Proof. omitted.

### 4.2 The Interference Effect

In this section, we discuss the other effect, the *interference effect*, which may also cause some node's delay solution from the MCMF algorithm to be larger than its label. Basically, the interference effect comes from the sequential processing of nodes' labels. During the labeling phase, we evaluate a node's label only based on the labels of nodes in its input cone. However, a node's label may be interfered by the *k*-cutsets of nodes outside its input cone. Consider the two input cones of nodes $n_1$ and $n_2$ in Fig. 6(a). Suppose $n_1$ is processed first and we find a *k*-cutset $C_1$ for node $n_1$. Then, we process node $n_2$ and also find a *k*-cutset $C_2$ for $n_2$. Because of additional bscs from $C_2$ are inserted inside $n_1$'s input cone, the delay of $n_1$ may be increased accidentally. Therefore, during the labeling phase, ignoring the effect of the *k*-cutsets of nodes outside $n_1$'s input cone may cause $n_1$'s label to be too optimistic. Fig. 6(b) is another case which also causes such effect. Similar to the bsc chain effect, it is possible that there exist other *k*-cutsets which do not have the interference effect such as Fig. 6(c) and Fig. 6(d).

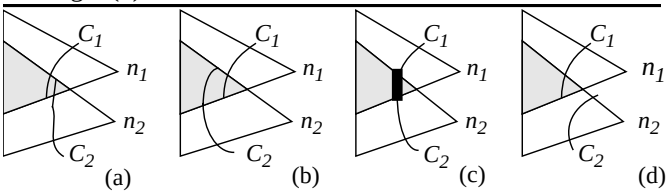


**Fig. 6** : Interference effect between different cutsets.

## 5 Experimental Results

In this experiment, we assume that the delays of all wires and bscs are 1; i.e., $d(e) = d(bsc) = 1$. Our experimental results are shown in Table 1 with the dependency constraint of 20. Column one gives the name of each circuit. Column two shows the critical path delay without inserting any bscs. Column three shows the largest label among all POs. The largest label of a circuit is a lower bound of the critical path delay. After inserting bscs by the algorithm, column four shows the circuit delay and column five shows the number of bscs inserted. We also re-implement the algorithm [16] and column six shows the results of delay and column seven shows the results of bscs needed. For example, in Table 1, the critical path delay of C5315 is originally 49 without inserting bscs. Under the dependency constraint of 20, the largest label is 50. Our heuristic requires 68 bscs to achieve the delay of 50 which must be an optimal solution. We also highlight our results which are the same as the lower bounds, *i.e.* optimal solutions.

## 6 Conclusions

In this paper, we first presented an estimation formula to estimate each node's label, *i.e.*, the lower bound of its delay. Since the bsc chain effect and the interference effect may occur during the bsc insertion process, the delay of some nodes may not achieve their labels by the formula. Then we also explored the reasons which make the two effects occur. To alleviate these effects, we further proposed some heuristics on which a bsc insertion algorithm was developed based. Finally, the experimental results of ISCAS85 benchmark circuits show that our heuristic algorithm can achieve or be very close to the labels (**optimal solution**).

Table 1: Dependency constraint = 20

| circuit | initial delay | largest label | OTP | | [16] | |
|---|---|---|---|---|---|---|
| | | | delay | #bscs | delay | #bscs |
| c432 | 17 | 19 | 20 | 46 | 23 | 20 |
| c499 | 11 | 12 | 12 | 8 | 12 | 8 |
| c880 | 23 | 24 | 24 | 30 | 29 | 15 |
| c1355 | 23 | 24 | 24 | 8 | 24 | 8 |
| c1908 | 40 | 41 | 41 | 20 | 42 | 14 |
| c2670 | 32 | 34 | 34 | 39 | 39 | 32 |
| c3540 | 47 | 48 | 50 | 92 | 51 | 68 |
| c5315 | 49 | 50 | 50 | 68 | 51 | 47 |
| c6288 | 124 | 127 | 128 | 73 | 133 | 55 |
| c7552 | 43 | 43 | 43 | 123 | 45 | 89 |

## 7 References

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital System Testing and Testable Design*, Computer Science Press, New York, 1990.

