Test Point Insertion: Scan Paths through Combinational Logic

Chih-chang Lin[1], Malgorzata Marek-Sadowska[2], Kwang-Ting Cheng[2], and Mike Tien-Chien Lee[3]

Mentor Graphics[1]
San Jose, CA 95131

Univ. of California[2]
Santa Barbara, CA 93106

Fujitsu Lab. of America[3]
Santa Clara, CA 95054

Abstract

We propose a low-overhead scan design methodology which employs a new test point insertion technique to establish scan paths through the functional logic. The technique re-uses the existing functional logic; as a result, the design-for-testability (DFT) overhead on area or timing can be minimized. In this paper we show an algorithm which considers the test point insertion for reducing the area overhead for the full scan design. We also discuss its application to timing-driven partial scan design.

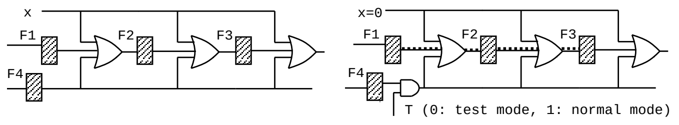
I. Introduction

Automatic test pattern generation (ATPG) for sequential circuits is a difficult problem because of the lack of direct controllability of the present state lines and direct observability of the next state lines. To enhance testability, design-for-testability (DFT) techniques aiming at improving controllability and observability of the state lines have been proposed, such as full scan [1, 2, 3] and partial scan [4, 5]. Both scan techniques facilitate testing of a sequential circuit by interconnecting selected flip-flops into a shift register during the test mode to directly control and observe the state lines. The complexity of ATPG is therefore reduced. However, the area and delay overheads imposed by conventional scan can be significant due to the extra scan multiplexers (MUXs) in the scan flip-flops (assuming that MUXed D flip-flops are used) and the extra routing area for the scan chains.

To alleviate the above DFT penalty, we propose a low-overhead scan design methodology which employs the *test point insertion* to establish the scan paths through the existing combinational logic. These test points are established by appropriately inserting a two-input AND gate or a two-input OR gate with a common test input. The essential idea is illustrated in Figure 1. Figure 1(a) shows a portion of a sequential circuit, where the boxes represent flip-flops. By inserting a test point at the output of F_4 and setting the primary input x to 0 during the test mode, a scan chain $F_1 \rightarrow F_2 \rightarrow F_3$ can be formed through the combinational logic, as shown in the dotted line of Figure 1(b). In this example, we established a partial scan chain involving three flip-flops using the functional logic and the area overhead is a two-input AND gate, while conventional scan design would require two multiplexers.

In our method, the cost of inserting a test point is one AND (OR) gate and a connection from the test input T, while converting a flip-flop into a MUXed scan flip-flop requires a multiplexer, a connection from another flip-flop, and a connection from the test input T. Inserting test points is advantageous in terms of area, if inserting k test points can successfully establish k more scan paths. A scan

[1]This work was conducted when the author was in University of California, Santa Barbara.



(a) portion of a sequential circuit. (b) insertion of a test point.

Fig. 1. Example of a test point insertion.

path here is defined as a physical path between two flip-flops that can be fully sensitized in the test mode. Moreover, the method of inserting test points can be applied for timing-driven scan design. For example, we can add test points away from the critical paths while still being able to establish scan paths through critical nets.

In this paper, we discuss two applications of using the test point insertion technique for scan design. First, we consider the full scan design environment, where the objective is to establish as many scan paths and use as few test points as possible. The advantage of our technique in this application is the reduction of area overhead. Next, we consider the partial scan design environment. The objective is to break cycles without degrading the performance of the design. In partial scan design, a flip-flop is selected by the cycle-breaking algorithms [4, 6, 7] sequentially and if timing constraints are not met for converting it into a MUXed scan flip-flop, test point insertion technique can be applied to avoid adding test circuitry onto the critical path to eliminate timing degradation. Due to the space limitation, some details are omitted. Please refer to [8].

II. Review and terminology

To improve the ATPG efficiency, a testpoint insertion technique [9] inserted a set of test-cells into a circuit to improve the observability and controllability of some selected internal signals. The size of a test-cell may be large and the compound effect of adding such cells may result in significant area overhead (a test-cell requires at least one flip-flop and two multiplexers). Our test point is simply a two-input AND or a two-input OR gate and the purpose of inserting test points is to establish a scan chain, which in turn makes scanned flip-flops fully observable and controllable.

The work in [10, 11, 12] presented algorithms to reduce scan overhead by attempting to merge scan MUXs into the combinational logic during logic synthesis. In [13], a scan design methodology called *free-scan* was proposed. By setting appropriate values at primary inputs during the test mode, some combinational paths between flip-flops can be sensitized and thus a portion of the scan chain can be established without any DFT overhead. In [14], the concept of embedded scan was proposed and attempts were made to embed the scan-multiplexers into the logic immediately preceding the scan flip-flops.

