Test Register Insertion with Minimum Hardware Cost

Albrecht P. Stroele

Institute of Computer Design and Fault Tolerance
University of Karlsruhe
D-76128 Karlsruhe, Germany

Hans-Joachim Wunderlich

Institute of Computer Structures
University of Siegen, Hölderlinstr. 3
D-57068 Siegen, Germany

Abstract

Implementing a built-in self-test by a "test per clock" scheme offers advantages concerning fault coverage, detection of delay faults, and test application time. Such a scheme is implemented by test registers, for instance BILBOs and CBILBOs, which are inserted into the circuit structure at appropriate places. An algorithm is presented which is able to find the cost optimal placement of test registers for nearly all the ISCAS'89 sequential benchmark circuits, and a suboptimal solution with slightly higher costs is obtained for all the circuits within a few minutes of computing time. The algorithm can also be applied to the Minimum Feedback Vertex Set problem in partial scan design, and an optimal solution is found for all the benchmark circuits.

The resulting self-testable circuits are analyzed. It is found that often CBILBOs lead to a minimum hardware overhead and also simplify test scheduling and test control.

1. Introduction

Built-in self-test (BIST) is one of the most important techniques for testing large and complex systems. Test registers are added at the primary inputs and outputs of a circuit, and some additional test hardware is inserted into the circuit. In a "test per scan" scheme, test registers feed and evaluate a (partial) scan path (see figure 1).

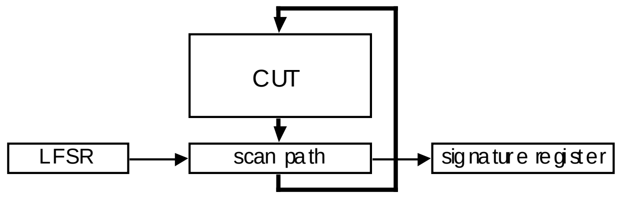

Figure 1: "Test per scan" scheme

Part of this work was supported by the European Commision under BRA 7107 ARCHIMEDES and by the DFG under grants Schm 623/5-1 and Wu 245/1-1.

It has been shown independently by several authors that breaking all cycles in the circuit structure bounds the length of the required test sequences to the sequential depth of the circuit [3, 7, 12, 14]. To keep the hardware overhead low the number of flip-flops that are integrated into the partial scan path in order to break all cycles should be as small as possible. If the topology of the storage elements is represented by a so-called *S-graph* whose vertices correspond to flip-flops and whose edges indicate combinational paths between flip-flops, then this problem is equivalent to finding a "Minimum Feedback Vertex Set" [6]. [4] presents an algorithm to compute the MFVS exactly using a branch and bound technique.

In a "test per clock" scheme, some system registers are enhanced such that they generate patterns or compact test responses in a special test mode. Examples of these multi-mode test registers are BILBO, GURT, and CBILBO [10, 17, 18]. A "test per clock" scheme has advantages with respect to test application time, delay testing and defect coverage, but it often requires a higher hardware overhead than the "test per scan" scheme.

In order to obtain testable subcircuits, the test registers must be placed at appropriate positions (e.g. [5]). However, the circuit structure obtained from breaking all cycles is not a priori suited to a "test per clock" scheme since during self-testing some test registers may have to generate patterns and compact test responses concurrently (e.g. test register T2 in figure 2).

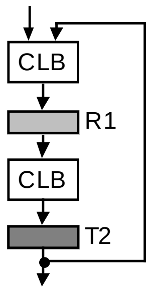

Figure 2: Part of a data path with register R1, test register T2, and combinational logic blocks (CLB)

This technique is called circular self-test path and leads to a sufficiently high fault coverage if the circuit is not random pattern resistant and all the states required for testing are reachable [11]. But in general, the patterns are not exhaustive, (weighted) random, or deterministic. In this case BILBOs have to be employed, and at least two of them are required in each cycle. In figure 2, register R_1 must also be enhanced to a BILBO register.

In particular, problems arise when a register feeds itself through combinational logic (*self-adjacent* register [9]), e.g. register R3 in figure 3. Here register R3 must be enhanced to a BILBO register T3, and an additional test register T4 of the BILBO type that is transparent in normal mode (see figure 4) must be inserted into the feedback path.

