Cost-Free Scan: A Low-Overhead Scan Path Design Methodology

Chih-chang Lin, Mike Tien-Chien Lee*, Malgorzata Marek-Sadowska, and Kuang-Chien Chen*

Electrical and Computer Engineering
Univ. of California
Santa Barbara, CA 93106

Fujitsu Laboratories of America*
77 Rio Robles
San Jose, CA 95134

Abstract

Conventional scan design imposes considerable area and delay overhead by using larger scan flip-flops and additional scan wires without utilizing the functionality of the combinational logic. We propose a novel low-overhead scan design methodology, called cost-free scan, which exploits the controllability of primary inputs to establish scan paths through the combinational logic. The methodology aims at reducing scan overhead by (1) analyzing the circuit to determine all the cost-free scan flip-flops, and (2) selecting the best primary input vector to establish the maximum number of cost-free scan flip-flops on the scan chain. Significant reduction in the scan overhead is achieved on ISCAS89 benchmarks, where in full scan environment, as many as 89% of the total flip-flops are found cost-free scannable, while in partial scan environment, reduction can be as high as 97% in scan flip-flops needed to break sequential loops.

1 Introduction

Automatic test pattern generation (ATPG) for sequential circuits is a difficult problem [1] because of the lack of direct controllability of the present state lines (PSLs) and direct observability of the next state lines (NSLs). To enhance testability, effective design-for-test (DFT) techniques aiming at improving controllability and observability of the state lines have been proposed, such as full scan [2] and partial scan [3, 4, 5, 6, 7, 8, 9, 10]. Both scan techniques facilitate testing of a sequential circuit by interconnecting the circuit's flip-flops into a shift register during the test mode to directly control and observe the state lines. 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 and the extra routing area for the scan chains. A general structure of conventional scan design is shown in Figure 1(a) for a portion of a sequential circuit, where flip-flop F_j is connected to flip-flop F_i through the combinational logic f, and MUXed scan D flip-flops are used for F_i and F_j. The combinational logic f is designed to perform functional operation in the normal mode, while extra logic in the MUXed scan flip-flops and the scan wire is used to establish the scan chain in the test mode, which imposes considerable penalty on circuit area and performance. Furthermore, the direct controllability of circuit primary inputs is usually not utilized in the test mode.

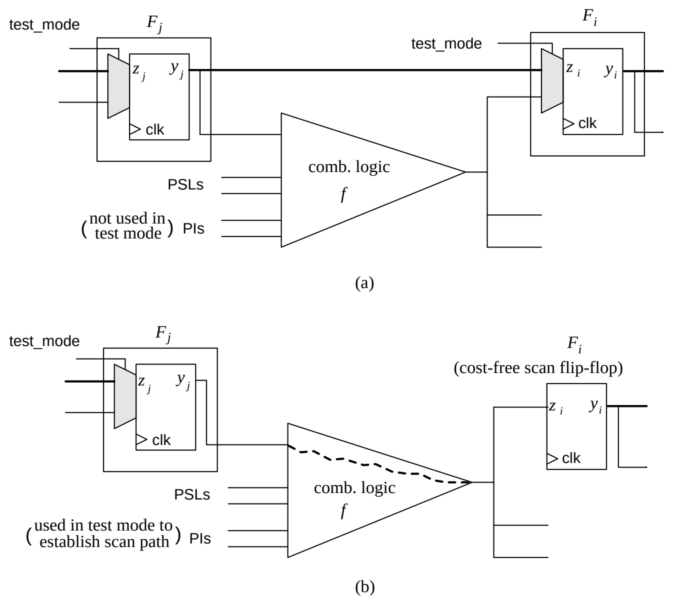

Figure 1: (a) general structure for conventional MUXed scan; (b) proposed free-scan design that exploits functional logic f and controllability of primary inputs (PIs) to establish the scan path (in dashed line).

