Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams

Masahiro FUJITA, Hisanori FUJISAWA, Nobuaki KAWATO

FUJITSU LABORATORIES LTD.

1015 Kamikodanaka, Nakahara-ku, Kawasaki 211, Japan

R.E. Bryant proposed a method to handle logic expressions [3], which is based on binary decision diagrams (BDD) with restriction: variable ordering is fixed throughout a diagram. Although the method has much more efficiency than other methods proposed so far, the efficiency heavily depends on variable ordering. We developed a simple but powerful algorithm for variable ordering. The algorithm tries to find a variable ordering which is very natural, i.e., minimizes the number of crosspoint of nets, when the circuit diagram is drawn. We applied this to the boolean comparison of ISCAS benchmark circuits for test pattern generation. The results show that BDD with our ordering method can verify almost all benchmark circuits in less than several CPU minutes, which is one hundred times or more faster than those reported in the literature. We also mention some techniques for circuit evaluation orderings.

1. Introduction

Boolean comparison is a technique to verify the equivalence of two combinational circuits. There are many cases where we must compare two circuits, such as, to check the correctness of synthesis results, to check the correctness of manual modifications, etc. Moreover, in minimization of multi-level logic circuits, almost all time (90%) is spent for equivalence check between the old and the modified circuits [1].

Several boolean comparison methods have been proposed and developed [5,6]. Unfortunately, the power of these methods is not necessarily satisfactory in terms of manageable circuit size and processing time.

R.E. Bryant recently proposed a method to handle logic expressions [3]. It is based on binary decision diagrams (BDD) with restriction: variable ordering is fixed throughout a diagram. 64-bit ALU circuits were shown to be verified within practical time, when appropriate variable ordering was given. We evaluated the method by ISCAS benchmark circuits for test pattern generations [1], and make sure the fact that the efficiency heavily depends on variable ordering of BDD. We developed a simple but powerful algorithm for variable ordering. The algorithm tries to find a variable ordering which is very natural, i.e., minimizes the number of crosspoint of nets, when the circuit diagram is drawn. Using this, all benchmark circuits except c6288 can be verified within reasonable time. Moreover, we also developed some techniques for circuits evaluation ordering, which shorten the verification time by about 30%.

We first briefly introduce Bryant's BDD in section 2, and then give a variable ordering algorithm and its performance in section 3. In section 4, we show some techniques for circuits evaluation ordering, and section 5 is a concluding remark.

2. Boolean Comparison Method Based on Binary Decision Diagram (BDD)

BDD indicates the binary decision diagram which has fixed ordering of input variables [3]. In the following, we will briefly explain BDD. Details can be found in [3]. An example circuit and its corresponding BDD are shown in Figure 1. BDD has two types of vertices. One has index(v) which is a variable number as attribute, and two vertices, low(v) and high(v), as children. The other has value(v) which is 0 or 1. Here v indicates a vertex. We call the former as non-terminal vertex and the latter as terminal vertex. In a BDD, index(v) corresponds to one of input variables which are ordered, and value of index(v) must be smaller than value of index(low(v)) and value of index(high(v)), which is the difference from other binary decision trees.

The BDD G which has a vertex v as root corresponds to the function F_v according to the next rule.

1) if v is a terminal vertex,
   a) if value(v) is 1, then F_v=1.
   b) if value(v) is 0, then F_v=0.
2) if v is non-terminal vertex (its index(v) is i),

$$F_v(X_1, \ldots X_n) = \overline{X_i} \cdot F_{low(v)}(X_1, \ldots X_n) + X_i \cdot F_{high(v)}(X_1, \ldots X_n)$$

If there is a vertex with low(v) = high(v), then such vertex is redundant and eliminated. Also, if two graphs are isomorphic, then they are translated to one graph. So the two circuits are logically equivalent, iff the two corresponding BDD's are isomorphic.

Boolean operations, such as, AND, OR, NOT, etc., are defined as 'apply' and 'reduce' operation for BDD's and so we can translate any circuit to the corresponding BDD in the following way:

First make BDD's for inputs terminals.