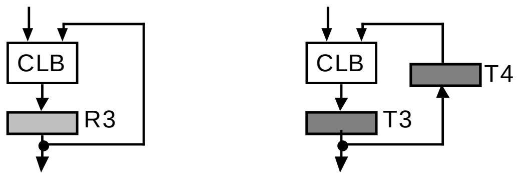

Figure 3: Part of a data path with self-adjacent register R3 (left), self-testable structure with BILBO register T3 and transparent BILBO register T4 (right)

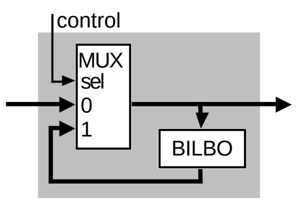

Figure 4: Structure of a transparent test register

Synthesis for testability approaches can reduce the number of cycles in a circuit [13, 15] and the number of self-adjacent registers [1]. But in general, cycles cannot be avoided completely.

To reduce the additional overhead of transparent test registers, some authors propose using three latches for each bit of the test register such that it can generate patterns and compact test responses independently. These test registers are called L3-BILBOs [8, 16] or CBILBOs [17]. In figure 3, it is sufficient to enhance R3 to a CBILBO, and an additional test register in the feedback path is not required.

At gate level, flip-flops can be enhanced to BILBO cells or CBILBO cells (1-bit elements of test registers), and additional BILBO cells and CBILBO cells that are transparent in normal mode can be inserted into arbitrary lines. After test cells have been inserted, each cycle of the circuit structure must contain at least one CBILBO cell or two BILBO cells. Regarding hardware costs, CBILBO cells are more expensive than BILBO cells, and inserting a whole transparent test cell into a line is more expensive than enhancing an existing flip-flop. Nevertheless, in some situations transparent test cells are useful. In order to make the circuit of figure 5 self-testable, both flip-flops FF1 and FF2 can be enhanced to CBILBO cells. But the same goal can also be achieved by inserting just one transparent CBILBO cell.

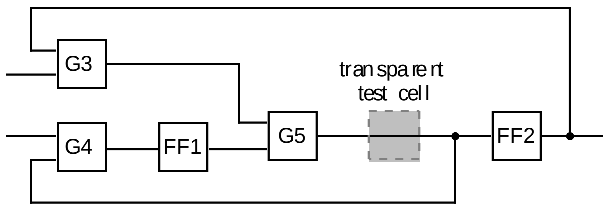

Figure 5: Circuit with two connected self-loops

In this paper, we present an exact algorithm that determines an appropriate placement of test cells and selects their types such that the total hardware cost of all the built-in test cells is minimum. As a special case, it also solves the MFVS problem for partial scan and the "test per scan" scheme.

In the next section the problem is described in a formal way, and graph transformations are introduced which are applied in a pre-processing step. Section 3 shows how the problem can be partitioned into subproblems that can be solved independently. In section 4 the branch and bound search is described. Experimental results follow in section 5. Section 6 gives some conclusions.

2. Problem statement and simplification

At gate level, a circuit is usually modeled by a directed graph $G = (V, E)$ with nodes V and edges $E \subset V \times V$. The nodes $V = I \cup O \cup V_C \cup V_S$ represent primary inputs (I), primary outputs (O), gates (V_C), and flip-flops (V_S). The edges describe the lines that connect these elements. The placement of test cells is described by labels *l* that have the following meaning:

for $v \in V_S$:

l(v) := \B\LC\{(\A\AL\CO2\VS0\HS10(0;if flip-flop v is not modified; 1;if v is enhanced to a BILBO cell; 2;if v is enhanced to a CBILBO cell))

for $v \in V_C$:

l(v) := \B\LC\{(\A\AL\CO2\VS0\HS10(0;if gate v is not modified; 1;if a transparent BILBO cell is; ;inserted at the output of gate v; 2 ; i f a transparent CBILBO cell is; ;inserted at the output of gate v))

In order to find an optimal placement, the following problem has to be solved:

"Minimum cost placement (MCP)"

Given: Circuit graph G = (V, E), costs of enhancing a flip-flop to a BILBO cell or a CBILBO cell, and costs of inserting a transparent BILBO cell or a transparent CBILBO cell at the output of a gate.

