A Partitioning-based Logic Optimization Method for Large Scale Circuits with Boolean Matrix

Yuichi Nakamura and Takeshi Yoshimura
C&C Research Laboratories, NEC Corporation
Miyazaki, Miyamae-ku, Kawasaki 216, Japan

Abstract—

This paper presents a new logic partitioning method for optimizing large scale circuits. The proposed method partitions a given circuit into transitive fanin-disjoint sub-circuits by matrix operations, so that various optimization methods can be applied to each partitioned sub-circuit instead of the whole circuit. Experimental results show that the proposed method achieves high-quality design comparable to the one optimized for the whole circuits, with much shorter time(1/20). Thus, the circuits with over 10,000 gates can be optimized by the proposed partitioning.

I. Introduction

Over the past decade, many techniques have been proposed for technology independent area optimization for combinational circuits. Now a day, the size of the circuits to be optimized is growing rapidly, and logic synthesis system is requested to optimize given circuit to meet area and delay constraints specified by users. Since logic optimization techniques use complicated operations based on Boolean algebra, they require huge amount of memory space and computation time for large circuits. Therefore conventional algorithms can optimize only limited size of the circuits.

For example, transduction method [1] can not optimize C6288 and C7552 in the MCNC benchmark set, because of the complicated structure in case of C6288 and large circuit size in case of C7552. On the other hand, the circuit flattening and 2-level minimization[2] method cannot process circuits with over about 16 fanins. It takes much time to decompose more than 10,000 literals circuit. Although such methods analyze the whole circuit to obtain circuit information, they deal with only very limited portions of the circuits in the optimization process.

There is the following property:

Property 1 If two circuits have no common transitive fanin(TFIN), there are no possibility for inter-circuits optimizations.

32nd ACM/IEEE Design Automation Conference ®

Thus, if the circuits are partitioned into sub-circuits which have no common transitive fanin, and each of them is optimized independently, the obtained results are expected to have almost the same quality as the ones obtained by whole circuit optimization.

First, this paper proposes a new partitioning method for circuit optimization based on the above property. The proposed partitioning technique finds and extracts sub-circuits which have disjoint transitive fanins each other, by using Boolean matrix based operations. After partitioning, the following optimization techniques are applied to the partitioned sub-circuits;

1. decomposition,
2. flattening, 2-level minimization, and decomposition,
3. transduction method after 2.

Next, the results of the computational experiments are shown, which are very encouraging.

In this paper, logic circuits are expressed by the Boolean Network[3] where each node represents a logic expression, and each edge represents fanin, fanout or logic connection.

II. Conventional Approaches

When a given circuit is too large, it is natural to partition the circuit into sub-circuits having suitable size to be optimized. Several partitioning techniques based on the re-convergent structure have been proposed[4][5](Figure 1).

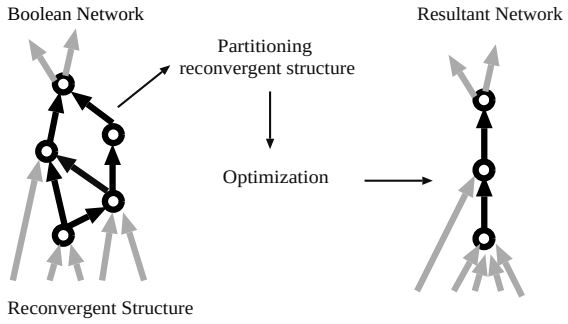

Figure 1: Re-convergency based partitioning

The re-convergent structure is defined as a sub-circuit which includes more than two paths starting from a node with multiple fanouts and converging at another nodes. It is highly expected that circuits with such structure have redundancies because the structure has usually many duplicate signals.

In these conventional methods, first, the circuits with the re-convergent structure are extracted from a given circuit. Next, the extracted sub-circuits are optimized by flattening based method. Then, the flattening process reduces redundancy in the re-convergent structure. After the process, a 2-level optimization algorithm is applied and decomposed for re-synthesis.

Since these methods are based only on re-convergent structure, there are following two disadvantages. One is that the method can not extract sub-circuits which have redundancies in the sub-circuits whose types are not re-convergent structures. For example a *half* re-convergent structure which has multiple fanout nodes but does not have the nodes at which the signal from the multiple fanouts converge. The other is that, the partitioned sub-circuits sometimes become too large and they cannot be optimized because of large computational time. In order to avoid these two disadvantages, we propose a new method which partitions the given circuits quickly by using common TFIN informations under limitation of the number of fanins in partitioned sub-circuits.

