AN EXACT ALGORITHM FOR SELECTING PARTIAL SCAN FLIP-FLOPS

Srimat T. Chakradhar
C & C Research Laboratories, NEC USA, 4 Independence Way, Princeton, NJ 08540
Arun Balakrishnan
RUTCOR, PO Box 5062, Rutgers University, New Brunswick, NJ 08903
Vishwani D. Agrawal
AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

ABSTRACT: We develop an exact algorithm for selecting flip-flops in partial scan designs to break all feedback cycles. The main ideas that allow us to solve this hard problem exactly for large, practical instances are - graph transformations, a partitioning scheme used in the branch and bound procedure, and pruning techniques based on an integer linear programming formulation of the minimum feedback vertex set (MFVS) problem. We have obtained optimum solutions for the ISCAS '89 benchmark circuits and several production VLSI circuits within reasonable computation time. For example, the optimal number of scan flip-flops required to eliminate all cycles except self-loops in the circuit s38417 is 374. This optimal solution was obtained in 32 CPU seconds on a SUN Sparc 2 workstation.

1. INTRODUCTION

Scan design is a widely used *design for testability* technique. In this technique, memory elements (flip-flops) are chained into a shift register (*scan chain*) during the test mode and hence, can be directly controlled and observed. In the *full scan design*, all flip-flops are included in the scan chain and so combinational test generation methods are sufficient. However, the full scan design often entails unacceptable penalties of area overhead and performance degradation. A practical alternative is to select a subset of the flip-flops for inclusion in the scan chain (*partial scan*). Thus, both area overhead and performance degradation can be significantly reduced, but the *partial scan* circuit requires the use of sequential test generation methods.

Existing approaches for selecting flip-flops for partial scan can be classified as testability analysis based [1, 2], test generation based [3, 4, 5, 6] and structural analysis based [7]. Cheng and Agrawal [7] observed that the feedback cycles among flip-flops are mainly responsible for test generation complexity. Further, they empirically observed that partial scan circuits with self-loops but no longer feedback cycles are easier to test than circuits that have the longer feedback cycles. A self-loop here refers to a situation where the output of a flip-flop, after passing through combinational logic, feeds back into the same flip-flop. Allowing self-loops in the partial scan circuit is particularly attractive since a large number of flip-flops in real designs have self-loops. Therefore, the scan overhead will be high if we had to break *all* feedbacks.

The approach proposed by Cheng and Agrawal [7] translates to a graph problem. Let $v_1, \ldots, v_n$ be the set of flip-flops in the circuit. The structural dependencies among flip-flops can be represented by a directed graph, called the S-graph. This graph has as many vertices as the number of flip-flops in the circuit. There exists an arc from vertex v_i to vertex v_j if there is a combinational path from flip-flop v_i to flip-flop v_j. Also, there is an arc from vertex v_i to itself (self-loop) if there is a combinational path from flip-flop v_i to itself. The problem of selecting flip-flops to break all feedback cycles is now equivalent to the problem of finding a set of vertices whose removal makes the S-graph acyclic. This is referred to as the *minimum feedback vertex set* (MFVS) problem. If we wish to allow self-loops in the partial scan circuit, then we must remove the self-loop arcs in the S-graph before computing the MFVS.

The MFVS problem belongs to the class of NP-hard problems. Several exact and heuristic methods have been proposed for this problem [8, 9, 10, 11, 12, 13, 14]. In this paper, we present a new exact algorithm. Experimental results in Section 6 show that our algorithm can compute the MFVS for S-graphs obtained from large sequential circuits in the ISCAS '89 benchmark set and several production VLSI circuits. This is the first time that optimal results are being reported for these circuits. Moreover, our algorithm uses less or comparable computing resources than the available heuristic methods.

This work involves several new ideas. We develop an MFVS-preserving graph transformation that defines a new class of graphs for which the MFVS problem can be solved in polynomial time complexity. Then, we propose a partitioning scheme in the branch and bound search procedure. This is essentially a new branching strategy. Finally, we propose a novel integer linear programming formulation for the MFVS problem. Such a formulation can be used directly to solve the MFVS problem or to find lower bounds on the cardinality of the MFVS. These lower bounds are useful in pruning the branch and bound search tree.

2. GRAPH TRANSFORMATIONS

Let V and A be the vertex and arc sets, respectively, of the S-graph. Also, let $n = |V|$ and $m = |A|$. For any arc $(v_i \rightarrow v_j)$, v_i is a predecessor of v_j and v_j is a successor of v_i. Let $remove(v_i)$ denote the process of removing all incoming and outgoing arcs of vertex v_i. Let $ignore(v_i)$ denote the following process: connect each predecessor of v_i to all its successors, $remove(v_i)$ and collapse multiple arcs (if any) into a single arc. If v_j is both a predeces-

sor and successor to v_i then a self-loop arc is created. The following three transformations are known to be MFVS-preserving [15]:

- **T1:** If v_i has a self-loop then $remove(v_i)$ and return v_i. A MFVS for the S-graph is obtained by adding v_i to any MFVS of the modified graph.
- **T2:** If v_i has either indegree or outdegree equal to 0 then $remove(v_i)$. The MFVS's of the S-graph and the modified graph are identical.
- **T3:** If v_i has either indegree or outdegree equal to 1 but no self-loop then $ignore(v_i)$. Any MFVS of the modified graph is a MFVS for the S-graph.

