EASILY TESTABLE PLA-BASED FINITE STATE MACHINES

by

Srinivas Devadas and A. Richard Newton

Memorandum No. UCB/ERL M88/47

19 July 1988

EASILY TESTABLE PLA-BASED FINITE STATE MACHINES

by

Srinivas Devadas and A. Richard Newton

Memorandum No. UCB/ERL M88/47

19 July 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

EASILY TESTABLE PLA-BASED FINITE STATE MACHINES

by

Srinivas Devadas and A. Richard Newton

Memorandum No. UCB/ERL M88/47

19 July 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Easily Testable PLA-based Finite State Machines

Srinivas Devadas*and A. Richard Newton

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

Abstract

In this paper, we outline a synthesis procedure, which beginning from a State Transition Graph description of a sequential machine, produces an optimized **easily testable** PLA-based logic implementation.

Previous approaches to synthesizing easily testable sequential machines have concentrated on the stuck-at fault model. For PLAs, an extended fault model called the crosspoint fault model is used. In this paper, we propose a procedure of **constrained state assignment and logic optimization** which guarantees testability for all combinationally irredundant crosspoint faults in a PLA-based finite state machine. No direct access to the flip-flops is required. The test sequences to detect these faults can be obtained using combinational test generation techniques alone. This procedure thus represents an alternative to a Scan Design methodology. We present results which illustrate the efficacy of this procedure – the area/performance penalties in return for easy testability are small.

1 Introduction

Test generation for sequential circuits has long been recognized as a difficult task [3]. Several approaches [2] [18] [16] [15] [17] [19] have been taken in the past to solve the problem of test generation for sequential circuits. They are either extensions to the classical D-Algorithm or based on random techniques [18] [17]. When the number of states of the circuit is large and the tests demand long input sequences, they can be quite ineffective for test generation.

For sequential circuits, design for testability has been a synonym for the use of full Scan Design techniques, such as the LSSD approach [8] pioneered by IBM. This method converts the difficult

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

problem of testing sequential circuits, into a much easier one, that of testing a combinational circuit. However, there are cases where the area and timing penalty associated with LSSD techniques are not acceptable to designers.

Logic synthesis and minimization techniques can, in principle, ensure fully and easily testable combinational and sequential circuit designs. In [10], a synthesis procedure which guaranteed fully testable irredundant combinational logic circuits was proposed. In [6], a procedure which produced a fully and easily testable logic-level sequential machines from State Transition Graph descriptions was proposed. The work in [6] showed that state assignment has a profound effect on the testability of a sequential machine.

Programmable Logic Arrays (PLAs) are used extensively in the design of complex VLSI systems. Sequential functions can be realized very efficiently by adding feedback registers to the PLA. Numerous programs for the optimal synthesis of PLA-based finite state machines have been developed (e.g. [9], [5]). Test generation and design-for-testability techniques for PLA structures have been active areas of research.

Due to a PLA's dense layout, PLA faults other than conventional stuck-at faults can occur easily and must be modeled. An extended model, the *crosspoint fault* model, has been proposed in [4] and [12]. The crosspoint-oriented test set covers many of the frequently occurring physical faults, including shorts between lines. Several PLA test generation techniques aimed at the crosspoint fault model have been proposed (e.g. [13], [7]). In particular, an exact and efficient technique which guarantees maximum fault coverage and identification of all redundant faults was proposed in [20].

Design-for-testability techniques (e.g. [11]) for PLAs require controllability of all inputs and observability of all outputs of the PLA. Synthesis approaches to producing easily testable sequential machines, without requiring direct access to the inputs/outputs of the circuit's memory elements, have not been aimed at the crosspoint fault model.

In this paper, we outline a synthesis procedure, which beginning from a State Transition Graph description of a sequential machine, produces an optimized easily testable PLA-based logic implementation. We propose a procedure of constrained state assignment and logic optimization which guarantees testability for all combinationally irredundant crosspoint faults in a PLA-based finite state machine. No direct access to the flip-flops is required. The test sequences to detect these faults can be obtained using combinational test generation techniques alone. This procedure thus represents an alternative to a Scan Design methodology. We present results which illustrate the

efficacy of this procedure – the area/performance penalties in return for easy non-scan testability are small.

