Partial Scan with Pre-selected Scan Signals *

Peichen Pan and C. L. Liu
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Abstract – A partial scan approach proposed recently selects scan signals without considering the availability of the flip-flops (FFs). Such an approach can greatly reduce the number of scan signals since maximum freedom is allowed in scan signal selection. To actually scan the selected signals, we, however, must make them FF-driving signals. In this paper, we study the problem of modifying and retiming a circuit to make a pre-selected set of scan signals FF-driving signals while preserving the set of cycles being broken. We present a new approach for solving this problem. Based on the new approach we design an efficient algorithm. Unlike a previous algorithm which inherently has no control over the area overhead incurred during the modification, our algorithm explicitly minimizes the area overhead. The algorithm has been implemented and encouraging results were obtained.

I. Introduction

Sequential circuits are hard to test mainly because state signals in the feedback loops are difficult to control and observe. Scan design has been proposed as a design-for-testability technique for sequential circuits [1]. Due to the high area and performance penalty associated with full scan, partial scan is often used in practice.

The main problem in partial scan design is to select a set of scan signals. Several methods have been proposed for this purpose [5, 6, 8, 11]. Among them, cycle-breaking has been demonstrated to be a very effective one [5, 6]. The goal of cycle-breaking is to select a small set of signals such that their removal will simplify the cycle structure of a circuit, or will break the cycles that cause many of the hard-to-detect faults. In conventional approaches, the positions of the FFs in a circuit are assumed to be fixed. Therefore, scan signals cannot be selected arbitrarily — they must be signals that drive FFs. In this case, the cycle structure of a circuit can be captured by a graph on the FFs, called *S-graph*. Breaking up the cycles in the circuit amounts to breaking up the cycles in the S-graph. Effective heuristics as well as exact algorithms have been proposed for selecting a minimum set of FFs to break up the cycles in an S-graph [2, 3, 9].

A recent approach to partial scan selects a best possible set of cycle breaking signals (under certain criteria) without considering the availability of the FFs. This allows maximum freedom for scan signal selection. Of course, to actually scan these pre-selected signals, each of them must be made to drive a FF. Recently, Chakradhar and Dey [4] showed that through a combination of retiming and logic replication, a desired FF configuration in which each pre-selected signal drives a FF can be obtained. They further proposed an iterative algorithm for doing so. Their algorithm, however, inherently has no control over the area overhead due to logic replication. Actually, it might introduce unnecessary replication. For example, because of the iterative nature of the algorithm some logic might be replicated more than once.

In this paper, we present a new approach to the problem of determining a FF configuration in which the pre-selected signals can actually be scanned. In our approach, the problem is transformed to that of determining a set of FF movements in the circuit, which, in turn, is reduced to a problem of breaking up the cycles in a special graph introduced in this paper. The FF movements can be carried out with possible logic replication to generate an equivalent circuit. It is guaranteed that the modified circuit has the *same* cycle structure as the original circuit and the pre-selected signals break the *same* set of cycles in both the original and the modified circuit. Since signals are made to drive FFs in a global way in our approach, it is guaranteed that no logic is replicated more than once. An algorithm based on the new approach is proposed. The algorithm explicitly minimizes the combinational and sequential area overheads incurred due to logic replication.

The remainder of this paper is organized as follows: In Section II we introduce some definitions and state the

*This work was partially supported by the NSF under grant MIP-9222408.

32nd ACM/IEEE Design Automation Conference ®

problem we want to solve. In Section III we present our new approach. In Section IV we present an algorithm based on the new approach. Section V shows experimental results. Finally, Section VI concludes this paper.