Each of the above transformations yields a modified graph with one less vertex. Starting from the S-graph, we can repeatedly apply transformations **T1, T2** and **T3** until no more transformations are applicable. The vertices returned by transformation **T1** during this process must be added to the MFVS of the final graph to obtain the MFVS of the given S-graph. The final graph resulting from such a process is unique irrespective of the order in which the transformations are applied [15]. Hence, the process of repeatedly applying the transformations **T1, T2** and **T3** is well defined and we refer to it as procedure $compress\_graph$. This procedure can be implemented to run in $O(m \log n)$ time as suggested in [16]. In particular, if $compress\_graph$ reduces the S-graph to an empty graph then the MFVS is determined in polynomial time complexity. In [15], the class of graphs that reduce to an empty graph after applying $compress\_graph$ was called the *two-way reducible* class.

We propose a MFVS-preserving transformation that can reduce S-graphs beyond what is possible using $compress\_graph$. There are two advantages of using our transformation. First, our transformation defines a new polynomial time solvable class of graphs that is strictly larger than the two-way reducible class and hence, we can determine the MFVS of some S-graphs which are not two-way reducible in polynomial time. Second, for arbitrary S-graphs, this transformation will give a final graph with fewer vertices than the final graph obtained through $compress\_graph$. Therefore, fewer vertices have to be considered in the branch and bound search.

Consider the graph S shown in Figure 1. Suppose this is the S-graph of a sequential circuit with thirteen flip-flops $x_1, \ldots, x_{13}$. First, we reduce S by using the procedure $compress\_graph$. Ver-

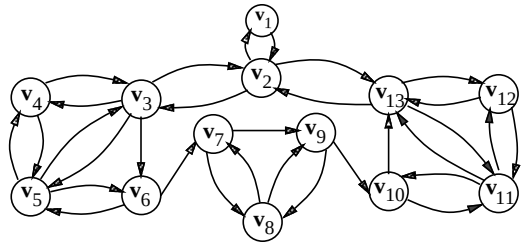


Figure 1: Example S-graph S.

tex v_1 is removed by transformation **T3**. This results in a self-loop on v_2. Vertex v_2 is then removed using transformation **T1**. The transformed graph S' is shown in Figure 2. None of the transformations are applicable at this point. Since S' is not empty, our S-graph does not belong to the two-way reducible class. However, the transformed graph S' can be further reduced based on the following observation:

Observation 1: It suffices to consider the strongly connected components (scc's) of the graph for finding the MFVS [8].

The graph S' has the following strongly connected components: $\{v_3, v_4, v_5, v_6\}$, $\{v_7, v_8, v_9\}$ and $\{v_{10}, v_{11}, v_{12}, v_{13}\}$. We extract the

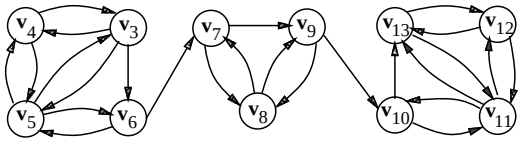


Figure 2: Example S-graph reduced by $compress\_graph(S)$.

strongly connected components by deleting arcs $(v_i \rightarrow v_j)$, where v_i and v_j belong to different scc's. The resulting graph S'' shown in Figure 3 can now be reduced to an empty graph by performing

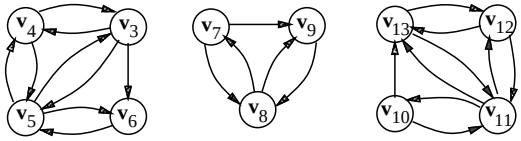


Figure 3: SCC's of graph in Figure 2.

the $compress\_graph(S'')$. For example, consider the component $\{v_3, v_4, v_5, v_6\}$ of S''. The sequence of transformations for this component is shown in Figure 4. Similarly, the other two scc's are also reduced to empty graphs.

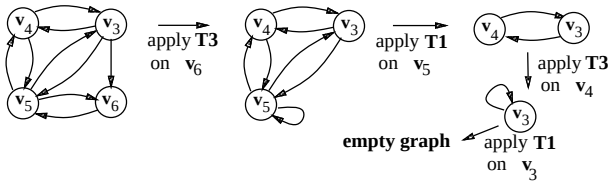


Figure 4: Reducing SCC $\{v_3, v_4, v_5, v_6\}$.