We define some terminology used in the following discussion. A connection is specified by a pair of gates $[g_{source}, g_{sink}]$, where g_{source} is g_{sink}'s fanin. A path is specified by a sequence of gates, $[g_1, \cdots, g_k]$, where g_i is g_{i+1}'s fanin. Side inputs of a path are a set of connections where their g_{sink}'s are on the path, while g_{source}'s are not. Given a gate g and one of its fanins f, we define a constant value v as sensitizing value for the connection $[f, g]$, if setting f to v does not determine the value of g. On the other hand, if it does, v is called a controlling value.

For static timing analysis, we adopt the timing models used in [15] where the delay across a gate g is modeled linearly by its block delay, driving power, and load as follows: *delay(g) = block(g) + drive(g)* $\times$ *load*, where *load* is the total capacitive load driven by gate g. The parameters *block(g)* and *drive(g)* are specified in the technology library. We define the slack time as the difference between the required and arrival times. The slack time of a connection determines how much extra delay can be added to this connection without degrading the overall performance of the circuit. The slack times of all gates have to be positive to guarantee the correctness of the circuit under the given cycle time.

III. Test point insertion

Suppose that a pair of flip-flops is connected through a combinational path. To include this path into a scan chain, all the side-inputs along this path must be set to sensitizing values. If a value of *zero* is desired at a connection c for disabling the connection in the test mode, we can insert a two-input AND gate at c with the test input T as one of its inputs. The value of T is assumed to be 1 in the normal mode and 0 in the test mode. On the other hand, if a value of *one* is desired, we can insert a two-input OR gate with T' as one of its inputs, where T' is the negation of T. To establish a scan path between two flip-flops may require more than one test points. The number of side-inputs along a selected combinational path is an upper bound on the requirement of the number of test points for establishing a scan path through the selected path.

In general, assigning a constant value at a connection (by inserting a test point) may potentially disable more than one side-input because the connection may have multiple fanouts. To efficiently utilize this methodology, we should analyze the circuit's topology and determine the global effect of inserting a particular test point. The objective is to decide at which connections test points should be inserted and what constant values they should be, so that we can establish as many scan paths as possible with as few test points as possible.

A. Test point insertion for full scan design

For a full scan design, the goal here is to use the test point insertion technique to establish as many scan paths (through functional logic) with as few test points as possible, and then use the conventional scan conversion (MUX insertion) for the missing scan paths in order to have a connected scan chain. We developed an algorithm, called TPGREED, for this purpose. TPGREED examines the combinational paths between flip-flops in the circuit and then, in a greedy way, sequentially inserts the test points with appropriate values. During the insertion, all the possible candidate locations are sorted according to their potential contribution in establishing scan paths. The details of the algorithm are as follows.

Given a sequential circuit, we build first a sparse matrix A, where the entry A_{ij} represents a set of combinational paths from flip-flop F_i to F_j. Since there might exist a large number of paths in the circuit and in general it is more costly to establish a scan path through a combinational path with a large number of side-inputs, we heuristically limit the number of paths for consideration and record only those paths with a number of side-inputs smaller than a user specified upper bound K_{bound} to save computation time.

Given a combinational path p_k in A_{ij}, let $|p_k|$ denote the number of side-inputs along this path. During the iteration of test point insertion and the forward implication of the assigned constants, side-inputs of p_k may have either sensitizing, controlling or unknown values. If there exists a side-input which has a controlling value, it will be impossible to build a scan path through p_k. We call such a path a *nullified* path and remove it from A_{ij}. On the other hand, if there is no side-input with a controlling value, we use w_k to denote the number of side-inputs which have an unknown value. The gain of setting one of the side-inputs to a sensitizing value is $1/w_k$. Notice that, for each path p_k, the number $|p_k|$ does not change while w_k decreases during the process. When w_k is reduced to zero, the path p_k successfully becomes a scan path.

Given a connection c, if we insert a test point with value v at c, forward implication of v at c may imply new values v_i's at some connections c_i's in c's fanout cone. We denote them as $\{(c_1, v_1), \cdots, (c_h, v_h)\}$. Some of the v_i's are controlling while others are sensitizing values. With these new constant values, paths passing through $c, c_1, \cdots, c_{h-1}$, or c_h will be nullified. Also, paths with c or c_i's as side-inputs and v or v_i's being controlling values will be nullified. On the other hand, paths with c or c_i's as side-inputs and v or v_i's being sensitizing values will have their w_k's reduced. We denote the set of such paths as S_c. Thus, the gain of inserting a test point with value v on c is as follows:

$$\sum_{j=1}^{n}(\mathrm{MAX}_{i=1}^{n}(\mathrm{MAX}_{p_k \in A_{ij}, p_k \in S_c} \frac{1}{w_k})), \qquad (1)$$

where n is the number of flip-flops. We sum the contribution of making flip-flops F_j's $(1 \leq j \leq n)$ as a part of the scan chain. Among all the paths in A_{ij}'s ending at a flip-flop F_j, we choose the maximal contribution instead of their summation because our objective is to establish exactly one path from a flip-flop to flip-flop F_j in the scan chain.

