VLSI DESIGN
1998, Vol. 7, No. 2, pp. 131–141
Reprints available directly from the publisher
Photocopying permitted by license only

© 1998 OPA (Overseas Publishers Association)
Amsterdam B.V. Published under license
under the Gordon and Breach Science
Publishers imprint.
Printed in India.

SCOAP-based Testability Analysis from Hierarchical Netlists

C. P. RAVIKUMAR[a,*] and H. JOSHI[b]

[a]*Department of Electrical Engineering, Indian Institute of Technology, New Delhi 110016, India;*
[b]*Synopsys, Inc., Milipitas, CA*

(Received 27 July 1995)

Circuits of VLSI complexity are designed using modules such as adders, multipliers, register files, memories, multiplexers, and busses. During the high-level design of such a circuit, it is important to be able to consider several alternative designs and compare them on counts of area, performance, and testability. While tools exist for area and delay estimation of module-level circuits, most of the testability analysis tools work on gate-level descriptions of the circuit. Thus an expensive operation of flattening the circuit becomes necessary to carry out testability analysis. In this paper, we describe a time and space-efficient technique for evaluating the well known SCOAP testability measure of a circuit from its hierarchical description with two or more levels of hierarchy. We introduce the notion of *SCOAP Expression Diagrams* for functional modules, which can be precomputed and stored as part of the module data base. Our hierarchical testability analysis program, HISCOAP, reads the SCOAP expression diagrams for the modules used in the circuit, and evaluates the SCOAP measure in a systematic manner. The program has been implemented on a Sun/SPARC workstation, and we present results on several benchmark circuits, both combinational and sequential. We show that our algorithm also has a straightforward parallel realization.

Keywords: Hierarchical testing, testability analysis, SCOAP, hierarchical netlist

1. INTRODUCTION

Testability-driven synthesis of digital systems requires a method to measure the testability of a circuit given its structural description. Thus various architectures for the same circuit can be compared and less testable designs can be rejected in favor of the more testable ones. High-level synthesis has traditionally considered cost functions such as area and performance. However, in the recent past, much effort has been expended on testability-oriented synthesis [9, 1, 6]. Many fast algorithms for estimation of area and performance have been reported in the literature; these estimation tools can easily take advantage of the hierarchy in the circuit to reduce the computation in arriving at an estimate. However, no testability analysis tool which works on a hierarchical

*Corresponding author.

description of the circuit is hitherto known. Since VLSI systems are built using building blocks such as adders, multipliers, register files, memories, multiplexers, and busses, traditional gate-level testability analysis tools will not be very useful for testability estimation. Flattening a hierarchical description to the gate-level is an expensive process, and is prohibitively expensive if it must be iterated for each alternate design.

Present work on testability-driven synthesis uses simple measures for testability, such as the number of self-loops in the structure graph of the circuit. The disadvantage of such a measure is that it is specific to the style of testing; thus, self-loops reduce the testability of a circuit when built-in self-test (BIST) is employed. Technology-independent testability analysis tools such as SCOAP [3], PREDICT [10], and STAFAN [4] would be more appropriate when the testability of the circuit must be gauged without committment to the test style. SCOAP [3] gives integral numerical estimates of the controllability and observability of signal lines in a given circuit; a low SCOAP value indicates a high testability and vice versa. PREDICT uses signal robabilities as estimates of controllability and observability [10]. STAFAN [4] estimates these probabilities through a statistical sampling of input vectors, and can be used to predict the fault coverage of a random test set without actually performing a fault simulation. A recently published comparative analysis of various testability analysis tools indicates that SCOAP gives more authentic measures of testability than many other estimation tools [2].

SCOAP has been effectively used as a testability analysis tool in solving the Partial Scan Design problem in [8]. The partial scan design problem is to determine the smallest subset of flip-flops which, when scanned, would give the maximum fault coverage; the SCOAP measure of the scanned circuit gives a good estimate of this fault coverage. Techniques such as Simulated Annealing [8] and Genetic Algorithms [7] have been used for solving the scan selection problem. In these algorithms, SCOAP measure is used as a part of the cost function that estimates the improvement in fault coverage by modifying the scan set. Since SCOAP is used repeatedly in such applications, it is desirable to have an efficient implementation of the algorithm. In this paper, we describe HISCOAP – a hierarchical implementation of SCOAP that operates on a hierarchical netlist description of the circuit. Our algorithm is efficient both in space as well as in computation time.

In the following section, we explain the theory behind HISCOAP algorithm, which involves the concept of SCOAP Expression Diagrams (SED). With an n-input, m-output logic module, a naive approach would require the storage of

- $2m$ SEDs, corresponding to the zero and one controllabilities of each primary output.
- n SEDs, required for computing the observabilities, one for each primary input.

