2

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 89-553
August 1989

Test Generation for Highly Sequential Circuits

Abhijit Ghosh, Srinivas Devadas, and A. Richard Newton

Abstract

We address the problem of generating test sequences for stuck-at faults in non-scan synchronous sequential circuits. We present a novel test procedure that exploits both the structure of the combinational logic in the circuit as well as the sequential behavior of the circuit. In contrast to previous approaches, we decompose the problem of sequential test generation into *three subproblems of combinational test generation, fault-free state justification and fault-free state differentiation.* We describe *fast algorithms for state justification and state differentiation* using the *ON-sets and OFF-sets* of flip-flop inputs and primary outputs. The decomposition of the testing problem into three subproblems rather than the traditional two, performing the justification and differentiation steps on the fault-free rather than the faulty machine and the use of efficient techniques for cube intersection results in significant performance improvements over previous approaches.

Microsystems Research Center Room 39-321
Massachusetts Institute of Technology
Cambridge Massachusetts 02139
Telephone (617) 253-8138

Acknowledgements

To be presented at the *International Conference of Computer Aided Design (ICCAD '89)* in November 1989. This work was supported in part by the Defense Advanced Research Projects Agency under contract number N00014-87-K-0825.

Author Information

Ghosh and Newton: Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.
Devadas: Department of Electrical Engineering and Computer Science, Room 36-848, MIT, Cambridge, MA 02139. (617) 253-0454.

Test Generation for Highly Sequential Circuits

Abhijit Ghosh Srinivas Devadas* A. Richard Newton

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Abstract

We address the problem of generating test sequences for stuck-at faults in non-scan synchronous sequential circuits. We present a novel test procedure that exploits both the structure of the combinational logic in the circuit as well as the sequential behavior of the circuit. In contrast to previous approaches, we decompose the problem of sequential test generation into *three subproblems of combinational test generation, fault-free state justification and fault-free state differentiation.* We describe *fast algorithms for state justification and state differentiation* using the *ON-sets and OFF-sets* of flip-flop inputs and primary outputs. The decomposition of the testing problem into three subproblems rather than the traditional two, performing the justification and differentiation steps on the fault-free rather than the faulty machine and the use of efficient techniques for cube intersection results in significant performance improvements over previous approaches.

1 Introduction

Test generation for sequential circuits has been recognized as a difficult problem [8] [5] [12]. Initial work in tackling this problem involved the use of both random [4] [15] and deterministic techniques [11] [10] [13] [16]. Due to the relative ineffectiveness of these approaches, a popular approach to enhancing the testability of sequential circuits has been the Scan Design methodology [6] [1].

Recently, there has been considerable progress made in this area. A PODEM-based [7] deterministic approach to sequential test generation was described in [9]. This approach uses the iterative array model for test generation and makes intelligent use of a partial State Transition Graph (STG) of the circuit while generating justification sequences for the faults under test. A heuristic, simulation-based test pattern generation algorithm was described in [2]. Both these approaches have been successful in generating tests for circuits with 1000-2000 gates and 50-75 flip-flops.

In contrast to previous approaches, the approach we present in this paper, involves decomposing the problem of sequential test generation into *three subproblems of combinational test generation, fault-free state justification and fault-free state differentiation.* Initially, prior to test generation, each of the flip-flop inputs (next state lines) and primary outputs are represented as separate cone circuits and complete or partial sum-of-product representations of the ON/OFF-sets of these cone circuits are extracted. The extraction is carried out via the use of the PODEM algorithm. Cover extraction is significantly faster than STG enumeration, which is used in [9].

Given a fault for which a test sequence has to be generated, we first generate a combinational test vector that propagates the effect of the fault to the next state lines or the primary outputs. We then perform a justification step, which involves finding a justification sequence for the state corresponding to the generated test vector. This step is carried out efficiently via *a sequence of cube intersections* on the complete or partial ON/OFF-set representations of the next state lines. Thus, a fault-free justification sequence is found. If the effect of the fault has been propagated to the next state lines alone, then we obtain the *faulty fault-free state pair* produced by the test vector. We obtain a fault-free differentiation or distinguishing sequence for this faulty fault-free state pair via another sequence of cube intersections, this time on the ON/OFF-set representations of the primary outputs.

