TABLE II
ROUTING RESULTS OF THE SECOND SET OF BENCHMARK CIRCUITS WITH $F_C = W$ AND $F_S = 3$

Circuit	SEGA			TRACER-fpga		
	#Tracks/ Channel	Max Delay	CPU sec.	#Tracks/ Channel	Max Delay	CPU sec.
alu4	15	168	77	11	458	171
apex7	13	78	23	8	147	34
term1	10	72	9	8	113	14
z03	14	192	253	11	642	477
example2	17	122	54	10	145	54
too_large	12	121	40	10	291	73
k2	17	243	161	14	745	526
vda	14	170	64	11	628	147
9symml	10	58	13	7	88	24
alu2	11	116	31	9	354	54

TRACER-fpga was able to achieve very dense routing for two suites of benchmark circuits using a reasonable amount of CPU time. The long wiring delay makes TRACER-fpga unsuitable for high-performance design. However, the track efficiency makes it suitable for slow-speed applications such as hardware emulation where the clock rate ranges from 1 to 10 MHz.

The largest circuit in our benchmark suites has 586 CLB's. As the technology advances, the number of CLB's in a circuit will increase. TRACER-fpga will slow down as the routing graph grows. It is important to improve the programming and data structuring efficiency so that TRACER-fpga can keep up with the technology progress. Furthermore, to improve the wiring delay of TRACER-fpga, we have to integrate the delay model into the expansion router. That is, instead of searching for the shortest connection, it should look for the fastest connection.

ACKNOWLEDGMENT

The authors would like to thank Professor S. Brown of the University of Toronto, Canada, for providing them with the CGE/SEGA package along with the benchmarks. They would also like to thank the reviewers for their suggestions that helped improve the quality of this paper.

REFERENCES

[1] *The Programmable Gate Array Data Book.* Xilinx Co., 1992.
[2] S. D. Brown, J. Rose, and Z. Vranesic, "A detailed router for field-programmable gate arrays," *Int. Conf. Computer-Aided Design*, pp. 382–385, 1990.
[3] G. G. Lemieux and S. D. Brown, "A detailed routing algorithm for allocating wire segments in field-programmable gate arrays," in *4th ACM/SIGDA Physical Design Wkshp*, Apr. 1993.
[4] Y.-A. Chen, Y.-L. Lin, and Y.-C. Hsu, "A new global router for ASIC design based on simulated evolution," in *Int. Symp. VLSI Technol., Syst., Applic.*, Taipei, Taiwan, May 1989, pp. 261–265.
[5] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 346–365, Sept. 1961.
[6] E. S. Kuh and M. Marek-Sadowska, "Global routing," in *Layout Design and Verification*, T. Ohtsuki, Ed. Amsterdam, Netherlands: North-Holland, 1985.

A Coordinated Circuit Partitioning and Test Generation Method for Pseudo-Exhaustive Testing of VLSI Circuits

Wen-Ben Jone and Christos A. Papachristou

***Abstract*— In this paper, we present a circuit partitioning and test pattern generation technique for pseudo-exhaustive built-in self-testing of VLSI circuits. The *circuit partitioning* process divides a given circuit into a set of subcircuits which can be exhaustively tested, while the *test pattern generation* process generates reduced exhaustive test patterns for each subcircuit using a linear feedback shift register (LFSR). In conventional approaches, these two problems are considered separately. However, in this paper, both problems are considered and solved in the same phase.**

A graph theoretic model of VLSI circuits is proposed. Based on this model, a circuit partitioning algorithm using the concept of *minimum vertex cut* is devised to partition the circuit into a set of exhaustively testable subcircuits with restricted hardware overhead. Each time a subcircuit is generated by the partitioning algorithm, the test pattern generation problem is considered. A new algorithm, based on the *subcircuit modification* technique, is proposed with the objective of generating reduced exhaustive test patterns of limited length (e.g., $\leq 2^{20}$) using LFSR's, for each of the subcircuits. This task is *embedded* in the circuit partitioning process itself, leading to an efficient and well-coordinated solution. Experiments using ISCAS benchmark circuit simulation have been conducted. The results demonstrate that the proposed method is very good.

***Index Terms*—Built-in self-testing; circuit partitioning; pseudo-exhaustive testing; graph theory; test pattern generation.**

I. INTRODUCTION

As the circuit technology is moving to high densities of integration, built-in self-testing (BIST) is a very important technique [1] in VLSI design for test (DFT). The main idea in this technique is to have the chip test itself. Many BIST techniques such as BILBO [2], syndrome testing [3], [4], autonomous testing [5], and self verification [6], have been proposed. In general, the above techniques can be divided into two different categories: random self-testing [7] and exhaustive self-testing [1].

In *exhaustive* self-testing, all combinations of the test patterns must be generated. This can be achieved using the linear feedback shift register (LFSR) by cycling through all 2^n - 1 LFSR states. Exhaustive self-testing provides a thorough test, in that it eliminates the need for a fault model or fault simulation process and, at the same time, it can achieve very high fault coverage [5], [8]. However, this technique may require prohibitively long test time for a circuit with many inputs. To shorten testing time, McCluskey [9] proposed a verification testing technique based on *reduced exhaustive test patterns* (RETP) by concurrently overlapping the exhaustive test patterns of different output cones. However, this technique will not help if most outputs depend on too many inputs. A more general solution is *pseudo-exhaustive* testing by partitioning a large circuit into subcircuits, such that each subcircuit has a sufficiently small number of inputs to make

Manuscript received September 22, 1992; revised January 19, 1994 and September 15, 1994. This work was supported in part by the Semiconductor Research Corp. (SRC), Contract 87-DJ-127, Sandia National Laboratories, Contract Sandia-54-9306, and the National Science Council, Taiwan, R.O.C., Contract 82-0404-E-194-041. This paper was recommended by Associate Editor K.-T. Chen.

W.-B. Jone is with the Department of Computer Science and Information Engineering, National Chung-Cheng University, Chiayi 62107, Taiwan, ROC.
C. A. Papachristou is with the Department of Computer Engineering & Science, Case Western Reserve University, Cleveland, OH 44106 USA.
IEEE Log Number 9406934.

RETP-based testing practical [5], [8]. A hundred percent of single and a high percentage of multiple stuck-at faults can be detected with a much lower test length (compared with exhaustive testing) if a good circuit segmentation can be found [5], [8].

Although the pseudo-exhaustive BIST method is very promising (extremely high fault coverage, no fault model needed), it is very difficult to find a good partitioning algorithm. If the circuit partitioning algorithm is not good enough, the number of interconnections among the subcircuits may be very large. Since each interconnection will induce extra hardware to physically separate the subcircuits in test mode, the hardware overhead may not be tolerable. In addition, it is very difficult to generate RETP using a built-in device such as a LFSR.

In this work a methodology for BIST based on partitioning and pseudo-exhaustive testing is proposed. The approach consists of (1) an algorithm for VLSI combinational circuit partitioning, and (2) a technique for RETP generation using LFSRs. The goal of our methodology is to lead to an efficient and well-coordinated solution to the VLSI BIST problem. The objectives of *circuit partitioning* are: (1) the number of RETP generated for each subcircuit is no larger than 2^n (n is 20 in general); (2) the total number of interconnections among the subcircuits should be as small as possible. The objective of *RETP generation* is to construct a suitable LFSR for generating RETP of limited test length, say 2^{20}, for each subcircuit. This task is included as a part of the circuit partitioning process. That is, each time a subcircuit is generated during the partitioning process, the RETP generator of the subcircuit will be derived using a subcircuit reduction technique. The idea of attacking circuit partitioning and RETP generation at the same time facilitates an integrated solution to both problems. Experiments by computer simulation have been conducted using ISCAS combinational benchmark circuits. The results demonstrate that the proposed method is very good, i.e., meeting the above objectives.

This paper contains the following major sections. In Section II, fundamental concepts are established. Section III is dedicated to the discussion of circuit partitioning; while RETP generation using LFSR is discussed in Section IV. Experimental results are presented in Section V and Section VI gives the concluding remarks.

II. Fundamental Concepts

Partitioning a circuit for the purpose of testing is said to be one of the most important problems in VLSI testing. Some work has already been done regarding combinational circuit partitioning for pseudo-exhaustive testing [10]–[20]. In this work, we focus on partitioning for pseudo-exhaustive testing of combinational logic circuits. The approach is based on a graph-theoretic model using *minimum vertex cut* to achieve a global suboptimal (local optimal) solution [20]. As mentioned, the basic aims of our approach regarding partitioning for pseudo-exhaustive testing of a circuit C are:

1) The partition technique should keep the hardware overhead as small as possible; i.e., the number of interconnections among the subcircuits should be manageable.
2) The RETP length of each subcircuit should be controllable (say no larger than 2^n); i.e., any signal (internal) line in each subcircuit (or segment) should be functionally dependent on no more than n primary inputs of the subcircuit.