The following procedure reduces a graph by repeated use of Observation 1.

Procedure **transform_graph(S)**
do {
$extract\_scc(S)$;
$mfvs\_list = mfvs\_list + compress\_graph(S)$;
} **while** (S is modified and S is not empty)
return(S and $mfvs\_list$);

The procedure $extract\_scc$ computes the strongly connected components of a graph and deletes arcs $(v_i \rightarrow v_j)$ where v_i and v_j belong to different scc's. Consider the application of procedure $transform\_graph$ to the example S-graph of Figure 1. Since S is strongly connected, procedure $extract\_scc(S)$ does not modify S. Next, we perform $compress\_graph(S)$, to obtain the graph shown in Figure 2. Since the procedure $compress\_graph$ modifies the graph S, we recompute the strongly connected components of the modified graph. Now procedure $extract\_scc(S)$ modifies S into the graph shown in Figure 3. Finally, $compress\_graph(S)$ reduces the graph shown in Figure 3 to the empty graph.

The $transform\_graph$ procedure can be implemented to run in time complexity $O(mn \log n)$. Observe that the procedure

$transform\_graph$ reduces the example S-graph, that did not belong to the two-way reducible class, to an empty graph. Consequently, we have defined a new class of graphs, strictly larger than the two-way reducible class, for which the MFVS can be computed in polynomial-time complexity. We call this class *scc compressible*. If the S-graph of a sequential circuit is not scc-compressible then $transform\_graph(S)$ will reduce it to a final graph containing one or more scc's. An scc that cannot be further reduced by the procedure $compress\_graph$ is called a *compressed scc*.

3. PARTITIONED BRANCH AND BOUND

We illustrate the partitioning scheme used in the branch and bound procedure with the graph S shown in Figure 5. This graph has seven vertices labeled $v_1, \ldots, v_7$. We note that performing $transform\_graph(S)$ does not reduce it any further. We begin by associating a Boolean variable x_i to each vertex v_i in the graph. For any assignment of 0-1 values to the Boolean variables, we can construct a vertex set that includes only those vertices for which the corresponding Boolean variables assume the value 1. The 0-1 assignments that correspond to feedback vertex sets are called *feasible* solutions and the rest are infeasible solutions. The feasible solutions include the optimum solutions corresponding to the MFVS's. The branch and bound procedure systematically searches the space of all 0-1 vectors for an optimum solution.

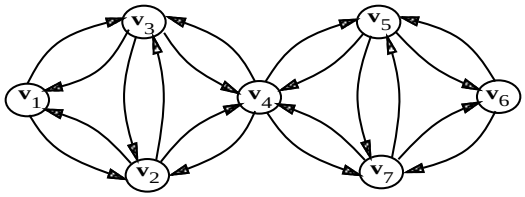


Figure 5: A compressed scc.

We begin the search with all Boolean variables unassigned. We define the *index set* to be a list of subscripts of all unassigned variables. The index set corresponding to the initial search space here is $N^0 = \{1, 2, 3, 4, 5, 6, 7\}$. We pick x_4 as our first decision variable and let x_4 = 1. This implies that we wish to include vertex v_4 in the feedback vertex set and hence, we remove this vertex from S. The modified graph S' is shown in Figure 6. Next, we try to

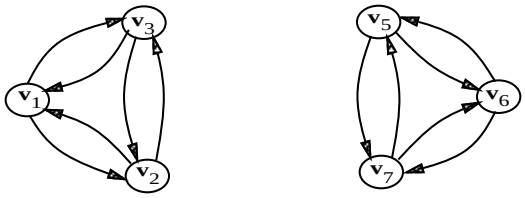


Figure 6: Graph after removing v_4.

reduce the graph S' by doing $transform\_graph(S')$, but it does not reduce any further. Now the index set corresponding to search space of S' is $N^1 = \{1, 2, 3, 5, 6, 7\}$. Conventionally, we may explore this search space as shown in Figure 7. However, the MFVS of S' can be found by solving the two strongly connected components $\{v_1, v_2, v_3\}$ and $\{v_5, v_6, v_7\}$, independently (Observation 1). Hence, we can partition the index set N^1 into {1,2,3} and {5,6,7} and independently explore the corresponding search spaces. This situation is shown in Figure 8. Partitioning the search space of N^1 has lead to an efficient search. Observe that the number of nodes in

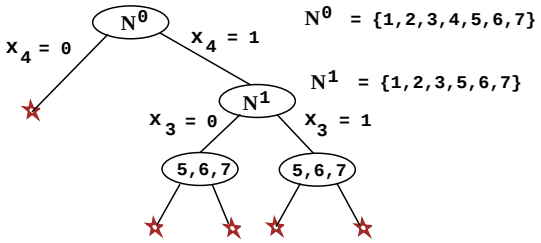


Figure 7: Conventional branch and bound.

the search tree of Figure 8 is 3 whereas the number of nodes in the search tree of Figure 7 is 4. In general, we can expect the search tree for the partitioned branch and bound to be much smaller than the search tree for the conventional branch and bound. Note the recursive use of Observation 1 within branch and bound search.