Find: Labeling *l* such that $_{v \in Z}$;$\Sigma l(v) \geq 2$ is true for each cycle Z of G, and the total cost associated with the labeling *l* is minimum.

Problem MCP includes selecting flip-flops for partial scan as a special case (cost of CBILBO cells smaller than cost of BILBO cells), and it is NP-complete. The authors of [4] developed an exact algorithm for selecting partial scan flip-flops. But their graph partitioning using strongly connected components cannot be applied here.

The presented algorithm uses a pre-processing step similar to [14] where iteratively all the nodes without predecessors or without successors are removed since they cannot be part of any cycle. Moreover, as the transistor cost of a transparent test cell is the same for all the gate inputs and outputs, it is sufficient to consider the outputs of combinational fanout-free regions for possible insertion of transparent test cells.

Every cycle of a directed graph G is completely included in a strongly connected component (SCC) of G [14]. Hence, all SCCs of G can be considered separately. A minimum cost placement for G consists of minimum cost placements for all the SCCs of G. The SCCs of G are extracted by deleting edges (v_i,v_j) where v_i and v_j belong to different SCCs.

3. Divide and conquer

At the beginning, the labels of all the nodes of G are reset, *l*(v) = 0 for all $v \in V$, corresponding to the circuit without any test cells. Then in each step a node v_i with *l*(v_i) = 0 is selected, and its label is set to 1 or 2. If the assigned label makes $_{v \in Z}$;$\Sigma l(v) \geq 2$ true for a cycle Z of G, this cycle does not have to be considered any more. If there are still cycles and two of them share a node v_i with *l*(v_i) = 0, then both cycles have to be considered together since a test cell placed at v_i would have an impact on both cycles. On the other hand, if it is possible to partition the cycles into subsets such that cycles from different subsets do not share any nodes with label 0, then these subsets can be considered separately. In this way we get subproblems that can be solved independently, and the optimal solution can be found much more efficiently. These subsets of cycles are called T-connected components (TCCs, connected during test application).

Definition: A T-connected component of a graph G is a minimal subgraph of G with the following characteristics:

- A TCC includes at least one cycle Z of G with $_{v \in Z}$;$\Sigma l(v) < 2$ (i.e. a cycle that contains no CBILBO cells and at most one BILBO cell).
- If G includes two cycles Z and Z' with $_{v \in Z}$;$\Sigma l(v) < 2$ and $_{v' \in Z'}$;$\Sigma l(v') < 2$, and if these cycles share at least one node with label 0, then all the nodes and edges of both cycles belong to the same TCC.

The labeled graph of figure 6 has two TCCs. The graph of figure 7, however, cannot be divided since it is composed of a single TCC. Figure 8 shows a graph that does not contain any TCCs.

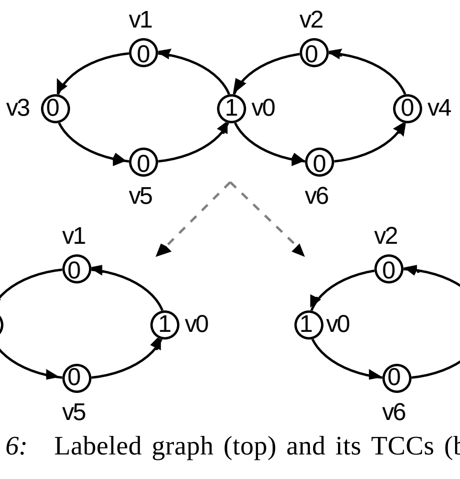

Figure 6: Labeled graph (top) and its TCCs (bottom)

