Initialization of Partially LBISTed Sequential Circuits

Isabelle Vogel, Marie-Lise Flottes, Christian Landrault

Initialization of Partially LBISTed Sequential Circuits [1]

I. Vogel, M.-L. Flottes, C. Landrault
Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier
LIRMM
161 rue Ada, 34392 Montpellier Cedex 5, France
{vogel flottes landraul}@lirmm.fr

Abstract:

Full scan is the most widely accepted and used DfT approach for large sequential machines. Nevertheless, in very dedicated cases it cannot be used mainly due to performance reasons as for example in high performance deeply pipelined CPU units. In this case full scan approach has to be replaced by partial scan. When trying to apply LogicBIST on partially scanned machines, the initialization problem of non-scan elements has to be solved. In this paper, we propose a nearly optimal algorithm to obtain a minimum set of memory elements to be initialized enabling to solve this initialization problem.

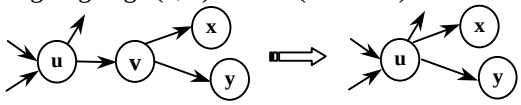
I. Introduction

It is demonstrated that cyclic structures are responsible for initialization problems. Thus, our approach consists in selecting at least one memory element per cycle in the circuit and to transform this element into an equivalent one with state initialization facility. In order to lower the impact of the initialization hardware on BIST overhead, the technique must lead to break all cycles with a minimum number of transformations. This problem, called the Minimum Feedback Vertex Set (MFVS), is known as NP-hard.
We propose a parameter-driven heuristic that can handle circuits with a large number of memory elements.
First, the circuit is modeled by a digraph G(V,E). V is the set of vertices representing the set of FFs and E is the set of edges representing the combinational paths between them. Then, several pre-process reduction tasks preserving the MFVS are applied on the graph. Finally a heuristic is used to solve the MFVS problem on the reduced graph.

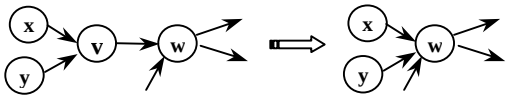
II. Graph reduction

The first pre-processing task consists in partitioning the graph into Strongly Connected Components (SCCs).
Each SCCs can be processed independently.

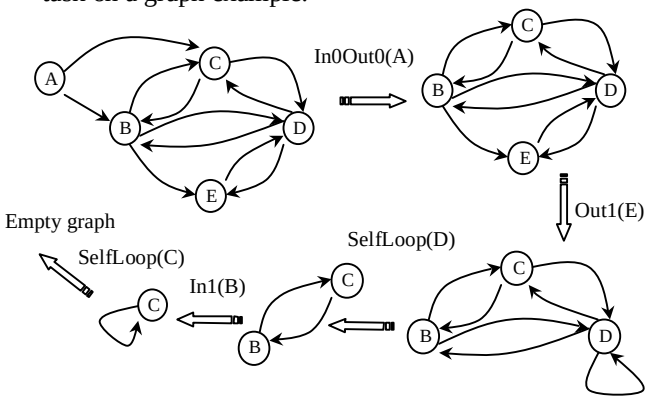
The MFVS of an initial graph G is the sum of the MFVSi computed on every SCCi in G.
Then we define 3 rules able to reduce the graphs.
Rule 1: **In0Out0(G(V,E))**
A vertex v_i without incoming or outgoing edge cannot belong to a cycle. The vertex v_i and its adjacent edges can be eliminated.
Rule 2**: In1(G(V,E))**
A vertex v in V such that there is one and only one incoming edge (u,v) to v, then every cycle including v necessarily includes the vertex u. The vertex v can be eliminated. In order to maintain all the paths via v, the outgoing edges from v are replaced by outgoing edges from u. The same stands for a vertex v such that there is one and only one outgoing edge (v,w) from v (rule 2bis).



Rules 2(bis): **Out1(G(V,E))**



Rule 3: **SelfLoop(G(V,E),MFVS)**
This rule removes all self-loops vertices from the graph and put them automatically into the solution set (MFVS).
Those three rules are applied iteratively on each SCC until there is no more possible reduction.
Next figure shows an application of this pre-process task on a graph example.



Note that if a sub-graph Gi is empty at the end of the pre-process, the MFVS problem is solved for

[1] This work was supported by Intel Corporation

this partition and the solution is optimal. Conversely, if a sub-graph cannot be reduced to an empty graph, this means that cycles are still present in this partition and have to be detected and broken. To do so, we use the heuristic defined below.