Justification or differentiation sequences that are valid in a fault-free machine are not necessarily valid in the faulty machine. *However, experimentation indicates that these sequences are either valid or themselves test sequences, more than 95% of the time.* Fault-free justification and differentiation can be performed much more efficiently than the same under faulty conditions since *information can be reused.* Further, the intersection of sum-of-product forms, which forms the basis for justification and differentiation, can be performed efficiently, via sophisticated data structures. Splitting the sequential test generation problem into *three* subproblems rather than the traditional two, also improves efficiency. Upto a factor of *15X* improvement in performance has been achieved for large sequential machines, over previous approaches.

We present basic definitions in Section 2. In Section 3, the three-step global strategy for test generation is described. The algorithms used in the cover enumeration and combinational test generation, state justification, and differentiation steps are described in Sections 4, 5, and 6 respectively. Detection of redundant faults is the subject of Section 7. In Section 8, we present preliminary experimental results, that indicate that this approach is more efficient than previous approaches and viable for larger circuits than previous approaches.

*Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge

2 Preliminaries

2.1 The Problem

A general sequential circuit consists of a combinational logic block and feedback registers that hold the state information. The problem of sequential test generation involves finding primary input sequences which can excite the fault and propagate its effect to the primary outputs. It is assumed that the present state and the next state lines are neither *controllable* nor *observable*. The following assumptions are made regarding the sequential circuit to be tested.

1. The machine is assumed to have a **reset state**, R. All test vectors are applied with this state as the starting state.
2. The fault model is assumed to be **single stuck-at.**
3. The memory elements are considered as distinct logic primitives and faults inside the memory elements are not considered. However, all faults on present state and next state lines are considered.

To detect a fault in a sequential machine, the machine has to be first placed in a state which can excite the fault and propagate its effect to the primary output or the next state lines. If the fault is not propagated to the primary outputs, then another sequence of input vectors is necessary to propagate the fault to the primary outputs. Thus unlike combinational test generation, sequential test generation requires multiple vector test sequences.

It has been shown [5] that a fault in a general sequential circuit may require a test sequence of up to 4^n input test vectors, where n is the number of memory elements in the machine. This shows that the search space for sequential test generation is very large. To add to the complexity, some faults in the circuit may be **redundant**; i.e., they cannot be detected by any test sequence. Since such faults are very difficult to identify, large amounts of effort can be spent in trying to generate tests for them.

2.2 Definitions

A **state** is a bit vector of length equal to the number of memory elements (latches or flip-flops) in the sequential circuit. A state with only 0s and 1s as bit values is called a **minterm** state. In general, a state is a **cube**: i.e., the values in the different bit positions may be 0, 1 or x (don't care).

A state is said to **cover** another state if the value of each bit position in the first state is either an x or is equal to the value of the corresponding bit position in the second state.

For the combinational logic in the sequential circuit, there are p primary inputs, n present state and next state lines, and q primary outputs. The combinational logic implements a multiple-output Boolean function $f : B^{p+n} \rightarrow B^{q+n}$. Each of the primary outputs or next state lines are single output functions of $p+n$ variables. The **ON-set**, $X_{ON} \subseteq B^{p+n}$, of a primary output or next state line is the set of input values such that primary output or next state line is 1. Similarly the **OFF-set**, $X_{OFF} \subseteq B^{p+n}$, is the set of input values for which the corresponding line is 0. The set of cubes C, is said to be a **cover** for a ON-set if $X_{ON} \subseteq C$ and C does not intersect X_{OFF}.

A space can be **enumerated** by exhaustively searching a set of cubes which add up to the universal cube corresponding to that space.

Minterm enumeration implies that each cube searched is a minterm. Minterm enumeration of an n-dimensional space implies that 2^n combinations have been searched. **Implicit enumeration** involves exhaustively searching an n-dimensional space via cubes such that the number of cubes searched is significantly less than 2^n.

An edge in the State Transition Graph is said to be **corrupted** by a stuck-at fault if the effect of the fault can be propagated to the primary outputs or next state lines by the input vector corresponding to the edge with the present state lines set to the fan-in state of the edge. An input vector to the combinational logic part of the machine that excites and propagates the effect of the fault to the primary output or the next state lines is called the **excitation vector** for the fault. The present state part of the input vector is the state whose fanout edge is corrupted by the fault, and is called the **excitation state**.

The process of finding an input sequence which takes the machine from the reset state to the fault excitation state is called **state justification**. The corresponding input sequence is called the **justification sequence** and set of states traversed during justification constitute the **justification path**. State justification may be **forward** state justification or **backward** state justification, depending on whether the search is conducted from the reset state to the excitation state or vice versa.