Basic definitions and terminologies used are given in Section 2. The crosspoint fault model is also described. In Section 3, the relationship between state assignment and testability of a sequential machine is discussed and the necessary conditions required for an easily testable PLA-based Moore or Mealy finite state machine are stated. In Section 4, we discuss how an existing state assignment algorithm can be modified to produce a constrained encoding satisfying the testability criterion. Results obtained thus far for the synthesis technique are presented in Section 5.

2 Preliminaries

2.1 Definitions

A variable is a symbol representing a single coordinate of the Boolean space (e.g. a). A literal is a variable or its negation (e.g. a or $\overline{a}$). A cube is a set C of literals such that $x \in C$ implies $\overline{x} \notin C$ (e.g., $\{a, b, \overline{c}\}$ is a cube, and $\{a, \overline{a}\}$ is not a cube). A cube represents the conjunction of its literals. The trivial cubes, written 0 and 1, represent the Boolean functions 0 and 1 respectively. An expression is a set f of cubes. For example, $\{\{a\}, \{b, \overline{c}\}\}$ is an expression consisting of the two cubes $\{a\}$ and $\{b, \overline{c}\}$. An expression represents the disjunction of its cubes.

A cube may also be written as a bit vector on a set of variables with each bit position representing a distinct variable. The values taken by each bit can be 1, 0 or 2 (don't care), signifying the true form, negated form and non-existence respectively of the variable corresponding to that position. A minterm is a cube with only 0 and 1 entries.

A minterm m_1 is said to dominate another minterm m_2 (written as $m_1 \supset m_2$) if for each bit position in the second minterm that contains a 1, the corresponding bit position in the first minterm also contains a 1.

A finite state machine is represented by its State Transition Graph (STG), $G(V, E, W(E))$ where V is the set of vertices corresponding to the set of states S, where $||S|| = N_S$ is the cardinality of the set of states of the FSM, an edge (v_i, v_j) joins v_i to v_j if there is a primary input that causes the FSM to evolve from state v_i to state v_j, and $W(E)$ is a set of labels attached to each edge, each label carrying the information of the value of the input that caused that transition and the values of the primary outputs corresponding to that transition. In general, the $W(E)$ labels are Boolean expressions.

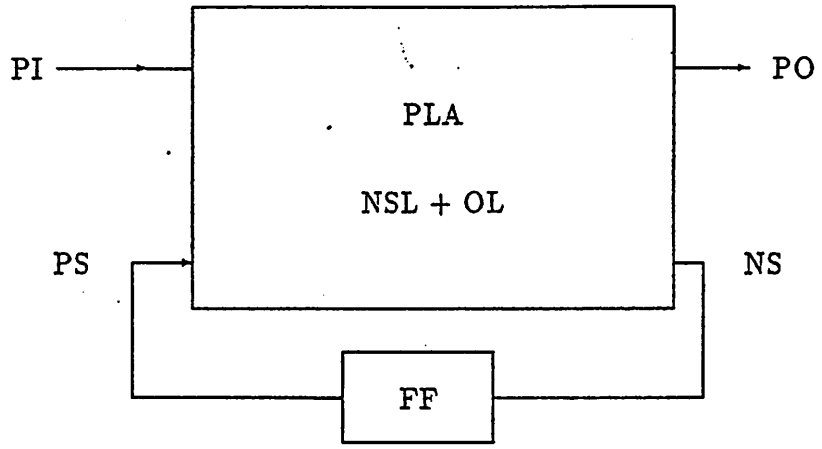

Figure 1: PLA-based Finite State Machine

Given n inputs to a machine, 2^n edges with minterm input labels fan out from each state. A STG where the next state and output labels for every possible transition from every state is defined corresponds to a **completely specified machine**. An **incompletely specified machine** is one where at least one transition edge from some state is not specified.

Given n latches in a sequential machine, 2^n possible states exist in the machine. A starting or initial state is assumed to exist for a machine, also called the **reset state**. A **R-reachable** finite state machine has a STG such that input sequences exist which place the machine in any of the 2^n states, beginning from the reset state.

A finite state machine is assumed to be implemented using a PLA and feedback registers as shown in Figure 1. The PLA implements both the output logic (OL) and next state logic (NSL) functions. Direct access is provided only to the primary inputs (PI) and primary outputs (PO). Tests are generated for faults in the PLA. A fault in the sequential machine is said to be **combinationally irredundant** if a primary input vector and present state exist that detect the fault at either the primary outputs or the next state lines. A fault in the PLA is said to be **sequentially irredundant** if a primary input test sequence exists which detects the fault at the primary outputs (The machine is assumed to be initially at the reset state).