We show that space optimization can be achieved by merging the $2m+n$SEDs into two, one for controllability and another for observability. We also examine other optimizations in memory space that are possible in storing the SED.

The theory behind the HISCOAP algorithm is developed in Section 2 and an implementation of HISCOAP is discussed in Section 3. Performance results of HISCOAP on some benchmark problems are described in Section 4. Section 5 presents our conclusions.

2. SCOAP FOR MODULAR CIRCUITS

Goldstein introduced the SCOAP testability measure for both combinational and sequential circuits in [3]. With each signal line l in the circuit, we associate six testability measures, $CC_0(l)$, $CC_1(l)$, $CO(l)$, $SC_0(l)$, $SC_1(l)$, and $SO(l)$. $CC_0(l)$ is the combinational 0-controllability of the signal line l; it represents the number of combinational assignments that must be made to other nodes in the circuit in order to set l to logic 0. The higher the value of $CC_0(l)$, the poorer the 0-controllability

of l. $CC_1(l)$, the combinational 1-controllability, is similarly defined. The combinational observability $CO(l)$ is a measure of (a) the number of combinational nodes between the node l and the primary-outputs, and (b) the number of combinational assignments necessary to propagate the value on l to one of the primary outputs. $SC_0(l)$ is the sequential 0-controllability of line l, and represents the number of time frames that will be required to justify the node l to 0. $SO(l)$ is the sequential observability of node l, and is defined in a manner similar to $CO(l)$, except that we count the number of sequential nodes and the number of sequential node assignments.

For basic standard cells such as AND, OR, NAND, NOR and NOT, D flip-flops, and fanout points, Goldstein showed that the controllabilities of the gate output can be related to the controllabilities of the gate inputs through simple algebraic expressions. These expressions have two types of operators, namely the "min" operator and the "+" operator. As an example, the controllabilities of the output c of a two-input AND gate can be related to the controllabilities of its two inputs a and b as follows.

$$CC_0(c) = \min(CC_0(a), CC_0(b)) + 1 \quad (1)$$

$$CC_1(c) = CC_1(a) + CC_1(b) + 1 \quad (2)$$

The above equations can be compactly represented in the form of a matrix as explained in [3]. If we have to compute the controllabilities for signal lines in a larger circuit composed of basic gates, we apply the expressions for each individual gate. The starting point of the evaluation are the primary inputs, whose combinational controllabilities are defined to be 1. The SCOAP expressions are then iteratively applied until a convergence is obtained [3]. Observabilities are similarly computed. The reader is referred to [3] for a more detailed exposition of SCOAP.

When large circuits of VLSI complexity are involved, the above technique can be quite expensive both in terms of CPU time and memory requirement, the reason being that SCOAP requires a gate-level netlist description as input. In this paper, we overcome this problem by computing SCOAP measures from a hierarchical netlist. Hierarchical descriptions of circuits are commonly used in practice due to their compactness and their usefulness in mixed-mode simulation. The HIS-COAP algorithm described in this paper makes it unnecessary to flatten the hierarchical netlist into a gate-level netlist. The one-time overhead in our approach is in the form of a preprocessing step, which computes SCOAP Expression Diagrams for each module.

2.1. SCOAP Expression Diagrams for Combinational Circuits

For a single output combinational circuit on n primary inputs labelled $X_1, X_2, \ldots, X_n$, we define two types of SCOAP Expression Diagrams for the output Y. The 0-SED of line Y is a diagrammatic representation of the expression for $CC_0(Y)$. Refer to Figure 1. The 0-SED is a directed acyclic graph with two types of nodes, labelled 'min' and '+'. Each node corresponds to the output of a gate in the circuit, and is labelled '+' or 'min'. The label for a node x is decided based on the nature of the gate whose output is x, and the value to which x must be set in order to drive Y to 0. Each edge of the form (i,j) in the 0-SED is given a weight, equal to the r-controllability of line i, if the line i must be driven to logic level r in order to set Y to 0. In the example circuit of Figure 1, in order to set the output f to 0, one of the signals d or e must be driven to 1. Thus we label the node f as 'min'. Further, the edges (e, f) and (d, f) are given weights equal to $CC1(e)$ and $CC1(d)$ respectively. The remaining nodes are similarly labelled and the edges similarly weighted.

The 1-SED for an output Y of an n-input combinational circuit is defined in a manner similar to 0-SED. Figure 2 shows the 1-SED for the output f of the sample circuit of Figure 1.