We use the following graph-theoretic model to represent the circuits. A combinational circuit C can be represented by a directed graph G_C containing two types of nodes: (1) square nodes, corresponding to inputs or outputs of C; (2) circle nodes, corresponding to gates or cells of C. A directed edge of G_C represents the signal flow between the corresponding components of C. Obviously, the boundaries of G_C consist of square nodes only, and G_C is acyclic. During the partitioning process, the interconnections of adjacent subcircuits in G_C are explicitly depicted by *black* square nodes as shown in Fig. 1. Thus, the black square nodes are distinguished from the blank square nodes in that the latter comprise the input/output boundary of G_C, and the former are inserted to separate the subcircuits in testing mode. In this approach, the black square nodes are implemented by control register cells [21], and hence, constitute circuit overhead, in accord with the BIST philosophy. Specifically, in testing mode, the control cells are linked to form LFSRs which perform test pattern generation and test result evaluation using signature analysis [1]. In normal mode, all test hardware is bypassed [2]. The circuit partitioning and RETP generation method can also be applied to existing test architectures.

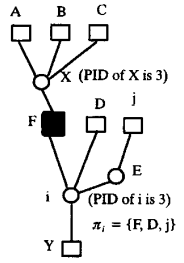


Fig. 1. The graph definitions.

The basic idea of the partitioning and RETP generation scheme is first to derive one segment S from the beginning boundary of G_C (next section), such that each gate in S depends on no more than n primary inputs. Next, an attempt is made to derive a LFSR L for generating the RETP for S (Section IV), with test pattern length limited to 2^n, $n \leq 20$, n is the maximum PID (defined in Section III) in S. If no such L can be found after several trials, S will be modified by reducing the size from its end boundary, so that L is able to generate the RETP for S. This is the reason why we do not partition the entire circuit into several segments at one time. Once the LFSR of S has been derived, S will be removed from G_C. The above process is repeated until G_C has been totally partitioned and all LFSRs of the associated segments have been derived.

III. Circuit Partitioning

Before describing the circuit partitioning method, the following concepts are needed. By definition, a circle node i in G_C is said to be *affected* by square node j if there exists at least one directed path from j to i and there exists no other square node in this path (Fig. 1). The *primary input degree* (PID) of a circle node i in G_C is the number of square nodes which affect node i (Fig. 1). Clearly, these square nodes may be located either internally or at the beginning/ending boundaries of G_C. Based on the graph model and the definitions given earlier, the partitioning problem can now be formulated in terms of the following requirements:

Given a directed acyclic graph G_C, partition the graph into disjoint segments by adding (black) square nodes at the segment interconnections, such that (1) Each segment contains no circle nodes with PID larger than n (n is 20 in our work); (2) The total number of square nodes added into G_C, after it has been completely partitioned, should be as small as possible.

It has been shown that the circuit partitioning for pseudo-exhaustive testing problem is NP-complete [21], so, a good heuristic algorithm is needed. Intuitively, a good way to partition a directed acyclic graph under the above two requirements would be using the minimum cut set. However, the basic idea of our partitioning technique is to use the

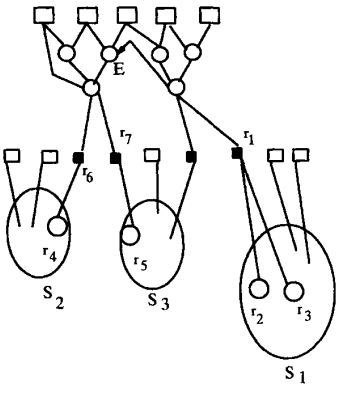


Fig. 2. Fanout absorption and backward routing line.

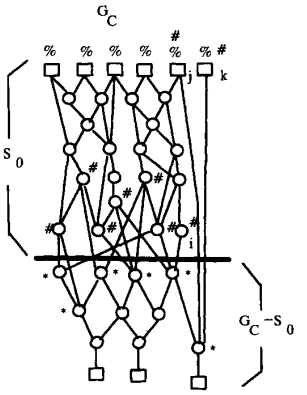


Fig. 3. Derivations of initial segment S_0.

minimum vertex cut of circuit G_C instead of the minimum cut set. Note that a *vertex cut* of directed acyclic graph G_C is a set of nodes whose removal will cut all paths from the beginning boundary to the ending boundary of G_C. This technique provides a local optimal solution, because the size of the minimum vertex cut is always less than that of the minimum cut set. Although this technique does not always guarantee global optimality, the use of the minimum vertex cut is a very good heuristic, according to our experiments (Section V). The benefits are that the number of square nodes (interconnections) can be significantly reduced and, further, the derivation of the LFSRs for generating RETP is facilitated (Section IV).

The proposed circuit partitioning (steps 1, 2, 3) and RETP generation (step 4) method is composed of the following five well-coordinated major steps.

Circuit Partitioning and RETP Generation Process:

1) Derivation of initial segment S_0 by a greedy method.
2) Minimum vertex cut derivation in S_0, and creation of another segment S_1.
3) Choice between S_0 and S_1 as the final segment version, called S.
4) Segment reduction of S. The RETP generator (LFSR) is derived for S and black square nodes are inserted at its ending boundary. Then S is removed from G_C.
5) Repeat step 1 through 4 on G_C until the entire G_C is partitioned.

The discussion of step 4 is in Section IV. The scenario of all other steps is described below.

III.1. Initial Segment Derivation

Given a directed acyclic graph G_C and the allowed maximum PID, n; the initial segment S_0 can be derived by traveling from (the square nodes of) the beginning boundary of G_C as deeply as possible until either no nodes with PID $\leq n$ can be traversed, or the square nodes at the ending boundary of G_C are reached. The best traveling strategy in this case is by means of a breadth first search. However, the *backward routing* lines will cause some trouble because each time a node is traversed, for example node E in Fig. 2, one or more of its immediate ancestors may not have been accessed yet. This problem is solved by levelizing G_C before starting the traveling process. This guarantees that once a node is accessed all its ancestors have been traversed, if the traveling process is performed level by level.

By definition, the *primary input set* Π_i of node i in G_C is a set of square nodes which affect i. If i is a square node, then $\Pi_i = i$. Clearly the PID of i is equal to the cardinality, $|\Pi_i|$, of Π_i. An example of primary input sets is in Fig. 1. The Π_i sets can be derived recursively in terms of the Π_i^k sets, $k = 1, 2, 3, \cdots$, i.e., the primary input sets of the immediate predecessors of node i. The following relations may be used for this purpose:

$\Pi_i = \cup_k \Pi_i^k$, k is an immediate predecessor of i

$$\max\{|\Pi_i^k|\} \leq |\Pi_i| = \left|\bigcup_k \Pi_i^k\right| \leq \sum_k |\Pi_i^k|.$$

By definition, node i in G_C is called *n-divergent* if $|\Pi_i| > n$. The set of all n-divergent nodes in G_C comprises the *n-divergent set.*

To obtain an initial segment S_0, it is natural to use a node traveling algorithm whose basis is a node queue ordered by the level of each node in an ascending sequence. Nodes are traveled level by level from the lowest level and those which are n-divergent are placed in the n-divergent set. It should be noticed that if node i is n-divergent then all the i-successors are obviously n-divergent. Thus, to speed up the process, the immediate successors of i are assigned *travel tags* only if i is not n-divergent.

The traveling process will stop when no more nodes can be traversed or all nodes left are n-divergent. As an example, the graph of Fig. 3 is traveled by this algorithm, $n = 3$. The complexity of the traveling algorithm is $O(fnN)$, where N is the number of nodes in G_C, n is the maximum primary input degree allowed in the partitioning process, and f is the maximal fan-ins in G_C [21]. Generally, the value of n and f is very small ($n \leq 20$, $f \leq 10$ in VLSI circuits), thus the computation time is quite low.