An edge in a State Transition Graph of a machine is said to be **corrupted** by a fault if either the fanout state or output label of this edge is changed because of the existence of a fault. A path in a State Transition Graph is said to be corrupted if at least one edge in the path has been corrupted.

2.2 Crosspoint Faults

The following faults are considered in the crosspoint fault model.

1. Growth/Missing contact faults in the input plane
2. Shrinkage/Extra contact faults in the input plane
3. Appearance/Extra contact faults in the output plane
4. Disappearance/Missing contact faults in the output plane
5. Output stuck-at-one faults

Except for fault type 5, essentially two types of faults are present, namely, the *missing contact* and *extra contact* faults. In the input plane, an additional contact on a row reflects an additional constraint placed on the cube corresponding to the row and has the effect of shrinking the set of vertices covered by the cube. On the contrary, a missing contact in the input plane removes a constraint and thus expands the set of vertices covered by the cube. A missing contact on the ith column of the output plane reflects a removal of a cube from the ON-set cover of the ith output function. The effect is then the shrinkage of the ON-set of that ith output function. By the same token, an extra contact in the output plane adds an additional cube to the output ON-set cover and thus enlarges the ON-set. In the sequel, to adopt a unified point of view on these faults, we call fault types 1 and 3 as $GROWTH$ faults and fault types 2 and 4 as $SHRINKAGE$ faults.

3 Easily Testable PLA-based Finite State Machines

Synthesizing a logic-level implementation of a finite state machine from a State Transition Graph description involves the steps of state minimization, state assignment and logic optimization. All three steps have a profound effect on the testability of the resulting logic implementation. In this section, we will first describe the relationships between state assignment and testability of a sequential machine. We will then focus on PLA-based finite state machines and give a synthesis procedure of constrained state assignment and logic optimization which ensures testability for all combinationally irredundant crosspoint faults in the machine. The procedure does not require that the State Transition Graph (STG) description be state minimal – equivalent states can exist in the original STG. However, for area efficiency and performance reasons, it is better to begin from

a state minimal representation. Finally, we will describe the origin of combinationally redundant crosspoint faults in a PLA.

3.1 Relationship between State Assignment and Testability

In order to detect a fault in a sequential machine, the machine has (1) to be placed in a state that can excite the fault and (2) the effect of the fault has to be propagated to the primary outputs. State assignment does not affect the first step, i.e. state justification but can have a profound effect on the second step of fault propagation.

We will concentrate on PLA-based Mealy finite state machines, since a Mealy machine can be viewed as a more general case of a Moore machine. The PLA implements both the output logic and next state logic functions. We will focus on combinationally irredundant crosspoint faults in the PLA – combinationally redundant faults cannot be made testable in a sequential machine even using full Scan Design or via state assignment.

For any combinationally irredundant fault, a present state, s, and a primary input vector, i, exist, which can propagate the effect of the fault to the next state lines (NS) or the primary outputs (PO). If the effect of the fault is propagated to PO, then the fault can be detected in the non-scan sequential machine via a justification sequence for s. That is, when the machine is in s, applying i will detect the fault. On the other hand, if the effect of the fault is propagated to NS but not PO, then we obtain a faulty next state q^F instead of the fault-free (true) next state q. We need to be able to distinguish q and q^F at the primary outputs. If q and q^F are equivalent states in the faulty machine then we cannot detect the fault.

Depending on the type of crosspoint fault under test, the codes of q and q^F will have certain relationships. If we can ensure via state assignment that any two states produced as a faulty fault-free pair are not equivalent (in the faulty machine) then any fault which is propagated to the next state lines will always be detectable at the primary outputs. The synthesis procedure described in the next section, does precisely this, in order to ensure testability for all combinationally irredundant crosspoint faults in the sequential machine.

3.2 The Synthesis Procedure

Definition 3.1 : *Two minterms m_1 and m_2 are said to be mutually-dominant if $m_1 \supset m_2$ or $m_2 \supset m_1$. Two minterms m_1 and m_2 which are not mutually-dominant are said to be mutually-nondominant if $m_1 \neq m_2$.*

Lemma 3.2 : *For any kind of irredundant crosspoint fault in a PLA, the faulty output vector and the true output vector are mutually-dominant.*

Proof: Consider a fault, F, in the PLA. If the fault is a *GROWTH* fault, then F adds to the ON-set of some outputs, but does not subtract from the ON-set of any output. Therefore, if F is detected by some input vector i, then for some subset of the outputs whose true value is 0 for i, the faulty value is 1. Outputs whose true value is 1, remain at 1. This means that o^F, the faulty output vector for i, dominates o, the true output vector for i.