We now explain how the 0-SED and 1-SED are useful in computing the controllabilities and

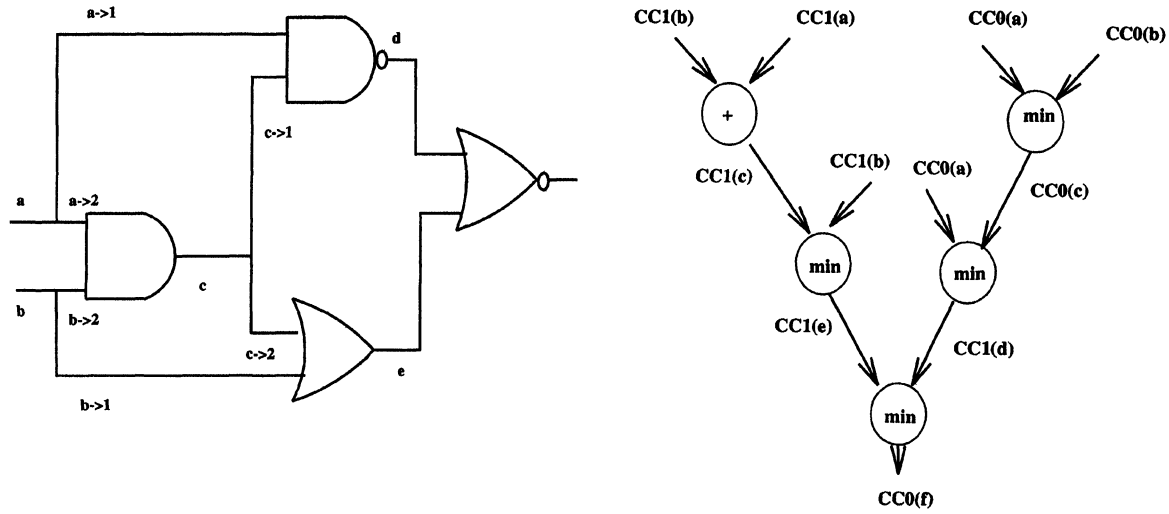


FIGURE 1 A sample circuit and the 0-SED for its output f.

observabilities of signal lines in a circuit. Let us first restrict ourselves to an n-input, 1-output combinational circuit. Given the 0-SED and 1-SED for the primary output Y, we can evaluate the SCOAP combinational controllabilities for all the lines by a simple top-down traversal of the 0-SED and the 1-SED. In the example of Figure 1, the calculations proceed as follows.

$$
\begin{aligned}
CC_1(b) &= 1 \\
CC_1(a) &= 1 \\
CC_0(a) &= 1 \\
CC_0(b) &= 1 \\
CC_1(c) &= 2 + \text{depth} \\
CC_0(c) &= 1 + \text{depth} \\
CC_1(e) &= \min(CC_1(b) + CC_1(c)) + \text{depth} \\
&= 1 + \text{depth} \\
CC_1(d) &= \min(CC_0(a) + CC_0(c)) + \text{depth} \\
&= 1 + \text{depth} \\
CC_0(f) &= \min(CC_1(e) + CC_1(d)) + \text{depth} \\
&= 1 + 2 \cdot \text{depth}
\end{aligned}
$$

The 1-SED of Figure 2 can be used similarly to compute the remaining controllabilities. Thus simple breadth-first traversal of the 0-SED and

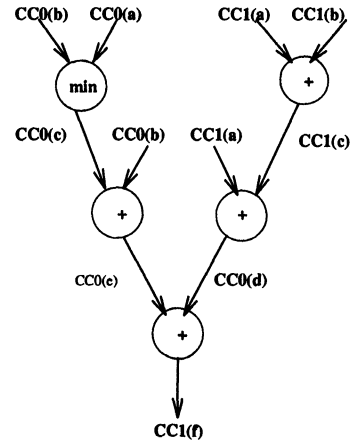


FIGURE 2 The 1-SED for output f of the sample circuit.

1-SED is sufficient to compute all the controllabilities in a combinational circuit.

2.1.1. *Storage Optimization*

We first note that both the 0-SED and 1-SED need not be stored in memory, since one can be obtained from the other through a *complement* operation. Complementing involves changing

```
procedure Control(Y)
(* Y is the output of a combinational circuit *)
begin
    Locate the 0-SED of Y;
    While traversing the 0-SED of Y top down
    begin
        Let x be the output of the current node;
        Let the label on edge x be CO_r(x);
        Compute CC_r(x);
    end
    Complement the 0-SED to obtain the 1-SED of Y;
    While traversing the 1-SED of Y top down
    begin
        Let x be the output of the current node;
        Let the label on edge x be CO_s(x); (* Note that s = 1 - r *)
        Compute CC_s(x);
    end
end
```

FIGURE 3 Computing SCOAP controllabilities from optimized SED.