The traveling algorithm produces the n-divergent set of G_C. The initial segment can be obtained as follows: let θ denote the set of nodes which are the immediate predecessors of the nodes in the n-divergent set; and each node i in θ has the following two properties: (1) i is not divergent, (2) i is assigned a tag. An example of θ (denoted by "#") and the 3-divergent set (denoted by "*") is shown in Fig. 3. If all the edges between θ and the n-divergent set are removed, G_C will be partitioned into two parts. The initial segment, S_0, is the directed acyclic subcircuit of G_C whose ending boundary is θ as shown in Fig. 3.

Theorem 3.1: The set θ and the initial segment, S_0, derived by the traveling algorithm satisfy: (1) θ is a vertex cut of G_C; (2) S_0 is the maximal initial segment whose nodes are not n-divergent (Proof: [21]).

Based on the above theorem, the initial segment S_0 can be derived at the end of the traveling process. The nodes in θ will form the ending boundary, and the square nodes in the beginning boundary of G_C will form the beginning boundary of S_0 (denoted by "%"). Note that the beginning and ending boundaries need not be disjoint,

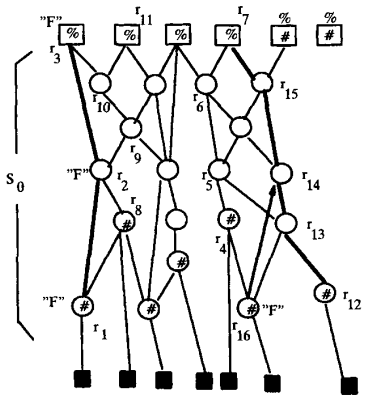


Fig. 4. Vertex path creation by a labeling technique.

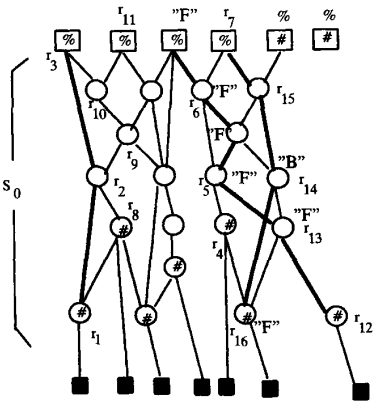


Fig. 5. Vertex path backtracking.

for example nodes j and k as shown in Fig. 3. In order to isolate S_0 from $G_C - S_0$, for each node i in θ, one square node should be inserted at the rear of i as shown in Fig. 4. Obviously, the number of required square nodes is $|\theta|$. The next step is to find a *better* segment based on S_0.

III.2. Minimum Vertex Cut Derivation

Since S_0 is the maximal initial segment that can be derived, the optimal segment should be embedded in S_0. This optimality can not be derived in polynomial time in a global sense. However, it is possible to derive a local optimal segment by generating the minimum vertex cut within S_0. Derivation of the minimum vertex cut for an undirected graph is known as the connectivity problem [22] and its complexity is still unknown. However, for the case of directed acyclic graphs, it can be solved with complexity $O(\mu N)$ where N is the number of nodes in S_0, and μ is the number of nodes in the ending boundary of S_0.

The motivation for using the minimum vertex cut, instead of the minimum cut set, is to greedily reduce the number of square nodes needed; because the size of the former is always smaller than that of the latter. The basic reason for reducing the overhead is the *fan-out absorption* that occurs on the square nodes of the vertex cut. Specifically, using the minimum cut set, each square node inserted fans out to only one interconnection between two segments. However, in the minimum vertex cut, a square node may fan out to several edges. For example, in Fig. 2, r_1 fans out directly to nodes r_2 and r_3, if they (r_3, r_2) belong to the same segment (say S_1). But, r_4 and r_5 belong to different segments (S_2, S_3), so it is necessary to place square nodes r_6 and r_7 in the respective fan out edges such that S_2, S_3 will be disjoint. This also makes *parallel* testing of segments S_2, S_3 possible [23]–[25]. Note that the minimum vertex cut may not always give a better solution than the minimum cut set, which may happen when the fan out edges of the vertex cut are directed to *many* different segments, after entirely partitioning G_C.

In order to keep the size of each segment as large as possible, the derivation of minimum vertex cut will be performed from the ending boundary (that is θ) to the beginning boundary of S_0 by *reversing* the direction of each edge. Based on this idea, if there are more than one minimum vertex cuts in S_0, the *lowest* one which induces the largest segment is always selected. By definition, a *vertex path* is a path of S_0 from the ending to the beginning boundary. A *vertex stream* is a set of disjoint vertex paths. In Fig. 4, vertex paths $r_1 \rightarrow r_2 \rightarrow r_3$ and $r_{12} \rightarrow r_{13} \rightarrow r_{14} \rightarrow r_{15} \rightarrow r_7$ (denoted by dark lines) comprise a vertex stream, while $r_1 \rightarrow r_2 \rightarrow r_3$ and $r_8 \rightarrow r_2 \rightarrow r_9 \rightarrow r_{10} \rightarrow r_{11}$ do not. It is required to find a maximal vertex stream in S_0. Based on this concept, the derivation of minimum vertex cut consists of two major parts:

1) Derive and label a maximal vertex stream,
2) Derive the minimum vertex cut from the labels of the maximal vertex stream.

III.2.1. Maximal Vertex Stream: Basically, a maximal vertex stream can be derived by the depth first search path expansion strategy with the aid of a labeling technique. For each node i in S_0 two labels, "B" and "F", are used to indicate the backward and forward path expansion from i, respectively. Before expanding forward (backward) from node i, the F (B) label of i is checked. If there is no F (B) label associated with i, then expand forward (backward) from i, and label i with F (B) to avoid expanding forward (backward) from i any longer. The process is quite straightforward by using greedy forward expansion if no existing paths are reached. A vertex path is found successfully when the beginning boundary of S_0 is reached. In Fig. 4, vertex path $r_1 \rightarrow r_2 \rightarrow r_3$ is created by labeling a series of "F" flags.

Once a new path is found, it is recorded and all labels are removed before going on to find the next paths of the maximum vertex stream. If an existing path is hit during the expansion process, some backtracking is needed because an existing path can not be shared by another path. For example, in Fig. 4 vertex path $r_{12} \rightarrow r_{13} \rightarrow r_{14} \rightarrow r_{15} \rightarrow r_7$ is hit at node r_{14} if a new path is developed from node r_{16} to r_{14}. The basic idea of backtracking is to let the developing path take over part of the existing path (the part from node r_{14} to the beginning boundary of S_0 in this example), and let the existing path develop the missing part that has been taken over. After backtracking to r_{13} as shown in Fig. 5, it is possible to expand forward from r_{13}; or continue backtracking if the forward tracking from r_{13} does not find a new path successfully. Each edge ($\overline{r_{13}, r_{14}}$ in this example) which is in an existing path and has been backtracked should be removed from the existing path, if the backtracking (and then forward tracking) successfully reaches the beginning boundary of S_0.

Fig. 5 demonstrates the new paths created by this process. The details of the path expansion procedure and the vertex stream algorithm are in Appendix I. The path expansion algorithm can be called repeatedly to find all the disjoint vertex paths that form the maximal vertex stream.

Theorem 3.2: The vertex stream algorithm finds all disjoint vertex paths in S_0 with complexity $O(\mu N)$ where N is the number of nodes in S_0, and μ is the number of nodes at the ending boundary of S_0 (Proof: Appendix II).

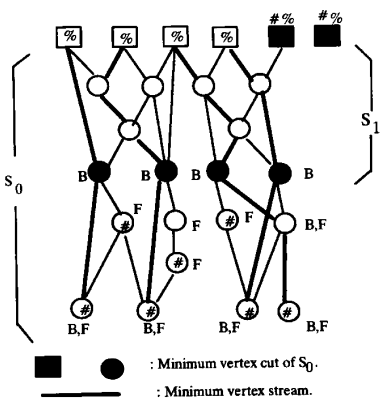


Fig. 6. Minimum vertex cut derivation.

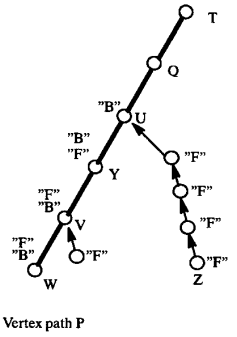


Fig. 7. Vertex path hitting.

As shown in Figs. 6 and 7, the maximal vertex stream process ends with two kinds of labeled nodes, hit node and backtracking node, for each vertex path. Node n is called a *hit* node if n is first visited by a forward traversal; while n is called a *backtracking* node if n is first visited by a backward traversal. According to our labeling technique, hit node is always labeled "B" first and backtracking node is labeled "F" first. Thus, node U in Fig. 7 is definitely a hit node, but node V is a hit node (if V is traveled before U) or a backtracking node (if V is traveled after U). When vertex path P is hit on nodes U or V in the last repetition of the maximal vertex stream process, a backtracking along P will be performed until another hit node (labeled "B") or the starting node of P is met. Note that node V will also be labeled "F" before the end of the maximal vertex stream process. Thus, there is one and only one node along each vertex path P labeled "B" but not "F".