III. Partitioning with Boolean Matrix

Definition

Boolean network(BN) is defined as $(PI \cup PO \cup INT, E)$, where PI, PO and INT are the set of primary inputs, the set of primary outputs and the set of internal nodes, respectively. Each node in INT represents the sum of products of Boolean equations. $E \subseteq \{(a,b) | a,b \in PI \cup PO \cup INT\}$ is the set of directed edges which presents fanin and fanout information. An edge $(a,b) \in E$ means that a is fanin of b.

Node Literal Cube Incidence(NLCI) Matrix[6] is known as a common cube extraction technique on a network. NLCI matrix is a 0-1 Boolean matrix with rows R and columns C, where each row and each column corresponds to a cube and a literal, respectively. Element at (i,j) is "1" if and only if a cube c corresponding to row, $i \in R$, has a literal x corresponding to column $j \in C$. *Rectangle* of NLCI matrix is defined as a subset of rows, R, where $|R| \geq 2$ and a subset of columns, C, where $|C| \geq 2$, such that $(i,j) = 1$ for all $i \in R$, $j \in C$. The rectangle in NLCI matrix presents common cube in networks.

The proposed method partitions the circuit using four matrices which are defined by extending the NLCI matrix.

NVI matrix Node Value Incidence (NVI) matrix is defined as a 0-1 Boolean matrix with rows R and columns C. Each row corresponds to a node $x \in INT \cup PO$, and each column corresponds to a node $a \in PI \cup INT$. The element at $(i,j) = 1$, if and only if a node x corresponding to a row $i \in R$ has a fanin node a corresponding to a column $j \in C$ (Figure 2).

TNVI matrix Transitive Node Value Incidence (TNVI) matrix is defined as a 0-1 Boolean matrix, where similar to NLCI matrix, each row corresponds to a node and each column corresponds to a element of TFIN. The element at $(i,j) = 1$, if and only if, a node z corresponding to row $i \in R$, has a TFIN , c, corresponding to column $j \in C$ (Figure 4).

p-TNVI matrix p-TNVI matrix is a TNVI matrix such that TFIN level which is the number of nodes along the path from each node to TFIN, does not exceed i. 2-TNVI matrix presents the fanin nodes, and the fanin nodes of fanins for a node.

p-$q$fanin-TNVI matrix p-$q$fanin-TNVI matrix is a p-TNVI matrix such that each column size, the number of TFIN of each node, does not exceed q.

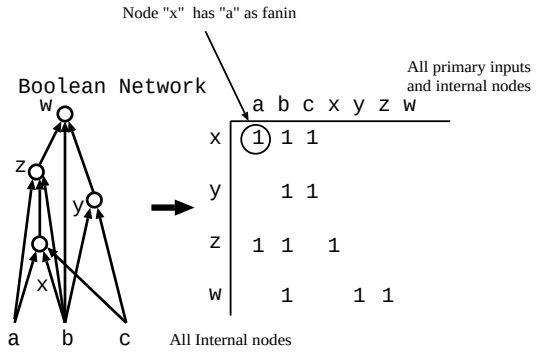

Figure 2: NVI matrix

Making Up Matrix

The NVI matrix can easily be built up by assigning each INT and PO node to each row, and each fanin of the PI and INT node to each column, and setting element (i,j) as 1 if the node corresponding to row, R_i, has a fanin corresponding to column, C_j. On the other hand, p-$q$fanin-TNVI matrix can be built up by tracing the fanin of nodes in matrix, as shown in the following. For the ease of explanation, the algorithm is presented for the case where $p = 2$ and $q = 16$ (Figure 3).

```
Making_up_2-16fanin-TNVI_Matrix
{
   construct NVI_matrix;
   alloc(2-16fanin-TNVImatrix);
   i = 0;
   set R(i) = i-th row in NVI matrix;
   foreach_row(R(i++)) {
      set X = node corresponding to R(i);
      /* Row R(i) corresponds to INT node X */
      k = 0;
      set C(j) = j-th column;
      /*  Row R(i) corresponds to
                    INT node or primary input Y */
      set EL(i, j) =  j-th column element of R(i);
      foreach_element(EL(i, j)) {
         set Y = node corresponding to C(j);
         if ( Y != primary input ) {
            Row(k++) = row corresponding to Y;
         }
      }
      foreach(Row(k)){
         Row(k) = row_or(Row(k), Row(k-1)); /* (a) */
      }
      TMP_ROW = row_or(R(i), Row(k));
      if(size(TMP_ROW <= 16 ){              /* (b) */
         insert TMP_ROW to 2-16fanin-TNVI;
      }else{
         insert R(i) to 2-16fanin-TNVI;
      }
   }
 }
}
```