This paper proposes a novel low-overhead scan design methodology which can exploit the controllability of primary inputs to establish scan paths through the combinational logic, even without using any MUXed scan flip-flops and the associated interconnect wires. The proposed method can reduce drastically the DFT overhead as well as its impact on circuit timing properties.

Our essential idea can be illustrated in 1(b), where a scan path from F_j to F_i is formed through the combinational logic f, as indicated by the dashed line, by a particular primary input vector supplied in the test mode. This primary input vector, called the **enabling vector**, must make the next state line z_i (the input of F_i) depend exclusively on y_j (the output of F_j) during scan, such that the content of F_j can be shifted into the non-scan flip-flop F_i through the scan path in f in one clock cycle. Since the non-scan flip-flop F_i costs no scan overhead for the scan order from F_j to F_i, F_i is called a **free-scan flip-flop**.

We aims at reducing scan overhead by (1) performing efficient analysis of the entire sequential circuit to determine all the free-scan flip-flops and the associated enabling vectors, and (2) selecting the best enabling vector to maximize the number of free-scan flip-flops for constructing the scan chain. A novel formulations are derived which allow efficient computation by using OBDD techniques [11, 12].

The proposed methodology can be incorporated with conventional full scan and partial scan designs. In a full scan environment, the remaining flip-flops not free-scannable are converted into MUXed scan flip-flops to complete the scan chain. In the case of partial scan environment, after the free-scan flip-flops are selected, our current implementation uses conventional loop-breaking algorithms [5, 6, 7, 8, 9, 10] to determine the MUXed scan flip-flops for breaking the remaining sequential loops. In both cases, scan overhead is reduced significantly. Experimental results on ISCAS89 benchmarks show that the reduction by free-scan can be as high as 97% in terms of the number of MUXed scan flip-flops needed by Lee-Reddy's partial scan algorithm [8].

Previous work in [13, 14, 15, 16] presented heuristics to reduce scan overhead by attempting to merge scan MUXs into the combinational logic during logic synthesis. However, the controllability of primary inputs was not considered in establishing scan paths through the combinational logic. Therefore, even in the case of Figure 1(b) where F_i is free-scannable, a scan MUX and the associated scan wire were added into the circuit.

Operations and representation of Boolean functions relevant to our discussion are defined in this following.

Definition 1 *The* **consensus** *of $f(x_1, \cdots, x_n)$ with respect to x_i is $\mathcal{C}_{x_i}(f) = f_{x_i} \wedge f_{\overline{x_i}}$, where f_{x_i} and $f_{\overline{x_i}}$ are cofactors of f w.r.t. x_i and $\overline{x_i}$, respectively.*

$\mathcal{C}_{x_i}(f)$ represents the component that is independent of x_i, and can be interpreted as $\forall_{x_i} f(x_1, \cdots, x_n)$. It represents all the $(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$ vectors that make f true regardless of the value of x_i. The consensus operation can be extended to a set of variables, as an iterative application of the consensus on each variable in the set.

A Boolean function can be efficiently represented in a compact form by using **ordered binary decision diagram** (OBDD) [11], where the same variable ordering is required when traversing along any OBDD paths. Efficient OBDD package for such Boolean function manipulations were developed in [12], and is used in the algorithms and the experiments presented in this paper.

2 Determining Free-Scan Flip-Flops

Suppose a sequential circuit has n flip-flops $F_1, \cdots, F_n$, where each F_i has input z_i as the next state line and output y_i as the present state line. To analyze this circuit for the presence of free-scan flip-flops, we first build a directed **dependency graph** G_D, where a vertex v_i corresponds to an F_i in the circuit, and an arc $< v_j, v_i >$ corresponds to topological connection by the combinational logic from F_j to F_i. Because a scan chain cannot contain loops, an arc that creates a self-loop $< v_j, v_j >$ is not included in G_D.