Then make BDD's for the nets of the circuit from input to output using the operation 'apply' and 'reduce' (please see [3] for the details of 'apply' and 'reduce').

Figure 1 is an example of the above translation. To compare two circuits, first BDD's for both circuits are calculated independently, and then they are checked to be isomorphic.

BDD calculation time dominates the whole processing time. The required time for one operation of 'apply' and 'reduce' is proportional to the product of each number of vertices in the BDD's to be operated, and so we must keep the size of BDD's as small as possible. As the size heavily depends on variable ordering, finding a good ordering is a key point of this boolean comparison method.

3. A Variable Ordering Algorithm and its Performance

The boolean comparison algorithm shown above was applied to ISCAS benchmark circuits for test pattern generation (some features are shown in Table 1). Two same circuits are processed, and outputs are compared one by one. The results are shown in Table 2. Programs are coded in C and executed on SUN3/260 workstation. Calculation is aborted when the number of nodes in BDD exceeds 100,000 or CPU time exceeds 50,000 seconds. SN181 in the table is a 4bit ALU. In all, four methods of variable ordering were applied: original ordering (the same appearing order in the original ISCAS data format), manual ordering (one who does not know logic gates drew a circuit diagram from net-list description, and the appearing order in the diagram was used), random ordering (randomly generated by a program), and our ordering algorithm (explained later). From the results of the first three ordering methods, we can see the following facts.

(1) Both size of BDD and execution time heavily depend on variable ordering.
(2) Manual ordering shows the best results.
(3) Even if we use the original order, the algorithm is much faster than other ones reported in the literature.

The question why manual ordering gives us good results is partially answered by the following two theorems.

Theorem 1 One of the best orderings for a tree network circuit (numbers of fanouts of all inputs and gates are 1) composed of only AND, OR, and NOT gates is acquired by the following procedure: Traverse the gates from an output to inputs with depth-first order (selection of input nets of a gate is arbitrary), and when we find an input, append that variable to the last of the current ordering (proof is omitted, and an extension to include EOR gates is easy).

Theorem 2 One of the best ordering for a circuit which is composed of only AND, OR, and NOT gates and only one input or gate has more than one fanout (numbers of fanouts of all other inputs and gates are 1) is acquired by the following procedure which is slightly modified from the procedure in theorem 1: Traverse the gates from an output to inputs with depth-first order, but a net which has more than 1 fanout is selected first, and when we find an input, append that variable to the last of the current ordering (proof is omitted, and an extension to include EOR gates is easy).

The above theorems mean that the number of crosspoints of nets in the circuit diagram should be small, that is, natural ordering acquired by drawing the circuit diagram gives us good performance. We conjecture that ISCAS original data is converted from circuit diagrams, and so, the original order is not so bad compared to the random ordering.

We developed an ordering algorithm from the theorems, as shown in Figure 2. It gives a natural ordering by depth-first traverse of a circuit (according to theorem 1) and inputs which have more than 1 fanout are ordered first (according to theorem 2). Sample execution trace of this algorithm to the circuit in Figure 1 is shown in Figure 3. Here we get a better ordering (number of nodes is 6) than that of Figure 1. The results of application of the ordering algorithm in Figure 2 are shown in the last columns in Table 2. The time in the table includes the required time for variable ordering (it is very small). The performance of the algorithm is great and all circuits except c6288 (which has too many levels) can be handled within practical time. This is surprising when we compare it with the results reported in [5,6]. The table shows that it is one hundred times or more faster than other known results. Moreover, our ordering algorithm gives us better performance than manual ordering in almost all circuits. Our algorithm give us worse ordering only in c432. c432 has a couple of many-fanin gates, and those mix up variable ordering. Therefore, we should traverse those gates only after other gates (which are not-many-fanin gates) are traversed. If we use this heuristic, the same ordering as manual one is acquired.

4. More Improvements

Two points of improvements were applied besides variable ordering. They are:

(1) Handling multiple output at the same time,

(2) When calculating BDD's from inputs to output, if the number of vertex for an internal net exceeds a certain limit, that net is cut and is treated as an external terminal. After calculating the BDD for the output and the cut net, they are composed to get the BDD for the original circuit.