In a sequential circuit, a fault may be redundant, i.e., untestable. There are two kinds of redundancies [9] in a sequential circuit — **combinational redundancies** and **sequential redundancies**. For a **combinationally redundant** fault, the effect of the fault cannot be propagated to the primary outputs or the next state lines, beginning from any state, with any input vector. A **sequentially redundant** fault is a fault that cannot be propagated to the primary outputs in 4^n time frames beginning from the reset state of the machine.

3 Global Strategy for Test Generation

Information about state transitions in sequential machines are traditionally represented using graphs. It is also possible to represent this information by the ON-sets and OFF-sets of all the next state lines and the primary outputs. *Connectivity* can be represented by the ON and OFF-sets of the next state lines alone.

The first step in test generation is the enumeration of the partial or complete (memory and CPU time permitting) ON and OFF-sets of each of the next state (NS) lines and primary outputs (POs) in the sequential circuit to be tested. Cover enumeration is done via a PODEM-based or DALG-based [14] enumeration algorithm. A limit on the number of cubes in each ON or OFF-set can be placed. This limit is used to restrict the amount of memory and CPU time used for enumeration. Cover enumeration is generally very fast and full covers of moderately large circuits can be extracted easily.

Given the complete (or partial) covers, test generation is a three-step process. These three steps consist of (1) combinational test generation for a fault treating the present state (PS) lines also as primary inputs (PIs) and the NS lines also as POs, (2) fault-free state justification and (3) fault-free state differentiation.

After combinational test generation, the excitation vector is examined to see if the present state part of the excitation vector covers the reset state. For this discussion, assume that the excitation state is a minterm state. If the excitation state is the reset state, then the fault can be excited from the reset state of the machine. If not, then the excitation state is justified using a backward justification algorithm. Backward justification is done by first finding all the fan-in states of the excitation state, via repeated cube intersection. If the reset state is a member of the set of fan-in states, then a one vector justification sequence is found. Else, the process is repeated for some state in the fan-in of the state being currently justified. Once a justification sequence is found, it is fault simulated to see if the required state is justified. If the required state is justified, then the justification sequence is a proper justification sequence in the faulty machine too. If the required state is not justified, *then some edge in the justification path must have been corrupted.* A part of the justification sequence can then be used as a justification sequence for the state whose fanout edge was corrupted by the fault. State differentiation can then be done between the corresponding true and faulty states. We thus have to perform only one fault-free justification to obtain a true faulty state pair.

If the effect of the fault under test has been propagated to the primary outputs by the combinational test vector, and if the excitation state can be justified in the faulty machine, then a successful test for the fault has been generated. If, however, the effect of the fault is propagated only to the next state lines, then the fault has to be propagated to some primary output by state differentiation. This is done by first finding an input vector that produces a different output on at least one primary output line for the true and the faulty states. Such a vector constitutes a single-vector differentiation sequence between the true and faulty states. If a single-vector differentiation sequence cannot be found, all the fanout states of the true and faulty states are found via repeated cube intersections. Then, for each pair of fanout states a single-vector differentiation sequence is sought (we are searching for an overall two-vector differentiation). If no such pair exists, then a pair of states fanning out from some fanout state pair is picked and differentiation between this pair is attempted. The differentiation sequence obtained is valid under fault free condition. After the differentiation sequence is obtained, the entire test sequence is fault simulated to see if the fault under test is detected. Experimental evidence indicates that more than 95% of the time, a test vector sequence generated is actually a test for the fault.

As in some combinational test generators [1] and in some sequential test generators [15], we have added a random test vector generation procedure as a front end to the deterministic test generation algorithm. Random vector test generation enables us to detect some of the easy to detect faults without much effort and therefore reduces test generation time.

By checking to see if the justification sequence is valid, prior to state differentiation, we obviate the need for generating more than one justification sequence in each pass of the algorithm.

All justification sequences and differentiation sequences generated are stored for use later. Thus parts of the State Transition Graph (STG) *that are required for test generation are explicitly enumerated.* Enumeration of only the required parts of the STG gives rise to significant memory and CPU time savings.