Based upon the cost function in Equation 1, we iteratively choose a connection and a value (c, v) with the highest gain as a test point and update the entries A_{ij}'s in the matrix A by removing the nullified paths. During the iteration of the insertion process, if the scan path $F_i \rightarrow F_j$ is established, we record this path as part of the final scan chain. Since the scan chain has to be acyclic, we also remove some entries A_{pq}'s if adding the path $F_p \rightarrow F_q$ to the scan chain would result in a cycle. For example, consider a sequential circuit with four flip-flops F_1, F_2, F_3 and F_4. Suppose that we have already established a path $F_1 \rightarrow F_2$. Assume adding a new test point will establish scan path $F_2 \rightarrow F_3$. Besides recording this new scan path in the scan chain, we remove A_{31}, because a path $F_3 \rightarrow F_1$ would result in a cycle $F_1 \rightarrow F_2 \rightarrow F_3 \rightarrow F_1$. We also have to remove all A_{i3}'s and A_{2j}'s, since each flip-flop in the scan chain should have only one incoming edge and one outgoing edge.

B. Input assignment

After performing the procedure described above, we know exactly at which connections (i.e., $c_1, \cdots, c_m$) test points should be inserted and also what values (i.e., $v_1, \cdots, v_m$) they should be. Before physically inserting AND (for value 0) or OR (for value 1) gates, we make attempt to set up as many of the values v_i's as possible at the

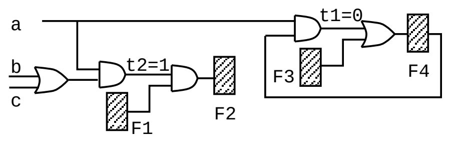


Fig. 2. Example of setting values for test points by assigning values at primary inputs.

connections c_i's by assigning appropriate values at the primary inputs to avoid inserting unnecessary test points. For example, Figure 2 shows a portion of a sequential circuit, where a, b and c are primary inputs. Assume that the insertion procedure decides to insert test points at connections $t1$ and $t2$ with values 0 and 1, respectively, to establish two scan paths, $F_1 \rightarrow F_2$ and $F_3 \rightarrow F_4$. We can use appropriate values at primary inputs to produce one of the desired constants (e.g. $a = 0$, or $a = 1$ & $b = 1$, or $a = 1$ & $c = 1$), and use one test point to achieve another desired constant. In general, an optimization algorithm is required to decide the optimal input assignment to maximize the number of signals with desired values without inserting test points. We adopt the algorithm described in [13] for this purpose.

C. Overall algorithm

Besides the circuit, users should provide two extra parameters K_{bound} and gain$_{bound}$. The parameter K_{bound} is used to limit the number of side-inputs for paths considered for establishing scan paths. The parameter gain$_{bound}$ is used to terminate the algorithm when the highest gain computed by Equation 1 for all candidate connections is smaller than gain$_{bound}$. During the iteration, some scan paths may be established. Besides adding them as a portion of the scan chain, we also have to make sure that the subsequent insertions will not destroy the established scan paths. In our current implementation, after a test point is inserted, we re-compute the gain of inserting a test point at each connection, before inserting the next one. This could cause high computation time. One possible solution is to have an incremental algorithm which only re-computes the gain of those affected connections. We also apply a procedure which determines the values for the primary inputs to reduce the number of required test points (as discussed in Section B).

D. Experimental results

We tested the proposed test point insertion method on a number of ISCAS89 and MCNC91 sequential benchmarks. All circuits are optimized by SIS *script.algebraic* script and mapped using technology libraries *nand-nor.genlib* and *mcnc_latch.genlib* for minimal area. In the current implementation, we can only handle primitive gates, including AND, OR, NAND, and NOR gates.