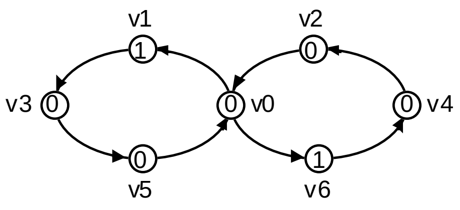

Figure 7: Labeled graph composed of a single TCC

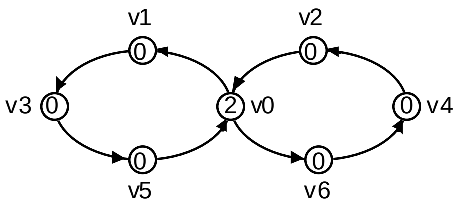

Figure 8: Labeled graph with an empty set of TCCs

The TCCs of a graph are unique. They describe a partition of the set of nodes with label 0 that are included in at least one cycle Z with $\sum_{v \in Z} l(v) < 2$. Exactly these nodes are the candidates for the insertion of further test cells. The TCCs do not contain any node with label 2. Nodes with label 1 may be included in more than one TCC. If a graph does not contain any nonzero labels, its TCCs and its SCCs agree.

4. Branch and bound search

The problem MCP is solved by a depth first search algorithm that is applied to each SCC of G separately. During the search, a tree is built whose nodes represent TCCs. The root node contains an SCC where all the nodes are labeled with 0 and for each node v the set of admissible labels, L(v), contains all the possible labels 0, 1, and 2. In the first level of the search tree, a node v_a is labeled using all the admissible labels from $L(v_a)$. The assignment of a label can divide the SCC into smaller TCCs (second level). Next a second node v_b is labeled (third level) and so on. Figure 9 shows an example.

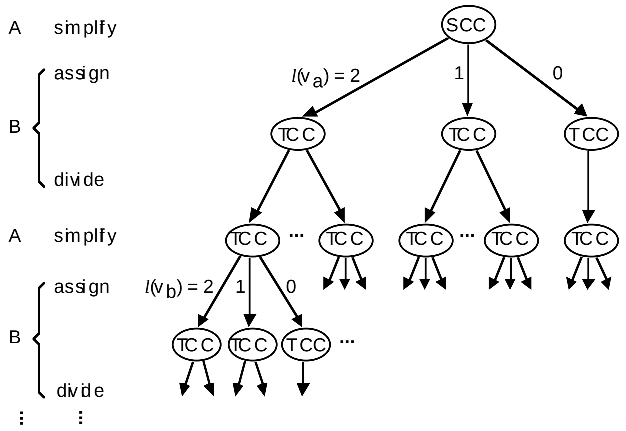

Figure 9: Search tree for a single SCC

The search is implemented by two alternating procedures. Procedure A restricts the admissible labels for the nodes and tries to simplify the graph by local transformations. Procedure B selects a node v that has not yet been considered and tries all the admissible values $l(v) \in L(v)$. When a specific label is assigned to node v, the set of admissible labels, L(v), is restricted to this single label. After the assignment, procedure B tries to divide the TCC into smaller ones. Then procedure A is called for each of the resulting (smaller) TCCs.

4.1 Procedure A

In procedure A, the following rules are applied until no more changes are feasible. An optimal solution for the modified graph is still an optimal one for the original graph.

Rule (i):
If v is a combinational node with $l(v) = 0$,
indegree(v) = 1 or outdegree(v) = 1,
and v is not part of a cycle with only one or two nodes,
then ignore node v, i.e. remove v and replace every pair of edges (w', v), (v, w") by an edge (w', w").

Rule (ii):
If v is a sequential node with $l(v) = 0$,
indegree(v) = outdegree(v) = 1,
and its direct predecessor and successor are two sequential nodes,
and v is not part of a cycle with only one or two nodes,
then ignore node v.

By local inspection some labels of nodes can be excluded as they cannot lead to an optimal solution:

Rule (iii):
If there is a self-loop (v, v),
then the only admissible label for node v is 2, $L(v) := L(v) \setminus \{0,1\}$.

Rule (iv):
If there is a cycle with two nodes, (v, v', v),
and $l(v') = 1$,
then label 0 is not admissible for v, $L(v) := L(v) \setminus \{0\}$.

As far as cycle breaking is concerned, increasing the label of a node v from 0 to 1 is equivalent to adding 1 to all the direct predeccessors of v that are labeled with 0 or 1. It is also equivalent to adding 1 to all the direct successors of v that are labeled with 0 or 1. Consequently, for a minimum cost labeling the node v must not get the label 1 if one of the equivalent options is less expensive.