Figure 3: Algorithm for building up 2-16fanin TNVI matrix

The TFIN of node x which corresponds to row $Row(i)$, can be calculated by applying logic-or operations (Figure 4) for the $Row(i)$ and the rows which is corresponding to the fanin nodes of node x (see line *(a)* in Figure 3).

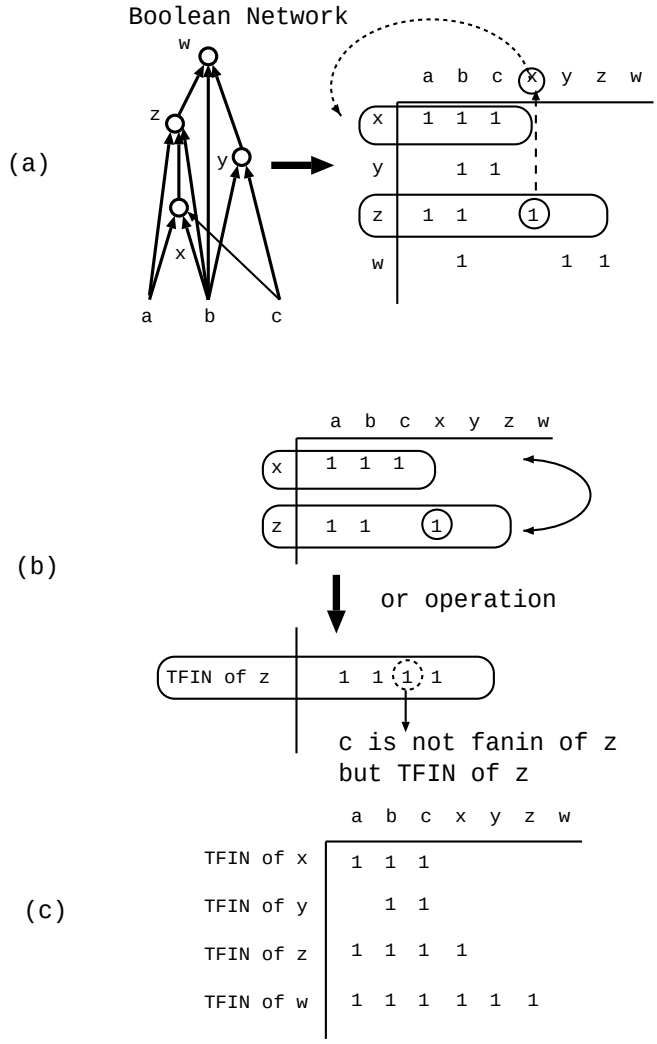

Figure 4: 2-fanin-NVI matrix

In order to find the fanin nodes of x, the following operations are applied; 1) search for the elements of $Row(i)$ and 2) find the nodes corresponding to the column of the elements.

Moreover, it is possible to give a limit of the number of fanins (see line *(b)* in Figure 3). Although, the algorithm is explained in Figure 3 where $p = 2$ and $q = 16$, it is easy to verify that p+1-TNVI can be constructed from p-TNVI by recursively applying the algorithm.

The p-TNVI matrix has the following property.

Property 2

A common TFIN between node x and y within p-level exists, if and only if, the result of logic-and operation for R_x and R_y is not empty, that is,

$$row\_and(R_x, R_y) \neq \phi$$

at rows corresponding to nodes x and y in p-TNVI matrix.

The computational complexity to make up a NVI matrix is $O(N)$, where N is the number of INT and PO nodes, since each row in NVI matrix is constructed by scanning fanins of one INT and PO node. In order to build p-TNVI matrix, it is required to construct matrix p times. Thus, the complexity is $O(pN)$. In actual problem, however, the complexity for making up p-TNVI is equivalent to O(N), since p is a constant and small enough.