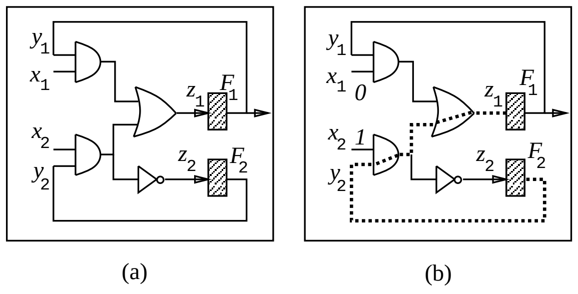

Figure 2: (a) example of a sequential circuit; (b) free-scan path (in dashed line) established from F_2 to F_1 by $(x_1 x_2)$ = 01.

2.1 Enabling Vector

For each arc $< v_j, v_i >$ in G_D, we check if there exists an enabling vector X at the primary inputs such that z_i (the input of F_i) depends only on y_j (the output of F_j), regardless of the values of other present state lines. The condition for such an X to exist is stated by the following theorem:

Theorem 1 *For a next state function $z_i = f(X, y_1, \cdots, y_n)$, where X represents the set of primary inputs and $y_1, \cdots, y_n$ are the present state lines, the necessary and sufficient condition for X to make z_i depend only on y_j is that X must be in the on-set of the Boolean function:*

$$K^{j,i}(X) = \mathcal{C}_{y_1,\cdots,y_n}(f_{y_j} \wedge \overline{f_{\overline{y_j}}}) + \mathcal{C}_{y_1,\cdots,y_n}(\overline{f_{y_j}} \wedge f_{\overline{y_j}}).$$

Proof. Based on the consensus definition, the on-set of the first term of $K^{j,i}(X)$ consists of all the X's under which $z_i = y_j$, regardless of all the other y_k, $k \neq j$. Similarly, the on-set of the second term consists of all the X's under which $z_i = \overline{y_j}$. Thus, the union of these two on-sets makes z_i depend only on y_j regardless of all the other y_k, $k \neq j$. ∎

Example 1 *Consider the sequential circuit in Figure 2(a) with two flip-flops F_1 and F_2. The next state functions are $z_1 = f_1(x_1, x_2, y_1, y_2) = x_1 y_1 + x_2 y_2$, and $z_2 = f_2(x_1, x_2, y_1, y_2) = \overline{x_2 y_2}$. So the G_D consists of two nodes v_1, v_2, and an arc $< v_2, v_1 >$. To establish a scan path from F_2 to F_1 through the combinational logic, we check if there exists any input vector which can make z_1 depend only on y_2. Since $f_{1_{y_2}} = x_1 y_1 + x_2$ and $f_{1_{\overline{y_2}}} = x_1 y_1$, we can obtain $K^{2,1}(X) = \mathcal{C}_{y_1}(f_{1_{y_2}} \wedge \overline{f_{1_{\overline{y_2}}}}) + \mathcal{C}_{y_1}(f_{1_{\overline{y_2}}} \wedge \overline{f_{1_{y_2}}}) = \overline{x_1} x_2$ by Theorem 1. So $(x_1 x_2) = 01$ is the only enabling vector. This vector can be verified by substituting into f_1, i.e., $z_1 = f_1(0, 1, y_1, y_2) = 0y_1 + 1y_2 = y_2$. The established scan path from F_2 to F_1 is depicted in the dashed line in Figure 2(b).* □

If $K^{j,i}(X)$ consists of the whole Boolean space of X, this means that for any X, z_i is equal to y_j or $\overline{y_j}$. In

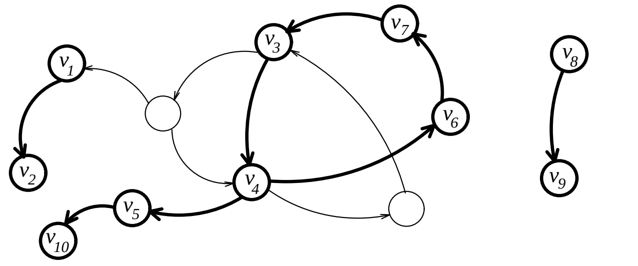

Figure 3: Example of G_{FS} and $G_{FS}(X)$, where $G_{FS}(X)$ is a subgraph of G_{FS} and consists of only the arcs established (in darkened line) by the enabling vector X.