III.2.2. Minimal Vertex Cut: At the end of the vertex stream algorithm, labels that have been assigned and not yet erased will be left in the graph (since no new paths can be found) as shown in Fig. 6. These labels are needed for deriving the minimum vertex cut. The basic idea is to search for every node whose backtracking label (B) is set but forward label (F) is not. These nodes will form the minimum vertex cut.

Theorem 3.4: Node V is in the minimum vertex cut of S_0 if: (1) V is in one of the vertex paths of the maximal vertex stream of S_0; and (2) V is labeled "B" but not "F" at the end of the vertex stream algorithm (Proof: Appendix III).

Based on the above theorem, the minimum vertex cut can be derived. It is interesting to see that each vertex path of the maximal vertex stream in S_0 contributes one and only one node to the minimum vertex cut of S_0. The entire minimum vertex cut process is dominated by the vertex stream algorithm, which has complexity $O(\mu N)$ where N is the number of nodes in S_0, and μ is the number of nodes at the ending boundary of S_0.

After the minimum vertex cut has been derived, a new segment is automatically generated as shown in Fig. 6, and is called S_1. The decision as to whether S_0 or S_1 is to be the final segment is discussed next.

III.3. Choice of S_0 and S_1

Once S_0 and S_1 are derived, the next step is to determine which one is to be chosen as the final version. The choice is based on the *hardware overhead*, i.e., the total numbers of square nodes induced by S_0 and S_1 respectively to partition the entire G_C. Let $N_1(S)$ denote the number of square nodes required to isolate S from $G_C - S$, and let $N_{opt}(G_C - S)$ be the minimum number of square nodes required to completely partition $G_C - S$. We will choose S_0 if

$$N_1(S_0) + N_{opt}(G_C - S_0) \leq N_1(S_1) + N_{opt}(G_C - S_1)$$

otherwise, S_1 will be chosen. Unfortunately, $N_{opt}(G_C - S)$ is unsolvable in polynomial time (actually, it is just the same problem with smaller size). Let $N_{grd}(G_C - S)$ denote the number of square nodes required to partition $G_C - S$ using the greedy traveling algorithm of Section III.1 repeatedly. Instead of using $N_{opt}(G_C - S)$ above, $N_{grd}(G_C - S)$ is adopted to be the worst case estimation of the performance for S_0 and S_1. Thus, S_0 will be chosen if

$$N_1(S_0) + N_{grd}(G_C - S_0) \leq N_1(S_1) + N_{grd}(G_C - S_1),$$

otherwise, S_1 will be selected. Note that the minimum vertex cut of S_0 is employed as the local optimum, while $N_1(S_1) + N_{grd}(G_C - S_1)$ and $N_1(S_0) + N_{grd}(G_C - S_0)$ are used as the global suboptimal evaluation for S_1 and S_0. The derivation of N_1 and N_{grd} is discussed next.

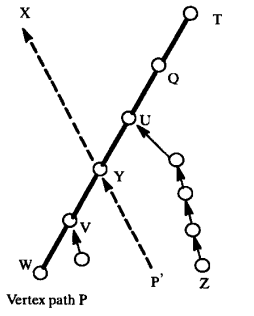
III.3.1. Derivation of $N_1(S_0)$, $N_{grd}(G_C - S_0)$: The derivation of $N_1(S_0)$ is performed at the ending boundary of S_0; and $N_{grd}(G_C - S_0)$ is derived by repeated application of $N_1(S_0)$ on $G_C - S_0$. To evaluate $N_1(S_0)$, each node i at the ending boundary of S_0 is considered. Let EB and BB denote the ending and beginning boundaries of S_0 respectively. There are two cases:

Case 1: i is not in EB $\cap$ BB. Here, no square nodes are attached to i as shown in Fig. 3, and this requires one extra square node i' to isolate i from the nodes on $G_C - S_0$ as shown in Fig. 9.

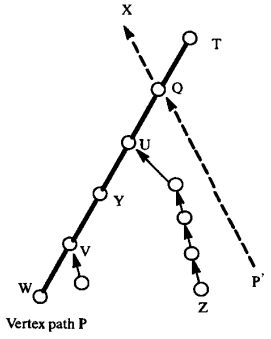
Case 2: j is in EB $\cap$ BB. Obviously, j is a square node as shown in Fig. 3. Let $\delta_1(j)$, $\delta_2(j)$ be the number of segments connected to j before and after S_0 is formed. If $\delta_2(j) = \delta_1(j)$ then no square nodes are added based on the self-testing strategy, because j does not feed more segments. If $\delta_2(j) > \delta_1(j)$ then one more control register should be added, because a new square node is needed to isolate the newly created segment from the others.

In Fig. 3, before S_0 is formed, node j feeds only one segment, G_C, hence $\delta_1(j) = 1$. After S_0 is formed, j feeds segment S_0 and $G_C - S_0$, so $\delta_2(j) = 2$. This requires one more square node j' to be attached to j as shown in Fig. 9. Again, before S_0 is formed, node k feeds one segment, G_C, only. After S_0 is formed, k feeds $G_C - S_0$ only; this makes $\delta_1(k) = \delta_2(k)$, and hence no square nodes are required as shown in Figs. 3 and 9 (actually, k does not feed S_0).

Based on the above two rules, the square nodes required to separate S_0 and $G_C - S_0$ are derived. The derivation of $N_{grd}(G_C - S_0)$ is done by repeated applications of the traveling algorithm and $N_1(S_0)$ derivation in $G_C - S_0$. Before $N_{grd}(G_C - S_0)$ is derived, S_0 is temporarily removed from G_C but the square nodes added between



(a)



(b)

Fig. 8. (a) Vertex cut failure (Case 2). (b) Vertex cut failure (Case 3).

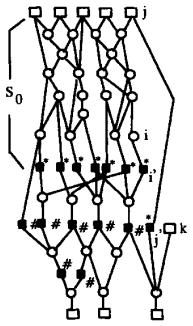


Fig. 9. Derivation of $N_1(S_0)$ and $N_{grd}(G_C - S_0)$.

S_0 and $G_C - S_0$ are not. The processes of initial segment derivation (Section III.1), $N_1(S_0)$ derivation, and segment S_0 removal are applied repeatedly to $G_C - S_0$ until it is completely partitioned as shown in Fig. 9. Nodes contributed to $N_1(S_0)$ and $N_{grd}(G_C - S_0)$ derivations are marked by "*" and "#" respectively. The value of $N_{grd}(G_C - S_0)$ is derived by the summation of $N_1(S_0)$ values obtained in each repetition.

III.3.2. Derivation of $N_1(S_1)$, $N_{grd}(G_C - S_1)$: The derivation of $N_1(S_1)$ is performed on the nodes of the minimum vertex cut, and that of $N_{grd}(G_C - S_1)$ by repeated applications of the traveling algorithm and $N_1(S_0)$ derivation on $G_C - S_1$. To evaluate $N_1(S_1)$, each node i on the minimum vertex cut is considered. Let VB denote the ending boundary of S_1. There are two cases:

Case 1: i is not in VB ∩ BB. Obviously, the addition of one square node i' at the rear of i so that i can be isolated from the nodes of $G_C - S_1$ is needed. This is shown in Fig. 10.

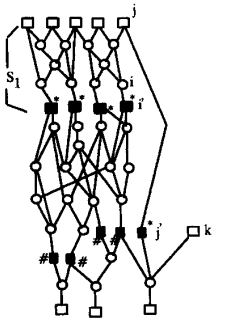


Fig. 10. Derivation of $N_1(S_1)$ and $N_{grd}(G_C - S_1)$.

Case 2: j is in VB ∩ BB. Let $\phi_1(j)$, $\phi_2(j)$ be the number of segments connected to j before and after S_1 is formed. If $\phi_2(j) > \phi_1(j)$, then one more square node will be added; otherwise no square nodes are required. The reason is similar to the one for $N_1(S_0)$, case 2. In Fig. 10, before S_1 is formed, node j fans its outputs to one segment, i.e., G_C; thus, $\phi_1(j) = 1$. After S_1 is formed, j fans its outputs to segments S_1 and $G_C - S_1$, respectively. Thus, $\phi_2(j) = 2$, and one more square node j' is added as shown. The same discussion (as k in Fig. 9) holds for node k.