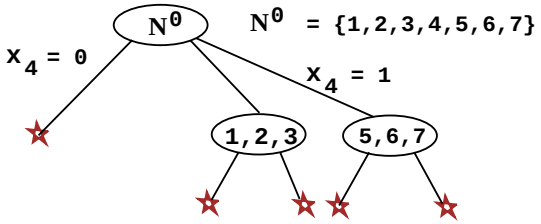


Figure 8: Partitioned branch and bound.

Any branch and bound procedure can be expressed as a sequence of moves through the search spaces. Let $S(N^0)$ denote the initial space of (Boolean) vectors and $S(N^i)$ for $i \geq 1$ denote the subsequent search spaces where N^0 and N^i are the index sets. The complexity of exploring a subspace is exponential in the size of its index set. In conventional branch and bound search, the search space is halved by fixing the value of one decision variable, since $|N^{i+1}| = |N^i| - 1$. For example, in Figure 7, $|N^1| = |N^0| - 1$. Sometimes, we could have $|N^{i+1}| = |N^i| - k$ for some small value of k. This can happen if fixing one variable *implicates* a few others. A variable x_i implicates a variable x_j if inclusion or exclusion of v_i from the MFVS forces us to either include or exclude v_j from the MFVS. Here every unit increase in the value of k reduces the subsequent search space by a factor of two.

In the partitioned search strategy, we partition the index set of the current search space into one or more pieces and proceed to solve them, independently. This results in the reduction of the search space by a large factor. For example, the size of the search space corresponding to the index set N^1 in the conventional branch and bound is $2^6 = 64$ whereas it is $2^3 + 2^3 = 16$ in the partitioned branch and bound. This can lead to a substantial reduction in the overall complexity of the search procedure since we apply this strategy recursively. Hence, the search is likely to be much faster. Here, the partitions are inherent as in the case of N^1. They depend on the compressed scc's present in the current graph. Since the search could branch multiway after each decision depending on the number of compressed scc's, we refer to this as the *multiway branch and bound* algorithm. We present the pseudo-code and other details involved in the implementation of this algorithm in Section 5.

4. PRUNING STRATEGIES

We need to compute good lower bounds on the cardinality of the MFVS of a graph to effectively prune our search. First, we show that the lower bound for any compressed scc is *two*. This

can be used as the trivial lower bound in the branch and bound procedure. We then formulate the MFVS problem as an *integer linear program* (ILP) [17]. This formulation can be used in several ways for computing lower bounds.

Claim 1: For any compressed scc S, $|MFVS(S)| \geq 2$.

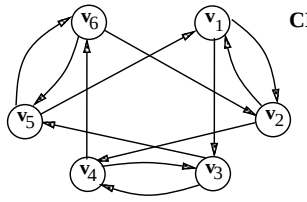
Proof: Let S be the given compressed scc. By contradiction, let us suppose that the size of optimum solution is 1. Let that vertex be *u*. If we remove *u* from S then the resulting graph must be acyclic. Clearly this graph has a vertex with in-degree 0. Let that vertex be *v*. This implies that the in-degree of *v* in S must have been ≤ 1 (the only possible arc being $u \rightarrow v$). This contradicts the assumption that S is compressed. ∎

4.1. ILP Formulation

Let $W = (w_1, w_2, \ldots, w_n)$ be *weights* associated with the vertices $v_1 \ldots v_n$ of the graph. We start with the following requirement: for every arc $(v_i \rightarrow v_j)$, we require that $w_i - w_j \geq 1$. By transitivity, this implies that, along any path in the graph, the weights assigned to its vertices must decrease. Clearly, the above requirement cannot be satisfied for all the arcs of a cycle. We will use this infeasibility to identify the vertex feedback sets of a graph. Consider an arc $(v_i \rightarrow v_j)$ that is part of some cycle. Let the weight of vertex v_j be n. Starting from vertex v_j, if we decrease the weights of vertices along the cycle by one unit, then the weight of vertex v_i cannot be less than 1. This is because the longest cycle can have at most n arcs. Therefore, if we add n to the weight of vertex v_i, then $(w_i + n) - w_j \geq 1$. In general, it is possible to satisfy the requirement $w_i - w_j \geq 1$ for all arcs on a cycle by adding n to the weight of any one vertex on the cycle. This requirement can be expressed as follows: for every $(v_i \rightarrow v_j)$ we require that $w_i - w_j + nx_i \geq 1$. Here, x_i is a Boolean variable associated with vertex v_i. If the weight of vertex v_i on a cycle has to be augmented by n, then the Boolean variable x_i assumes the value 1. Otherwise, its value is 0. To find a MFVS, it suffices to minimize $\sum x_i$. The ILP formulation for the MFVS problem is as follows:

Minimize $\sum x_i$
Subject to: $w_i - w_j + nx_i \geq 1 \; \forall \; (v_i \rightarrow v_j) \in A$
where $0 \leq w_i \leq n-1$ and x_i are Boolean.