this case, F_j and F_i are connected either directly as in a shift register or through the combinational logic which is redundant and can be reduced to a wire or just an inverter. This type of free-scan flip-flop F_i is called the **shift flip-flop**, and can be identified easily when the OBDD of $K^{j,i}(X)$ is equal to 1.

2.2 Free-Scan Graph

After the enabling vectors are computed for each $< v_j, v_i >$ in G_D, a subgraph can be built from the dependency graph G_D, called the **free-scan graph** G_{FS}, such that an arc $< v_j, v_i >$ is in G_{FS} if and only if $< v_j, v_i >$ is in G_D and the on-set $K^{j,i}(X)$ is not empty. An arc $< v_j, v_i >$ in G_{FS} is said **established** by an enabling vector X as a free-scan path if $X \in K^{j,i}(X)$.

Since the final scan chain can use only one enabling vector to complete the connection in the test mode, it is of interest to know which arcs in G_{FS} are established by this vector. We define for a particular X, a subgraph $G_{FS}(X)$ of G_{FS} consisting of the arcs in G_{FS} that are established by X. An example of $G_{FS}(X)$ is shown in the darkened lines in Figure 3.

$G_{FS}(X)$ captures the free-scan paths in a circuit for a given input vector X. It can have cycles. But the scan chain has to be acyclic. Therefore, in order to break all the cycles in $G_{FS}(X)$ while maintaining as many free-scan flip-flops on the scan chain as possible, a minimum set of free-scan flip-flops needs to be converted into MUXed scan flip-flops. For example, v_6 in Figure 3 can be selected to break the cycle. The problem of finding such a minimum set is referred as the *minimum feedback vertex set* (MFVS) problem, and in general is an NP-hard problem [8]. Fortunately, $G_{FS}(X)$ has the following special properties by which a simple linear-time algorithm can be developed to break all the cycles optimally.

Lemma 1 *The in-degree of a node in $G_{FS}(X)$ must be at most 1.*

Proof. Based on Theorem 1, the enabling vector X must not make the next state line depend on more than one present state line. So there cannot be more than one arc incident to a node in $G_{FS}(X)$. ∎

Theorem 2 *Any disjoint component of $G_{FS}(X)$ can have at most 1 cycle.*

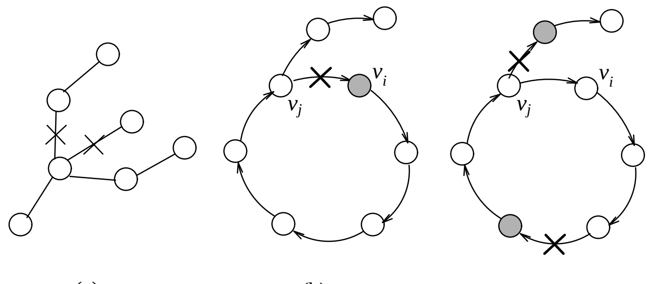

Figure 4: Disconnect arcs to eliminate branches and/or break cycles when constructing the scan chain (the black nodes correspond to MUXed scan flip-flops).

Proof. Suppose there can be two cycles in a disjoint component of $G_{FS}(X)$. If these two cycles share some node v_i, v_i must have in-degree greater than 1. Otherwise, if they do not share any nodes, there must be a path in the disjoint component from one cycle incident to another cycle at some node v_i. Then node v_i must have in-degree greater than 1. Both situations contradict Lemma 1. So any disjoint component can have at most 1 cycle. ∎

Therefore, cycle detection in each disjoint component of $G_{FS}(X)$ can be done in linear time by traversing along each arc. If a visited node is visited again during traversal, a cycle is detected. If there are totally l cycles in $G_{FS}(X)$, l nodes must be selected to break them, one for each cycle. The selection of such cycle breaking nodes can also be done in linear time.