Again, the derivation of $N_1(S_1)$ and $N_{grd}(G_C - S_1)$ is marked by "*" and "#" respectively. Similar to $N_{grd}(G_C - S_0)$, the derivation of $N_{grd}(G_C - S_1)$ is by repeated applications of the traveling algorithm and $N_1(S_0)$ derivation in $G_C - S_1$ as shown in Fig. 10.

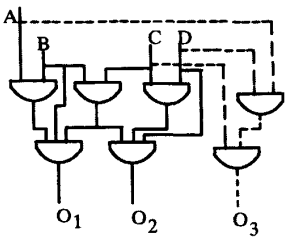
At the end of the segment choice phase, the final version of initial segment S is derived. For the example in Fig. 3, $N_1(S_0) = 8$ and $N_{grd}(G_C - S_0) = 7$ are shown in Fig. 9; while $N_1(S_1) = 5$ and $N_{grd}(G_C - S_1) = 4$ are shown in Fig. 10. Thus S_1 is chosen as the final version of S. At this time, the RETP generation for S is performed (as discussed in the next section), and then S is removed from G_C (the square nodes added between S and $G_C - S$ should not be removed). The whole process is repeated until G_C is completely partitioned. At the end of the partitioning process, the graph will be partitioned into three segments as shown in Fig. 10, in case that no segment reduction is required during the RETP generation phase of each segment. Totally, nine square nodes are added to physically isolate the three segments. We show next that the number of square nodes induced by our circuit partitioning algorithm is always less than or equal to the one obtained by the greedy method.

Theorem 3.4: The number of square nodes induced by the circuit partitioning algorithm is always less than or equal to the one obtained by the greedy method (Proof: Appendix IV).

IV. Reduced Exhaustive Test Pattern Generation

After a segment S has been generated, all registers which provide inputs to S are linked together, in testing mode, to form a LFSR which is used as the RETP generator of S [26]–[28]. The objective is to derive a LFSR which can generate RETP efficiently with limited length (say 2^{20}). The set of length 2^r RETP contains $2^r - 1$ nonzero test patterns generated by a r-stage LFSR, and the all-zero test pattern (generally) given by scan chains. To simplify the discussion, we say that a LFSR generates all RETP though the all-zero test pattern is, in fact, excluded. The output results of S will be compressed into a set of digital signatures, and separately stored in the other segments' RETP generators. Note that the RETP generation process is incorporated in the circuit partitioning process discussed in the previous section.

The rationale of the RETP generation process is as follows. Suppose each output of S depends on no more than n inputs. Then, it is desirable to derive a suitable LFSR, with sequence length 2^n, to



Reduced Exhaustive Test Patterns	Exhaustive Test Patterns
ABCD	ABCD
0000	0000
0010	0001
0100	0010
0110	0011
1001	0100
1011	⋮
1101	⋮
1111	1111
Length = 8	Length = 16

Fig. 11. Reduced exhaustive test pattern generation.

jointly exercise each output exhaustively. Unfortunately, such a LFSR may not always exist for an arbitrary circuit. Consider the circuit in Fig. 11 without including output O_3 and the dotted lines. Each circuit output depends on three inputs (i.e., PID=3), although there are totally four inputs in the circuit. Thus, it is possible to generate a set of RETP (by a LFSR) to exhaustively exercise each output with sequence length 2^3, not 2^4, as shown in Fig. 11. Now, suppose output O_3 is inserted into Fig. 11. Although every output still depends on three inputs, no LFSR can generate exhaustive test patterns for O_1, O_2, O_3, with sequence length 2^3. If the RETP length of each segment is confined to no larger than 2^3, segment modification can not be avoided.

In this section, the RETP generation problem is considered based on the work of [28]. The objective is to develop an efficient method to solve the RETP generation problem even if no LFSR (generated from a primitive polynomial) is able to provide RETP to S. The approach is to modify segment S by reducing the size of S from its ending boundary, provided the penalty of this reduction is tolerable.

The connection relation between a primary output i and its primary input positions in segment S, i.e., those inputs that affect i, can be represented by

$$T_i = \{K_{i,1}, K_{i,2}, \cdots K_{i,q}\},$$

where $K_{i,j}$ is the position number (counted from left to right) of the j-th primary input affecting primary output i, and q is the total number of primary inputs affecting i. For the example in Fig. 12, it is obvious to have $K_{i,1}$ = 1, $K_{i,2}$ = 2, $K_{i,3}$ = 3, $K_{i,4}$ = 5. By definition, a primitive polynomial $P(X)$ of degree n over $GF(2)$ is an *irreducible* polynomial (can not be factored over $GF(2)$), provided that $N = 2^n - 1$ is the smallest positive value of k such that $X^k - 1$ is divisible by $P(X)$. The next problem is to derive a primitive polynomial $P(X)$ which can *exhaustively exercise* the T_i's for all primary outputs [28]. After the primitive polynomial $P(X)$ has been derived, the LFSR can be generated by *feeding back* the content of each register j if X^j appears in $P(X)$. The RETP length will be 2^n if the degree of $P(X)$ is n.

According to [28], $P(X)$ exhaustively exercises the set $T = \{K_1, K_2, \cdots, K_r\}$, if and only if the r residue classes $[X^{K_i} \bmod P(X)]$ are linearly independent over $GF(2)$. The result will be employed in the following analysis. Let $T[S] = \{T_i | i$ is a primary output of $S\}$. The problem is to derive a primitive polynomial $P(X)$ of degree n (n is the maximum PID in S) which enables the corresponding LFSR to exhaustively exercise all of the T_i's in $T[S]$. There is no systematic way to derive $P(X)$, and the conventional approach in [28] is by trial and error, i.e., by randomly generating (or selecting) several polynomials, one by one, until finding one that can exhaustively exercise $T[S]$. The difficulty is that, for some segments, the derivation of $P(X)$ with a specified degree n (say n = 20) which can exhaustively exercise $T[S]$ is very time-consuming and, even worse, there may be no such a polynomial with degree n.

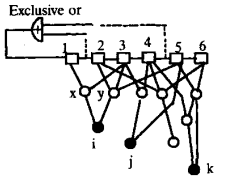


Fig. 12. Segment reduction.

The objective is then to find a good method such that not only the derivation of too many $P(X)$'s can be avoided, but also an appropriate $P(X)$ can always be found by segment modification (if needed). The problem can now be formulated as below.

Segment Reduction Problem:: Find a nth degree primitive polynomial $P(X)$ such that $P(X)$ can exhaustively exercise $T[S]$. If $P(X)$ can not be found, reduce segment S from the ending boundary until $T[S]$ can be exhaustively exercised by $P(X)$, while keeping the increase of primary outputs as small as possible. This is an NP-complete problem (a formal proof is not given here). Instead of the optimal solution, a good heuristic algorithm will be provided.

Given a connection relation T_i, after performing each division $X^{K_{i,j}}/P(X)$, some of the residues may be independent of any other residues, while some may be dependent on one another. To ease the discussion, elements $k_1, k_2, \cdots, k_q$ are said to be linearly dependent with respect to $P(X)$, if $MOD(X^{k_1}, P(X))$, $MOD(X^{k_2}, P(X))$, $\cdots$, $MOD(X^{k_q}, P(X))$ are linearly dependent. If there exist no dependent elements, with respect to $P(X)$, in every T_i of $T[S]$, then $P(X)$ is the desired primitive polynomial. Otherwise, it is possible to achieve independent elements by properly reducing the segment from the output end.

By definition, the cone of i contains all nodes and edges located on paths from the primary inputs to i as shown in Fig. 12. The basic idea of this approach is to shrink backwards the cone of i (level by level) until we find a cut-set whose nodes do not have dependent primary inputs. For example, in Fig. 12, suppose T_i = {1, 2, 3, 5} is not independent with respect to a specified $P(X)$. If the circuit is compressed up one level from node i, the original primary output i will be replaced by two new primary outputs x, y. Suppose inputs 1, 2, 3 are pairwise linearly independent; since x is affected by inputs 1, 3, and y is affected by 2, 3, they have been made linearly independent with respect to the new primary outputs (x, y). The result of this operation is to reduce the segment to one whose end boundary is the above cut-set. If this operation is repeated for all primary outputs which contain some dependent primary inputs, and

join the corresponding cut-sets, a reduced segment S' which can be exhaustively exercised by $P(X)$ is generated.