II. Preliminaries

A (synchronous) sequential circuit consists of combinational logic separated by FFs. FFs are assumed to be edge-triggered D-type flip-flops and are synchronized by a global clock. A sequential circuit can be modeled as an edge weighted directed (multi-)graph. The vertices are the primary inputs (PIs), the primary outputs (POs), and the logic gates. The edges are the interconnections. There is an edge from u (*head*) to v (*tail*) with weight t if the output of u, after passing through t FFs, is an input to v. We will use $w(e)$ to denote the weight of edge e. An edge corresponds to a signal — the signal passing through the corresponding interconnection in the circuit.

Retiming is the process of repositioning the FFs without modifying the structure of a circuit [10]. A retiming r of a circuit is a mapping from the vertices to the integers such that $r(v) = 0$ for a PI or PO v. $r(v) = t$ means that t FFs are removed from each outgoing edge of v, and t FFs are added to each incoming edge of v. Thus, in the circuit retimed by r, the weight of edge $e = (u, v)$, denoted $w_r(e)$, becomes $w(e) + r(v) - r(u)$. The functionality of the circuit will not be changed by retiming. However, to be physically realizable it is required that the edges in the retimed circuit have nonnegative weights. Such a retiming is called a *legal* retiming. For now on, when we say retiming, we always mean legal retiming.

In practice, the FFs on the outgoing edges of a vertex can be shared. As a result, the number of FFs on the outgoing edges of a vertex is the maximum of the numbers of FFs on the edges instead of their sum. Similarly, if we want to scan several outgoing edges of a vertex, only one scan FF is needed. In our graph model of a circuit, we do not explicitly model FF sharing because retiming can change the FF count on an edge. However, in counting the FFs as well as the scan FFs, we assume that sharing is employed implicitly.

Unlike retiming which does not modify the structure of a circuit, *duplicating*, on the other hand, is a process that structurally modifies a circuit while preserving its functionality. Informally, duplicating a vertex means to create a copy of the vertex and move some of the outgoing edges to the copy. Here we give the formal definition for duplicating a vertex v: Suppose the outgoing edges of v are divided into two groups, E_1 and E_2. A copy of v (which has the same functionality as v), denoted v', is added. For each edge $e = (x, v)$ a copy (x, v') with weight $w(e)$ is added. For each edge $(v, y) \in E_2$ it is changed to (v', y) with the weight unchanged. Figure 1 illustrates the process.

To *satisfy* an edge, we mean to position a FF on the edge. The problem addressed in this paper can be formally defined as follows: Given a circuit and a set of

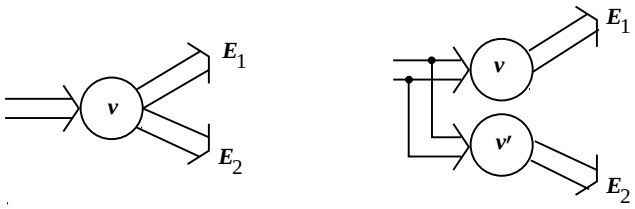

Figure 1: duplicating a vertex.

edges, $\mathcal{S}$, in the circuit, structurally modify the circuit using duplicating and retiming such that the edges in $\mathcal{S}$ are satisfied while the cycles broken by the edges in $\mathcal{S}$ are preserved. We call such a modified circuit an $\mathcal{S}$-*configuration* of the original circuit. The edges in $\mathcal{S}$ will be referred to as the $\mathcal{S}$-*edges*. Our primary objective is to find an $\mathcal{S}$-configuration with a small number of duplicated vertices. A secondary objective is to reduce the number of FFs in the $\mathcal{S}$-configuration.

In general, retiming alone is not able to satisfy all the $\mathcal{S}$-edges.

Theorem 1 *There is a retiming that can satisfy all the $\mathcal{S}$-edges iff (1) For each path from a PI to a PO, the number of $\mathcal{S}$-edges does not exceed the number of FFs, and (2) For each cycle, the number of $\mathcal{S}$-edges does not exceed the number of FFs.*

An $\mathcal{S}$-configuration cannot be obtained through retiming alone when some cycles and/or paths (from PIs to POs) contains too many $\mathcal{S}$-edges. However, as far as cycle-breaking is concerned, each cycle needs to contain only one satisfied $\mathcal{S}$-edge while a path from a PI to a PO does not need to contain any at all.