If F is a *SHRINKAGE* fault, then F subtracts from the ON-set of some outputs, but does not add to the ON-set of any output. Therefore, if F is detected by some input vector i, then for some subset of the outputs whose true value is 1, the faulty value is 0. Outputs whose true value is 0, remain at 0. This means that $o \supset o^F$.

Finally, a crosspoint fault of type 5, namely an output stuck-at-one fault if detected will produce a o^F which differs in one bit from o (a 1 instead of a 0). Again, $o^F \supset o$. □

We now give a procedure of constrained state assignment, summarized in Theorem 3.3, which ensures that faulty fault-free next state pairs are always propagated to the primary outputs within one clock cycle.

Theorem 3.3 : *Given a n-latch logic-level implementation of a PLA-based finite state machine (shown in Figure 1), if (1) the machine is R-reachable and (2) if the state encoding of the machine is such that each pair of states which do not produce mutually-nondominating primary outputs for at least one primary input vector are assigned mutually-nondominating codes, the machine is testable for all combinationally irredundant crosspoint faults.*

Proof: Consider a fault F in the PLA. Since the fault is combinationally irredundant a primary input vector i_1 and a present state s exist which detect the fault at either the primary outputs or at the next state lines. If the fault is detectable at the next state lines, then the faulty next state produced q^F instead of the true next state q are mutually-dominating by Lemma 3.2. By Condition 2, a primary input vector i_2 will exist which will distinguish q^F and q in the next cycle, since states which cannot be distinguished in the true machine are given mutually-nondominating codes and never allowed to appear as faulty fault-free pairs. We know that s can be justified in the true machine because of Condition 1. However, the justification sequence may have been corrupted by the fault. Also, we have to show that the distinguishing vector i_2 also holds in presence of the fault.

The distinguishing vector i_2 is such that q and q^F produce mutually-nondominating primary output vectors o_1 and o_2 on applying i_2 in the true machine. o_2 may be corrupted due to the fault, the vector produced may be ${o_2}^F \neq o_2$. By Lemma 3.2, ${o_2}^F$ and o_2 are mutually-dominating. Therefore, o_1 has to be different from ${o_2}^F$, since o_1 and o_2 are mutually-nondominating. This means that the distinguishing vector i_2 holds in faulty conditions as well. Note that if o_1 and o_2 were distinct but mutually-dominating this is not the case.

We have a justification sequence for s, namely I. This path may or may not be corrupted due to F. If the path is not corrupted, we can detect F by applying i_2 on reaching s. If the path is corrupted, it means that for some edge in the path, F has been propagated to the primary outputs or next state lines. If F is propagated to the primary outputs, we detect F even before reaching s. Else, if F has been propagated to the next state lines, we obtain a faulty and fault-free next state pair n and n^F. We know that n and n^F can be distinguished with some input vector i_3 even under faulty conditions. □

We now show that Condition 2, which requires mutually-nondominating codes to be assigned to some state pairs can be satisfied quite easily.

Lemma 3.4 : *Given n bits, there are* ${C^n}_1$, ${C^n}_2$, .. ${C^n}_{n-1}$ *sets of mutually-nondominating codes, where* ${C^n}_p = \frac{n!}{(n-p)!\ p!}$. *The maximum number of mutually-nondominating codes given an n bits is* ${C^n}_{n/2}$ *if n is even and* ${C^n}_{(n+1)/2}$ *if n is odd.*

For example, given a 3 bits, we have the following sets of multiple mutually-nondominating codes (001, 010, 100) and (011, 101, 110) whose cardinalities correspond to ${C^3}_1$ and ${C^3}_2$ respectively.

In general, State Transition Graph specifications of machines have reset states. However, a STG specification of a machine need not necessarily have $N_s = 2^k$ states, $k = 1, 2..$ etc. Given the number of encoding bits to be used, n ($n \geq \lceil log(N_s) \rceil$), the number of states in a STG can be raised to 2^n. We have to ensure that these new states are reachable from the reset state to satisfy the R-reachability condition. Given a single unspecified transition edge (minterm or cube) from a single state in the original STG, edges can be added to the STG so as to ensure that all the added states are reachable (If the machine is completely specified, an extra input has to be added). Most STGs encountered in practical design have a large number of transitions that are not specified. It should be noted that these extra states may be equivalent to other previously existing states in the STG. We do not require state minimality as a condition for easy testability, but we require all states to be reachable.