IV. Logic Optimization by Logic Partitioning

In this section, two logic optimization algorithms based on the proposed logic partitioning, are described. In the first algorithm, a given circuit is partitioned into sub-circuits whose fanins(not transitive) are disjoint each other. Then the partitioned sub-circuits are optimized by a logic decomposition technique.

In the second algorithm, a given circuit is partitioned into sub-circuits whose TFIN are disjoint each other. Then the partitioned sub-circuits are optimized by flattening, 2-level minimization, decomposition and transduction techniques.

Common Fanin Based Partitioning

From the property presented in section III, it is shown that node x and y have common fanins, if the result of logic-and operation between two Rows, R_x and R_y in NVI matrix corresponding to node x and y, respectively, is not empty. In the rectangle (R, C) extracted from NVI matrix, rows R correspond to the nodes which have common fanins among them and column C corresponds to the common fanins.

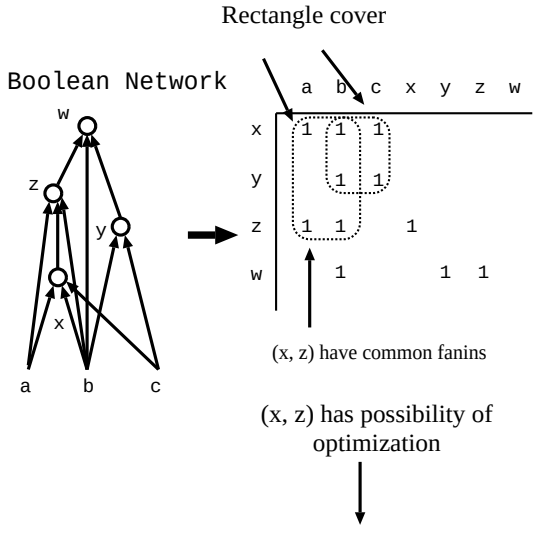

Figure 5: Rectangle extract

In the example of Figure 5, two rectangles $((x, z), (a, b))$ and $((x, y), (b, c))$ are extracted. As for the former one, both x and z have common fanins a and b. Thus, proposed algorithm calculates the rectangle in NVI matrix, and extracts the sub-circuit corresponding to R from a whole circuit. Then, logic optimization algorithms are applied to the partitioned sub-circuits.

In order to find a maximal node set with common fanins, the proposed algorithm finds a maximal rectangle, because the rows of the rectangle in NVI matrix corresponds to a node set with common fanins . The rectangle covering problem is NP-hard even in Boolean matrix. However, there is a very fast heuristic rectangle covering algorithm named pingpong[6], which is used in common cube extraction. The proposed algorithm applies the pingpong heuristic to find the rectangle.

The following algorithm finds a rectangle, then extracts the sub-circuit corresponding to the row of the rectangle, from the given circuit(Figure 6). The extracted circuit is handed over total logic optimization process.

```
Logic_partitioning_algorithm_for_logic_optimization
{
  construct NVI matrix;
     i = 0;
     NVI(0) = NVI;
     rect(R, C) = pingpong(NVI(0));
     while( rect(R, C) == NULL ){
        /* rectangle extraction */
        Nodes = nodes corresponding to R;
        /* select the partitioned nodes*/
        Opt_nodes = optimization(Nodes);
         if ( # of literals(Nodes)
              >= # of literals(Opt_nodes)) { /* (c) */
              replace Nodes by Opt_nodes;
         }
        NVI(i++) = remove R from NVI(i);
         /* reshape the NVI(i)*/
        rect(R, C) = pingpong(NVI(i));
     }
}
```

Figure 6: Algorithm for partitioning method for logic optimization by using NVI matrix

A comparison between the sub-circuits before and after optimization gives a guarantee not to increase the number of literals by the subsequent replacement, see line (c) in Figure 6. After extraction, whether the replacement is successful or not, the rows of the rectangle are deleted from NVI matrix. The process continues till NVI matrix does include no rectangle. In case of decomposition, this comparison is not required because the number of literals must be decreased by the decomposition.

Table 1 shows the results of the proposed method, " Partitioning decomposition", for MCNC complex benchmarks. It applied the fast_extract[7] which is one of fast logic decomposition algorithms, to the partitioned subcircuits. For comparison, the results of the fast_extract for "Whole circuit decomposition" are also shown. In Table 1, "# of literals" is the number of literals in factored form and "time" is CPU time on Sparc Station 2(28.5 MIPS, 32MB memory).