4 Cover Extraction and Combinational Test Generation

The input to the program is the combinational logic specification of the finite state machine, with the latch inputs and outputs properly identified. For each primary output and next state line, the ON-set and OFF-set are derived by setting the corresponding line to a 1 or 0 and using PODEM [7] to implicitly enumerate the input combinations that can set the line to a 1 or 0. Since on every backtrack PODEM sets an input line to a value different from what it had previously, the cover of the ON and OFF-sets are guaranteed to be single cube containment minimal. For almost all examples, covers are of tractable size and enumeration takes a very small fraction of the total test generation time.

Given a fault for which a test sequence is to be generated, the first step in sequential test generation is to generate a combinational test vector for the fault – the circuit is considered to be combinational with inputs being the PIs and the PS lines and the outputs being the POs and NS lines. A cube test vector is generated using as many don't care entries in the present state part as possible. This is done because state justification is easier for states represented by large cubes. Combinational test generation is based on the decision tree concept of the test pattern generation algorithm PODEM. It uses the 9-valued simulation, as in STALLION. The fault is first excited by setting the faulty wire to a value different from the faulty value. At first, this value is justified by setting some of the inputs to the combinational logic block. The algorithm then tries to propagate the effect of the fault to the primary outputs, failing which it tries to propagate the effect of the fault to the next state lines. If the fault is combinationally redundant, then the effect of the fault cannot be propagated to either the NS or PO lines. Since the goal is to generate a maximal cube for the PS lines, all NS and PO lines may not be set to a 1 or 0 at the end of test generation ome lines may still be left unknown.

If a new test has to be generated because the previous excitation state [illegible] not be justified, then the new excitation state should be *disjoint* the previous excitation state. Thus for a particular fault, each test generated has an excitation state which is disjoint from all previous excitation states for the fault.

5 Justification

Combinational test generation produces a test vector with as many don't care entries in the present state part as possible. Any state in the group of states S_1 corresponding to the excitation states for the fault has to be justified. If the reset state R is already covered by S_1, then the fault can be excited from the reset state and a justification sequence is not needed.

The state justification algorithm first attempts to find a single vector justification sequence from the reset state R of the machine to any of the states (minterms) in S_1. If complete covers of the next state lines are available, the entire fanin of S_1 can be found via cube intersections. S_1 is represented as a bit-vector with 0, 1 and x entries. If the position corresponding to a PS line has a 1 (0), the ON-set (OFF-set) of the corresponding NS line is picked. Bit positions with x's are ignored.

The intersection of the ON and OFF-sets of the NS lines with 0s and 1s gives the fanin edges (both PI and PS vectors) to the states in S_1. The intersection can be computed dynamically, checking each cube, c, produced to see if the PS part of the cube covers the reset state R. If such a cube(s) is found, a single vector justification sequence from R to S_1 is obtained, corresponding to the PI part of the cube c.

If no such cube is found, then it means that a single vector justification sequence does not exist for any of the states in S_1. Thus an N-vector sequence with $N > 1$ has to be found. This is done by heuristically selecting a group of states S_2 which exist in the fanin of S_1 and attempting to justify some state in S_2, via a single vector justification sequence. While inspecting each cube c formed from the intersection of the ON and OFF-sets, the *largest* cube (cube with most don't care entries) that is *disjoint* from S_1 and which is not already in the potential justification path is picked to be S_2. Disjointness is required, since if $S_2 \subseteq S_1$, then we know that a single vector justification sequence does not exist for S_2. Also, the new state selected should not be in the potential justification path built so far, inorder to prevent cycles during justification.

The justification sequence that is constructed is valid under fault-free conditions, and may be invalid under faulty conditions. If a justification sequence is invalid under faulty conditions, it means that the effect of the fault has already been propagated to the NS lines or the POs. Empirical evidence has shown that over 99% of the time, in real circuits, a justification sequence, valid in a fault-free machine, is also valid in the faulty machine or is in itself a *test sequence* for the fault. In the unlikely event that a justification sequence is neither valid in the faulty machine nor a test sequence in itself, we obtain a valid justification sequence from the invalid one as described earlier.

All cubes are represented as bit vectors which makes storage and operations on cubes very efficient. Cube and cover data structures used are similar to those used in ESPRESSO [3]. Using proper bit notations for 0s, 1s and x's, cube intersection can be performed by bitwise AND operations, which can be done very efficiently. A cube is therefore a collection of unsigned integers whose bits represent the value of the corresponding variable. The cubes in each cover are ordered so that the cubes that cover the reset state are before those that do not. Amongst cubes that cover the reset state, the larger ones are placed before the smaller ones. This helps in finding a fanin state that covers the reset state as early as possible.