Furthermore, if a node has out-degree m larger than 1, only one outgoing arc can be maintained in the scan chain while the other $m-1$ arcs must be disconnected to eliminate the branch, as is shown in Figure 4(a). If this node is also in a cycle, the outgoing arc in the cycle should be disconnected to break the cycle, because the branch can be eliminated at the same time. In this case, one MUXed scan flip-flop is needed. Otherwise, two MUXed scan flip-flops are needed, one for breaking the cycle, the other for eliminating the branch. This idea is illustrated by the following example.

Example 2 *Consider the graphs in Figures 4(b) and 4(c) where node v_j has out-degree 2 and is contained in a cycle. So, if $< v_j, v_i >$ is disconnected, the cycle is broken and the branch is eliminated, as shown in Figure 4(b). Note that in order to disconnect arc $< v_j, v_i >$, the flip-flop corresponding to node v_i is converted to a MUXed scan flip-flop. Otherwise, disconnecting any other arc in the cycle will need an additional MUXed scan flip-flop to eliminate the branch, as is the case in Figure 4(c).* □

2.3 Constructing Free-Scan Path

Based on the ideas discussed in Section 2.2, a linear-time algorithm to construct free-scan paths, called `construct_free_scan_path`, is developed which can select the minimum set of nodes in $G_{FS}(X)$ for cycle breaking as well as branch elimination. This algorithm is presented in Figure 5.

```
construct_free_scan_path($G_{FS}(X)$)
{
  for (each disjoint component $dc \in G_{FS}(X)$) {
    /* break the cycle in $dc$ */
    if (($c$ = cycle_detect($dc$)) $\neq \emptyset$) {
      for (each arc $< v_j, v_i > \in c$)
        if (out_degree($v_j$) > 1) /* branch detected */
          break;
      disconnect($< v_j, v_i >$);
    } /* $dc$ is acyclic now */
    /* eliminate the remaining branches in $dc$ */
    for (each node $v_j \in dc$)
      if (out_degree($v_j$) > 1)
        disconnect(all except one outgoing arc from $v_j$);
  }
  return(all free scan paths);
}
```

Figure 5: Algorithm of constructing free-scan paths by breaking the cycle and eliminating the branches in $G_{FS}(X)$.

The algorithm takes as input a graph $G_{FS}(X)$, which consists of all the arcs established by an enabling vector X. For each disjoint component dc in $G_{FS}(X)$, if it contains a cycle c, and c has an arc $< v_j, v_i >$ with v_j's out-degree larger than 1, $< v_j, v_i >$ is selected to be disconnected. Otherwise, an arbitrary arc in c is selected to be disconnected. So these steps make dc acyclic. Then, for each node v_j in dc, if v_j's out-degree is still larger than 1, all except one outgoing arc from v_j are disconnected to eliminate the branches. Finally, all the constructed free-scan paths are returned. The time complexity of this algorithm is linear to the number of arcs in $G_{FS}(X)$.

3 Selecting the Best Enabling Vector

In order to minimize the scan overhead, we want to find the best enabling vector X^* to maximize the number of free-scan flip-flops. Given an input vector X, all the $K^{j,i}(X)$'s that equal to 1 are determined for the corresponding arcs $< v_j, v_i >$'s established in $G_{FS}(X)$. A variable $q_{j,i}$ is assigned to represent $K^{j,i}(X)$ for arc connectivity. That is, $q_{j,i} = 1$ if $< v_j, v_i >$ is established by X; otherwise, $q_{j,i} = 0$. So, the mapping relation from X to arc connectivity $(q_{1,2}, \cdots, q_{n-1,n})$ can be determined by the following characteristic function:

$$\mathcal{X}(X, q_{1,2}, \cdots, q_{n-1,n}) = \prod_{<v_j,v_i> \ \in \ G_{FS}} (q_{j,i} \equiv K^{j,i}(X)).$$