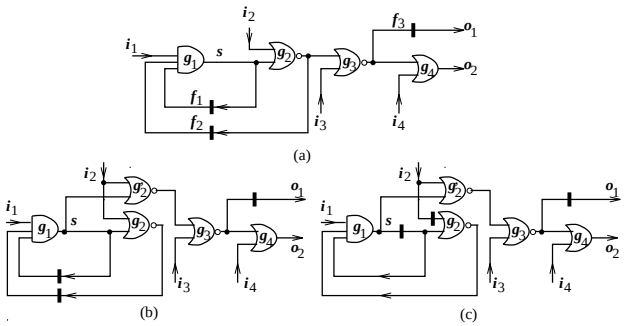

Figure 2: An example of using duplicating and retiming to obtain an $\mathcal{S}$-configuration

We now use an example to illustrate how duplication can be used to obtain an $\mathcal{S}$-configuration for a circuit violating the conditions in Theorem 1. Consider the circuit in Figure 2(a) with s being the only $\mathcal{S}$-edge. Here, s breaks two cycles: one containing f_1 and the other containing f_2. Retiming cannot position a FF on s. The

reason is that s is also on paths $g_1g_2g_3$ and $i_2g_2g_3$, and neither of them contains a FF. Notice that since these two paths are not in any cycle, there is no need to put a FF on s for these two paths. Therefore, we duplicate g_2 as shown in Figure 2(b). We then retime g_2 by 1 in the duplicated circuit as shown in Figure 2(c). Now there is a FF on s. Note that s cuts the same set of cycles in both circuits in Figure 2(a) and (c). Therefore, the circuit in Figure 2(c) is an $\mathcal{S}$-configuration of the circuit in Figure 2(a). As is evident from this example, duplication can facilitate retiming because a vertex and its copy can be retimed differently in the duplicated circuit.

In the next section, we will present a systematic way to identify a sub-circuit for duplication in order to obtain an $\mathcal{S}$-configuration of a circuit.

III. A New Approach

In this section, we propose a new approach to the problem of determining an $\mathcal{S}$-configuration of a circuit. Our algorithm presented in the next section is based on a refinement of the ideas presented here.

We view the $\mathcal{S}$-edges as demands and the FFs as resources. The ultimate question is how to utilize the resources to meet the demands. In our approach, we directly determine the FF movements in order to satisfy the $\mathcal{S}$-edges. First, we examine how to carry out a single FF movement. Suppose s is an $\mathcal{S}$-edge and f is a FF. We want to move f onto s. Let A_s and A_f denote the sets of cycles passing through s and f, respectively. Let $C_{s,f}$ denote the set of vertices on the paths from the head of s to the tail of the edge containing f. (See Figure 3(a).) To move f onto s, $C_{s,f}$ is duplicated as $C_{s,f}$ and $C'_{s,f}$, and then the vertices in $C_{s,f}$ are retimed by 1 as shown in Figure 3(b). Obviously, the set of cycles cut by the FF on s in Figure 3(b) is $A_{s,f} = A_s \cap A_f$. Therefore, the problem of determining an $\mathcal{S}$-configuration becomes that of selecting a set of FF movements such that $\cup A_{s,f}$ (over all selected FF movements) contains all the cycles cut by the edges in $\mathcal{S}$. The combinational area overhead is determined by $\cup C_{s,f}$. From Figure 3(b) it is also clear that the sequential area overhead (FFs) is determined by the vertices outside of $\cup C_{s,f}$ which connect to one or more vertices inside of $\cup C_{s,f}$.