The results are shown in Table 3. Variable ordering for (1) are generated by the algorithm in Figure 3, and the cut limit is decided by sample execution. We can indicate several interesting points:

(1) A good variable ordering is not necessarily good for other outputs in the circuit. We can not find any good variable ordering for all outputs of c7552 by the algorithm in Figure 3.

(2) Number of vertex for the output may increase when some nets are cut and are treated as external terminals. We can not find any good cut points for c3540, because any cuts cause not a little increase of the number of nodes in the BDD for the output.

5. Conclusions

We have shown that the boolean comparison method based on BDD with the ordering algorithm in Figure 2 has surprisingly good performance, when we apply it to ISCAS benchmark circuits and compare the results with those of other methods reported in the literature. We also applied the method to real circuits, including control circuits of a main frame computer, and got the same performance as those of ISCAS data.

We have a plan to apply the method to other topics which are closely related to boolean comparison, such as temporal logic based hardware verification [4] and minimization of multi-level boolean expressions [1].

References

[1] D. Bostick, G.D. Hachtel, R.M. Jacoby, M.R. Lightner, P. Moceyunas, C.R. Morrison, and D. Ravenscrofit. The boulder optimal logic design system. In *International Conference on Computer Aided Design*, November 1987.

[2] F. Brglez and H. Fujiwara. A nertral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *IEEE International Symposium on Circuits and Systems*, June 1985.

[3] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computer*, C-35(8):667--691, August 1986.

[4] M. Fujita, H. Tanaka, and T. Moto-oka. Logic design assistance with temporal logic. In *IFIP 7th Computer Hardware Description Languages and their Application*, IFIP, August 1985.

[5] G.D. Hachtel and R.M. Jacoby. Algorithms for multi-level tautology and equivalence. In *IEEE International Symposium on Circuits and Systems*, June 1985.

[6] R. Wei and A.L. Sangiovanni-Vincentelli. Proteus: a logic verification system for combinational circuits. In *International Test Conference*, 1986.

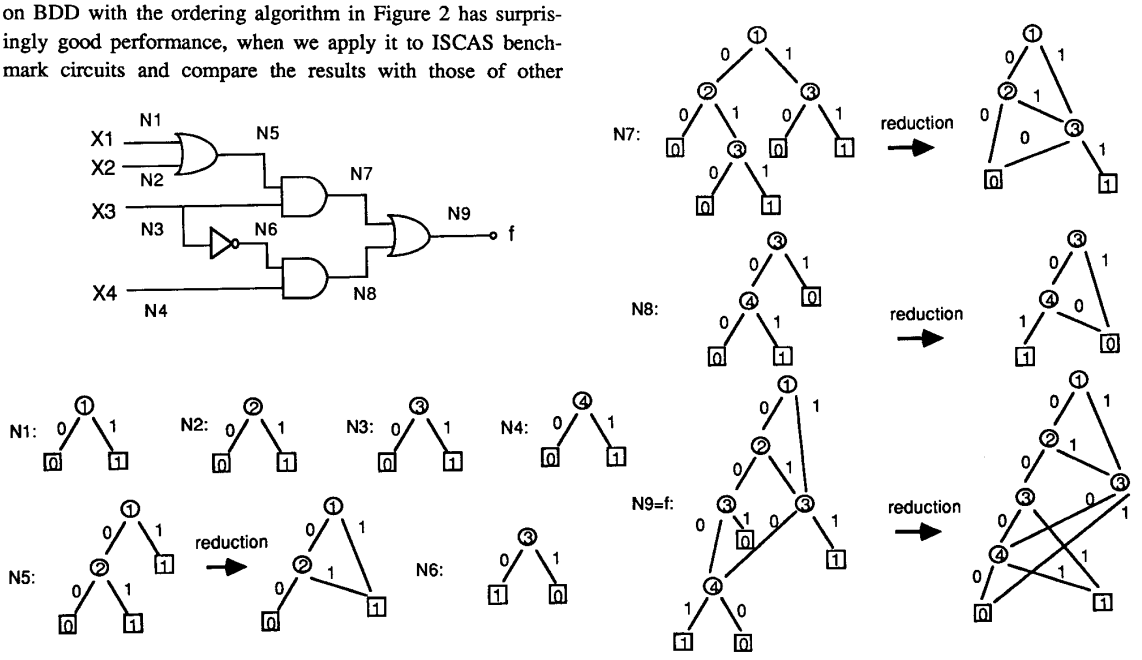
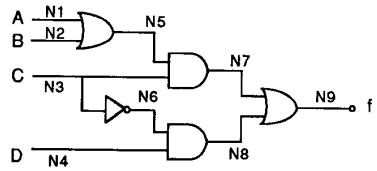