6 State Differentiation

This step is only required if the initial combinational test vector generated for the fault under test propagates the effect of the fault to the NS lines alone.

Typically, in sequential test pattern generators, a propagation sequence that propagates the effect of the fault to the POs, is found using a test generation algorithm like PODEM, on multiple time-frames (or clock cycles) using the iterative-array model [5]. The first vector in this sequence propagates the effect of the fault to the POs or the NS lines. Since the fault is present in each time-frame, propagation from the PS lines of the second time-frame to the POs of the second time-frame is attempted under faulty conditions (to take into account the multiple-fault effect).

We believe, from our experience with fault-free justification, that generating a propagation sequence (in an iterative array model of the sequential circuit) where only the logic in the first time-frame contains the fault would result in a propagation sequence that is almost always valid under faulty conditions (i.e. the fault is present in all time-frames). Arising from this observation, we propose a method of fault-free *state differentiation*, given the test vector that propagates the effect of the fault to the NS lines.

After justification, the faulty fault-free state pair $(s_1{}^T, s_1{}^F)$ given by the test vector have to be differentiated. $s_1{}^T$ and $s_1{}^F$ are guaranteed to differ in at least one bit. Some entries in $s_1{}^T$ and $s_1{}^F$ may be unknown values. This means that in the general case, *groups* of states may have to be differentiated, rather than a minterm state pair.

State differentiation can be performed using the partial or complete covers of the POs and the NS lines. The procedure for single-vector differentiation is as follows:

1. Pick a (new) output.

2. Inspect the covers of the ON-set and OFF-set of the output and search for an PI combination, i_1, which appears concatenated with $s_1{}^T$ (or $c \supseteq s_1{}^T$) in the ON-set and concatenated with $s_1{}^F$ (or $c \supseteq s_1{}^F$) in the OFF-set (or vice versa). If such an input combination is found for some output, then a fault-free state differentiation sequence can be constructed. Exit with the input combination. If not, a single-vector state differentiation sequence cannot be found for $(s_1{}^T, s_1{}^F)$.

CKT	#inp	#out	#gate	#lat	#eqv. faults
sse	7	7	130	6	368
planet	7	19	606	6	1482
mult1	9	9	170	15	336
sbc	40	56	1011	28	1985
stage	131	64	2700	64	7227
key	62	48	1342	56	8520
pewd	117	101	1873	112	3183
dsip	228	197	3654	224	6781

Table 1: Statistics for Example Circuits

Multiple-vector differentiation sequences can be searched for in the following fashion. N is the number of vectors in the current sequence.

3. $N = 1$.

4. Pick a NS line and attempt to find an PI vector, i_N, that produces a 1 (0) when concatenated with s and a 0 (1) when concatenated with s^F. Try another NS line if a vector cannot be found for the picked one. If an input combination cannot be found for any such NS line, then a state differentiation sequence cannot be found for $(s_N{}^T, s_N{}^F)$.

5. Find the state pair $(s_{N+1}{}^T, s_{N+1}{}^F)$ given by the fanout states of the PI vector i_N for the state pair $(s_N{}^T, s_N{}^F)$. $N = N + 1$. Attempt to find a single vector propagation sequence for the state pair $(s_{N+1}{}^T, s_{N+1}{}^F)$.

The algorithm thus attempts to find a single vector sequence, then a two-vector sequence and so on. The NS lines are selected in a heuristic order that uses topological information as to the location of the fault with respect to the different NS lines and POs.

7 Identification of Redundant Faults

The difficulty in sequential test generation lies not only in the generation of difficult to detect but testable faults, but also in the identification of redundant faults. Low fault coverage on certain examples does not necessarily mean that the test generator is inadequate, if it can be shown that the fault coverage is close to the maximum possible value. In general, identification of redundant faults require astronomical amounts of CPU time, as the total search space has to be enumerated before a fault can be pronounced redundant. As defined in Section 2, there are two classes of redundant fault – combinationally redundant and sequentially redundant. Combinationally redundant faults are detected during combinational test generation, and are easier to find than sequentially redundant fault.