The results of test point insertion are shown in Table I. We report the number of flip-flops in the circuit (A), the number of test point inserted (B), the number of test points' values which can be setup freely by primary inputs (C), and the number of scan paths established (D). The CPU time is measured on a SUN SPARC 5 with 128 Megabyte memory. In our experiments, the parameters K_{bound} and gain$_{bound}$ are set to 10 and 0.5. For example, we inserted 137 test points in circuit *s15850* to establish 244 scan paths. Among the 137 test points, we can use primary inputs to set up two of them. So the actual number of required test points is 135. Assuming that the area costs of inserting a multiplexer and a test point are 2 and 1, the reduction of area overhead will be

$$1 - \frac{2(A-D)+(B-C)}{2A}.$$

TABLE I
EXPERIMENT RESULTS FOR ISCAS89 AND MCNC91 CIRCUITS.

circuit	#FF A	#insertion B	#free C	#scan paths D	reduction	CPU (sec)
s5378	152	28	3	62	32.6%	171
s9234	135	35	1	57	29.6%	296
s13207	453	120	2	196	30.2%	1151
s15850	540	137	2	244	32.7%	3907
s35932	1728	3	3	1440	83.3%	3019
s38417	1636	169	8	448	22.5%	6852
s38584	1294	164	1	1133	81.3%	15324
bigkey	224	115	3	112	25.0%	576
dsip	224	4	3	168	74.8%	52
mult32a	32	31	1	31	50.0%	24
mult32b	61	31	1	31	26.2%	26

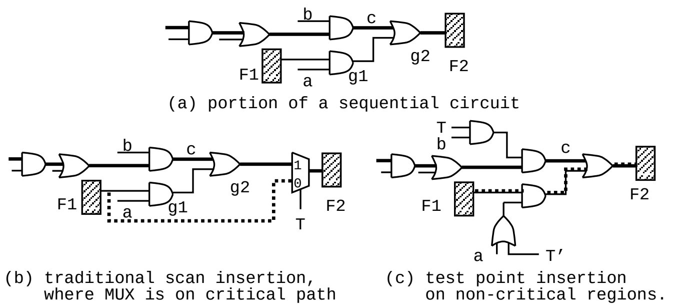
If we use MUXed D flip-flops, the area overhead can be approximated as $2A$. For our method, the term $(B - C)$ represents the number of test points inserted and term $(A - D)$ represents the number of remaining flip-flops which requires a multiplexer for each of them.

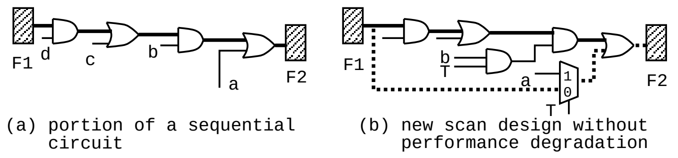
The amount of the reduction depends on a circuit's structure, the logic synthesis algorithm, and our test point insertion algorithm. In the case of *s35932*, as much as 83% in the area overhead reduction can be achieved. The computation time for *s38584* is quite high. This is because the number of paths considered in our algorithm is huge (270463). Possible ways to reduce the computation time are to have a smaller K_{bound}, or have an incremental algorithm for re-computing the gains as discussed in Section C.

IV. TIMING-DRIVEN SCAN PATH DESIGN BY TEST POINT INSERTION

Although partial scan has a lower overhead in terms of area, it may not be so when we consider timing issues. In [7], a timing-driven partial scan flip-flop selection algorithm was proposed. There, a flip-flop with a slack time less than the gate delay of a multiplexer is not allowed for selection, even if it has high gains for breaking cycles. As a result, the number of selected flip-flops for breaking cycles is usually larger than the case in which timing issues are not considered. Moreover, there are circuits that have no cycle-breaking solutions without degrading the performance. Here, we enhance the timing-driven partial scan design methodology [7] by combining the cycle-breaking algorithm and the test point insertion method.

If we scan a flip-flop by converting it to a MUXed scan flip-flop, where the slack time of the flip-flop is less than the gate delay of a multiplexer, such a conversion will result in timing degradation. However, by incorporating the test point insertion technique, we may scan the flip-flop without any timing penalty. Figure 3(a) shows a portion of a sequential circuit, where the bold lines denote a critical path. To scan the flip-flop F_2 by inserting a MUX directly behind F_2 will increase the critical delay and result in timing degradation, as shown in Figure 3(b). However, there exists a combinational path from F_1 to F_2. To make this combinational path $F_1 \rightarrow g1 \rightarrow g2 \rightarrow F_2$ a scan path, all the side-inputs, a and c, must have sensitizing values in the test mode. To achieve this, we insert a test point (OR gate) at a. However, we cannot insert a test point at c without degrading the performance, since c is on a critical path. Instead, we can insert a test point (AND gate) at b, which in turn will induce a sensitizing value 0 at c. The insertion of test points at a and b causes no timing violation and establishes a scan path from F_1 to F_2. The result is shown in Figure 3(c).


b
c
g2
F2
F1
a
g1
(a) portion of a sequential circuit
b
c
g2
F1
a
g1
T
F2
T
b
c
F2
F1
a
T'
(b) traditional scan insertion, where MUX is on critical path
(c) test point insertion on non-critical regions.
Fig. 3. Example of test point insertion for timing-driven scan path design. The bold line represents critical path, while the dotted line represents the scan path.


F1
d
c
b
a
F2
F1
b
T
a
T
F2
(a) portion of a sequential circuit
(b) new scan design without performance degradation
Fig. 4. Example of MUX and test point insertion for timing-driven scan path design.