Figure 1 A circuit and its translation into BDD (Binary Decision Diagram)

```
{ Initially all nets must be marked off }
procedure makeOrder(N);
begin
  foreach I ∈ Set of all input nets of the gate to which N is connected do
  begin
    if I is marked then continue;
    if I is directly connected to a primal input then
      if I is connected more than one gate then
      begin
        FANOUT2UP := I;
        if I is not in ORDER then ORDER := append(ORDER,I);
      end else FANOUT1LIST := append(FANOUT1LIST, I);
    else makeOrder(I);
  end;
  if FANOUT2UP <> undef then
  begin
    Insert FANOUT1LIST into ORDER after FANOUT2UP;
    FANOUT1LIST := NIL;
  end;
  Mark N;
  return;
end;
```

Figure 2 Variable ordering algorithm



traverse of circuit	FANOUT2UP	FANOUT1LIST	ORDER
N9	undef	NIL	NIL
N7	undef	NIL	NIL
N5	undef	NIL	NIL
N1	undef	A	NIL
N2	undef	A,B	NIL
N3	C	NIL	C
N8	C	NIL	C,A,B
N6	C	NIL	C,A,B
N4	C	D	C,A,B
finish	C	NIL	C,A,B,D

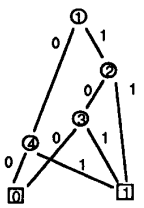


Figure 3 An execution trace of the variable ordering algorithm

Name	Inputs	Outputs	Levels	Gates	Max gates for single output	Max inputs for single output
SN181	14	8	9	34	55	14
c432	36	7	7	203	145	36
c499	41	32	11	275	102	41
c880	60	26	24	469	130	45
c1355	41	32	24	619	322	41
c1908	33	25	40	938	557	33
c2670	233	140	32	1566	828	122
c3540	50	22	47	1741	1433	50
c5315	178	123	49	2608	937	67
c6288	32	32	124	2480	2327	32
c7552	207	108	43	3827	1096	194

Table 1 ISCAS benchmark circuits [2]

Name	Original order		Manual order		Random order		Our algorithm order	
	Time	Maximum nodes	Time	Maximum nodes	Time	Maximum nodes	Time	Maximum nodes
SN181	8	339	7	197	8	578	6	213
c432	109	1146	53	442	unable	>100000	1423	6196
c499	928	9020	673	4661	unable	>100000	673	4661
c880	unable	>100000	85	3421	unable	>100000	55	3359
c1355	675	9020	——	——	unable	>100000	1009	4661
c1908	965	2912	1800	4505	>50000	——	778	3076
c2670	unable	>100000	——	——	unable	>100000	338	14763
c3540	unable	>100000	——	——	unable	>100000	12620	53460
c5315	2140	11807	——	——	unable	>100000	498	3441
c6288	unable	>100000	——	——	unable	>100000	unable	>100000
c7552	unable	>100000	——	——	unable	>100000	383	2096

—— : not tried
SUN3/260 CPU TIME, second
>100000 : number of nodes exceeds 100,000
>50000 : CPU TIME exceeds 50,000 seconds

Table 2 Boolean comparison results of benchmark circuits with various variable orderings

Name	Multiple output Time	Cut & compose Time
SN181	3	7
c432	529	732
c499	312	——
c880	34	46
c1355	——	627
c1908	441	736
c2670	336	280
c3540	7613	unable
c5315	253	460
c6288	unable	unable
c7552	unable	365

Ordering is automatically generated.
The aboves are applied separately.

Table 3 Results of composition and multiple output handling