As an example, consider the graph shown in Figure 9. This is a compressed scc. Let $W = (w_1, \ldots, w_6)$ and $X = (x_1, \ldots, x_6)$. The ILP formulation for this graph is as follows:



CHORDLESS CYCLES
C_1 = {1,2}
C_2 = {3,4}
C_3 = {5,6}
C_4 = {1,3,5}
C_5 = {2,4,6}

Figure 9: Example to illustrate ILP formulation.

Minimize $x_1 + x_2 + x_3 + x_4 + x_5 + x_6$
Subject to:

$$w_1 - w_2 + 6x_1 \geq 1 \quad w_2 - w_1 + 6x_2 \geq 1$$
$$w_1 - w_3 + 6x_1 \geq 1 \quad w_5 - w_1 + 6x_5 \geq 1$$
$$w_2 - w_4 + 6x_2 \geq 1 \quad w_6 - w_2 + 6x_6 \geq 1$$
$$w_3 - w_4 + 6x_3 \geq 1 \quad w_3 - w_5 + 6x_3 \geq 1$$
$$w_4 - w_3 + 6x_4 \geq 1 \quad w_4 - w_6 + 6x_4 \geq 1$$
$$w_5 - w_6 + 6x_5 \geq 1 \quad w_6 - w_2 + 6x_6 \geq 1$$
$$0 \leq w_i \leq 5, \quad x_i \text{ are Boolean.}$$

An optimum solution to the ILP corresponds to an MFVS of the graph. In the present example, the solution $X = (0, 1, 1, 0, 0, 1)$ corresponds to the MFVS $\{v_2, v_3, v_6\}$. Observe that the graph has many MFVS.

Claim 2: There exists a one-to-one correspondence between the feedback vertex sets of the graph and the feasible solutions to X of the above formulation.

Proof: Given F, a feedback vertex set of the graph, remove all outgoing arcs of the vertices in F and delete the inequalities corresponding to these arcs. Since the graph is acyclic now, the resulting formulation has a feasible solution for (W, X) with $X = 0$. Let $x_i = 1$ for all the vertices in F and we get a feasible solution corresponding to F. Alternatively, let (W^1, X^1) be a feasible solution to the formulation. Let $F = \{i : x_i = 1\}$. We show by contradiction that F is a feedback vertex set. Suppose it is not, then the graph obtained by removing all vertices in F has a cycle C. If we add inequalities corresponding to all arcs in the cycle C, then $\sum_{v_i \in C} x_i \geq k/n$ (k is the length of the cycle). However, this is a contradiction because x_i on the cycle have a value 0. ∎

Claim 3: The optimum solution to the above formulation gives the MFVS.

Proof: Let X^* be the given optimum solution. From the previous claim it is clear that there exists a feedback vertex set, say F^*, corresponding to X^*. Suppose F' is another feedback vertex set with fewer vertices. Then, again by previous claim, we get a feasible solution X' corresponding to F' with smaller objective value than X^*. This contradicts the assumption that X^* is optimum. ∎

The linear programming (LP) relaxation to this formulation is one way to determine lower bounds on the cardinality of the MFVS. Unfortunately, the lower bounds returned by the relaxation are weak. In the relaxation, we allow $0 \leq x_i \leq 1$ and hence, $w_i = 0 \; \forall \; i$ and $x_i = 1/n \; \forall \; i$ is a feasible solution. This implies that the optimum value returned by the LP relaxation will always be ≤ 1. This is especially weak considering that we have already shown a lower bound of *two* for compressed scc's. We can enhance the above formulation by adding more constraints like facets [17] derived from the graph or cuts derived from the above formulation.

One such constraint is the cycle constraint. For every cycle C in the graph, we can add an inequality $\sum_{v_i \in C} x_i \geq 1$ to the above formulation. Since two-cycles are easy to identify, one can add all the two-cycle constraints and use the corresponding LP relaxation to compute lower bounds. The LP relaxation after such enhancements will yield better bounds. Consider again the example shown in Figure 9. We add the following inequalities for the two cycles C_1, C_2 and C_3: $w_1 + w_2 \geq 1$, $w_3 + w_4 \geq 1$ and $w_5 + w_6 \geq 1$. The optimal solution to the LP relaxation after adding these constraints is 3 whereas it was 1 before. In general, if we add the cycle constraints corresponding to cycles with more than two vertices, then we can obtain better lower bounds.