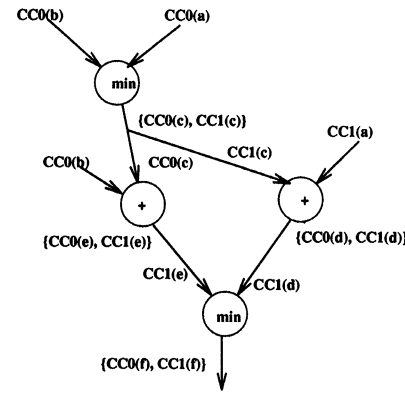


FIGURE 4 SED-C for primary output f of sample circuit.

every '+' label to a 'min', changing every 'min' label to a '+', changing the weight $CC_1(x)$ to $CC_0(x)$ and changing the weight $CC_0(x)$ to $CC_1(x)$. The above technique works for all circuits composed of NAND, NOR, AND, OR, and NOT gates only. For an example, the reader is referred to the 0-SED of Figure 1; complementing it using the rules mentioned above yields the 1-SED of Figure 2. When this optimization technique is used, the procedure of Figure 3 would suffice for computing all the SCOAP controllabilities.

Further storage optimization can be achieved in storing the 0-SED itself, by noting the following. If, for some signal x, both $CC_0(x)$ and $CC_1(x)$ appear as edge weights in the 0-SED, then effort is duplicated in computing them again from the 1-SED. We can eliminate this redundancy by deleting the subgraph supported on the edge weighted by $CC_1(x)$ from the 0-SED. To illustrate this point, we refer the reader to the 0-SED in Figure 1. Note that both $CC_1(c)$ and $CC_0(c)$ appear in the 0-SED. We can eliminate the nodes supported by $CC_1(c)$ to obtain the compact SED shown in Figure 4. However, we must add a fanout edge (marked $CC_1(c)$ in the Figure) to enable the computation of all the controllability values that depend on $CC_1(c)$. Using the compact 0-SED, the computation of controllability values is modified as shown in Figure 5. We refer to the compact SED as SED-C in this paper.

```
procedure Control-C (Y)
(* Y is the output of a combinational circuit *)
begin
    Locate the compact SED of Y;
    While traversing SED-C of Y top down
    begin
        Let x be the output of current node;
        compute CC_0(x);
        complement the node label;
        compute CC_1(x);
    end
end
```

FIGURE 5 Computation of controllabilities from SED-C.

We now consider the general case of an n-input, m-output combinational circuit and explain how the above optimizations can be used. Consider the compact SED-C for two primary outputs $Y1$ and $Y2$; there are controllability terms that are common to their SED-C. Instead of recomputing these controllabilities while traversing the SED-C of $Y2$, the controllability values can be passed on directly as inputs to the SED-C of $Y2$. To enable this, we create links from the SED-C of $Y1$ to SED-C of $Y2$.

2.2. SED for Observability Calculation

After the signal controllabilities have been computed, the signal observabilities can be calculated using algebraic expressions as discussed in [3]. Once again, we use an expression diagram for representing the set of observability expressions associated with a primary input X. For the input a

of the sample circuit in Figure 1, the relevant SED is shown in Figure 6. The SED-O is also an acyclic directed graph with nodes labelled '+' or 'min'. The edges of an SED-O are weighted with the appropriate controllability or observability value.

It may be observed that some observability values appear on more than one SED-O in a given circuit. For instance, $CO(e)$ and $CO(c)$, which are computed while traversing the SED-O for primary input a, may be directly used in traversing the SED-O for primary input b. In order to enable this reuse, we add links from the SED-O of a to SED-O of b. Thus, insome sense, the SED-O for all the primary inputs are threaded together into one observability expression diagram. Figure 7 shows the complete observability expression diagram for all the primary inputs of the sample circuit of Figure 1.

2.3. Sequential Circuits

The SCOAP expression diagrams SED-C and SED-O introduced in the previous section apply only to directed acyclic structure graphs. The same expression diagrams can be used for computing testability values for sequential circuits (namely, SC_1, SC_0, and SO); for this, the SED-C and SED-O are to be traversed iteratively until the testability values converge, as suggested in [3]. When extending the SED-C and SED-O to sequential circuits, we must bear in mind the following two points.

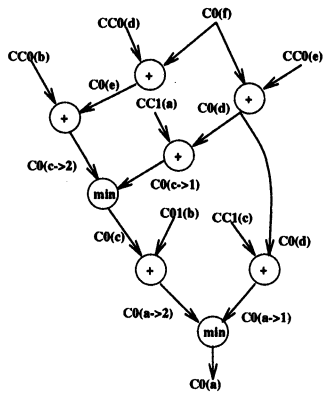


FIGURE 6 SED-O for primary input a of sample circuit. The notation $a->1$ is used to indicate the first fanout branch of signal a.