We define *cost1*(v) and *cost2*(v) as the costs for labels l(v) = 1 and l(v) = 2, respectively, and *in1*(v) as the cost for adding 1 to all the direct predecessors of v that are labeled with 0 or 1. *out1*(v) is the cost for adding 1 to all the direct successors of v that are labeled with 0 or 1.
An admissible labeling is restricted by the following rule:
Rule (v):
If l(v) = 0 and *cost1*(v) $\geq$ min {*in1*(v), *out1*(v)},
then label 1 is not admissible for v, L(v) := L(v) \ {1}.

If later the labels of some predecessors or successors of v are increased from 0 to 1, this restriction will still hold.

Similarly, increasing the label of a node v from 0 to 2 can be considered. Let *in2*(v) be the total cost for increasing the labels of all the direct predecessors of v to 2, and let *out2*(v) be the total cost for increasing the labels of all the direct successors of v to 2. Then analogously to rule (v) the following rule is established:
Rule (vi):
If l(v) = 0 and there is not a self-loop (v, v),
and *cost2*(v) $\geq$ min {*in2*(v), *out2*(v)},
then label 2 is not admissible for v, L(v) := L(v) \ {2}.

After rules (v) and (vi) have been applied, it is possible that for some nodes the only admissible label is 0. If these nodes are involved in self-loops or have a label different from 0, a solution with this (partial) labeling does not exist, and the search tree can be pruned (see subsection 4.3). Otherwise these nodes can be eliminated from further consideration by the following rule.
Rule (vii):
If L(v) = {0} and there is not a self-loop (v, v),
then ignore node v.

After all possible restrictions and simplifications have been made, procedure B follows.

4.2 Procedure B

Procedure B selects a node v that has not yet been considered for labeling (hence with label 0) and tries all the admissible assignments $l(\mathrm{v}) \in L(\mathrm{v})$. The node v is selected by the following criteria:

- The number of admissible labels for v, $|L(\mathrm{v})|$, should be minimum.
- Setting $l(\mathrm{v}) = 2$ should make it possible to divide the TCC into smaller TCCs, and the largest of these smaller TCCs should contain a minimal number of nodes.

The first of these two points is most important. Of course, selecting a node v with $|L(\mathrm{v})| = 1$ is best since no decisions are required that may have to be reversed later.

If the newly assigned label is 1 or 2, it is tried to divide the TCC into smaller TCCs, and procedure A is called for each of these. If the new label is $l(\mathrm{v}) = 0$, the TCC cannot be divided. In this case we set $L(\mathrm{v}) := \{0\}$, and when procedure A is called, the application of rule (vii) reduces the graph by ignoring node v.

4.3 Pruning the search tree

At each node of the search tree the cost of the current labeling is computed. If this sum (*current_cost*) equals or exceeds the value obtained by the best solution found so far (*best_cost*), sons of this node in the search tree need not be investigated and the search tree can be pruned. Also if there is a node v with l(v) $\neq$ 0 and l(v) $\notin$ L(v) or a node v with no admissible label, L(v) = Ø, the current (partial) labeling cannot be extended to a minimum cost labeling, and the search tree can be pruned at this point. Using these criteria for pruning, most of the search space may be skipped while it is still guaranteed that an optimal solution will be found.

After pruning, backtracking may be necessary. Starting from the current node, the search tree is traversed backward until a node is encountered where procedure B made a choice among several possible assignments of labels. A different assignment is made, and the search algorithm continues by again calling procedure A and procedure B alternatingly.

As the underlying problem is NP-hard, some circuits might be intractable by the exact algorithm. In this case, good suboptimal solutions are obtained by a heuristic approach. A parameter $quality \leq 1$ is introduced. This factor is used during two steps of the algorithm. In procedure A, the rules (v) and (vi), which restrict the admissible labels, are modified to
Rule (v'):

If $l(\mathrm{v}) = 0$
and *cost1*(v) $\geq$ min {*in1*(v), *out1*(v)} * *quality*,
then label 1 is not admissible for v, $L(\mathrm{v}) := L(\mathrm{v}) \setminus \{1\}$.