This means for each on-set minterm $(X, q_{1,2}, \cdots, q_{n-1,n})$ of $\mathcal{X}$, $q_{j,i}$ is 1 if X can establish an arc $< v_j, v_i >$; otherwise, $q_{j,i}$ is 0. $\mathcal{X}$ can be represented efficiently in OBDD with variable ordering X followed by $q_{j,i}$'s. The number of free-scan flip-flops for each X can be determined along the OBDD traversal.

As mentioned previously, due to the variable ordering in the OBDD of $\mathcal{X}$, the $q_{j,i}$ vertices are visited before

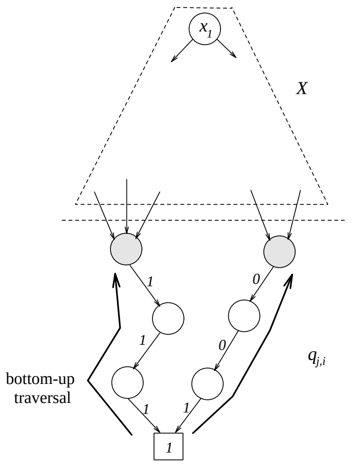

Figure 6: OBDD of characteristic function $\mathcal{X}$; the bottom-up traversal halts at the shaded vertices to compare the number of free-scan flip-flops.

the x_k vertices during the bottom-up depth-first traversal. If the traversal halts right before visiting any x_k vertex, the partially traversed OBDD path consists of only $q_{j,i}$ vertices. Therefore, this partially traversed path corresponds to a $(q_{1,2}, \cdots, q_{n-1,n})$ vector, which in turn corresponds to a $G_{FS}(X)$, although X is not yet determined because the traversal halted. Figure 6 shows an example of OBDD with two partially traversed paths ending at the shaded $q_{j,i}$ vertices, where one path corresponds to a $G_{FS}(X)$ with 3 arcs, while the other one, with 1 arc. The number of free-scan flip-flops in a $G_{FS}(X)$ can be determined at the shaded vertex by invoking `construct_free_scan_path` developed in Section 2.3. So, by comparing the numbers of free-scan flip-flops at the shaded vertices, the $G_{FS}(X)$ with the maximum number of free-scan flip-flops is determined. Then the bottom-up depth-first traversal resumes from the corresponding shaded vertex to find an X. Note that there can be more than one such X, and any of them can be picked as the best enabling vector X^*. The free-scan flip-flops established can be obtained by invoking `construct_free_scan_path(`$G_{FS}(X^*)$`)`.

4 Free-Scan Methodology

4.1 Free-Scan Design

The complete methodology of our proposed free-scan design is summarized in Figure 7. It can be used with conventional full scan and partial scan test strategies. First, the free-scan graph G_{FS} is built from the circuit's dependency graph G_D based on Theorem 1. The characteristic function $\mathcal{X}$ is then created in OBDD representation for each arc in G_{FS}. The best enabling vector X^* maximizing the number of free-scan flip-flops is selected by traversing the OBDD of $\mathcal{X}$. Free-scan paths are constructed next by algorithm `construct_free_scan_path` presented in Section 2.3. In case of full scan environment, all of the remaining flip-flops are converted into MUXed scan flip-flops. For partial scan test strategy, since the free-scan flip-flops selected may not break all the sequential

```
free_scan($G_D$)
{
  /* 1. build $G_{FS}$ based on Theorem 1 */
  $G_{FS} = \emptyset$;
  for (each arc $< v_j, v_i > \in G_D$) {
    compute OBDD of $K^{j,i}(X)$;
    if ($K^{j,i}(X) \neq \emptyset$)
      $G_{FS} = G_{FS} \cup < v_j, v_i >$;
  }
  /* 2. select the best enabling vector */
  OBDD$_{\mathcal{X}}$ = OBDD of $\mathcal{X}$;
  $X^*$ = the best enabling vector by traversing OBDD$_{\mathcal{X}}$;
  /* 3. construct free-scan paths */
  FF$_{FS}$ = construct_free_scan_path($G_{FS}(X^*)$);
  FF = total flip-flops from $G_D$;
  if (full scan)
    for (each flip-flop $ff \in$ FF − FF$_{FS}$)
      convert $ff$ into MUXed scan flip-flop;
  else if (partial scan) {
    FF$_{PS}$ = sequential_loop_breaking(FF − FF$_{FS}$);
    for (each flip-flop $ff \in$ FF$_{PS}$)
      convert $ff$ into MUXed scan flip-flop;
  }
}
```