[2] V. D. Agrawal and S. C. Seth, *Test Generation for VLSI Chips*, IEEE Computer Society, 1988.

[3] P. H. Bardell, W. H. McAnney and J. Savir, *Built-In Test for VLSI: Pseudo-random Techniques*, John Wiley and Sons, New York, 1987.

[4] Z. Barzilai, J. Savir, G. Markowsky and M. G. Smith, "The Weighted Syndrome Sums Approach to VLSI Testing", *IEEE Trans. on Computers*, vol. C-30, no. 12, pp.996-1000, Dec. 1981.

[5] S. N. Bhatt, F. R. K. Chung and A. L. Rosenburg, "Partitioning Circuits for Improved Testability", *Proc. of 4th MIT Conf. on Advanced Research in VLSI*, pp. 91-106, Apr. 1986.

[6] J. Cong and Y. Ding, FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs, *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 13, no. 1, Jan. 1994.

[7] T. Cormen, C. Leiserson, and R. Rivest, *Algorithms*, Cambridge, MA: MIT press, 1990.

[8] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Prentice Hall, Inc., 1974.

[9] S. Devadas, A. Ghosh and K. Keutzer, *Logic Synthesis*, McGraw-Hill, Inc., 1994.

[10] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton, NJ: Princeton Univ. Press, 1962.

[11] W. B. Jone and C. A. Papachristou, "A Coordinate Circuit Partitioning and Test Generation Method for Pseudo-Exhaustive Testing of VLSI Circuits", *IEEE Trans. on CAD*, vol. 14, no. 3, pp. 374-384, Mar. 1995.

[12] E. J. McCluskey, "Verification Testing - A Pseudoexhaustive Test Technique", *IEEE Trans. on Computers*, vol. C-33, no. 6, pp. 541-546, Jun. 1984.

[13] E. J. McCluskey and S. Bozorgui-Nesbat, " Design for Autonomous Test", *IEEE Trans. on Computers*, vol. C-30, no. 11, pp. 866-875, Nov. 1981.

[14] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Inc., 1994.

[15] M. W. Roberts and P. K. Lala, "An Algorithm for the Partitioning of Logic Circuits", *IEE Proc.*, vol. 131, pt. E, no. 4, Jul. 1984.

[16] R. Srinivasan, S. K. Gupta and M. A. Breuer, "An Efficient Partitioning Strategy for Pseudo-Exhaustive Testing", *Proc. of Design Automation Conf.*, pp. 242-248, Jun. 1993.

0-7803-5482-6/99/$10.00 ©2000 IEEE

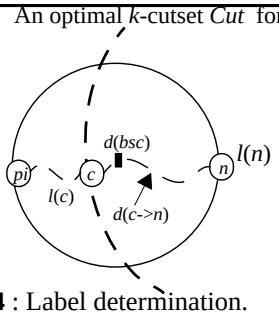


**Fig. 4** : Label determination.

define a node whose label satisfies EQ 2 and EQ 3 to be a *timing feasible* node for $bl(n)$.

$$l(c) \le bl(n)\text{-}d(c\text{->}n)\text{-}d(bsc) \text{ if } c \text{ is not a PI} \quad \text{(EQ 2)}$$

$$l(pi) \le bl(n)\text{-}d(pi\text{->}n) \quad \text{(EQ 3)}$$

In order to achieve $l(n) = bl(n)$, each node in the *k*-cutset must be a timing feasible node for $bl(n)$. Note that, for each node *c* in the input cone of node *n*, the values of $l(c)$, $d(c\text{->}n)$, $d(bsc)$ and $bl(n)$ can be pre-computed. Hence, we can easily determine whether node *c* is timing feasible for $bl(n)$.

Return to Fig. 3 where nodes *h, i, j, k,* and *l* have been labeled and label of *m* is under consideration. Since $l(j) = 1$ and $l(l) = 3$, according to Definition 1 and EQ 1, we have $bl(m) = 4$ and $4 \le l(m) \le 5$. In order to achieve $l(m) = bl(m) = 4$, we can find that node *j* is a timing feasible node because $l(j) \le bl(m)\text{-}d(j\text{->}m)\text{-}d(bsc)$, *i.e.*, $1 \le 4\text{-}1\text{-}1$. Similarly, node *i* is also a timing feasible node for $bl(m) = 4$. On the other hand, node *l* is not a timing feasible node for $bl(m) = 4$ because $l(l) > bl(m)\text{-}d(l\text{->}m)\text{-}d(bsc)$, *i.e.*, $3 > 4\text{-}1\text{-}1$. Also, all PIs which are considered as bscs are timing feasible from EQ 3. In Fig. 3, all timing feasible nodes for $l(m)=4$ are marked.