III. The GRASP Heuristic

The proposed method is issued from the meta-heuristic: **G**reedy **R**andomized **A**daptive **S**earch **P**rocedure (GRASP) proposed in[3]. It is composed of two main steps. The first one, `ConstructSol`, builds up a solution -a set of nodes to break all the cycles-. This initial solution is not necessarily optimal. Then a second procedure, `LocalSearch`, explores the solution space from this initial solution in order to find a local minimum. These two procedures are run several times in order to explore different parts of the solution space. The CPU time increases with the user defined number of iteration (`MaxIter`). The variable `BestSol` stores the best solution computed so far.

```
GRASP(G,MaxIter)
  BestSol=∅;
  For k=1 to MaxIter
    InitSol=ConstructSol(G,α,clkseed);
    Sol=LocalSearch(InitSol);
    BestSol=UpdateSol(BestSol,Sol);
end for;
```

a) Construction of an initial solution

The list of FFs to include in the initial solution is built up iteratively. The first task consists in selecting the best vertex candidates for breaking all the cycles (`MakeCL`). Then, the vertex v_i to include in the initial solution is randomly selected among this list CL (`RandomSelect`). After each selection, the graph is updated by removing the selected node (`UpdateGraph`). Finally, the procedure `ReduceGraphSize` containing all previously defined rules is applied before the next selection. The process iterates until the graph is empty.

```
Solution ConstructSol(G,α,clkseed)
   InitSol=∅;
   While G≠∅
       CL=MakeCL(G,α);
       vertex=RandomSelect(CL,clkseed);
       InitSol=InitSol∪{vertex};
       UpdateGraph(G);
       ReduceGraphSize(G,InitSol);
   end while;
   return InitSol
```

The candidate list CL is build up according to a gain representing the ability for a vertex to cut many cycles. The function used to evaluate this gain is: $F(v_i) = In(v_i) * Out(v_i)$. A vertex v_i is included in the list CL if $\mathbf{F(v_i) \in [Fmin + \alpha(Fmax-Fmin), Fmax]}$ where α is a user tuning parameter. If α=1, the algorithm is purely greedy. If α=0, the selection is purely random.

When the graph includes few vertices, it is recommended to chose α closed to zero and a high `MaxIter` value in order generate the optimal solution. Conversely, it is recommended to chose α closed to one when the graph includes numerous vertices.

b) Local search in the initial solution proximity

The exploration of the solution space consists in checking if every vertex in the initial solution is redundant or not. The `LocalSearch` procedure returns a local minimum.

```
Solution LocalSearch(InitSol)
   For every vertex v_i in InitSol
      G'=New(G,v_i,InitSol);
      NewReduceGraphSize(G');
      If G'=∅
         Remove(v_i,InitSol)
   end for
   return InitSol
```

IV. Experimental results

Experimental results have been performed on some ISCAS'89 benchmarks with α=0.5 and MaxIter=1.

Circuit	FFs	MFVS opti [2]		Our solution	
		(#FFs)	CPU (sec)	(#FFs)	CPU (sec)
s5378	179	30	13.4	30	0.27
s9234	228	152	1.0	152	0.21
s13207	669	310	3.7	310	0.5
s15850	597	441	8.4	441	2.81
s35932	1728	306	12.6	306	0.97
s38417	1636	1080	44.6	1080	3.84
s38584	1452	1115	37.1	1115	24.93

Table 1: MFVS experimental results

Columns 5 & 6 report our results. We obtain the optimal solution in shorter CPU time.

V. Conclusion

We have described an algorithm enabling to solve the problem of the initialization of non-scan memory elements in partial scan approach. Based on a minimum edge cutting algorithm, this algorithm provides nearly optimum results in acceptable CPU time. In a very near future, it will be used to solve initialization problems on deeply pipelined CPU units provided by INTEL in the framework of this collaboration work.

References

[1] S. T. Chakradhar A. Balakrishnan V. Agrawal, "An Exact Algorithm for Selecting Partial Scan Flip-Flops", Design Automation Conf., 1994, pp 81-86.

[2] G. Kiefer H.J Wunderlich, "Deterministic BIST with Partial Scan", Journal Of Electronic Testing: Theory and Applications (JETTA), June 2000, vol 16, n°3, pp 169-177.

[3] P. M. Pardalos T. Qian M.G.C. Resende. A Greedy Randomized Adaptative Search Procedure for the Feedback Vertex Set. Journal of Combinatorial Optimization, 2:399-412, 1999.