Table 1: Logic decomposition

	Whole circuit decomposition		Partitioning decomposition	
	# of literals	time sec	# of literals	time sec
C6288	4226	85	4229	16
C7552	2543	34	2551	12
S15850	3952	60	3983	22
S38584	13621	501	13767	33
clma	20455	460	20639	21

The results of Table 1 show that the difference of the number of literals between the whole decomposition and the partitioned decomposition is at most 1%, and that, the proposed partitioning based method is 15-22 times faster than the other when a given circuit has more than 10,000 literals.

TFIN based partitioning

TFIN based partitioning for logic optimization can be implemented by replacing NVI matrix with TNVI matrix in above common fanin (not transitive) partitioning algorithm.

In general, the reasonable limit of circuit size is about 16 fanins for optimization by flattening and 2-level minimization. Thus, the proposed technique partitions up to 16 fanins sub-circuits which do not have common fanins. Table 2 and Table 3 show the results of proposed method which uses the TFIN based partitioning. The optimization algorithms used in the experiments are flattening, 2-level minimization and decomposition.

For TNVI-Matrix, 5 level and 16 fanin are given as constraint. For comparison to the proposed method, the results of standard script of UCB's SIS are added to the tables. The results of SIS are obtained by applying SIS's transduction method, *full_simplify* after *sct.boolean* [8]. In MCNC benchmark data, since C6288 is a multiplier and C7552, S15850, and S38417 are too large, it is difficult to construct BDD[9] for optimization. Therefore, the transduction algorithm cannot be applied to such circuits.

Table 2 shows the results of the comparison of *full_simplify* after *sct.boolean* and the proposed method for the circuits.

Table 2 : Compared with full_simplify

	full_simplify			Proposed		
	# of lits	delay ns	CPU sec	# of lits	delay ns	CPU sec
C3540	1248	59	1543	1290	47	45
C5313	1709	50	1316	1745	37	72

In the results of Table 2, though the proposed method produced a little larger circuits, their computation time are much shorter than *sct.boolean* and *full_simplify*. The differences of the numbers of literals are less than 3%, and the proposed method is more than about 20 times faster than *full_simplify*. The results of *full_simplify* and the proposed method are mapped with real CMOS technology library, and the amounts of static delay are calculated. The amounts of circuits delay of the proposed method are up to 26% smaller than *full_simplify*, since the proposed method based on flattening for limited area reduces logic levels instead of transforming of gates and nets.

If the proposed partitioning method extract good subcircuits, the method of flattening, 2-level logic minimization and decomposition can produce the similar result as transduction method, since local flattening and resynthesis by 2-level minimization and decomposition can delete the same redundancies as transduction. The reason why *full_simplify* can generate slightly better results in area than proposed method, is that BDD have the wider logic information than flattening, and the proposed partitioning method sometimes fails to find good sub-circuits.

Since the proposed method optimizes the circuits within a limited area, it fails to find out the possibility for the optimization across two partitioned sub-circuits. In order to avoid this disadvantage, it is effective to apply the proposed method repeatedly. For example, the TVNI

matrix in Figure 7 has two rectangles, A and B. Since A and B have equal size, the proposed algorithm can select either one. If B is selected, all nodes (a, b, c, d, e) are optimized. However, when A is selected, the nodes (c, d, e) are optimized and new nodes (r, s) are generated. After this operation, B' is selected and the nodes (a, b) are optimized to produce the new nodes (p, q).

In the next iteration, the new TVNI matrix is constructed, the new rectangle is selected and the the nodes (p, q, r, s) are optimized. Thus, optimizing the two parts ((a, b) and (c, d, e)) obtains similar results as optimizing the one part (a, b, c, d, e).