Now, let us check the feasibility of $bl(n)$ for node *n*'s label. Assume all the labels of nodes in *n*'s input cone $Cone(n)$, except *n*, are already computed. Also, by applying EQ 2 and EQ 3, we can find all timing feasible nodes in $Cone(n)$. With these information, we are ready to use the MFMC algorithm to resolve the yes/no problem, *i.e.*, to decide whether there is a *k*-cutset containing only timing feasible nodes for $bl(n)$. To apply the MFMC algorithm, we transform $Cone(n)$ into a weighted graph as follows. We first add two nodes, the source node *S* and the destination node *D*. We also add the edges connecting *S* to each PI and the edges connecting node *n* to *D*. Then, the weight of each timing feasible node for $bl(n)$ is set to *1* and others to *infinite*. When applying the MFMC algorithm on this transformed graph, the MFMC algorithm tries to select a cutset whose total weight is minimum. The weights which we assign on the transformed graph cause the MFMC algorithm to select (cut) only timing feasible nodes because the weights of timing feasible nodes are much smaller than others. In addition, the MFMC algorithm will try to find the minimum number of possible timing feasible nodes to form a cutset. If the size of the selected cutset is greater than *k*, it means that there does not exist a *k*-cutset which contains only timing feasible nodes, *i.e.*, $l(n) = bl(n)$ is not feasible. On the other hand, if the cutset size is not greater than *k*, though in many cases, $l(n) = bl(n)$ is feasible, there still exists some case, it is NOT feasible. We detail this discussion in Section 4.1.

## 4 Bsc Insertion

The bsc insertion process is iteratively performed from POs to PIs. Initially, we put all POs into a processing list. In each iteration, we select a node *n* with the maximal label from the list. After finding a *k*-cutset for node *n*, we insert bscs on the fanout stems of nodes in the *k*-cutset. Then, node *n* is removed from the processing list and the nodes in the *k*-cutset are added into the list. This process continues until there is no node in the processing list.

In the following subsections, we discuss two effects, *the bsc chain effect* and *the interference effect*, which may cause some node's delay to be larger than its label after bsc insertion. Also, these two effects are explained in details and the heuristics to alleviate these two effects are also discussed.

### 4.1 The Bsc Chain Effect

If the cut size returned from MFMC is greater than *k*, we set $l(n) = bl(n)+1$. On the other hand, if the cut size from MFMC is not greater than *k*, we set $l(n)$ to $bl(n)$. However, the assignment of $l(n) = bl(n)$ for the later case (cut size $\le k$) may be too optimistic. The reason is as follows. Even though each node in the *k*-cutset suggested by MFMC is timing feasible, all together they may not be timing feasible for $l(n) = bl(n)$. For example, assume all labels have been obtained and consider to insert bscs for node *l* in Fig. 5(a) where the dependency constraint is *3*. The MFMC algorithm may return the 3-cutset {*f, g, j*} for *l* as shown in Fig. 5(b). Since the cutset size is equal to the dependency constraint, 3, the labeling algorithm sets $l(l)$ to *5*. Note that, in the configuration of this 3-cutset, node *j* is in the fanout of node *f* so there are two bscs in the critical path $a\text{->}f\text{->}j\text{->}i\text{->}l$. In this special case, after inserting bscs for this 3-cutset, the delay of node *l* is *6* (= $2*d(bsc)+4*d(e)$) which is larger than its label, *5*. In this example, the *3*-cutset has the configuration that *f* is in the fanin of *j*. If the configuration of the *k*-cutset for some node *n* contains more than one node in a path as in the example, then the labeling algorithm may wrongly estimate the delay of node *n*. We call such situation the *bsc chain effect*. However, there may exist other *k*-cutsets which can cause *n*'s delay to be the same as its label.