Figure 7: Complete free-scan design methodology.

loops, a conventional sequential-loop-breaking algorithm [5, 6, 7, 8, 9, 10] can follow to break the remaining loops.

4.2 Test Strategy

To test a circuit with our free-scan design methodology, test vectors for the entire circuit are generated by ATPG in the same way as in the conventional full scan or partial scan environment. When test vectors are scanned in and test responses are scanned out in the test mode, the best enabling vector X^* must be applied at the circuit's primary inputs to maintain the complete scan chain.

Moreover, since the free-scan path is a part of the combinational logic, it is necessary to test the logic on the free-scan path separately, prior to testing the entire circuit. This is because, for example, an error caused by a fault in the free-scan logic could be cancelled when it is scanned out through the same free-scan logic. This results in error compensation.

Testing the logic on a free-scan path can be accomplished by scanning in a sequence of alternating 0's and 1's and scanning them out while the primary inputs are fixed to X^*. By examining the scanned out result, the fault in the combinational logic responsible for scanning in and scanning out can hence be detected.

5 Experimental Results

We tested the proposed free-scan methodology on a number of ISCAS89 sequential benchmarks, in both full scan and partial scan environments. Table 1 gives the circuit statistics on the number of primary inputs (`#I`), the number of primary outputs (`#O`), the number of total flip-flops (`#total FF`), the number of arcs in G_D (G_D `#arc`), the number of arcs in G_{FS} (G_{FS} `#arc`), and the number of shift flip-flops detected by using Theorem 1, which are connected in a shift register fashion (`#shift FF`). The circuits with names appended with $*$ were optimized by logic optimizer SIS-1.2 [17] using **script.algebraic** script.

Table 1: ISCAS89 benchmark circuits statistics.

circuit	#I	#O	#total FF	G_D #arc	G_{FS} #arc	#shift FF
s641	35	23	19	100	9	0
s1196	14	14	18	20	12	0
s1423	17	5	74	1694	9	0
s5378	35	49	164	1194	53	47
s9234	36	39	211	2546	57	31
s13207	31	121	669	3406	284	156
s13207*	31	121	453	2442	102	94
s15850	14	87	597	14925	179	47
s15850*	14	87	540	14472	159	34
s35932	35	320	1728	4475	1728	0
s35932*	35	320	1728	4219	1728	0
s38584	12	278	1452	16880	1327	164
s38584*	12	278	1294	13507	1173	152

Table 2: Percentage of saving in MUXed scan flip-flops by free-scan (FS) in a full scan environment.

circuit	#total FF	#FS FF (%saving)	CPU time
s641	19	6 (32%)	2.8s
s1196	18	3 (17%)	2.7s
s1423	74	5 (7%)	203.2s
s5378	164	49 (30%)	106.3s
s9234	211	39 (18%)	413.0s
s13207	669	157 (23%)	224.3s
s13207*	453	96 (21%)	220.1s
s15850	597	127 (21%)	5118.0s
s15850*	540	114 (21%)	2681.6s
s35932	1728	1431 (83%)	1544.9s
s35932*	1728	1431 (83%)	902.2s
s38584	1452	1288 (89%)	3588.8s
s38584*	1294	1134 (88%)	3667.6s

In full scan environment

Table 2 shows the results on the number of free-scan flip-flops found by our `construct_free_scan_path` algorithm (`#FS FF`), its percentage of the total flip-flops, which is also the percentage of saving in MUXed scan flip-flops compared with full scan (`(%saving)`), and the CPU time on SUN SPARC 20 for computing the free-scan flip-flops (`CPU time`). We can see that very high saving percentage in MUXed scan flip-flops (89% for s38584) can be achieved by the proposed free-scan methodology.