The above transformation has one disadvantage. That is, since the scan path is from F_1 to F_2, we have to scan F_1 too in order to have a connected scan chain. In the partial scan environment, scanning F_1 might not help break cycles. Also, there is no guarantee that we can scan F_1 without timing degradation. To overcome this problem, we can consider insertion of MUX's as well. The MUX's need not be placed immediately behind the scan flip-flops. We only insert them at connections with enough slack times. If necessary, we may also insert test points at the corresponding side-inputs to sensitize the scan path. For example, in Figure 4(a), we can insert a multiplexer at a, and a test point at b to establish a scan path F_1 to F_2 (Figure 4(b)). Notice that, using the above transformation, the predecessor of F_2 in the scan chain need not be F_1 and could be any other flip-flop.

A. Topological feasibility analysis

Given a flip-flop selected by the cycle-breaking algorithm for scan, we derive the formula in Figure 5 to check if we can scan it without timing degradation. For simplicity, we assume that a gate has one of the following five types: AND, OR, INVERTER, FLIP-FLOP or INPUT. The gate delays of a multiplexer, a two-input AND and a two-input OR are t_{mux}, t_{and} and t_{or}, respectively. The slack time $slack(c_i)$ of a connection c_i is computed as the difference between the required and arrival times, while the gate type, $gate\_type(c_i)$, is the gate type of c_i's source gate. The $fanin(c_i)$ denotes the set of fanins of c_i's source gate.

To scan a flip-flop, some connection in its fanin cone has to carry the signal from the scan chain. Such a signal is denoted as $scan_{in}$. Also, some connections have to be set to 1 or 0. For example, to convert the circuit in Figure 4(a) to Figure 4(b), we assign a constant value 0 at b and assign a as the $scan_{in}$. We define $cost(c_i, value)$ as the area cost of assigning a connection c_i as $scan_{in}$, 1 or 0, where $value$ is $scan_{in}$, 1 or 0. In Equation 2 of Figure 5, if the slack time of c_i is greater than the gate delay of a multiplexer, we simply insert a multiplexer and the cost is the area of a multiplexer. Otherwise, we recursively check if we can use c_j (one of c_i's fanins) to be part of the scan path (assigning it to $scan_{in}$) and make other fanins c_k's ($k \neq j$) have sensitizing values (assigning them to 1 or 0). Since there may exist multiple solutions, we choose the one with

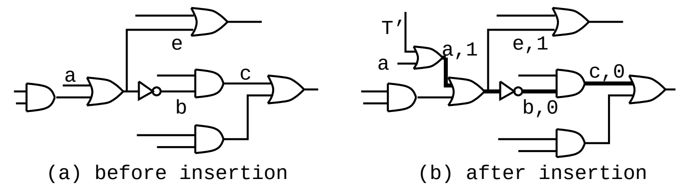


Fig. 6. Example for classification of constants.

a minimal area overhead. If the gate type of c_i is FLIP-FLOP, the cost will be ∞ since it has no fanins to allow further recursion. Equation 3 and Equation 4 are defined similarly. For a flip-flop with fanin connection c_i, if the cost function $cost(c_i, scan_{in})$ is not ∞, we can scan this flip-flop without timing degradation.

The selection of scan flip-flops and the insertion of test points are done sequentially. It is important to keep track of the created scan paths and make sure that the subsequent insertions will not destroy the previous efforts. That is, there are some connections which have constant values associated with them due to the previous insertions. We classify them into two categories: *desired constants* and *side-effect constants*. For example, in Figure 6(a), to make the connection c to be 0 in the test mode, we can insert an AND gate at c (set c to 0), insert an AND gate at b (set b to 0), or insert a OR gate at a (set a to 1). Assume that the slack times of b and c do not satisfy the requirement, while the slack time of a does. A test point can be inserted at a (Figure 6(b)). As a result, we have $a = 1$, $e = 1$, $b = 0$ and $c = 0$. Among them, $a = 1$, $b = 0$ and $c = 0$ are *desired constants* (as shown in the bold line of Figure 6(b)) while $e = 1$ is a *side-effect constant*. To preserve the efforts of this insertion, the desired constants should not be changed by subsequent test point insertions. On the other hand, we are free to change the constant value of side-effect constants.

There is a problem in using the recursive operations defined above. That is, when a test point is inserted at a connection c, the slack times of gates in c's fanin or fanout cone may be affected. Consequently, the function $slack(c_i)$ is not a constant value but depends on the decisions made in the previous recursions. Taking such update into account will result in a very complicated recursive process. To simplify this problem, we restrict the application of recursion only to the *non-reconvergent fanin regions* as defined below. With this restriction, we don't have to update the slack times during the recursion and the result is guaranteed to be correct. Notice that since we restrict our solution space to non-reconvergent fanin region, the obtained solution might be a sub-optimal solution.