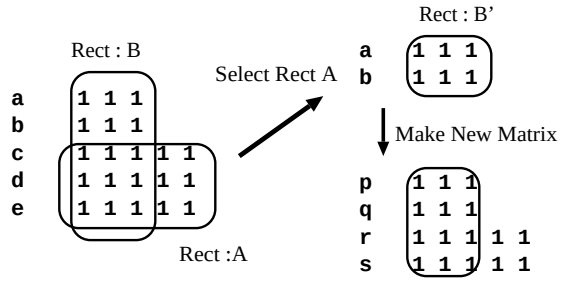

Figure 7: Rectangle Selection

Table 3 : Compared with *sct.boolean*

	sct.boolean			Proposed		
	# of lits	delay ns	CPU sec	# of lits	delay ns	CPU sec
C6288	3550	151	1484	3389	136	161
C7552	2297	63	989	2157	52	144
S15850	2413	67	964	2280	58	361
S38584	12847	189	20348	12156	177	1892
clma	11932	66	25952	11197	61	2351

Table 3 shows that comparison of results obtained by applying the proposed method 3 times repeatedly and SIS's *sct.boolean* for large circuits where BDD based approach cannot be applied. Table 3 shows that the results of proposed methods are up to 7% smaller in area, up to 18% shorter in delay and 10 times faster than *sct.boolean*.

Table 4 shows the effectiveness of applying the *full_simplify* algorithm to each partitioned sub-circuit to obtain the smaller results after the optimization shown in Table 3.

By means of partitioning, full_simplify can be applied to each partitioned sub-circuit, even for the circuits such as C6288 or C7552 which are too large to apply transduction algorithms to whole circuits. Since partitioned sub-circuits are small enough, full_simplify can optimize them after flattening, 2-level minimization and decomposition.

Table 4 : Partially *full_simplify*

	# of lits	# of cells	delay	CPU
C6288	2975	2505	141	332
C7552	2094	2205	46	251
S15850	2217	6241	69	983
S38584	12008	19312	185	4857
clma	11072	12041	64	6648

The results of Table 4 show that the area of circuits is slightly decreased by using transduction method after decomposition. However, the delay and CPU time are increased since the transduction method can transform the sub-circuits having wider area. If a well-balanced circuit is required in a short time, it is not necessary to apply the transduction method after the proposed partitioning.

V. Conclusion

This paper proposed a new partitioning method to optimize large scale circuits. The method makes up the Boolean matrixes and extracts the rectangle from the matrices so as to partition a given circuit to sub-circuits which have no common fanins each other. The following optimization methods apply to partitioned sub-circuits.

1. decomposition,
2. flattening, 2-level minimization, and decomposition,
3. transduction method after 2nd optimization.

In the case of 1, the proposed method can decompose about 22 times faster than the decomposition method for a whole circuit in case of over 20,000 literals circuit. In the case of 2, the method can produce a slightly larger but much shorter delay results than a a transduction method. Moreover the smaller and shorter delay results are obtained than SIS's standard script in a short time.

In the case of 3, the method is applied to the circuits which are too large for the transduction method to handle, and gets smaller results than the case 2.

The proposed method can handle more than 100,000 literal circuits in a practical CPU time and get high performance circuits, because the method can run in almost linear time to a given circuit size and optimize the small sub-circuits in detail.

References

[1] S. Muroga, Y. Kambayashi, H. C. Lai and J. N. Culliney, " The transduction method-design of logic networks based on permissible function," IEEE Transaction on Computer C-38(10) October 1989, pp.1404-1424.

[2] R.K.Brayton, G.D. Hachel, C.McMullen and A. Sangiovanni-Vincentalli, "Logic minimization algorithms for VLSI synthesis," Kluwer Academic Publishers, Boston, 1984.

[3] R. K. Brayton, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A Multiple-Level Logic Optimization System", IEEE Transaction on CAD, CAD-6(6) July 1987, pp.1062-1081.

[4] S. Dey, F. Beglez, and G. Kedem, "Corolla Based Circuit Partitioning and Resynthesis", 27th ACM/IEEE DAC 1990, pp.607-615.

[5] Y. Nakamura, K. Wakabayashi, and T. Fujita, "A Partitioning Method for Area Optimization by tree analysis" Logic Synthesis and Optimization, Kluwer Academic Publishers 1993, pp. 127-144.

[6] G. De Micheli, A. Sangiovanni-Vincentelli, and P. Antognetti "Design System for VLSI Circuits : Logic Synthesis and Silicon Compilation" Martinus Nijhoff Publishers, 1987, pp.197-248.

[7] J. Rajski, and J. Vasudevamurthy, " Testability Preserving Transformations in Multi-level Logic Synthesis", IEEE ITC 1990, pp.265-273.

[8] H. Savoj, R. K. Brayton, and H. J. Touati, "Extracting Local Don't Cares for Network Optimization" IEEE ICCAD 1991, pp.514-518.

[9] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Transaction on Computers, C-35(8) August 1986, pp677-691.