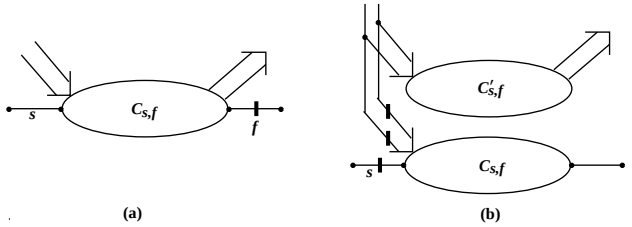

Figure 3: Using duplicating and retiming to carry out a single FF movement.

We now determine what kind of FF movements are good for reducing area overheads. Suppose there is a cycle that contains both s and f. Let p be the path from s to f in the cycle. If there is another $\mathcal{S}$-edge s_1 in p, obviously, $C_{s_1,f} \subseteq C_{s,f}$. To reduce combinational area overhead, it is desirable to move f onto s_1 instead of s. For a similar reason, if there is a FF f_1 on p, it is desirable to move f_1 instead of f onto s. In summary, f is moved onto s to cut a cycle only if there is no other $\mathcal{S}$-edges or FFs on the path from s to f in the cycle.

We define a directed graph called the *M-graph* which is introduced to identify (i) all possible FF movements, and (ii) all the cycles in $\mathcal{C}$ that are cut by the $\mathcal{S}$-edges. Let $\mathcal{F}$ denote the set of FFs in the circuit. The vertex set of the M-graph is $\mathcal{S} \cup \mathcal{F}$. There are three types of edges in the M-graph: from $\mathcal{S}$-edges to FFs, from $\mathcal{S}$-edges to $\mathcal{S}$-edges, and from FFs to $\mathcal{S}$-edges. The edges from $\mathcal{S}$-edges to FFs are referred to as *M-edges* which are introduced to denote the possible FF movements. There is an M-edge from $s \in \mathcal{S}$ to $f \in \mathcal{F}$ if there is a path in $\mathcal{C}$ from s to f containing no other element in $\mathcal{S} \cup \mathcal{F}$. All other edges are introduced to identify the cycles cut by the $\mathcal{S}$-edges in the circuit. There is an edge from $s_1 \in \mathcal{S}$ to $s_2 \in \mathcal{S}$ if there is a path in $\mathcal{C}$ from s_1 to s_2 containing no other element in $\mathcal{S} \cup \mathcal{F}$, and there is an edge from $f \in \mathcal{F}$ to $s \in \mathcal{S}$ if there is path in $\mathcal{C}$ from f to s containing no other element in $\mathcal{S}$. Note that there is no edge among the vertices in $\mathcal{F}$ in the M-graph. As an example, for the circuit in Figure 4(a), Figure 4(b) shows its M-graph.

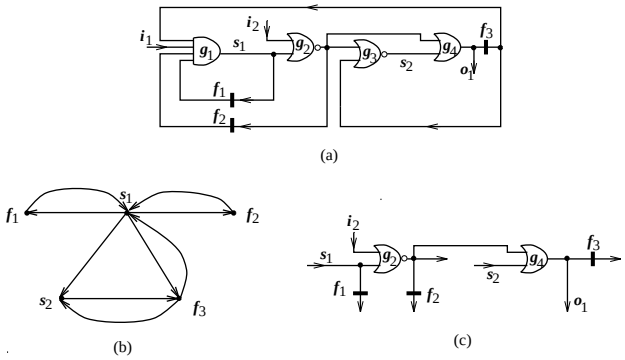

Figure 4: (a) A sequential circuit, (b) the corresponding M-graph, (c) the sub-circuit of a cover.

Lemma 2 *For each cycle containing one or more $\mathcal{S}$-edges in $\mathcal{C}$, there is a unique corresponding cycle containing at least one M-edge in the M-graph.*

As an example, for a cycle in $\mathcal{C}$ as shown in Figure 5(a), there is a corresponding cycle in the M-graph as shown in Figure 5(b). On the other hand, a cycle in $\mathcal{C}$ containing no $\mathcal{S}$-edge has no counterpart in the M-graph.