Rule (vi'):

If $l(\mathrm{v}) = 0$ and there is not a self-loop (v, v),
and *cost2*(v) $\geq$ min {*in2*(v), *out2*(v)} * *quality*,
then label 2 is not admissible for v, $L(\mathrm{v}) := L(\mathrm{v}) \setminus \{2\}$.

The second step where this parameter is applied occurs during pruning the search tree. Here the tree is pruned if *current_cost* > *best_cost* * *quality*. *quality* = 1 yields an optimal placement. Usually the costs of the suboptimal solutions are distinctly below the limit \F(*optimal_cost*;*quality*^{2}) as shown in the next section.

5. Experimental results

The presented algorithm has been applied to the ISCAS'89 benchmark circuits [2]. The minimum cost placement strongly depends on the hardware overheads associated with the built-in test cells:

c_B:	cost of replacing a D-flip-flop by a BILBO cell
c_{Bt}:	cost of inserting a transparent BILBO cell
c_C:	cost of replacing a D-flip-flop by a CBILBO cell
c_{Ct}:	cost of inserting a transparent CBILBO cell

For simplicity, it is assumed that the costs c_B, c_C, c_{Bt}, and c_{Ct} do not depend on the locations in the netlist. Technology, design style, and the cell library have an impact on the cost distribution.

For $c_B + c_{Bt} < c_C$, $2c_{Bt} < c_{Ct}$, a minimum cost solution does not have any CBILBO cells since a pair of BILBO cells is always more favorable than a single CBILBO cell. For $c_C < c_B$, $c_{Ct} < c_{Bt}$, an optimal solution does not have any BILBO cells. In this case the problem reduces to determining a minimum feedback vertex set.

The most interesting situations occur when $c_B < c_C < c_B + c_{Bt}$ and $c_{Bt} < c_{Ct} < 2c_{Bt}$ hold. In order to get realistic values, the hardware overheads were estimated for two different design styles. The resulting cost distributions are

parameter set I: $c_B = 10$, $c_C = 25$, $c_{Bt} = 30$, $c_{Ct} = 40$

parameter set II: $c_B = 10$, $c_C = 35$, $c_{Bt} = 30$, $c_{Ct} = 55$

The experiments have been performed in order to validate the presented algorithm, and also in order to investigate optimal self-testable structures.

5.1 Validation

For the validation of the algorithm, we are interested in provably optimal solutions, computing times, and the impact of the factor *quality* on the costs of the found solutions. The solutions and computing times on a SUN Sparc 10 workstation are listed in table 1 for parameter set I, and in table 2 for parameter set II. The first column denotes the circuit, and #B, #Bt, #C, and #Ct are the number of BILBO cells, transparent BILBO cells, CBILBO cells, and transparent CBILBO cells, respectively. "Cost"

circuit	#B	#Bt	#C	#Ct	cost	q	CPU sec
s27	-	-	1	1	65	1	<1
s208	-	-	8	-	200	1	<1
s298	-	-	14	-	350	1	<1
s344	-	-	15	-	375	1	<1
s349	-	-	15	-	375	1	<1
s382	-	-	15	-	375	1	<1
s386	-	-	6	-	150	1	<1
s400	-	-	15	-	375	1	<1
s420	-	-	16	-	400	1	<1
s444	-	-	15	-	375	1	<1
s510	-	-	6	-	150	1	<1
s526	-	-	21	-	525	1	<1
s526n	-	-	21	-	525	1	<1
s641	-	-	7	4	335	1	<1
s713	-	-	7	4	335	1	<1
s820	-	-	5	-	125	1	<1
s832	-	-	5	-	125	1	<1
s838	-	-	32	-	800	1	<1
s953	-	-	6	-	150	1	<1
s1196	-	-	-	-	0	1	<1
s1238	-	-	-	-	0	1	<1
s1423	-	-	71	-	1775	1	<1
s1488	-	-	6	-	150	1	<1
s1494	-	-	6	-	150	1	<1
s5378	2	-	29	-	745	2/3	1
s9234	-	-	152	-	3800	2/3	14
s13207	-	-	308	1	7740	1	17
s15850	6	-	438	-	11010	1	4
s35932	36	-	288	-	7560	1	31
s38417	4	-	1048	8	26560	1	28
s38584	3	-	1094	9	27740	2/3	500

Table 1: Test cell placement for parameter set I

is the sum of the overheads for these cells, and q is the value of the parameter *quality* where this solution is found. The last column gives the computing time.