In partial scan environment

Since the free-scan flip-flops selected may not break all the sequential loops for partial scan, we used Lee-Reddy's algorithm [8], denoted by LR, to select a minimal set in

Table 3: Percentage of saving in MUXed scan flip-flops by free-scan (FS) in a partial scan environment (LR: Lee-Reddy's loop-breaking algorithm.)

circuit	#MUXed scan FF	
	LR only	FS + LR (%saving)
s641	7	3 (57%)
s1196	0	0 n/a
s1423	22	21 (5%)
s5378	30	6 (80%)
s9234	53	43 (19%)
s13207	59	55 (7%)
s13207*	474	331 (30%)
s15850	90	82 (9%)
s15850*	90	82 (9%)
s35932	306	9 (97%)
s35932*	306	9 (97%)
s38584	294	29 (90%)
s38584*	177	11 (94%)

the remaining flip-flops for MUXed scan to break all the loops. Table 3 shows the results on the number of MUXed scan flip-flops selected by LR without free-scan (`LR only`) or with free-scan (`FS + LR`). The reduction percentage by `FS + LR` over `LR only` is shown as well (`(%saving)`). We can see that very high reduction percentage in MUXed scan flip-flops (97% for s35932) can be achieved by free-scan. Note that the G_D of s1196 is acyclic already, so no loop needs to break.

6 Conclusions

This paper proposed a novel low-overhead scan path design methodology, called free-scan, which allows to reuse the combinational logic for scan by exploiting the controllability of primary inputs. It can be incorporated with conventional full scan and partial scan designs to reduce the DFT overhead drastically. Formulations were presented to optimally compute the free-scan flip-flops and the best enabling vector, using OBDD techniques. Significant results on ISCAS89 benchmarks showed that (1) in full scan environment, as many as 89% of the total flip-flops are found free-scannable without using MUXed scan, and (2) in partial scan environment, reduction can be as high as 97% in MUXed scan flip-flops needed to break sequential loops.

Acknowledgements

This work was supported in part by the National Science Foundation grants MIP9117328 and MIP9419119, and by California MICRO program. The second author would like to thank Mr. Takashi Aikyo of Fujitsu Ltd. for interesting discussion.

References

[1] F. C. Hennie, "Fault detecting experiments for sequential circuits," *Proc. 5th Ann. Symp. Switching Circuit Theory & Logic Design*, pp. 95–110, 1964.

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

[3] E. Trischler, "Incomplete scan path with an automatic test generation approach," *Int. Test Conf.*, pp. 153–162, 1980.

[4] M. Abramovici, J. J. Kulikowski, and R. K. Roy, "The best flip-flops to scan," *Int. Test Conf.*, pp. 166–173, 1991.

[5] V. Chickermane and J. H. Patel, "An optimization based approach to the partial scan design problem," *Int. Test Conf.*, pp. 377–386, 1991.

[6] V. Chickermane and J. H. Patel, "A fault oriented partial scan design approach," *ICCAD*, pp. 400–403, 1991.

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

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

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

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

[11] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Computers*, vol. C-35, pp. 667–691, 1986.

[12] K.S. Brace, R.L. Rudell and R.E. Bryant, "Efficiency implementation of a BDD package," *ACM/IEEE Design Automation Conf.*, pp. 40–45, 1989.

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

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

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

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

[17] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," *ICCD*, pp. 328–333, 1992.