There are thus three steps in producing a PLA logic specification for the output logic and next state logic functions. This specification is then optimized using a two-level logic minimizer like ESPRESSO [1]. These steps are (1) raising the number of states in the State Transition Graph to 2^n, where n is the number of latches (2) obtaining constraints for the state assignment on the basis of state fanouts and (3) state assignment obeying the constraint relations generated. A straightforward solution exists for Step 1, however the optimality of the eventual implementation depends on the choices made during this step. For example, in Step 1, transition edges connecting original states in the STG to the new states can be added in a variety of ways. The new states can be connected in a chain or separately connected from the original states. In Step 3, an optimal state assignment which minimizes combinational logic while meeting the dominance constraints has to be found. This step is further discussed in Section 4.

To generate tests for the sequential machine, test vectors are generated for all irredundant crosspoint faults using a program like PLATYPUS [20]. Then, justification paths are obtained from the STG using simple breadth-first search. These paths concatenated with the test vectors applied to the primary inputs of a non-scan sequential machine will detect all the crosspoint faults in the machine so as to be observable at the primary outputs.

This procedure has ensured that a faulty state is always propagated to the primary outputs in a single clock cycle via state assignment. This can, in fact, be generalized to multiple-vector propagation. That is, state assignment constraints can be derived which ensure that a faulty state is propagated to the primary outputs in at most P clock cycles ($P \geq 1$). A state assignment algorithm can construct an optimal encoding which satisfies these constraints. For large P, the constraints are less stringent but more difficult to state succinctly.

A re-statement of Condition 2 in Theorem 3.3 to ensure testability via P-vector propagation sequences can be made. The re-statement for $P = 2$ is given below.

Definition 3.5 : *Two states q_1 and q_2 are said to be m-distinguishable if a primary input vector exists which produces two mutually-nondominating primary outputs o_1 and o_2 when the machine is in q_1 and q_2 respectively.*

The state encoding of the machine should be such that each pair of states which cannot be m-distinguishable should be assigned mutually-nondominating codes *or* the following should hold for any pair of states (q_1, q_2) which are not m-distinguishable and have mutually-dominating codes. An input combination should exist which drives the fault-free machine from q_1 and q_2 to states s_1

and s_2 respectively, such that

1. s_1 and s_2 are m-distinguishable and
2. If $q_2 \supset q_1$, then for all $s_2' \supset s2$, s_2' should be m-distinguishable from s_1. Similarly, if $q_2 \subset q_1$, then for all $s_2' \subset s2$, s_2' should be m-distinguishable from s_1.

3.3 Combinationally Redundant Crosspoint Faults

A two-level or multi-level circuit can be made irredundant for all single stuck-at faults. Such circuits are called prime and irredundant circuits. Logic minimization programs like ESPRESSO can ensure prime and irredundant two-level covers. However, since the crosspoint fault model is a superset of the stuck-at fault model, PLAs implementing prime and irredundant covers may not be testable for all possible crosspoint faults. However, typically a large percentage of crosspoint faults can be made testable via optimization [20]. All crosspoint faults of type 1, 4 and 5 can be guaranteed to be testable via logic minimization.

4 Constrained State Encoding

State assignment is the process of assigning binary codes to the internal states of a finite automaton. The problem of optimal state assignment is to find an encoding of states which minimizes the combinational logic part of the sequential machine.

The combinational logic part of the sequential machine can be implemented using a Programmable Logic Array (PLA) or using multi-level logic. State assignment techniques targeting both these implementations have been proposed (e.g. [9] [5]). The program MUSTANG [5] produces a state assignment that heuristically minimizes the number of literals in the combinational logic *after* multiple-level logic optimization. However, it is also effective in minimizing the number of product terms in an optimized PLA implementation.

The technique used by MUSTANG is based on maximizing common factors in the logic in an effort to reduce the area of the network. A weighted graph whose nodes represent each state of the machine is constructed. The weights between the edges in the graph reflect the "gains" in coding the corresponding states with uni-distant codes.

An embedding algorithm is used to assign binary codes to the states (nodes in the graph) so as to maximize the overall gain. The algorithm iteratively selects groups of states to be encoded. These states are given minimally-distant codes from the unassigned codes.