If FF f is to be moved onto $\mathcal{S}$-edge s, the logic between s and f may be duplicated so that the cycles and paths that contain s but not f can "bypass" all the FFs

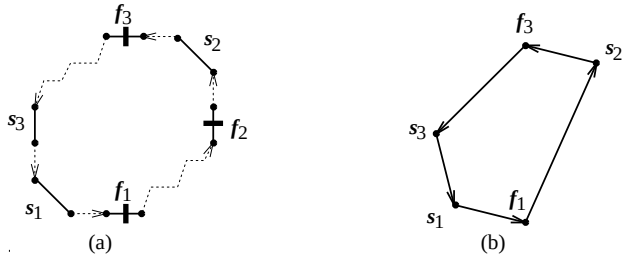

Figure 5: (a) Sketch of a cycle (dotted lines are paths containing no FF or $\mathcal{S}$-edge), (b) the corresponding cycle in the M-graph.

introduced by moving f onto s. (See Figure 3.) The *associated sub-circuit* of an M-edge (s, f) consists of all gates (together with their interconnections) on the paths from s to f which contains no other element in $\mathcal{S} \cup \mathcal{F}$ in the circuit. The inputs to the sub-circuit are the edges that come from a gate (or PI) not in the sub-circuit and the outputs are the edges that go to a gate (or PO) not in the sub-circuit. As a result, s is one of the inputs to the sub-circuit. As a convention, f is also included in the sub-circuit. The edge that f is on is, of course, an output of the sub-circuit. Similarly, we define the *associated sub-circuit* of a set of M-edges to be the union of the associated sub-circuits of all the M-edges in the set. Again, The inputs are the edges that come from a gate (or PI) not in the sub-circuit and the outputs are the edges that go to a gate (or PO) not in the sub-circuit.

A set of M-edges is called a *cover* if the removal of the edges in the set breaks all the cycles in the M-graph. In other words, a cover is a set of FF movements which, if carried out, guarantee that all the cycles passing through one or more $\mathcal{S}$-edges contain at least one satisfied $\mathcal{S}$-edge. For example, for the cycle in Figure 5(a), we can either move f_1 onto s_1 or move f_3 onto s_2. In terms of covers, either (s_1, f_1) or (s_2, f_3) must be included in a cover in order to break the corresponding cycle in the M-graph in Figure 5(b).

Let $\mathcal{M}$ be a cover and $\mathcal{C}_1$ be its associated sub-circuit. See Figure 6(a). We introduce a copy of $\mathcal{C}_1$, $\mathcal{C}_1'$, and move the edges without FFs to $\mathcal{C}_1'$ as shown in Figure 6(b). This circuit is, then, retimed by moving the FFs onto the input edges of $\mathcal{C}_1$ as shown in Figure 6(c). Let $\mathcal{C}_{\mathcal{M}}$ denote the final circuit in Figure 6(c). Note that the $\mathcal{S}$-edges that are inputs to $\mathcal{C}_1$ drive FFs in $\mathcal{C}_{\mathcal{M}}$.

Theorem 3 *$\mathcal{C}_{\mathcal{M}}$ is an $\mathcal{S}$-configuration of $\mathcal{C}$.*

As an example, for the circuit in Figure 2(a), its M-graph is shown in Figure 7(a) and edges (s, f_1) and (s, f_2) form a cover. Figure 7(b) shows the sub-circuit associated with the cover. The $\mathcal{S}$-configuration produced by our approach is exactly the one in Figure 2(c). Notice that our approach automatically detects that it is unnecessary to move f_3 onto s. As another example, for