The derivation time of $P(X)$ depends on the tradeoff between the time required for validation of the primitive polynomial and the test circuit overhead. If hardware overhead is a secondary issue, it is always possible to reduce segments as above for RETP generation using LFSR. There is a drawback, however, in that the number of interconnections may be very large, thereby, many square nodes may be induced. A good derivation-time/hardware tradeoff can be achieved by the following heuristic procedure. First, choose one primitive polynomial called $P_i(X)$ randomly, then test whether it can exhaustively exercise $T[S]$ or not. If yes, then the desired polynomial has been derived, and the search is terminated. Otherwise, S is reduced from the ending boundary until all (new) primary outputs can be exhaustively exercised; and a new segment called S^i is generated. The process is repeated at most K times, where K depends on the tradeoff desired. If in the end of this derivation, no $P_i(X)$'s can exhaustively exercise S, then the $P_i(X)$ whose corresponding S^i induces the smallest number of square nodes between S^i and $G_C - S^i$ is chosen. This S^i is set to be the final segment and, thus, it is removed from G_C.

The idea can be described by the algorithm shown below.

Segment Reduction Algorithm:

1) Randomly select a n-th degree primitive polynomial $P(X)$ from a primitive polynomial table [29]. If all related primary inputs of each primary output are linearly independent with respect to $P(X)$, terminate the process; else go to the next step.
2) The primary outputs which correspond to some linearly dependent primary inputs are put into an ordered queue (in decreasing order of the node levels).
3) Dequeue one primary output Z, compress one level for Z, and get new primary outputs. Check whether the new primary outputs are exhaustively exercised by $P(X)$ or not. The new primary outputs which are not exhaustively exercised are placed into the queue.
4) Repeat step 3 until queue is empty.
5) A segment is now derived and recorded. If K primitive polynomials have been tested, go to next step; else go to step 1.
6) Select the primitive polynomial $P_i(X)$ whose corresponding segment S^i has minimal number of output nodes in its ending boundary.

The testing of linearly independent relations for each primary output can be implemented by Gaussian elimination in $O(m^3)$ where m is the maximal PID of the segment. Fig. 12 demonstrates the basic idea of the above algorithm. Suppose a primitive polynomial with degree four (maximum PID = 4) is randomly chosen. After checking linear dependency for each output, assume outputs i, j and k (black nodes) can not be exhaustively exercised. Then place i, j, k into the queue ordered by the levels of nodes (descending order), and reduce the subcircuit from k, then j, then i.

It should be emphasized that once the final version of the first segment has been derived, the partitioning process will be applied repeatedly on the remaining graph (obtained by removing this segment) to generate the rest of the partition segments.

Theorem 4.1: The segment reduction algorithm has complexity $O(KNn^3)$, where N is the number of nodes in the determined segment, K is the number of primitive polynomials tested, and n is the maximum PID of S (Proof: Appendix V).

Since the value of n is always very small ($\leq$20), the complexity is tolerable.

An important point to make is that, although the derivation of $P(X)$ for each segment is tedious, the proposed partitioning technique has *accelerated* the derivation. This is due to utilizing the minimum vertex-cut during circuit graph partitioning (in Section III), and it significantly reduces the number of primary outputs (i.e., the number of T_i's) and simplifies the input-output relations (i.e., the number of elements in each T_i). To further illustrate this point, let S_0 denote the segment generated by the greedy method and S_1 be the segment induced by the minimum vertex cut as discussed before. The following theorem holds.

Theorem 4.2: If a LFSR can exhaustively exercise S_0, then it can exhaustively exercise S_1 also, but the converse is not always true (Proof: obvious).

Apparently, the derivation of LFSR for S_1 is much easier than for S_0. Thus, the partitioning algorithm described earlier not only reduces the number of square nodes (overhead) but also facilitates the derivation of LFSRs for RETP generation.

The RETP generation process is incorporated in the circuit partitioning process. At the end of both processes, the whole circuit is partitioned into segments and the RETP generator of each segment is derived. Solving the RETP generation problem without consideration of circuit partitioning is a difficult problem. Unlike conventional approaches, the advantage of the proposed approach in pursuing circuit partitioning and test pattern generation at the same time is that it leads to easier and better solutions of both problems.

V. Experimental Results

To evaluate the performance of the proposed circuit partitioning and RETP generation method, experiments were conducted using ISCAS combinational benchmark circuits [30]. We implemented the algorithm in C and run on a Sun Sparc System 2. The results are presented in the table of Appendix VI using six primitive polynomials for segment reduction and PID = 20. Column three gives the number of levels for each circuit, and columns four to seven describe the number of black square nodes inserted during each repetition of the partitioning and RETP generation process.

For example, greedy expansion from the primary inputs of C432 determines an initial segment with 54 black square nodes inserted to isolate S_0 from $G_C - S_0$ (i.e., $N_1(S_0)$ = 54). Worst case analysis estimates 31 more square nodes to entirely partition the circuit for pseudo-exhaustive testing (i.e., $N_{grd}(G_c - S_0)$ = 31). The minimal vertex cut is then derived to improve the initial segment. Unfortunately, the minimal vertex cut is composed of the primary inputs of C432 (i.e., $N_1(S_1)$ = 0 and $N_{grd}(G_c - S_1)$ = 85). Finally, S_0 is selected as the first segment. Six degree-20 primitive polynomials are used to determine a LFSR which exhaustively exercises S_0. No segment reduction is required in this case since one of the six primitive polynomials satisfies the input-output relations. Segment S_0 is then removed from C432 and the partitioning process is continued. This time, we have $N_1(S_0)$ = 18 and $N_{grd}(G_c - S_0)$ = 13. The minimal vertex cut of S_0 is composed of the ending boundary of S_0. Again, S_0 is determined as the final segment and a primitive polynomial is derived to exhaustively exercise S_0. As shown in Appendix VI, the partitioning and RETP generation process terminates by dividing C432 into four segments with totally 81 square nodes added (Column 8). Note that segment reduction only occurs in the cases of C3540 (the 2nd segment) and C6288 (the 1st segment).

It is quite interesting to find that the heuristic of using minimal vertex cut has significantly improved the hardware overhead in most circuits, except C1908 and C6288, as shown in Appendix VI. For circuit C5315, the greedy method requires 280 (272+8) square nodes inserted to accomplish pseudo-exhaustive testing by assuming that primitive polynomials can always be found for the segments. Our method results in adding 102 square nodes which save about 63.6%

of hardware overhead. In the cases of C499 and C1355, the hardware overhead is saved at least 50%. The square nodes added will affect the critical path delay. Based on our circuit partitioning scheme, the effect of critical path delay can be roughly estimated using the number of segment boundaries. For example, C432 has been partitioned into 4 segments with 3 boundaries. This hints that 3 extra delays will be induced to critical paths. From the benchmark circuit simulation, it can be observed that most circuits can be partitioned only by 2 or 3 boundaries. Thus, the effect of added square nodes to the critical path delay is quite tolerable. The CPU time consumed for circuit partitioning and RETP generation is small for all circuits, and the number of nodes inserted to achieve pseudo-exhaustive testing is also tolerable for most ISCAS benchmark circuits, especially large ones. Performance of the proposed method can be further improved by:

1) Using a stronger heuristic for circuit partitioning [14]; or
2) Giving more primitive polynomials for LFSR derivation [29].

VI. Conclusions

In this paper, circuit partitioning for pseudo-exhaustive testing and RETP generation using LFSR has been presented. A graph-theoretic model of VLSI circuit is proposed. Based on this model a circuit partitioning algorithm is devised, to partition the circuit into a set of exhaustively testable subcircuits with restricted hardware overhead. A new approach based on the subcircuit modification technique is proposed, with the objective of generating RETP of limited length(2^{20}) using LFSRs for each of the subcircuits. This task is embedded in the circuit partitioning process itself, leading to an efficient and well-coordinated solution. The overall self-testing time can be further reduced by parallel test scheduling [23]–[25].

The entire approach can be designed to be embedded in existing VLSI CAD tools. Given a circuit, the circuit partitioning and RETP generation algorithm can evaluate the hardware overhead based on the circuit topology and gate organization. Users may then decide to apply the proposed scheme if the circuit overhead is tolerable for them. The simulation results are particularly encouraging.