Definition 1 *Given a connection c, we define its non-reconvergent fanin region to be a set of connections in its fanin cone, so that each connection has exactly one path to c.*

See the circuit in Figure 7 for illustration. The dotted region is the non-reconvergent fanin region of the connection c. Although the gate $g1$ has two fanouts, a and e, there is only one path from $g1$ to c passing through a. As a result, the connections a, b and d are in the non-reconvergent fanin region of c. On the other hands, since the gate $g3$ has two paths to c, the connections j and k are not in.

Lemma 1 *The non-reconvergent fanin region of a connection c forms a tree rooted at c.*

Theorem 1 *Given a connection c, using Equations 2,3 and 4 recursively while restricting the connections at its non-reconvergent fanin regions, the value slack() for each connection can be considered as a constant. In other words, we don't have to update the slack time during the recursions.*

$$cost(c_i, scan_{in}) = \begin{cases} area(\text{MUX}) & \text{if } slack(c_i) > t_{mux} \\ \text{MIN}_{c_j \in fanin(c_i)}(cost(c_j, scan_{in}) + & \text{if } gate\_type(c_i) = \text{AND} \\ \quad \sum_{c_k \in fanin(c_i), c_k \neq c_j}(cost(c_k, 1))) & \\ \text{MIN}_{c_j \in fanin(c_i)}(cost(c_j, scan_{in}) + & \text{if } gate\_type(c_i) = \text{OR} \\ \quad \sum_{c_k \in fanin(c_i), c_k \neq c_j}(cost(c_k, 0))) & \\ cost(fanin(c_i), scan_{in}) & \text{if } gate\_type(c_i) = \text{INVERTER} \\ \infty & \text{if } gate\_type(c_i) = \text{FLIP-FLOP} \end{cases} \quad (2)$$

$$cost(c_i, 0) = \begin{cases} area(\text{AND}) & \text{if } slack(c_i) > t_{and} \\ \text{MIN}_{c_j \in fanin(c_i)}(cost(c_j, 0)) & \text{if } gate\_type(c_i) = \text{AND} \\ \sum_{c_j \in fanin(c_i)}(cost(c_j, 0)) & \text{if } gate\_type(c_i) = \text{OR} \\ cost(fanin(c_i), 1) & \text{if } gate\_type(c_i) = \text{INVERTER} \\ \infty & \text{if } gate\_type(c_i) = \text{FLIP-FLOP} \end{cases} \quad (3)$$

$$cost(c_i, 1) = \begin{cases} area(\text{OR}) & \text{if } slack(c_i) > t_{or} \\ \text{MIN}_{c_j \in fanin(c_i)}(cost(c_j, 1)) & \text{if } gate\_type(c_i) = \text{OR} \\ \sum_{c_j \in fanin(c_i)}(cost(c_j, 1)) & \text{if } gate\_type(c_i) = \text{AND} \\ cost(fanin(c_i), 0) & \text{if } gate\_type(c_i) = \text{INVERTER} \\ \infty & \text{if } gate\_type(c_i) = \text{FLIP-FLOP} \end{cases} \quad (4)$$

Fig. 5. The definition of $cost(c_i, value)$, where $value$ is $scan_{in}$, 1 or 0.

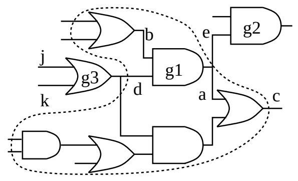


Fig. 7. Example of non-reconvergent fanin region.

The non-reconvergent fanin region of a connection c can be constructed in linear time in terms of its size by using breadth-first-traversal from the connection c toward the inputs.

B. Timing-driven partial scan algorithm

The overall algorithm integrates a conventional cycle-breaking algorithm [6] and our test point insertion algorithm. The cycling-breaking algorithm used here is originally from [6] and then modified by [7]. It consists of two major steps: (1) graph reduction and (2) heuristic selection. In the graph reduction step there are five operations. The first three (source operation, sink operation, self-loop operation) are exactly the same as the ones given in [6] while the last two reduction operations (unit-in operation and unit-out operation) are modified to take into account the slack times of the flip-flops. In the heuristic selection step, the algorithm chooses the one with maximal summation of the fanins and fanouts. For more details, please refer to [6, 7].

In our algorithm, we examine the topological structure of the given circuit and build the flip-flop connectivity graph excluding self-loops. Given a flip-flop *ff* selected by the cycle breaking procedure for scan, Equations 2,3,4 are performed to find a zero-performance degradation solution to scan *ff* in the non-reconvergent fanin region of *ff*. If such a solution exists, we always find it and return the set of test points. The algorithm then inserts appropriate MUX, AND or OR gates into the circuit and performs an incremental static timing analysis for the next run. If there exist no zero-performance degradation solutions, it returns NULL and the algorithm will mark this flip-flop and instruct cycle breaking procedure to choose another one. It continues until no cycles are left in the resulting graph or all flip-flops left have been marked. If there still exist cycles in G, we know there is no zero-performance degradation solution for this circuit. The algorithm then iteratively selects a flip-flop with minimal timing degradation using the equations similar to the ones described in Equations 2,3 and 4.