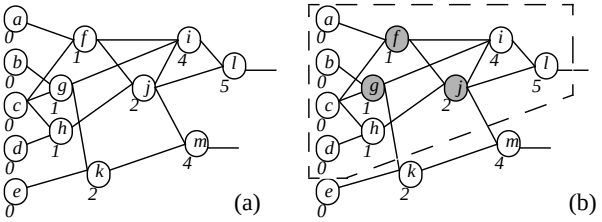


**Fig. 5** : An example to illustrate the bsc insertion algorithm.

**Lemma 2**: Suppose the labels of all the nodes in *n*'s input cone are the same as the optimal delays under the dependency constraint, *k*. If the *k*-cutset of node *n* suggested by the MFMC algorithm has no bsc chain effect, then *n*'s label $l(n)$ set by the labeling algorithm is optimal.

achieves the same delay results as the lower bounds; *i.e.*, optimal solutions.

This paper is organized as follows: Section 2 describes the delay model with the associated delay computation. Section 3 describes an algorithm computing a delay lower bound for each node under the dependency constraint. The approximation algorithm to guide the bsc insertion process is developed in Section 4. Finally, the experimental results and conclusions are given in Sections 5 and 6.

## 2 Delay Model and Delay Computation

In this section, we describe the delay model of bsc insertion. We said edge *e (v-> u)* is an (immediate) *fanin edge* of *u* and node *v* is an (immediate) *fanin node of u*. In addition, node *w* or edge *e* is a *transitive* fanin of node *u* if there exists a path from *w* or *e* to *u*. The set of nodes which are transitive fanins of node *v* is referred to as the *input cone* of node *v, Cone*(*v*). When a bsc is inserted on edge *e*, our delay model assumes that one *constant* delay penalty, *d*(*bsc*), is added to the delay of edge *e*. Consider again the circuit in Fig. 2(a). Assuming that edge delay *d*(*e*) and bsc delay penalty *d*(*bsc*) both are *1*, and we can obtain the delay of each node shown outside the circle. In Fig. 2(a), without inserting any bsc, the delay of node *k* is *3* because the longest path from PIs to *k* is *a->g->j->k* which contains 3 edge delays. In Fig. 2(c), after inserting bscs $b_1$, $b_2$, and $b_3$, the delay of node *k* becomes *4* because of adding bsc delay.

## 3 Delay Lower Bound Computation

The proposed algorithm consists of two phases: the labeling phase and the bsc insertion phase. We first give some definitions used in our discussion. For simplicity, we assume that a bsc has been inserted on the fanout stem of each PI and let the *dependency set* of node *n*, denoted *dep*(*n*), be the set of bscs on which node *n* depends and |*dep*(*n*)| be the *size* of *dep*(*n*). The bscs of *dep*(*n*) can be viewed as a "cut set" which separates node *n* from the PIs. If $|dep(n)| \le k$, the set of nodes on the fanout stems of which the bscs of *dep*(*n*) are inserted are called a *k*-cutset of node *n*. For example, in Fig. 2(c), the bsc set {$b_1$, $b_2$, $b_3$} is node *l*'s dependency set and the node set {*j, h, i*} is node *l*'s *3*-cutset.

The label of node *n*, *l*(*n*), is the (estimated) minimum delay of node *n* under dependency constraint *k*. Initially, all PIs' labels are set to *0* and then we compute the label of each node in the topological order from PIs to POs. Hence, before computing the label of node *n*, all the labels of nodes in node *n*'s input cone have been already done. Note that there are many possible k-cutsets for node *n.* To determine the minimum delay of node *n, i.e.*, the label, we need to find an "optimal" *k*-cutset for node *n*. In the following, we discuss an important property: the possible range of a node's label can be determined from the labels of its (immediate) fanin nodes.

**Definition 1**: The *best_label*, *bl*(*n*), of node *n* is defined as $bl(n) = \mathrm{MAX}\{l(n_i)+d(e_i) \mid \text{where } e_i\ (n_i \text{-> } n) \text{ is a fanin edge of } n\}$.

**Lemma 1**: The label *l*(*n*) of node *n* has the property:

$$bl(n) \le l(n) \le bl(n)+d(bsc). \quad \text{(EQ 1)}$$

Proof. omitted.

