akers, "Binary Decision Diagram," IEEE Trans. Computers, Vol. 27, pp. 509-516, 1978.