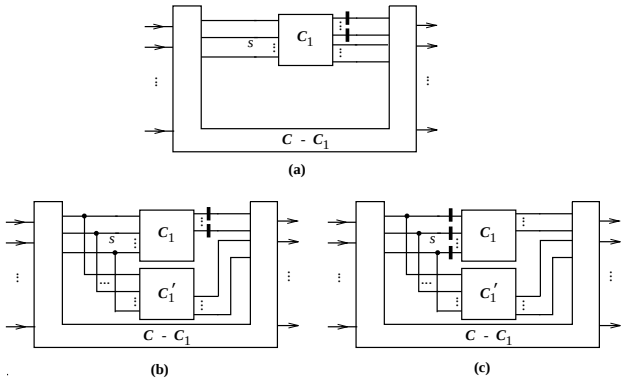

Figure 6: (a) Circuit $\mathcal{C}$, (b) $\mathcal{C}$ after duplication, (c) $\mathcal{C}_{\mathcal{M}}$.

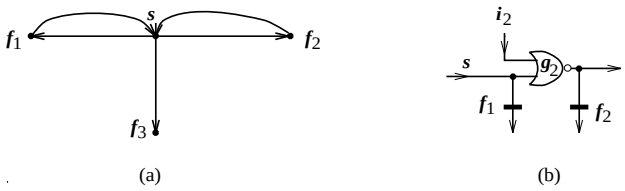

Figure 7: (a) M-graph, (b) sub-circuit associated with a cover.

the circuit in Figure 4(a), the only cover in the M-graph in Figure 4(b) is the set consisting of all M-edges. Figure 4(c) shows the sub-circuit associated with this cover and Figure 8 shows the corresponding $\mathcal{S}$-configuration.

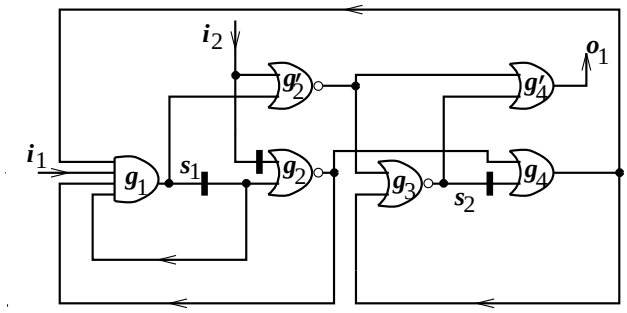

Figure 8: $\mathcal{S}$-configuration of the circuit in Figure 4.

IV. An Algorithm

We now present an algorithm for determining an $\mathcal{S}$-configuration based on finding a cover with small associated sub-circuit. Figure 9 is a high-level description of the algorithm. The main steps are described as follows:

Input: A circuit $\mathcal{C}$ and a set of edges $\mathcal{S}$.
Output: An $\mathcal{S}$-configuration of $\mathcal{C}$.
1. Determine an initial FF configuration by retiming the circuit twice;
2. Compute a cover with small associated sub-circuit in the M-graph of the retimed circuit;
3. Simultaneously identify the logic to be duplicated and minimize the number of FFs in the resultant $\mathcal{S}$-configuration by another retiming;
4. Generate the $\mathcal{S}$-configuration.

Figure 9: Algorithm for determining an $\mathcal{S}$-configuration.

Determine an initial FF configuration: Different initial FF configurations result in different M-graphs. The initial FF configuration also affects the sub-circuits associated with the covers. Intuitively, the closer the FFs to the $\mathcal{S}$-edges, the smaller the sub-circuits associated with the covers will be. We determine an initial FF configuration by retiming $\mathcal{C}$ twice. The purpose of the first retiming is to satisfy as many $\mathcal{S}$-edges as possible. The purpose of the second retiming is to move the FFs toward the unsatisfied $\mathcal{S}$-edges as much as possible while preserving the satisfied ones.