We now use an example to illustrate Lemma 1. Consider the circuit of Fig. 3, where the number outside a node is the corresponding node's label which we are about to compute. Again, let the delay of an edge, *d*(*e*), and the bsc delay penalty, *d*(*bsc*), be both 1. Here, we assume the dependency constraint is *4*. Initially, the labels of all the PIs are set to *0*. Our labeling algorithm will start from PIs to POs. For node *h*, it is easy to see that there is no need to insert bscs for node *h* so the label of *h* is equal to *1*. Similarly, for nodes *i, j, k,* and *l*, we have the labels *l*(*i*) = *1*, *l*(*j*) = *1*, *l*(*k*) = *2*, and *l*(*l*) = 3. For node *m* which has 7 PIs in its transitive fanins, we do need to insert bscs for *m*. Since *m*'s immediate fanin nodes are *l* and *j*, from Definition 1, we have $bl(m) = \mathrm{MAX}\{l(l)+d(e), l(j)+d(e)\} = \mathrm{MAX}\{3+1, 1+1\} = 4$. Then, by EQ 1, we can find out that the range of *m*'s label is $4 = bl(m) \le l(m) \le bl(m)+d(bsc) = 5$. Hence, there are only two choices for *m*'s label, *i.e.*, either *l*(*m*) = *4* or *l*(*m*) = *5*. In later discussion, we will show that there does not exist a solution which makes the delay of *m* to be *4* under the dependency constraint *4*, so *m*'s label must be *5*.

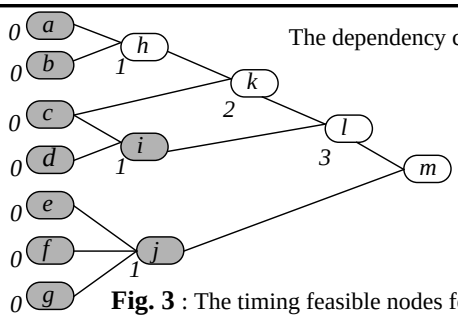


**Fig. 3** : The timing feasible nodes for *bl*(*m*) = *4*.

From Lemma 1, we can obtain a possible range of node *n*'s label, *l*(*n*), whose best value is *bl*(*n*) and worst value is *bl*(*n*)+*d*(*bsc*). Without losing generality, let us assume that the delay penalty *d*(*bsc*) and the delay *d*(*e*) of each edge *e* are integers. If the delay penalty *d*(*bsc*) is *1*, according to EQ 1, we can have only two choices for node *n's* label *l*(*n*): one is *bl*(*n*) and the other is *bl*(*n*)+*1*. Therefore, the problem of finding node *n*'s label has been changed to the yes/no problem of determining whether *bl*(*n*) is achievable. Note that *bl*(*n*)+*1* is always achievable by inserting bscs on the fanin edges of node *n*. Luckily, this yes/no problem has an approximate solution using the maximal-flow minimal-cut (MFMC) algorithm. Our algorithm can be easily extended when *d*(*bsc*) and *d*(*e*) are not integers.

Let *d*(*c->n*) be the longest path delay from *c* to *n* without involving any bsc delay penalty. If node *c* is selected into an optimal *k*-cutset of node *n*, we have *l*(*c*)+*d*(*c->n*)+*d*(*bsc*) ≤ *l*(*n*) in Fig. 4. Therefore, to achieve the best_label *bl*(*n*) of node *n*, if node *c* (other than PIs) is selected into the *k*-cutset, its label, *l*(*c*), must be less than or equal to the value of *bl(n)-d*(*c->n*)-*d*(*bsc*). In addition, because a PI can be considered as a bsc, we do not need to insert bsc at the fanout of a PI. We

# A Timing-Driven Pseudo-Exhaustive Testing of VLSI Circuits

S. C. Chang and J. C. Rau

Department of Computer Science and Information Engineering

National Chung-Cheng University

Chiayi, Taiwan, R. O. C.