Appendix I

```
/* FORWARD—backtracking forward (i.e., forward
     after backtracking occurs)
     forward—normal forward (nonbacktracking forward) */
if (direction is forward or FORWARD)
   if ((i is labeled "F") or ((direction is forward) and
     (i is in an existing path)
     and (i is labeled "B"))) then return FALSE;
   if ((d irection is FORWARD) or (i is not in any
     existing path)) then
       label "F" to i;
       if (i  is not in the beginning boundary) then
            for each child j of i
                if  ((lead j - i is not in any path) and
                   (j is not labeled "F") and path (j,
                       forward)) then
                   add lead j-i to the path;
                   return TRUE
       else
            return TRUE
   else
       if (i is labeled "B") then return FALSE;
       label "B" to i;
       let j be the father of i in the hit path;
       if (path(j, backward)) then
            remove j-i from the path;
            return TRUE
else
   result = path(i, FORWARD);
   if ((result is FALSE) and ((i is not in the
       ending boundary) or
        (i is not start node of existing path))) then
            if (i is labeled "B") then return FALSE;
            label "B" to i;
            let j be the father of i in the hit path;
            if (path(j, backward)) then
                remove j-i from the path;
                return TRUE;
   return result
return FALSE;
```

[VERTEX STREAM ALGORITHM]

```
enqueue (all of the nodes in the ending boundary of S);
i = dequeue();
While (i is not null)
   find = FALSE; path = empty;
   find = path (i, forward);
   if (find = TRUE)
        record the path;
        erase all of the labels;
        insert all of the nodes which have been
        dequeued back into the queue;
   i = dequeue();
```

Appendix II

Proof: The labeling technique guarantees to find all disjoint vertex paths. In the derivation of each vertex path, the labeling technique will traverse from the end boundary and try to reach the beginning boundary of S_0. In the traversal process, each node in S_0 can be traveled at most twice (forwardly, and backwardly), so the complexity of deriving a vertex path is at most $2 \cdot N$ where N is the number of nodes in S_0. Since the number of vertex paths can not be larger than the number of nodes, μ, in the end boundary of S_0, the complexity of deriving all vertex paths can not be larger than $2 \cdot N \cdot \mu$, which proves the theorem. QED

Appendix III

Proof: Let set M contain all nodes labeled "B" but not "F". We claim that M is the minimal vertex cut of S_0. That is, removing all nodes in M out of segment S_0 will disconnect all paths from the beginning boundary (BB) to the ending boundary (EB) of S_0. If not, there exists at least one path P' from EB to BB which is discussed as follows.

1) P' does not hit any vertex path.
   Our labeling technique will not stop since a new vertex path P' can be generated. This contradicts the fact that no new vertex paths can be found, when the maximal vertex stream process terminates.
2) P' hits vertex path P somewhere between W and U (but not including U).
   As shown in Fig. 8(a), the labeling technique will find vertex paths $W \rightarrow V \rightarrow Y \rightarrow X$ and $Z \rightarrow U \rightarrow T$ to replace vertex path P. Again, this contracts the fact that no new vertex paths can be found. Note that path P' may hit several vertex paths which can be discussed similarly.
3) P' hits vertex path P somewhere between U and T (but not including U).
   As shown in Fig. 8(b), P' hits vertex path P on node Q and backtracking occurs. According to our labeling technique, node U will be labeled "B" and "F". Contradiction. The case

in which P' hits several vertex paths can also be discussed similarly.

It is easy to verify that M is the minimal set to form the minimal vertex cut. QED

APPENDIX IV

Proof: Let S_i denote the i-th segment derived by our partitioning algorithm in G_C, β_i denote the set of square nodes used to separate S_i from $G_C - S_1 - \cdots - S_i$, and $\gamma_{i,j}$ denote the set of square nodes induced by the jth segment when greedily expand $G_C - S_1 - \cdots - S_i$ from β_i. Note that β_0 is the beginning boundary of G_C.

Assume G_C is partitioned into n segments by our algorithm. According to the strategies in the algorithm, we have.

$$|\beta_1| + |\gamma_{1,1}| + \cdots + |\gamma_{1,a_1}| \le |\gamma_{0,1}| + \cdots + |\gamma_{0,a_0}| \quad (1)$$

$$|\beta_2| + |\gamma_{2,1}| + \cdots + |\gamma_{2,a_2}| \le |\gamma_{1,1}| + \cdots + |\gamma_{1,a_1}| \quad (2)$$

$$\cdots$$

$$|\beta_i| + |\gamma_{i,1}| + \cdots + |\gamma_{i,a_i}| \le |\gamma_{i-1,1}| + \cdots + |\gamma_{i-1,a_{i-1}}| \quad (i)$$

$$|\beta_{i+1}| + |\gamma_{i+1,1}| + \cdots + |\gamma_{i+1,a_{i+1}}| \le |\gamma_{i,1}| + \cdots + |\gamma_{i,a_i}| \quad (i+1)$$

$$\cdots$$

$$|\beta_{n-1}| + |\gamma_{n-1,1}| + \cdots + |\gamma_{n-1,a_{n-1}}| \le |\gamma_{n-2,1}| + \cdots + |\gamma_{n-2,a_{n-2}}| \quad (n-1)$$

$$|\beta_n| \le |\gamma_{n-1,1}| + \cdots + |\gamma_{n-1,a_{n-1}}| \quad (n)$$

where $a_k \ge 1$ for n - $1 \ge k \ge 0$. The semantics of the equations can be described as follows: Each time when S_i is derived, we have S_i, β_i, $\gamma_{i,1}$, ... , γ_{i,a_i} available. The next step is to derive S_{i+1} by deriving the minimum vertex cut V_i between β_i and$\gamma_{i,1}$ in G_C. Note that V_i may equal to $\gamma_{i,1}$. In this case, V_i can not improve the performance of $\gamma_{i,1}$, and S_{i+1} is derived by choosing $\gamma_{i,1}$ as its ending boundary. This makes $\beta_{i+1} = \gamma_{i,1}$, and $\gamma_{i+1,1} = \gamma_{i,2}$, ..., $\gamma_{i+1,a_{i+1}} = \gamma_{i,a_i}$, the "=" relation in equation $(i + 1)$ holds in this case.

If V_i does not equal to $\gamma_{i,1}$, we evaluate its performance by the partitioning algorithm shown above. If the performance of V_i is better than $\gamma_{i,1}$, then V_i is chosen to be β_{i+1} and we have.

$$|\beta_{i+1}| + |\gamma_{i+1,1}| + \cdots + |\gamma_{i+1,a_{i+1}}| < |\gamma_{i,1}| + \cdots + |\gamma_{i,a_i}|.$$

If the performance of V_i is worse than that of $\gamma_{i,1}$, then $\gamma_{i,1}$ is chosen to be β_{i+1}, and the discussion is the same as the case that V_i equals to $\gamma_{i,1}$.

By eliminating the same $\gamma_{i,j}$'s in both sides of the above equations, we have.

$$|\beta_1| + \cdots + |\beta_n| \le |\gamma_{0,1}| + \cdots + |\gamma_{0,a_0}|.$$

Since $\gamma_{0,1}$, $\cdots$, γ_{0,a_0} are the set of square nodes induced by the greedy method in G_C, and β_1, $\cdots$, β_n are those induced by our partitioning algorithm, we proved the theorem. QED

APPENDIX V

Proof: For each selected primitive polynomial, the number of nodes reduced from the ending boundary of the determined segment, S, is at most N, and the computation complexity of checking the linear dependency (or independency) for each node reduction is $O(n^3)$, obviously, for each primitive polynomial, the complexity is bounded by $O(Nn^3)$. Since K primitive polynomials tested, the overall computation complexity is, of course, $O(KNn^3)$. We proved the theorem. QED

APPENDIX VI

CKT	# of PI	# of level	$N_1(S_0)$	$N_{gnd}(G_c - S_0)$	$N_1(S_1)$	$N_{gnd}(G_c - S_1)$	Nodes inserted	CPU time
C432	36	18	54	31	0	85	54	3.95s
			18	13	18	13	18	
			13	0	9	0	9/81	
C499	41	12	80	0	40	0	40	2.71s
C880	60	25	60	2	24	6	24	5.60s
			6	0	6	0	6/30	
C1355	41	25	80	0	40	0	40	12.57s
C1908	33	41	99	6	0	105	99	23.10s
			6	0	6	0	6/105	
C2670	157	33	110	9	59	9	59	46.64s
			9	0	7	0	7/66	
C3540	50	48	168	65	44	117	44	184.28s
			86	31	36	105	91*	
			31	0	18	0	18/153	
C5315	178	50	272	8	81	120	81	327.61s
			120	0	21	0	21/102	
C6288	32	125	93	36	0	129	99*	118.21s
			26	12	26	12	26	
			11	1	11	1	11	
			1	0	1	0	1/137	
C7552	206	44	243	18	114	147	243	531.99s
			18	0	13	0	13/256	

* Segment reduction

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their constructive and helpful suggestions.