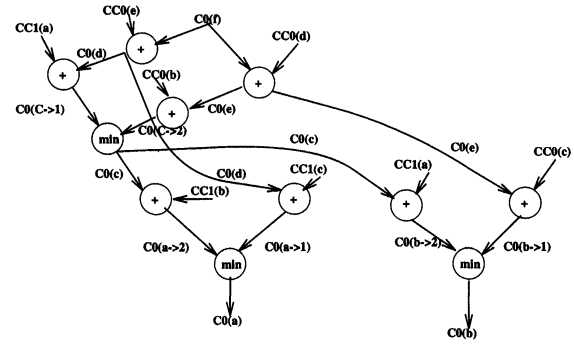


FIGURE 7 SED-O for the entire sample circuit.

- The flip-flops are represented by nodes that are labelled '*'. (In this paper, we shall assume that only D-type flip-flops are used as memory elements in the circuit. We also allow permit feedback paths without memory elements in the sequential circuit). In an SED-C, such a node receives the controllabilities of three input lines, namely, the D-input, the reset line, and the clock line; the node has a single output edge corresponding to the controllability value of the Q output. $SC_0(Q)$ and $SC_1(Q)$ can be related to the controllabilities of the three inputs D, *reset*, and *clock* using the expressions given in [3]. In an SED-O, a node with label '*' receives four input lines corresponding to the three controllabilities and one observability (namely SC_1(Clock), SC_0(Clock), and SC_0(Reset), $SO(Q)$).
- Due to feedback, there will be cycles in the resulting SED-C and SED-O. The testability values on the feedback lines are unknown to begin with, and must be initialized to ∞. Further, to make use of the top-down traversal algorithm introduced in the previous section, we need to break the cycles in the SED-C and SED-O by removing the edges.

A sequential circuit is shown in Figure 8 and the SED-C and SED-O for this circuit are given in Figure 9.

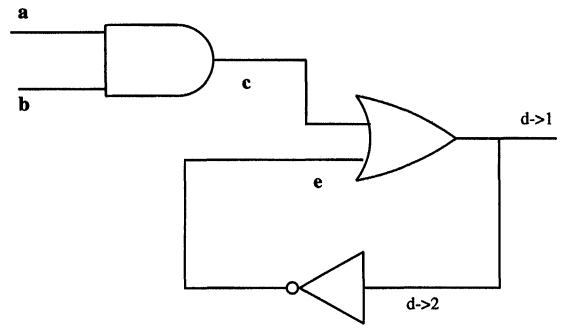


FIGURE 8 A sequential circuit.

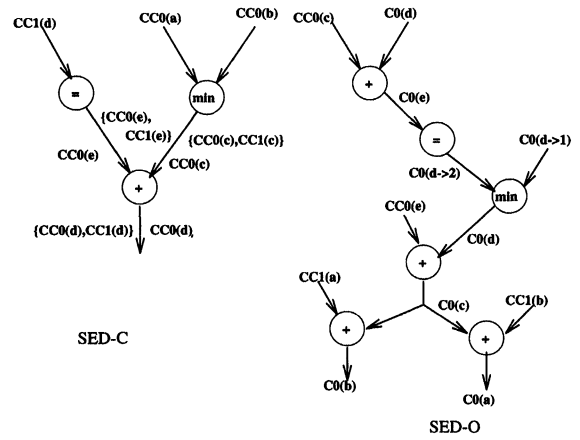


FIGURE 9 SED-C and SED-O for the example sequential circuit.

2.4. Storage Requirements

For a combinational circuit with n primary inputs, g gates each with a fan-in of f or less, m primary outputs, and I internal nodes, the SED-C contains $I+m$ nodes and no more than $f \cdot g$ edges. If the number of fanout branches is fo, The SED-O contains $I+n+fo$ nodes and no more than $f \cdot (I+n+fo)+n$ edges. Since a fixed amount of storage is required per node and per edge, we can conclude that the space complexity for combinational circuits is linear in the number of gates in the module. For a sequential circuit, the storage requirements for SED-C and SED-O are of the same order of magnitude.