It is possible to detect a subset of the sequentially redundant faults using **Theorem 1** of [9]. We use the same theorem for the detection of sequentially redundant faults. However, since we do not use the State Transition Graph of the machine, the conditions of **Theorem 1** are verified differently. A state (or group of states) cannot be justified if the total number of fan-in cubes determined during the justification procedure is zero. An unjustifiable state is a state that cannot be reached in the true machine (invalid state). We generate all possible tests for the fault, with disjoint initial states. If all of the initial states have no fan-in cubes after intersection, then the fault under test is redundant. In the next section, we show that this new method can establish redundancy for a larger fraction of faults than the method of [9].

8 Test Generation Results Using STEED

The test generation algorithm described in the previous section have been implemented in the program STEED. It consists of about 10,000 lines of C code and runs in a VAX-UNIX environment.

Results and time profiles using STEED for eleven finite state machines which are described in Table 1 are given in Tables 2 and 3, respectively. In Table 4, comparisons are drawn against STALLION [9] and the test generator, CONTEST, described in [2]. In the tables *m* and *s* stands for minutes and seconds, respectively. For each example in Table 1, the number of inputs (#inp), number of outputs (#out), number of gates (#gate), number of latches (#lat), and number of equivalent faults (#eqv. faults) are indicated.

CKT	#test seq.	#vec	max. seq. len.	succ (%)	fault cov. (%)	red. fault (%)	tfc (%)	CPU time
sse	52	336	10	100.0	80.98	19.02	100.0	25.1s
planet	57	1046	30	100.0	96.56	3.44	100.0	5.85m
mult4	9	94	50	100.0	98.51	1.19	99.70	11.9s
sbc	102	1364	50	95.0	95.42	3.22	98.64	75m
stage	58	155	20	100.0	92.36	7.64	100.0	50.87m
key	454	1669	20	100.0	94.75	5.25	100.0	419m
pewd	11	221	40	100.0	100.0	0	100.0	6.58m
dsip	8	212	40	100.0	99.99	0.01	100.0	25.8m

Table 2: Test Generation Results for Circuits

CKT	Cover Enum.	Justify	Differ	Test Gen.	Fault Sim.	Total Time
sse	0.3s	0.8s	0.3s	10.7s	13.9s	25.1s
planet	1.9s	3.0s	3.0s	36.9s	311.9s	5.85m
mult4	1.4s	3.5s	0.01s	4.59s	5.12s	11.9s
sbc	1.71m	21m	26m	50m	23m	75m
stage	3.38m	15.3m	0.0s	27.3m	20.07m	50.8m
key	21.3s	21.27s	174.75s	59.0m	354.6m	419m
pewd	31.9s	0.23s	8.0s	9.49s	5.18m	6.58m
dsip	120.2s	0.34s	12.8s	14.9s	23.46m	25.8m

Table 3: Time Profiles for Example Circuit

In Table 2, the number of test sequences (#test seq.), the total number of test vectors (#vect), the maximum test sequence length (max. seq. len), the percentage when a potential test sequence generated for a fault actually detected the fault (%succ), the fault coverage, the percentage of provably redundant faults (using the redundancy procedure), the total fault coverage including detected and provably redundant faults (tfc), and the CPU time on a VAX 11/8800 are indicated for each example.

CPU times for enumeration of covers, justification, differentiation, total test generation, fault simulation and miscellaneous setup operations and for the entire test generation process are given in Table 3.

In Table 4, total test generation time and fault coverage of STALLION [9], CONTEST [2] and STEED are compared. As can be seen, our test generation technique obtains close to the maximum possible fault coverage in all the examples. It takes significantly smaller time than STALLION [9] and CONTEST [2] to achieve the same fault coverage. For the large examples, very significant speed-ups were obtained.

The fraction of time spent in cover enumeration is very small, much smaller than the corresponding time required for even partial STG enumeration in STALLION. Test generation times are in most cases small, and fault simulation dominates the total test generation time in most cases. It is worthwhile to note that the success rate, i.e. the percentage of times that a potential test sequence is valid, is 100% for the largest examples. For the examples **sbc** and **key**, which have *huge* STGs, STALLION is unable to establish sequential redundancy for faults, but STEED can.

The results for STALLION shown in this paper have been derived using a newer and improved version of the program that uses better fault collapsing and hierarchical enumeration. Thus, the results quoted in this paper do not agree with those in [9].