With the cost distribution of parameter set I, only a small number of BILBO cells is used, and mainly CBILBO cells are placed. Parameter set II increases the costs for CBILBOs, results are shown in table 2.

It is seen that the efficiency of the algorithm depends on the cost distribution. For parameter set I, an optimal solution is found for nearly all circuits with the exception of three whereas with parameter set II five circuits are hard to handle. Moreover, in all these hard cases the value of the *quality* factor, q, is lower than in table 1.

The impact of q on the costs is investigated in table 3 where the costs are compared for different values of q. For

circuit	#B	#Bt	#C	#Ct	cost	q	CPU sec
s27	2	1	1	-	85	1	<1
s208	-	-	8	-	280	1	<1
s298	-	-	14	-	490	1	<1
s344	-	-	15	-	525	1	<1
s349	-	-	15	-	525	1	<1
s382	-	-	15	-	525	1	<1
s386	-	-	6	-	210	1	<1
s400	-	-	15	-	525	1	<1
s420	-	-	16	-	560	1	<1
s444	-	-	15	-	525	1	<1
s510	-	-	6	-	210	1	<1
s526	-	-	21	-	735	1	<1
s526n	-	-	21	-	735	1	<1
s641	8	4	7	-	445	1	<1
s713	8	4	7	-	445	1	<1
s820	-	-	5	-	175	1	<1
s832	-	-	5	-	175	1	<1
s838	-	-	32	-	1120	1	<1
s953	-	-	6	-	210	1	<1
s1196	-	-	-	-	0	1	<1
s1238	-	-	-	-	0	1	<1
s1423	-	-	71	-	2485	1	1
s1488	-	-	6	-	210	1	<1
s1494	-	-	6	-	210	1	<1
s5378	-	-	30	-	1050	1/3	<1
s9234	-	-	46	106	7440	1/3	4
s13207	-	-	310	-	10850	1/2	22
s15850	6	-	438	-	15390	1	4
s35932	36	-	288	-	10440	1	29
s38417	11	4	1049	4	37165	1/2	211
s38584	52	9	1079	-	38555	1/2	549

Table 2: Test cell placement for parameter set II

values q ≥ 0.5 the heuristic solutions are very close to the minimum cost solution found by q = 1, but for q < 0,5 there is a distinct loss in quality for some circuits.

In another experiment the algorithm has been applied to the S-graphs of the circuits. If the costs for CBILBO cells are set less than the costs for BILBO cells, $c_C < c_B$, then the MFVS-problem for implementing a partial scan path is solved. The algorithm is not tuned to handle S-graphs and the MFVS-problem, as it is designed for a much more general problem. But for all the benchmark circuits it found the provably optimal solution of the MFVS-problem within a few minutes of computing time.

Some authors propose not to break self-loops for a partial scan design [3]. This problem can be solved by removing the self-loop edges from the S-graph. Also for

circuit	q = 1	q = 5/6	q = 4/6	q = 3/6	q = 2/6
s1423	2485	2485	2485	2485	2645
s1488	210	210	210	210	210
s1494	210	210	210	210	210
s5378	-	-	-	-	1050
s9234	-	-	-	-	7440
s13207	-	-	-	10850	12755
s15850	15390	15390	15390	15390	19335
s35932	10440	10440	10440	10440	10710
s38417	-	-	-	37165	46090
s38584	-	-	-	38555	39410

Table 3: Costs for solutions found with parameter set II and different values of the *quality* factor

these modified graphs the algorithm provides an optimal solution for nearly all the benchmark circuits with the exception of s15850 and s38584 where the factor *quality* must be reduced. The optimal solutions found agree with the numbers reported in [4].

5.2 BIST configurations

The solutions for the ISCAS'89 circuits give new insight in the structure of self-testable circuits with a "test per clock" technique and minimal hardware overhead. In most of the circuits only CBILBO cells have to be placed, and in the remaining circuits only very few BILBO cells are required. We assumed a CBILBO cell up to 3.5 times more expensive than a BILBO cell, but the overall hardware overhead of a CBILBO based approach is still smaller than a BILBO solution.