3. HISCOAP ALGORITHM

Consider a hierarchical netlist description of a circuit such as an 8-bit array multiplier. The circuit is composed of multiplier cells connected using a regular interconnection pattern. In order to compute the SCOAP testability of such a modular design, we can take advantage of a two-level hierarchy in the description. The top level describes the interconnection among the multiplier cells, and the second level describes the gate-level netlist of a single cell. A single copy of SED-C and SED-O for the multiplier cell is sufficient for computing the SCOAP testability values for all the lines in the multiplier circuit. The HISCOAP algorithm works on the hierarchical netlist description of the multiplier. The overall HISCOAP algorithm is described below in pseduo-code (procedure HISCOAP-NonFeedBack). The algorithm uses breadth-first traversal for visiting all the modules in the circuit. This algorithm applies only to circuits without any feedback. (Note that a non-feedback circuit does not mean the circuit is purely combinational. The modules of the circuit may be sequential circuits such as shift registers, serial adders, etc. However, no feedback lines must exist from one module to another). It is easy to extend the algorithm of HISCOAP-NonFeedBack to the case when the circuit does have feedback lines. The alteration required is to initialize the controllability values on all the non-PI lines to ∞, initialize the observability values on all non-PO lines to ∞. Within the breadth-first traversal, we do not test for the condition that all input controllabilities are available or all output observabilities are available. The entire algorithm is repeated until the testability values on all the lines have converged to some value.

```
procedure HISCOAP-NonFeedBack ( )
begin
  for each type of module M in the circuit do
    if SED-C(M) or SED-O (M) not already
    available
      in module database then
```

```
        begin
          construct SED-C (M) and SED-O (M);
          enter the expression diagrams in module
          database;
        end
    Queue := NULL;
    for each primary input I in the circuit do
      add the module fed by I to Queue;
    while (Queue is not empty) do
    begin
      E := delete (Queue);
      if module instance E is colored then skip;
      if all input controllabilities to module E
      are available then
    begin
      color module instance E;
      Find the SED-C for module type of E;
      Using the input controllabilities of E,
      initialize the edge weights of SED-C;
      Traverse SED-C and find the controllabilities
        on internal nodes associated with E;
      Update the output controllabilities in the
        testability table;
      for each primary output O of module E
        add the module fed by O to Queue;
    end else
      add E to Queue;
end
Queue := NULL;
for each primary output O in the circuit do
  add the module whose output is O to Queue;
Remove the color on all module instances;
while (Queue is not empty) do
begin
  E := delete (Queue);
  if module E is colored then skip;
  if all output observabilities of module E
    are available then
  begin
      color module E;
      Find the SED-O for module type of E;
      Using the input controllabilities and
        output observability of E,
      initialize the edge weights of SED-O;
      Traverse SED-O and find the observabil-
      ities
        on internal nodes associated with E;
      Update the input observabilities in the
        testability table;
      for each primary input I of module E
        add the module which feeds I to Queue;
    end else
      add E to Queue;
  end
end
```

4. EXPERIMENTAL RESULTS

The HISCOAP algorithm described in the previous section has been implemented on a Sun/SPARC workstation in programming language C. The software is composed of about 3000 lines of code.

Both feedback and non-feedback circuits can be handled by our program. We have tested the program on a number of benchmark circuits such as

- *n*-bit Carry Look Ahead Adder (CLA) composed of 4-bit CLA modules. We use the notation add*n*_hl to describe these circuits.
- *n*-bit CLA composed of full-adders and Carry Lookahead circuitry. We use the notation add*n* to describe these circuits.
- *n*-bit array multiplier composed of multiplier cells. We use the notation mult*n*_hl to describe these circuits.
- Bit-serial adder composed of shift registers and a full adder.
- Some ISCAS 89 sequential benchmarks.

We have compared the performance of the HISCOAP algorithm with that of a gate-level implementation of SCOAP. In order to accomplish this, we have implemented a circuit flattener which uses macro expansion to convert a hierarchical description into a gate-level description.

Table I compares the running time of HISCOAP working on a hierarchical netlist with that of the conventional SCOAP algorithm working on a gate-level netlist. Several observations can be made from this table. First,the running time of the HISCOAP algorithm grows modestly with increasing circuit size. Thus, the execution time for a 64-bit CLA composed of 4-bit CLAs grows only by a factor of 3 in comparison to the running time for a 4-bit CLA. Thus, a 16-fold increase in circuit size does not reflect in the execution time of HISCOAP. This can be contrasted with the run time of the SCOAP algorithm, where the execution time increases by a factor of 87. Secondly, we note that HISCOAP offers a substantial reduction in execution time when compared to the SCOAP algorithm. For instance, the gate-level SCOAP algorithm requires 12.28 seconds (excluding the time for flattening the hierarchical description) for handling the circuit add32_hl, whereas HISCOAP takes only 0.33 seconds for the same circuit. The computational overhead in HISCOAP includes the reading of SED files and initialization of the SED data structures. Thus, if the circuit size is small in comparison to the module size, this overhead can overshadow the advantages of hierarchical implementation of SCOAP. For instance, HISCOAP shows a speedup of less than 1 for the circuit add4_hl. We conclude that HISCOAP is only meritorious when used with circuits which have a large number of copies of the same module. There is also an overhead of storage space in HISCOAP – the SED data structures must be stored for each type of module. This overhead will be negligible when the circuit has many copies of the same type of module. Finally, the complexity of each module also plays a crucial role in determining the execution time of HISCOAP. In the circuits add$n$_hl, the module is fairly complex by itself, namely, a 4-bit CLA. On the other hand, in the circuits add$n$, the module is a much simpler one, namely, a full adder. HISCOAP works best when the circuit has a number of copies of a complex module, as in the case of add32_hl (execution time is 300 ms, module is 4-bit CLA). In contrast, consider the add32 circuit, where the module is a full adder and the execution time is 500 ms. When there is only one copy of a complex module, as in the case of add4_hl, the execution time of HISCOAP is somewhat larger than that of gate-level SCOAP, due to overheads in maintaining the SCOAP expression diagrams.