REFERENCES

[1] E. J. McCluskey, "Built-in self-test techniques," *IEEE Design & Test*, pp. 21–28, Apr. 1985.
[2] B. Konemann, J. Mucha, and G. Zwiehoff, "Built-in logic block observation techniques," in *Proc. Int. Test Conf.*,Cherry Hill, NJ, 1979, pp. 37–41.
[3] J. Savir, "Syndrome-testable design of combinational circuits," *IEEE Trans. Comput.*, vol. C-29, no. 6, pp. 442–451, June 1980.
[4] Z. Barzilai, J. Savir, G. Markowsky, and M. G. Smith, "The weighted syndrome sums approach to VLSI testing," *IEEE Trans. Comput.*, vol. C-30, no. 12, pp. 996–1000, Dec. 1981.
[5] E. J. McCluskey and S. Bozorgui-Nesbat, "Design for autonomous test," *IEEE Trans. Comput.*, vol. C-30, no.11, pp. 866–875, Nov. 1981.
[6] R. M. Sedmak, "Implementation techniques for self-verification," in *Proc. Int. Test Conf.*, 1980, pp. 267–278.
[7] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudo-Random Techniques*, Wiley, 1987.
[8] E. C. Archambean and E. J. McCluskey, "Fault coverage of pseudo-exhaustive testing," in *Proc. Int. Symp. Fault-Tolerant Computing*, 1984, pp. 141–145.
[9] E. J. McCluskey, "Verification testing—A pseudo-exhaustive test technique," *IEEE Trans. Comput.*, pp. 541–546, June 1984.
[10] S. N. Bharr, F. R. K. Chung, and A. L. Rosenberg, "Partitioning circuits for improved testability," in *Proc. 4th MIT Conf. Advanced Res. VLSI*, Apr. 1986, pp. 91–106.
[11] M. W. Roberts and P. K. Lala, "An algorithm for the partitioning of logic circuits," *IEE Proc.*, vol. 131, pt. E, no. 4, pp. 113–118, July 1984.
[12] I. Shperling and E. J. McCluskey, "Circuit segmentation for pseudo-exhaustive testing via simulation annealing," in *Proc. Int. Conf. Computer-Aided Design*, 1987, pp. 148–151.
[13] J. G. Udell and E. J. McCluskey, "Efficient circuit segmentation for pseudo-exhaustive test," *Proc. Int Conf. Computer-Aided Design*, 1987, pp. 148–151.
[14] D. Kagaris, F. Makedon, and S. Tragoudas, "On minimizing hardware overhead for pseudo-exhaustive circuit testability," in *Proc. IEEE Int. Conf. Comput. Design*, 1992, pp. 358–364.
[15] R. Srinivasan, S. K. Gupta, and M. A. Breuer, "An efficient strategy for pseudo-exhaustive testing," in *Proc. ACM/IEEE Design Automat. Conf.*, 1993, pp. 242–248.
[16] J. G. Udell, Jr., and E. J. McCluskey, "Pseudo-exhaustive testing and segmentation: Formal definitions and extended fault coverage results," in *Proc. IEEE Int. Symp. Fault-Tolerant Computing*, 1989, pp. 292–298.
[17] S. Hellebrand and H. J. Wunderlich, "Tools and devices supporting the pseudo-exhaustive test," in *Proc. Euro. Design Automat. Conf.*, 1990, pp. 13–17.
[18] J. G. Udell, Jr., "Efficient segmentation for pseudo-exhaustive BIST," in *Proc. IEEE Custom Integrated Circuits*, 1992, pp. 13.6.1–13.6.6.
[19] E. Wu, "PEST: A tool for implementing pseudo-exhaustive self-test," *AT&T Tech. J.*, pp. 87–100, 1991.
[20] W. B. Jone and C. A. Papachristou, "A coordinated approach to partitioning and test pattern generation for pseudo-exhaustive testing," in *Proc. ACM/IEEE Design Automat. Conf.*, 1989, pp. 525–530.
[21] W. B. Jone, Methodology of partitioning and exhaustive test pattern generation for built-in self-testing of VLSI circuits, Ph.D. Dissertation, Dept. of Computer Eng. and Sci., Case Western Reserve Univ., Aug. 1987.
[22] N. Deo, *Graph Theory with Application to Engineering and Computer Science*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1974.
[23] C. R. Kime and K. K. Saluja, "Test scheduling in testable VLSI circuits," in *Proc. Int. Symp. Fault-Tolerant Computing*, 1982, pp. 406–412.
[24] C. L. Craig, C. R. Kime, and K. K. Saluja, "Test scheduling and control for VLSI built-in self-test," *IEEE Trans. Comput.*, vol. 37, no. 9, pp. 1099–1109, Sept. 1988.
[25] W. B. Jone, C. A. Papachristou, and M. Pereira, "A scheme for overlaying concurrent testing of VLSI circuits," in *Proc. ACM/IEEE Design Automat. Conf.*, 1989, pp. 531–536.
[26] L. T. Wang and E. J. McCluskey, "Condensed linear feedback shift register (LFSR) testing—A pseudo-exhaustive test technique," *IEEE Trans. Comput.*, vol. C-35, no. 4, pp. 367–370, Apr. 1986.
[27] D. T. Tang and C. L. Chen, "Logic test pattern generation using linear codes," *IEEE Trans. Comput.*, vol. C-33, no. 9, pp. 845–850, Sept. 1984.
[28] Z. Barzilai, D. Coppersmith, and A. L. Rosenberg, "Exhaustive generation of bit patterns with applications to VLSI self-testing," *IEEE Trans. Comput.*, vol. C-32, no. 2, pp. 190–194, Feb. 1984.
[29] W. W. Peterson, *Error Correcting Codes*. Cambridge, MA: MIT Press, 1961.
[30] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN," in *Proc. IEEE Symp. Circuits Syst.*, 1985, pp. 695–698.

Timing Models for Gallium Arsenide Direct-Coupled FET Logic Circuits

Ayman I. Kayssi and Karem A. Sakallah

***Abstract*— In this paper we derive delay and transition time macromodels for GaAs DCFL logic gates. The macromodels are derived by a systematic application of dimensional analysis aimed at finding suitable minimal functional forms that capture the effects of all relevant parameters. The process is illustrated through a detailed step-by-step account of the macromodel development for DCFL inverters. Based on different modeling approximations, one- and two-argument macromodel functions are derived and compared. The inverter macromodel is then used as a basis for developing timing macromodels for superbuffers and NOR gates. The NOR gate macromodels account for the simultaneous and near-simultaneous switching of two inputs, with an extension to multiple inputs.**

I. INTRODUCTION

We describe in this paper the macromodels developed to facilitate the timing verification of a gallium arsenide (GaAs) microprocessor that was designed at the University of Michigan [15], [23]. The microprocessor's CPU implements a subset of the MIPS R2000 instruction set [9]. Fabrication was provided, through MOSIS [22], by Vitesse Semiconductor Corporation. Physical design, including cell generators, parasitic extraction, and static timing analysis, was done using the Cascade Design Automation system [5]. This system, primarily geared towards CMOS design, was extended to handle GaAs Direct-Coupled FET Logic (DCFL) circuits.

The structural similarity between the GaAs DCFL and nMOS logic families led us to conjecture that the forms of their gate delay models would also be similar. In MOS technology (both nMOS and CMOS), gate delay is commonly expressed as a linear combination of two components: a fixed intrinsic delay Δ_{int}, and a variable load-dependent delay αC_L where C_L is the total capacitive load on the gate and α is the incremental increase in gate delay per

Manuscript received September 20, 1993; revised August 26, 1994. This work was supported in part by DARPA Grant DAAL03-90-C-0028, by NSF Grant MIP-9014058, and by the University Research Board of the American University of Beirut. This paper was recommended by Associate Editor J. White.

A. I. Kayssi is with the Department of Electrical Engineering, American University of Beirut, New York Office, New York, NY 10022 USA.

K. A. Sakallah is with the Advanced Computer Architecture Laboratory, Department of EECS, University of Michigan, Ann Arbor, MI 48109 USA.

IEEE Log Number 9406942.