Compute a cover in the M-graph: To determine a cover in an M-graph, we construct another directed graph called *SM-graph*. The vertices in the SM-graph are the M-edges. There is an edge from an M-edge u to an M-edge v in the SM-graph if there is a path containing no other M-edge from u to v in the M-graph. It can be easily verified that a cover in the M-graph corresponds to a feedback vertex set in the corresponding MS-graph and vice versa. To solve the problem optimally is costly since the minimum feedback vertex set problem (which is known to be NP-hard) is obviously a special case of this problem. We, therefore, resort to heuristics. Our heuristic algorithm is similar to many of the heuristics used for determining a minimum feed-back vertex set.

Simultaneously identify the logic to be duplicated and minimize the number of FFs: In this step, we determine a retiming of $\mathcal{C}$ aiming at minimizing the number of FFs in the resultant $\mathcal{S}$-configuration and possibly further reducing logic duplication. Refer to Figure 6(c). The FFs on the outputs of $\mathcal{C}_1$ are moved onto the inputs of $\mathcal{C}_1$ in the $\mathcal{S}$-configuration. Therefore, in order to minimize the number of FFs in the $\mathcal{S}$-configuration, we only need to minimize the number of FFs in the remaining part of $\mathcal{C}$. In the meantime, the FFs currently on the outputs of $\mathcal{C}_1$ are retained. This retiming can, again, be determined by solving an integer linear program which is the dual of a min-cost flow problem. Let r_3 be the retiming. r_3 is applied to $\mathcal{C}$.

V. Experimental Results

The algorithm proposed in this paper was implemented in C. To test our algorithm, we also implemented an algorithm for selecting the scan FFs called **SELECT_FF**, and an algorithm for selecting the scan signals called **SELECT_SIGNAL**

We first briefly describe the algorithms used in **SELECT_FF** and **SELECT_SIGNAL**. **SELECT_FF** implements the heuristic for cutting the cycles except self-loops in an S-graph described in [9]. When no graph transformation can be applied, the vertex with the maximum sum of in-degree and out-degree is picked. **SELECT_SIGNAL** implements a heuristic for selecting a set of edges to cut the cycles in a circuit. It is almost the same as the heuristic in **SELECT_FF**. The difference is that in **SELECT_SIGNAL**, the algorithm stops if the S-graph of the remainder of the circuit contains no cycle except self-loops.

The three programs are integrated into one package, which will be referred to as Pspss. The input to Pspss is a circuit in *blif* format [12]. The outputs are the the $\mathcal{S}$-configuration (again in *blif* format) determined by our algorithm as well as the scan FFs of the $\mathcal{S}$-configuration determined by **SELECT_FF**.

We tested Pspss on several LSW'91 sequential multi-level benchmark circuits [12]. The results are summarized in Table I. In Table I, under the column **initial** we list the number of literals (lits), the number of FFs (FF), and the optimal number of scan FFs (scan) quoted from [3] for each initial circuit; under the column **final**, we list the corresponding values for the $\mathcal{S}$-configuration produced by Pspss. Note that for the final circuits, the numbers of scan FFs are determined by **SELECT_FF**, which is a heuristic. The column **time** lists the total computation time of the integrated package for each circuit on a SPARCstation 10. Overall, our method was able to reduce the number of scan FFs significantly with moderate increase in area as measured by the number of literals and the number of FFs. As an example, for circuit s9234, the initial circuit has 7971 literals, 211 FFs, and 53 scan FFs (optimal), Pspss produced an equivalent circuit with 8022 literals, 226 FFs, and 32 scan FFs (**SELECT_FF**). As another example, for circuit s38417, the initial circuit has 32246 literals, 1636 FFs, and 374 scan FFs (optimal), Pspss produced an equivalent circuit with 32250 literals, 1618 FFs, and 135 scan FFs (**SELECT_FF**).

Table II compares the increases in literals and FFs in the equivalent circuits produced by the algorithm in [4] (Props) with that produced by our package (Pspss) for four large benchmark circuits . As can be seen, the area increases for the equivalent circuits produced by our method are much less. For example, for circuit s9234, in the equivalent circuit produced by Props the numbers of literals and FFs are increased by 207 and 48, respectively. However, in the circuit produced by Pspss the respective increases are 51 and 15. It should be noted that Props and Pspss may use different scan signals for s9234 and s38417.