Table II shows the results of HISCOAP on n-bit array multiplier circuits, $4 \leq n \leq 32$. When HISCOAP was executed on a gate-level description of the 4-bit multiplier, the execution time (without counting the time for flattening) was 330 ms. HISCOAP took 5450 ms to process a gate-level description of an 8-bit array multiplier. We do not have HISCOAP results for gate-level 16-bit and 32-bit array multipliers since the flattening operation took an excessively large amount of time and had to be aborted.

The core memory requirement of HISCOAP is substantially lower than that of SCOAP. This was clearly apparent in one of our experiments dealing

TABLE I Comparison of Execution Times of SCOAP and HISCOAP for CLA circuits. The time for flattering the hierarchical circuit description is not included

Circuit	Number of Modules	Number of Gates	Times (ms) (HISCOAP)	Time (ms) (SCOAP)
add4_hl	1	54	190	140
add8_hl	2	108	210	290
add12_hl	3	162	230	720
add32_hl	8	432	330	3510
add64_hl	16	864	580	12280
add4	5	54	100	140
add8	10	108	150	290
add12	15	162	200	720
add32	40	432	500	3510

TABLE II Results on n-bit array multiplier circuits

n	Number of Modules	Number of Gates	Execution Time (ms)	Memory (pages * ticks)
4	16	208	100	777
8	64	416	340	1719
16	256	3328	2440	13881
32	1024	13312	29000	175742

TABLE III Sample runs on sequential circuits

Circuit	Number of Modules	Types of Modules	Number of Standard Cells	Time (ms)	Memory (pages * ticks)
Serial Adder	4	3	47	100	1418
s27	1	1	13	80	731

with the 32-bit array multiplier. The gate-level description of the circuit demanded a large amount of core memory, leading to disk swapping, in turn leading to an excessively large run time. When we attempted to flatten the 32-bit multiplier circuit on an HP/9000 computer with 4 MB of main memory, the program failed to complete even after 4 hours of execution time. On the other hand, the HISCOAP algorithm was able to handle the 32-bit multiplier example in 29 seconds.

Although several ISCAS combinational and sequential benchmarks were available to us, we did not attempt extensive experimentation on these benchmarks since these circuits are gate-level circuits. To the best of our knowledge, no benchmark circuits are available in public domain for hierarchical testing. However, we constructed some fictitious circuits by duplicating two or more copies of circuits such as s27, and adding interconnections between them. Table III shows the results on two sequential circuits – a serial adder (using a full adder, 3 shift registers, and a flip-flop as modules) and the circuit s27 using NAND gates as modules.

5. CONCLUSIONS

We have presented a hierarchical implementation of the well known SCOAP testability analysis program. Our algorithm, called HISCOAP, has been implemented on a Sun/SPARC workstation and has been tested against a number of large benchmark examples. The hierarchical algorithm applies to both sequential and combinational circuits. The algorithm takes a hierarchical description of the circuit as input. Presently, a two-level hierarchy is employed in the hierarchical netlist description. It is easy to extend the ideas presented in this paper to handle multiple levels of hierarchy. Such an extension would imply that the SCOAP Expression Diagrams must themselves be implemented in a hierarchical fashion (See [5] for details). The HISCOAP algorithm gives a two-sided advantage over the conventional SCOAP algorithm; first, the run time of the HISCOAP algorithm is significantly smaller. Secondly, the memory requirement of the HISCOAP algorithm is substantially lower, which in turn gives it an advantage in terms of execution time due to reduced disk swap operations.

Acknowledgements

We thank the two anonymous referees whose feedback helped us in improving the first draft of this paper. Initial discussions with Mr. H. Rasheed were useful. We thank Dr. A. Muzumder and Dr. R. Jain for providing us with *gmalloc.c*. We are thankful to the Computing Services Center of IIT Delhi for providing the facilities to carry out this work.

References

[1] Avra, L. (1991). Allocation and assignment in high-level synthesis for self-testable data paths, In *Proceedings of International Test Conference*, pp. 463–472.

[2] Chandra, S. J. and Patel, J. H., Experimental evaluation of testability measures for test generation, *IEEE Transactions on CAD*, **8**(1), 93–97, January 1989.

[3] Goldstein, L. H. (1979). Controllability/observability analysis of digital circuits, *IEEE Transactions on Circuits and Systems*, **26**, 685–693.