9 Conclusions

A novel approach to test generation for sequential circuits have been presented in this paper. We have developed an efficient deterministic algorithm for test generation that uses fault-free state justification and fault-free state differentiation. The efficacy of our method stems from the efficient use of covers of the ON and OFF sets of the primary output and next state lines instead of the STG, for state justification and differentiation. Experimental results indicate that faults that require a long input sequences are handled efficiently. We have successfully generated tests for finite state machines with a large number of latches within reasonable amounts of CPU time and have obtained close to maximum amount of fault coverage. We have also demonstrated that our algorithms require significantly smaller time than the test generator described in [9] and [2] while maintaining or improving fault coverage. It was also demonstrated that a larger class of sequentially redundant faults can be determined during test generation.

CKT	CONTEST		STALLION		STEED	
	F.C.	Time	F.C.	Time	F.C.	Time
sse	99.56	291s	99.8	26.6s	100.0	27.3s
planet	98.42	52m	99.95	6.58m	100.0	5.85m
mult4	97.41	838s	99.21	42.4s	99.70	11.9s
sbc	–	–	98.34	80.0m	98.64	75m
stage	–	–	100.0	154m	100.0	50.8m
key	–	–	35.95	> 900m	100.0	419m
pewd	–	–	100.0	117m	100.0	6.58m
dsip	–	–	99.99	350m	100.0	25.8m

Table 4: Comparisons With STALLION and CONTEST

10 Acknowledgements

The interesting discussions with Tony Ma and Tim Cheng on sequential test generation are acknowledged. This research was supported in part by the Defense Advanced Research Projects Agency under contracts N00014-87-K-0825 and N00039-87-C-0182, Digital Equipment Corporation, AT&T Bell Laboratories and Semiconductor Research Corporation. Their support is gratefully acknowledged.

References

[1] V. D. Agarwal, S. K. Jain, and D. M. Singer. Automation in Design for Testability. In *Proc. of Custom Integrated Circuit Conference*, May 1984.

[2] V. D. Agrawal, K-T. Cheng, and P. Agrawal. CONTEST: A Concurrent Test Generator for Sequential Circuits. In *Proc. of 25th Design Automation Conference*, pages 84–89, June 1988.

[3] R. K. Brayton, G. D. Hachtel, Curt McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[4] M. A. Breuer. A Random and an Algorithmic Technique for fault detection and test generation. In *IEEE Transactions on Computers*, pages 1366–1370, November 1971.

[5] M. A. Breuer and A. D. Friedman. *Diagnosis and Reliable Design of Digital Systems*. Computer Science Press, 1976.

[6] E. B. Eichelberger and T. W. Williams. A Logic Design Structure for LSI Testability. In *Proc. 14th Design Automation Conference*, pages 462–468, June 1977.

[7] P. Goel. An Implicit Enumeration Algorithm to generate tests for combinational logic circuits. In *IEEE Transactions on Computers*, pages 215–222, March 1981.

[8] F. C. Hennie. Fault Detecting Experiments for Sequential Circuits. In *Proc. of 5th Annual Symposium and Switching Theory and Logical Design*, pages 95–110, November 1974.

[9] H-K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli. Test Generation for Sequential Circuits. In *IEEE Transactions on CAD*, pages 1081–1093, October 1988.

[10] S. Mallela and S. Wu. A Sequential Test Generation System. In *Proc. of International Test Conference*, pages 57–61, October 1985.

[11] R. Marlett. EBT: A Comprehensive Test Generation System for Highly Sequential Circuits. In *Proc. of 15th Design Automation Conference*, pages 332–338, June 1978.

[12] A. Miczo. The sequential ATPG: A Theoretical Limit. In *Proc. of Int'l Test Conference*, pages 143–147, October 1983.

[13] S. Nitta, M. Kawamura, and K. Hirabayashi. Test Generation by Activation and Defect-Drive (TEGAD). In *INTEGRATION Journal*, pages 2–12, 1985.

[14] J. P. Roth. Diagnosis of Automata Failures: a calculus and a method. In *IBM journal of Research and Development*, pages 278–291, July 1966.

[15] H. D. Schnurmann, E. Lindbloom, and R. G. Carpenter. The Weighted Random Test-Pattern Generator. In *IEEE Transactions on Computers*, pages 695–700, July 1975.

[16] S. Shteingart, A. W. Nagle, and J. Grason. RTG: Automatic Register Level Test Generator. In *Proc. of 22nd Design Automation Conference*, pages 803–807, June 1985.