5. MULTIWAY BRANCH AND BOUND

We combine the MFVS-preserving graph transformation, partitioned search strategy and ILP-based lower bounding technique to

obtain an exact algorithm for computing the MFVS. Given an S-graph we first reduce it by using the procedure $transform\_graph$. If the resulting graph is not empty, then we solve each compressed scc of this graph independently by using the partitioned search strategy of Section 3. The following procedure computes the MFVS of an S-graph:

Procedure **MFVS(S)**
$mfvs\_list$ = $transform\_graph(S)$;
if (S is not empty)
  for (each scc S_i)
    $mfvs\_list$ = $mfvs\_list$ + $SOLVE\_SCC(S_i)$;
return($mfvs\_list$);

The optimum solution for each scc S_i is obtained by using the recursive procedure $SOLVE\_SCC$. Let x_j be the first decision variable for scc S_1. The right branch for decision variable x_j corresponds to the case in which vertex v_j is included in the feedback vertex set ($remove(v_j)$). The left branch corresponds to the case in which vertex v_j is excluded from the feedback vertex set ($ignore(v_j)$). We compute the best possible feedback vertex set for both the right and left branches and store them in $r\_fvs[v_j]$ and $l\_fvs[v_j]$, respectively. The cardinality of the MFVS of the scc S_1 is equal to the minimum of $|r\_fvs[v_j]|$ and $|l\_fvs[v_j]|$.

The vertex v_j can be included in the feedback vertex set by using procedure $remove(v_j)$. Similarly, vertex v_j can be excluded from the feedback vertex set by using the procedure $ignore(v_j)$. In either case, the scc S_1 has changed and we use the procedure $transform\_graph(S_1)$ to further reduce S_1. After including v_j, if the reduced S_1 is not empty, then we recursively apply procedure $SOLVE\_SCC$ to each of its components. After that, we restore (backtrack) S_1 to the state it was before we removed vertex v_j. For the left branch, after excluding v_j, we can compute a lower bound on the cardinality of the MFVS for the transformed graph. If $|l\_fvs[v_j]| + lower\_bound(v_j) \geq |r\_fvs[v_j]|$, then there is no need to explore the left branch. Otherwise, we again recursively apply the procedure $SOLVE\_SCC$ to each of its components. After that, we restore (backtrack) S_1 to the state it was before we ignored vertex v_j. For the lower bound computation we can either use the trivial lower bound of 2 or the lower bound returned by the enhanced LP relaxation explained in the previous section.

Procedure **SOLVE_SCC(S_i)**
Pick a variable x_j from scc S_i;
$r\_fvs[v_j] = \phi$; $l\_fvs[v_j] = \phi$;
$remove(v_j)$; /* Include v_j in feedback vertex set */
$r\_fvs[v_j]$ = $transform\_graph(S_i) + v_j$;
if (S_i is not empty)
  for (each scc of S_i)
    $r\_fvs[v_j]$ = $r\_fvs[v_j]$ + $SOLVE\_SCC(scc)$;
backtrack; /* Restore S_i to its state before $remove(v_j)$ */
$ignore(v_j)$; /* Exclude v_j from feedback vertex set */
$l\_fvs[v_j]$ = $transform\_graph(S_i)$;
if ($|l\_fvs[v_j]| + lower\_bound(v_j) < |r\_fvs[v_j]|$) {
  if (S_i is not empty)
    for (each scc of S_i)
      $l\_fvs[v_j]$ = $l\_fvs[v_j]$ + $SOLVE\_SCC(scc)$;
  backtrack; /* Restore S_i to its state before $ignore(v_j)$ */
  return(minimum ($r\_fvs[v_j]$,$l\_fvs[v_j]$));
} **else** {
  backtrack; /* Restore S_i to its state before $ignore(v_j)$ */
  return($r\_fvs[v_j]$); }

6. RESULTS

We implemented the procedure **MFVS** presented in Section 5 in a C language program called PSCAN. We compare our algorithm with three other state-of-the art methods: Lee and Reddy [9], Pascant [11] and Opus [5]. All three are heuristic approaches and do not guarantee an MFVS. They also cannot determine how close their solution is to the optimum solution. However, they all select flip-flops to break all cycles (except self-loops) in the S-graph. All experiments were performed on a Sparc2 SUN workstation.

Table 1 reports results of experiments on the ISCAS 89 benchmark set of circuits. The number of flip-flops in each circuit is shown in the column *FF*. The number of scan flip-flops selected by various methods and the CPU seconds required to find the scan flip-flops are shown in columns *Scan Flip-Flops* and *CPU sec*, respectively. A 0 under *CPU sec* means less than 0.1 seconds. Under columns *LR, Pa, Op* and PSCAN, we report data obtained by Lee and Reddy, Pascant, Opus and PSCAN algorithms, respectively. The results in the column *LR* were communicated to us by the authors. They have recently modified their heuristics to obtain better (smaller) feedback vertex sets. Also, the table does not show the CPU seconds required by their algorithm. Their program requires less than 3 seconds for all the circuits in the table. Results for the exact algorithm of Smith and Walford are not included here since that technique is practical only for small circuits [9]. The $'-'$ in Table 1 indicates that the feedback vertex set reported by the corresponding program did not break all cycles.