[4] Jain, S. K. and Agarwal, V. D., Statistical fault analysis, *IEEE Design and Test of Computers*, pp. 38–44, February 1985.

[5] Joshi, H., Hiscoap: A hierarchical implementation of scoap testability measure for VLSI circuits, Master's thesis, Indian Institute of Technology, Delhi, December 1993.

[6] Papachristou, C. A. *et al.*, A framework for high-level synthesis with self-testability, Technical report, Computer Engineering and Science Department, Case Institute of Technology, Case Western Reserve University, February 1991, CES-91-03.

[7] Ravikumar, C. P. and Rajarajan, R., Genetic algorithms for scan design problems in vlsi circuits, Technical report, Department of Electrical Engineering, IIT Delhi, India, December 1993.

[8] Ravikumar, C. P. and Rasheed, H., Simulated annealing for target-oriented partial scan. In *Proc. of International Conference on VLSI Design*, January 1994, Calcutta, India.

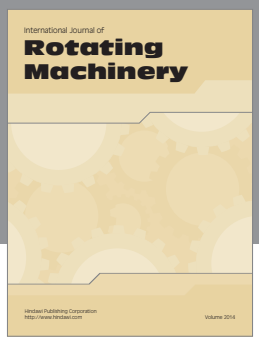
[9] Ravikumar, C. P. and Saxena, V. (1995). TOGAPS–a Testability Oriented Genetic Algorithm for Pipeline Synthesis, Accepted for publication in the International Journal of VLSI Design.

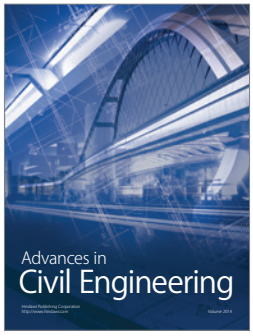
[10] Seth, S. C., Pan, L. and Agarwal, V. D. (1985). Predict–probabilistic estimation of digital circuit testability, In *Fault Tolerant Computing Symposium Digest of Papers*, pp. 220–225.

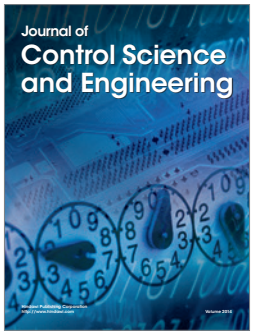
Authors' Biographies

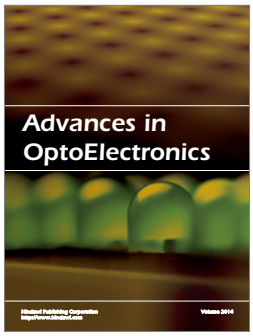
C. P. Ravikumar obtained his Ph.D. in Computer Engineering from the Department of Electrical Engineering Systems, University of Southern California, in 1991. Since then, he is an Assistant Professor in the Department of Electrical Engineering at the Indian Institute of Technology, Delhi. He is the Indian editor of the International Journal of VLSI Design and is on the editorial board of the *Computers and Informatics* journal. His research interests are in the areas of VLSI Design Automation and Testing, Combinatorial Optimization, Interconnection networks for multiprocessors, and parallel algorithms.

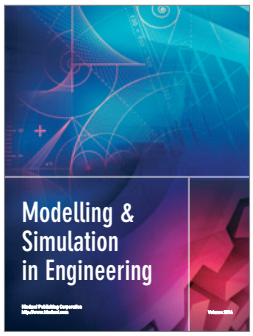
Hemant Joshi obtained his Master's degree in Computer Technology in December 1993 from the Department of Electrical Engineering, Indian Institute of Technology. He is currently an engineer working with Synopsys, Inc. His research interests are in the areas of VLSI design, testing, and HDLS.

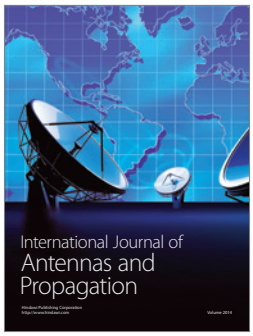


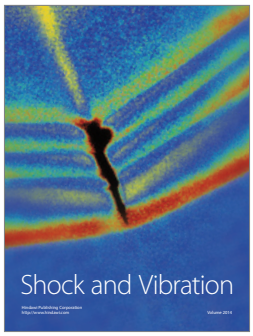


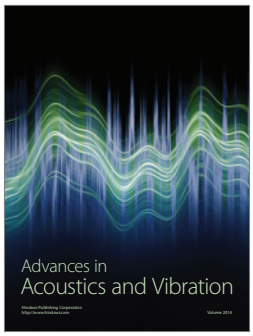












Hindawi

Submit your manuscripts at
http://www.hindawi.com