CBILBOs have even more advantages. As they generate patterns and compact responses concurrently, a single test session is sufficient and the test application time is shortened. This makes self-test control very simple, and hardware is saved for both the self-testable data path and the test control logic.

6. Conclusion

An exact algorithm has been presented that selects flip-flops to be incorporated into a multi-mode test register or into a partial scan path in order to implement a "test per clock" or a "test per scan" scheme with minimum hardware overhead. The algorithm finds provably optimal solutions for nearly all the ISCAS'89 benchmark circuits, and the remaining circuits can be handled by a heuristic version very efficiently. It also considers test cells within the combinational logic, and takes into account that the hardware costs of test cells depend on their type. The MFVS problem of partial scan design is solved as a special case.

In a "test per clock" scheme the hardware minimal solution consists mainly of CBILBO type test cells such that also test scheduling and test control are simplified.

7. References

[1] L. Avra: "Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths", International Test Conference, 1991, pp. 463-472, 1991

[2] F. Brglez, D. Bryan, K. Kozminski: "Combinational Profiles of Sequential Benchmark Circuits", Int. Symp. on Circuits and Systems, 1989, pp. 1929-1934

[3] K.-T. Cheng, V. D. Agrawal: "A Partial Scan Method for Sequential Circuits with Feedback", IEEE Trans. on Computers, Vol. 39, No. 4, April 1990, pp. 544- 547

[4] S. T. Chakradhar, A. Balakrishnan, V. D. Agrawal: "An Exact Algorithm for Determining Partial Scan Flip-Flops", Design Automation Conf., San Diego, 1994

[5] S. Chiou, C. Papachristou, H. Harmani: "A data path synthesis method for self-testable designs", Design Automation Conference, 1991, pp. 378-384

[6] M. R. Garey, D. S. Johnson: "Computers and Intractability", Freeman, New York, 1979

[7] R. Gupta, R. Gupta, M. A. Breuer: "The BALLAST Methodology for Structured Partial Scan Design", IEEE Transactions on Computers, April 1990, pp. 538-544

[8] S. Das Gupta, R. G. Walther, E. B. Eichelberger, T. W. Williams: "An Enhancement to LSSD and Some Applications of LSSD in Reliability, Availability and Serviceability", FTCS, 1981, pp. 32-34

[9] C. L. Hudson, G. D. Peterson: "Parallel Self-Test with Pseudo-Random Test Patterns", International Test Conference, Washington DC, 1987, pp. 954-963

[10] B. Koenemann, J. Mucha, G. Zwiehoff: "Built-In Logic Block Observation Techniques", Test Conference, Cherry Hill NJ, 1979, pp. 37-41

[11] A. Krasniewski, S. Pilarski: "Circular Self-Test Path: A Low Cost BIST Technique for VLSI Circuits", Transactions on CAD, Jan. 1989, pp. 46-55

[12] A. Kunzmann, H.-J. Wunderlich: "An analytical approach to the partial scan problem", J. of Electronic Testing: Theory and Applications, 1990, pp. 163-174

[13] T.-C.Lee, N. K. Jha, W. H. Wolf: "Behavioral synthesis for easy testability in data path scheduling", Design Automation Conference, 1993, pp. 292-297

[14] D. H. Lee, S. M. Reddy: "On Determining Scan Flip-Flops in Partial-Scan Designs", International Conference on CAD, 1990, pp. 322-325

[15] A. Mujumdar, R. Jain, K. Saluja: "Behavioral Synthesis of Testable Designs", International Symposium on Fault-Tolerant Computing, 1994, pp. 436-445

[16] M. J. Ohletz, T. W. Williams, J. P. Mucha: "Overhead in Scan and Self-Test Designs", International Test Conference, 1987, pp. 460-470

[17] L.-T. Wang, E. J. McCluskey: "Concurrent Built-in Logic Block Observer (CBILBO)", Int. Symposium on Circuits and Systems, 1986, pp. 1054-1057

[18] H.-J. Wunderlich: "Self Test Using Unequiprobable Random Patterns", FTCS, 1987, pp. 258-263