C. Experimental results

TABLE II
STATISTICS ON ISCAS89 AND MCNC91 BENCHMARK CIRCUITS AFTER DELAY OPTIMIZATION. CIRCUITS MARKED '*' ARE OPTIMIZED WITHOUT USING FULL_SIMPLIFY.

circuit	#I	#O	#FF	area	delay (ns)
s5378	35	49	163	4286.0	26.9
s9234	36	39	135	3619.0	29.5
s13207	31	121	453	8511.0	35.8
s15850	14	87	540	13442.0	54.7
s35932*	35	320	1728	40881.0	31.0
s38417*	28	106	1462	40611.0	42.4
s38584*	12	278	1449	36646.0	39.6
bigkey*	262	197	224	14461.0	27.8
dsip*	228	197	224	8288.0	23.1
mult32a*	33	1	32	1655.0	95.8
mult32b*	32	1	61	1505.0	12.2

We have implemented a prototype system, named TPTIME, based on the SIS-1.2 [15] package. The experimental results for a number of ISCAS89 and MCNC91 sequential benchmarks and the experimental setup are described as follows.

All the circuits are first optimized by SIS *script.delay* script and then mapped for minimal delay. Since we target the timing-driven partial scan design, it is more reasonable to optimize the original circuits for minimal delay. The longest delay of the optimized circuit is used as the circuit timing constraint. The technology libraries used for mapping are based on *nand-nor.genlib* and *mcnc_latch.genlib* from SIS-1.2 package. We choose *nand-nor.genlib* because the current implementation can only handle primitive gates. To facilitate test point insertion (adding AND, OR and MUX gates into the circuit), we appended three entries in the technology library in order to perform static timing analysis in SIS-1.2. Each library cell's *drive(g)* is set to 0.2 and the input capacitive load is set to 1. For example, inserting a multiplexer at a connection will decrease its slack time by 2.2, since its block delay is 2.0 and the extra 0.2 is due to the fanout of the multiplexer. The statistics of the SIS-1.2 optimized circuits are shown in Table II. Notice that the test input T might have many fanouts and consequently its large capacitive load would cause timing problems. Fortunately, in the mission (normal) mode, since the value of T is fixed to 1, the paths from T to test points or MUX are false paths. Therefore, we should disable the paths originating from T during the static timing analysis.

Three different experiments were performed for each optimized circuit. First, we ran the Lee-Reddy [6] cycle-breaking algorithm(CB) which does not take timing into

TABLE III
Experimental results on ISCAS89 and MCNC91 circuits for timing-driven partial scan design.

	CB					TD-CB					TPTIME					
circuit	#FF	area		delay		#FF	area		delay		#FF	area		delay		time (CPU)
s5378	29	4431.0	3.4%	29.0	7.8%	29	4431.0	3.4%	26.9	0.0%	29	4431.0	3.4%	26.9	0.0%	44.6s
s9234	24	3739.0	3.3%	31.6	7.1%	25	3744.0	3.5%	29.5	0.0%	24	3754.0	3.7%	29.5	0.0%	33.2s
s13207	41	8716.0	2.4%	38.0	6.1%	42	8721.0	2.5%	35.8	0.0%	42	8721.0	2.5%	35.8	0.0%	113.0s
s15850	91	13897.0	3.4%	56.9	4.0%	91	13897.0	3.4%	55.9	2.2%	91	13909.5	3.5%	54.7	0.0%	441.6s
s35932*	306	42411.0	3.7%	33.2	7.1%	306	42411.0	3.7%	31.0	0.0%	306	42411.0	3.7%	31.0	0.0%	11217.4s
s38417*	366	42441.0	4.5%	44.6	5.2%	388	42551.0	4.8%	44.6	5.2%	382	43351.0	6.7%	44.2	4.2%	8727.2s
s38584*	175	37521.0	2.4%	41.8	5.6%	233	37811.0	3.2%	41.4	4.5%	183	37808.5	3.2%	40.6	2.5%	3384.2s
bigkey*	112	15021.0	3.9%	30.0	7.9%	112	15021.0	3.9%	30.0	7.9%	112	15686.0	8.5%	28.7	3.2%	778.4s
dsip*	150	9038.0	9.0%	25.3	9.5%	180	9188.0	10.8%	25.3	9.5%	162	10555.5	27.4%	23.1	0.0%	688.6s
mult32a*	16	1735.0	4.8%	97.9	2.2%	17	1740.0	5.1%	97.9	2.2%	16	1740.0	5.1%	95.8	0.0%	13.8s
mult32b*	2	1515.0	0.6%	14.2	16.4%	22	1616.0	7.4%	14.2	16.4%	19	1647.5	9.5%	12.2	0.0%	14.9s