TABLE I
EXPERIMENTAL RESULTS

circuit	Initial			Final			time
	lits	FF	scan	lits	FF	scan	sec.
s344	269	15	5	269	17	4	0.4
s349	273	15	5	273	17	4	0.4
s382	306	21	9	306	22	3	0.5
s400	352	21	9	352	22	3	0.6
s444	320	21	9	320	22	3	0.5
s1423	1164	74	21	1164	86	8	5.4
s9234	7971	211	53	8022	226	32	158.6
s13207*	11234	638	58	11333	489	35	233.0
s15850*	13658	534	88	13767	572	41	374.6
s38417	32246	1636	374	32250	1618	135	2680.6

*For this circuit we use the scan signals provided by Chakradhar and Dey.

TABLE II
COMPARISON BETWEEN PROPS [4] AND PSPSS

circuit	lits incr.		FF incr.	
	PROPS	PSPSS	PROPS	PSPSS
s9234	207	51	48	15
s13207	206	99	-51	-149
s15850	253	109	35	38
s38417	31	4	-80	-18

VI. CONCLUSIONS

In this paper, we study the problem of structurally modifying a circuit to make a pre-selected set of signals FF-driving signals so that they can be scanned while preserving the set of cycles being broken. We present a new approach for solving this problem. Based on the new approach we design an efficient algorithm. Our algorithm explicitly minimizes the area overhead. The algorithm has been implemented and encouraging results were obtained.

ACKNOWLEDGMENT

We thank Drs. Chakradhar and Dey of NEC USA for providing their scan signals for some benchmark circuits. We thank Prof. Goldberg at Stanford University for permitting us to use his min-cost flow program.

References

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

[2] P. Ashar and S. Malik, "Implicit computation of minimum-cost feedback-vertex sets for partial scan and other applications," in *Proc. 31st ACM/IEEE Design Automation Conf.,* 1994, pp. 77–80.

[3] S. T. Chakradhar, A. Balakrishnan, and V. D. Agrawal, "An exact algorithm for determining partial scan flip-flops," in *Proc. 31st ACM/IEEE Design Automation Conf.,* 1994, pp. 81–86.

[4] S. T. Chakradhar and S. Dey, "Resynthesis and retiming for optimum partial scan," in *Proc. 31st ACM/IEEE Design Automation Conf.,* 1994, pp. 87–93.

[5] K. T. Cheng and V. D. Agrawal, "A partial scan method for sequential circuits with feedback," in *IEEE Trans. on Computers*, vol. 39, no. 4, pp. 544-548, 1990.

[6] V. Chickermane and J. H. Patel, "A fault oriented partial scan design approach," in *Digest Int'l. Conf. on Computer-Aided Design,* 1991, pp. 400-403.

[7] D. Kagaris and S. Tragoudas, "Partial scan with retiming," in *Proc. 30th ACM/IEEE Design Automation Conf.,* 1993, pp. 249-254.

[8] K. S. Kim and C. R. Kime, "Partial scan by use of empirical testability," in *Digest Int'l. Conf. on Computer-Aided Design,* 1990, pp. 314-317.

[9] D. H. Lee and S. M. Reddy, "On determining scan flip-flops in partial-scan designs," in *Digest Int'l. Conf. on Computer-Aided Design,* 1990, pp. 322-325.

[10] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. 3rd Caltech Conf. on VLSI,* 1983, pp. 87-116.

[11] P. S. Parikh and M. Abramovici, "A cost-based approach to partial scan," in *Proc. 30th ACM/IEEE Design Automation Conf.,* 1993, pp. 255-259.

[12] S. Yang, "Logic synthesis and optimization benchmarks user guide, version 3.0", *MCNC Technical Report*, Jan, 1991.