Optimal results for large circuits are being reported for the first time here. PSCAN computes the MFVS of large S-graphs (obtained from circuits like **s15850**, **s38417**, **s38584** and others) in less than one minute. The CPU time required by PSCAN to compute the optimal solution is comparable or better than the time required by the heuristic methods to compute an approximate solution. For small circuits, the feedback vertex sets produced by the heuristic methods are optimal. For the larger circuits, the heuristic methods produce sub-optimal results. As an example, consider the circuit **s15850**. This circuit has 597 flip-flops. Lee and Reddy's technique computed a feedback vertex set of size 89. Pascant and Opus computed feedback vertex sets of size 101 and 91, respectively. PSCAN computed the optimal feedback vertex set of size 88.

We also compared our algorithm with the conventional branch and bound method. If we do not include our graph transformation procedure and pruning techniques, then the conventional branch and bound algorithm is unable to determine the MFVS for the S-graphs of large circuits like **s15850** and **s38417** even after running on a Sparc 2 SUN workstation for more than three days. If we include the graph transformation procedure and the pruning techniques, then the conventional branch and bound algorithm is able to determine the MFVS for all the S-graphs except for the S-graph of **s38417**. However, for the cases where MFVS is obtained, this method requires significantly more CPU seconds. As shown in Table 1, by using partitioned branch and bound, PSCAN is able to compute the MFVS for the S-graphs of **s15850** and **s38417** in less than a minute of CPU time.

If we also break self-loops, then the size of the feedback vertex set obtained by the heuristic methods is the same as the optimal solution determined by PSCAN for all benchmark circuits. This is

Table 1: Partial scan results.

Ckt	FF	Scan Flip Flops				CPU Sec.		
		LR	Pa	Op	PSCAN	Pa	Op	PSCAN
s208	8	0	0	0	0	0	0	0
s298	14	1	1	1	1	0	0.1	0.1
s344	15	5	5	5	5	0	0.1	0.1
s349	15	5	5	5	5	0	0.1	0.1
s382	21	9	9	9	9	0	0	0
s386	6	5	5	5	5	0	0.1	0.1
s400	21	9	9	9	9	0	0	0
s420	16	0	0	0	0	0	0.4	0
s444	21	9	9	9	9	0	0	0
s510	6	5	5	5	5	0	0.1	0.1
s526	21	3	4	3	3	0	0	0
s526n	21	3	4	3	3	0	0	0
s641	19	7	7	7	7	0	0.3	0.1
s713	19	7	7	7	7	0	0.3	0.1
s820	5	4	4	4	4	0	0	0
s832	5	4	4	4	4	0	0	0
s838	32	0	0	0	0	0	0.4	0
s953	29	5	5	5	5	0	0.1	0.1
s1196	18	0	0	0	0	0	0.4	0
s1238	18	0	0	0	0	0	0.4	0
s1423	74	21	37	22	21	0.3	1.0	0.9
s1488	6	5	5	5	5	0	0.5	0.1
s1494	6	5	5	5	5	0	0.5	0.1
s5378	179	30	32	30	30	0.2	0.3	0.1
s9234	228	53	65	-	53	0.8	6.6	0.9
s13207	669	59	71	-	59	1.5	24.2	0.4
s15850	597	89	101	91	88	6.4	18.0	48.0
s35932	1728	306	306	306	306	9.3	27.3	0.5
s38417	1636	374	400	380	374	43.3	909.0	32.8
s38584	1452	218	376	222	218	33.0	55.8	7.7

Table 2: Results for production VLSI circuits.

Ckt	Gates	PI	PO	FF	Scan FF	% Scan	Sec
ckt1	12054	18	30	873	83	9.5	1.2
ckt2	30272	104	82	3020	479	16.0	1.1
ckt3	53279	26	83	2810	1117	40.0	12.5

because these circuits have many self-loops. The graph obtained after deleting self-loop vertices is much smaller and is either already acyclic or has very few cycles.

7. RESULTS FOR PRODUCTION VLSI CIRCUITS

Several production VLSI circuits were also processed by PSCAN. These circuits contain logic gates, tri-state buffers, bidirectional IO buffers and buses. Table 2 reports results on a few circuits. Self-loops were ignored for all circuits. The number of flip-flops selected for partial scan is shown in column *Scan FF*. The percentage of flip-flops selected for partial scan is shown in column *% Scan*. Our experiments show that 10% to 40% of the flip-flops will have to be scanned to break all cycles. If we consider the overhead of full-scan design to be about 30%, then the area overhead for partial scan can be estimated to be between 3% to 12%. Note that the CPU seconds required to determine the optimum number of scan flip-flops is extremely low. We are unable to report the results for the heuristic methods since the corresponding C programs provided by the authors are unable to parse these circuit descriptions. They appear to have preset limits on several characteristics of the input circuit like the maximum number of flip-flops or maximum fanin of a gate.