account. Second, we ran the timing-driven cycle-breaking (TD-CB) algorithm shown in [7]. Third, we ran our program (TPTIME). The results are shown in Table III. For each experiment, we report the number of selected flip-flops, and the area and delay of the resulting circuit. As we can see, without taking timing into account, the first method (CB) selected fewer flip-flops and had smaller area overhead, but all the tested circuits have timing degradation ranging from 2.2% to 16.4%. On the other hand, the timing-driven cycle-breaking algorithm (TD-CB) selected more flip-flops and had a larger area overhead, but the timing degradations for the tested circuits are smaller, ranging from 0.0% to 16.4%.

Our method (TPTIME) incorporates test point insertion technique to scan timing-critical flip-flops. Compared to CB, TPTIME has a larger area overhead due to the extra AND or OR gates. However, compared to TD-CB, since the number of selected flip-flops is smaller, the area overhead may be less. In term of timing degradation, our method TPTIME obtains the best results among the three methods. In most cases, there is no timing degradation at all.

V. Conclusion

In this paper, we propose a low-overhead scan design methodology which employs the test point insertion technique to establish scan paths through the functional logic. Applications for reducing either area or timing overhead are addressed and the experimental results demonstrate its usefulness.

Since the scan path is a part of the combinational logic, it is necessary to test the scan path prior to testing the entire circuit. This can be accomplished by scanning in a sequence of alternating 0's and 1's and scanning them out [8]. If there are some discrepancy between the scan-in and scan-out data, we know that the circuit is faulty. Moreover, by examining the scan-out data, certain faults (in the combinational logic) which affect the correctness of the scan chain can be tested before the application of scan tests.

Acknowledgements

This work was supported under NSF grant MIP 9419119 and MICRO/Fujitsu.

References

[1] M. J. Y. Williams and J. B. Angell, "Enhancing testability of large scale integrated circuits via test points and additional logic," *IEEE Trans. Computers*, vol. C-22, pp. 46–60, Jan. 1973.

[2] E. B. Eichelberger and T. W. Williams, "A logic design structure for LSI testability," *J. Design Automat. Fault-Tolerant Comput.*, vol. 2, pp. 165–178, May 1978.

[3] V. D. Agrawal, S. K. Jain, and D. M. Singer, "Automation in design for testability," *IEEE Custom Integrated Circuits Conference*, pp. 159–163, 1984.

[4] K.-T. Cheng and V. D. Agrawal, "A partial scan method for sequential circuits with feedback," *IEEE Trans. Computers*, vol. 39, pp. 544–548, Apr. 1990.

[5] R. Gupta, R. Gupta, and M. A. Breuer, "The ballast methodology for structural partial scan design," *IEEE Trans. Computers*, pp. 538–543, Apr. 1990.

[6] D. H. Lee and S. M. Reddy, "On determining scan flip-flops in partial-scan designs," *Proc. ICCAD*, pp. 322–325, 1990.

[7] J.-Y. Jou and K.-T. Cheng, "Timing-driven partial scan," *Proc. ICCAD*, pp. 404–407, 1991.

[8] C.-C. Lin, M. Marek-Sadowska, K.-T. Cheng, and M. T.-C. Lee, "Test-mode point insertion: Scan paths through combinational logic," Report UCSB-ECE-CAD-95-10, University of California, Santa Barbara, 1995.

[9] H. H. S. Gundlach and K. D. Muller-Glaser, "On automatic testpoint insertion in sequential circuits," *Int. Test Conf.*, pp. 1072–1079, 1990.

[10] S. M. Reddy and R. Dandapani, "Scan design using standard flip-flops," *IEEE Design and Test of Computers*, pp. 52–54, 1987.

[11] B. Vinnakota and N. K. Jha, "Synthesis of sequential circuits for parallel scan," *Proc. European Conf. Design Automation*, pp. 289–293, 1992.

[12] S. Bhatia and N. K. Jha, "Synthesis of sequential circuits for easy testability through performance-oriented parallel partial scan," *Proc. ICCD*, pp. 151–154, 1993.

[13] C.-C. Lin, M. T.-C. Lee, M. Marek-Sadowska, and K.-C. Chen, "Cost-free scan: A low-overhead scan path design methodology," *Proc. ICCAD*, pp. 528–533, 1995.

[14] H. Cox, "On synthesizing circuits with implicit testability constraints," *Int. Test Conf.*, pp. 989–998, 1994.

[15] "SIS: A system for sequential circuit synthesis," Report M92/41, University of California, Berkeley, 1992.