**Abstract**

The object of this paper is to reduce the delay penalty of bsc insertion for pseudo-exhaustive testing. We first propose a tight delay lower bound algorithm which estimates the minimum circuit delay for each node after bsc insertion. By understanding how the lower bound algorithm lose optimality, we can propose a bsc insertion heuristic which tries to insert bscs so that the final delay is as close to the lower bound as possible. Our experiments show that the results of our heuristic are either optimal because they are the same as the delay lower bounds or they are very close to the optimal solutions.

## 1 Introduction

Exhaustive testing, one method of *Built-In Self-Test* (BIST) [1][2][3], becomes very attractive because that the method can guarantee the detection of all non-redundant combinational faults. Unfortunately, the test time of exhaustively testing a combinational circuit increases exponentially to the number of primary inputs (PIs) of the circuit, which makes this test method unpractical for circuits with large PIs. To pseudo-exhaustively test a circuit using $2^k$ patterns, some *bypass storage cells* (bscs) may be inserted so that the dependency of each node is less than or equal to the pre-determined value *k* called the *dependency constraint*. A *bsc* is transparent in the normal mode and acts as both pesudo-input and pseudo-output in the test mode. The bsc structure is presented in Fig. 1. During the normal mode, the signal, *test* is set to 0 and all bscs become transparent; that is, they do not affect the circuit's function. In each cycle of the test mode (*test = 1*), a test pattern is serially shifted to the flip-flops of bscs, and the output responses is directed to the output response analyzer (ORA) which is analyzed later.

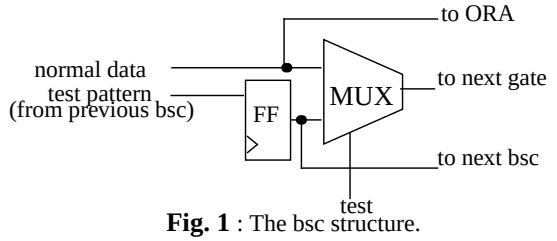


**Fig. 1** : The bsc structure.

However, adding bscs can increase the circuit size. Most previous work [5][11][15] [16] attempts to reduce the number of bscs under a given dependency constraint. Basically, based on the area consideration, the bsc insertion methods for pseudo-exhaustive testing can be divided into two categories. One uses the constrained partitioning strategy which partitions a circuit into several disjoint sub-circuits [5][15][16]. The other uses the un-constrained partitioning strategy which makes no such constraint [11]. In this paper, our strategy of bsc insertion is based on the first one.

In addition to the area overhead of inserted bscs, the inserted bscs may be placed on critical paths and therefore can worsen the circuit delay. Consider the circuit of Fig. 2(a) and assume the path *a->g->j->k->l* is critical. Fig. 2(b) shows a solution for pseudo-exhaustively testing the circuit. Despite the inserted bscs do not affect circuit's original function in the normal mode, the circuit delay is increased due to the placement of two bscs ($b_1$ and $b_2$) on the critical path. Instead, if we insert bscs as in Fig. 2(c), where only one bsc ($b_1$) is placed on the critical path, the critical path delay is less than the delay in Fig. 2(b). Note that the maximum dependency size of both circuits is 3.

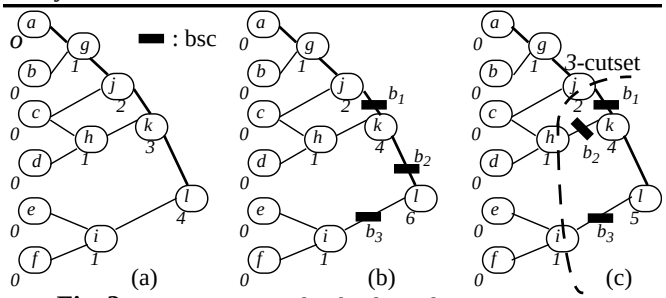


**Fig. 2** : Bsc insertion under the dependency constraint *3*.

The purpose of this paper is to minimize the delay penalty resulted from inserting bscs under a given dependency constraint. Our algorithm consists of two phases. In the first phase, we discuss an algorithm which finds a *lower bound* of the minimum delay of a node after inserting bscs. With the lower bound information and the understanding of how the optimality may lose, we then develop a heuristic to guide bsc insertion so that the delay result (after inserting bscs) is as close to the lower bound as possible. Our experimental results show that, for many benchmark circuits, our heuristic