8. CONCLUSION

Our exact algorithm computes the MFVS of the flip-flop dependency graph of a sequential circuit. As expected, our algorithm has worst case exponential complexity. However, our experimental results show that the method outperforms the available heuristic methods for large practical instances of the partial scan problem. The presented solution has other applications. For example, the proposed technique has been used to solve MFVS problems arising in the resynthesis and retiming of sequential circuits for enhanced testability [18].

Acknowledgment - We are grateful to J. Patel, S. Reddy and K. Sreenivas for providing C programs that implement their heuristics. We thank S. Bhawmik and T. J. Chakraborty for their assistance with the Pascant program. We also thank S. Rothweiler for assistance with the program PSCAN.

REFERENCES

[1] E. Trischler, "Incomplete Scan Path with an Automatic Test Generation Methodology," in *Proc. of the Intl. Test Conf.*, pp. 153 – 162, 1980.

[2] M. Abramovici, J. J. Kulikowski, and R. K. Roy, "The Best Flip-Flops to Scan," in *Proc. of the Intl. Test Conf.*, pp. 166–173, 1991.

[3] V. D. Agrawal, K. T. Cheng, D. D. Johnson, and T. Lin, "Designing Circuits with Partial Scan," *IEEE Design and Test of Computers*, vol. 5, pp. 8–15, April 1988.

[4] H.-K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli, "An Incomplete Scan Design Approach to Test Generation for Sequential Machines," in *Proc. of the Intl. Test Conf.*, pp. 730 – 734, 1988.

[5] V. Chickermane and J. H. Patel, "A Fault Oriented Partial Scan Design Approach," in *Proc. of the Intl. Conf. on Computer-Aided Design*, pp. 400 – 403, November 1991.

[6] P. S. Parikh and M. Abramovici, "A Cost Based Approach to Partial Scan," in *Proc. of the 30th ACM/IEEE Design Automation Conf.*, June 1993.

[7] K. T. Cheng and V. D. Agrawal, "A Partial Scan Method for Sequential Circuits with Feedback," *IEEE Transactions on Computers*, vol. 39, pp. 544 – 548, April 1990.

[8] G. W. Smith and R. B. Walford, "The Identification of Minimum Feedback Vertex Set of a Directed Graph," *IEEE Transactions on Circuits and Systems*, vol. 22, pp. 9 – 14, January 1975.

[9] D. Lee and S. Reddy, "On Determining Scan Flip-Flops in Partial-Scan Designs," in *Proc. of the Intl. Conf. on Computer-Aided Design*, pp. 322 – 325, November 1990.

[10] A. Kunzmann and H. J. Wunderlich, "An Analytical Approach to the Partial Scan Problem," *J. of Electronic Testing: Theory and Applications*, vol. 1, pp. 163–174, 1990.

[11] S. Bhawmik, C. J. Lin, K. T. Cheng, and V. D. Agrawal, "Pascant: A Partial Scan and Test Generation System," in *Custom Integrated Circuits Conf.*, pp. 17.3.1 – 17.3.4, 1991.

[12] S. Park and S. B. Akers, "A Graph Theoretic Approach to Partial Scan Design by K-Cycle Elimination," in *Proc. of the Intl. Test Conf.*, pp. 303–311, 1992.

[13] S. E. Tai and D. Bhattacharya, "A Three Stage Partial Scan Design Method using the Sequential Circuit Flow Graph," in *Proc. of the 7th Intl. Conf. on VLSI Design*, pp. 101–106, January 1994.

[14] P. Ashar and S. Malik, "Implicit Computation of Minimum-Cost Feedback-Vertex Sets for Partial Scan and Other Applications," in *Proc. of the 31st ACM/IEEE Design Automation Conf.*, June 1994.

[15] E. L. Lloyd, M. L. Soffa, and C. C. Wang, "On Locating Minimum Feedback Vertex Sets," *J. of Computer and System Sciences*, vol. 37, pp. 293 – 311, 1975.

[16] H. Levy and L. Low, "A Contraction Algorithm for Finding Small Cycle Cutsets," *J. of Algorithms*, vol. 9, pp. 470 – 493, 1988.

[17] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization Algorithms and Complexity*. Englewood Cliffs, New Jersey: Prentice Hall, 1982.

[18] S. T. Chakradhar and S. Dey, "Resynthesis and Retiming for Optimum Partial Scan," in *Proc. of the 31st ACM/IEEE Design Automation Conf.*, June 1994.