For our problem, the graph construction part remains the same. During embedding, when a group of states is selected, they are checked for mutual-nondominance constraints. A minimally-distant set of codes satisfying these constraints is then assigned to the states. The more complex, but less stringent, constraints given by multiple-vector propagation can also be accommodated.

5 Results

Results obtained on five State Transition Graphs from the MCNC 1987 Logic Synthesis Workshop benchmark set, whose statistics are given in Table 1, are given in Table 2. First, the machines were encoded and optimized disregarding testability. The number of product terms in the PLA, the fault coverage obtained and the test generation time are given in Table 2 under the column labeled OPTIMIZE. In Table 2, m stands for CPU-minutes and s for CPU-seconds on a VAX 11/8650. Then, each of the machines were synthesized using the procedure described in Section 3. Again, the number of product terms, fault coverage obtained and the test generation time are given. The example scf is a Moore machine, the others Mealy machines. In all cases, single-vector propagation constraints were placed on the state assignment program.

For the optimized machine, sequential test generation was accomplished as follows:

1. A present state and a primary input vector which propagates the effect of the fault to the primary outputs or the next state lines is found, if such a vector exists, using PLATYPUS [20].

2. A fault-free justification sequence for the required present state is found via breadth-first search on the State Transition Graph of the machine.

3. If the fault has been propagated to the next state lines, then a fault-free distinguishing sequence is found for the true and faulty states in the State Transition Graph. Such a sequence may not exist if the true and faulty states are equivalent. If this is the case, a new test vector is generated which produces a different true and faulty state pair, if possible.

4. The justification sequence, the combinational test vector and the distinguishing sequence are concatenated to produce a possible test sequence for the fault. The sequence may not be valid because the justification and/or distinguishing sequence may be invalid under fault conditions. The sequence is fault simulated on the circuit to check if the fault is indeed detected at the primary outputs.

5. If the sequence does not detect the fault, a different distinguishing sequence for the true-faulty state pair is tried, if possible.

A combinationally irredundant fault may not be detected using this procedure because (a) a distinguishing sequence may not exist for a true-faulty state pair since no constraints have been placed on the state assignment and (b) even if a distinguishing sequence exists, it may not hold under fault conditions.

The constrained synthesis procedure ensures that distinguishing sequences always exist and always hold under fault conditions. Test generation for the testable machine was accomplished as follows:

1. Same as Step 1 described above.

2. Same as Step 2 described above.

3. A single distinguishing vector which produces mutually-nondominating outputs is found for each true-faulty state pair (such a vector is guaranteed to exist).

4. The justification sequences are checked to see if they are valid under fault conditions. If the sequence is valid, a test sequence is constructed by concatenating the sequence with the combinational test vector and the distinguishing vector. If the justification sequence is invalid, and is not a test sequence for the fault by itself, a new distinguishing vector (which is guaranteed to exist) is found for the true-faulty state pair that is generated by the first corrupted edge in the sequence. The shortened justification sequence concatenated with the distinguishing vector constitutes a test sequence for the fault.

Sequential test generation for the testable machine is faster because typically more than one distinguishing sequence has to be tried to produce a test sequence for the fault in the optimized machine. Also, rather than having to fault simulate the entire test sequence in the optimized machine, only the justification sequences have to be fault simulated in the testable machine.

In all cases, the maximum possible fault coverage was achieved in the testable machine, i.e. all combinationally irredundant crosspoint faults are detectable in the sequential machine. The area penalties incurred are due to two reasons: (1) the constraints imposed during state assignment (2) the addition of extra edges to the STG to obtain R-reachability. As can be seen the area penalties are quite small, and compare favorably to Scan Design approaches. The gain in fault coverage and test generation times more than offsets the area penalty.

EX	#inp	#out	#states	#latches	#edges
sse	7	7	13	4	59
tbk	6	3	16	4	787
scf	27	54	97	7	168
dfile	2	1	24	5	99
planet	7	19	48	6	118

Table 1: Statistics of Benchmark Examples

EX	I - OPTIMIZE			II - TESTABLE		
	#prod.	fault cov.	tpg time	#prod.	fault cov.	tpg time
sse	33	89.56	6.4s	36	92.12	3.6s
tbk	56	90.21	22.1s	61	95.83	15.8s
scf	145	93.31	6.2m	154	96.07	3.3m
dfile	51	94.12	16.2s	54	98.81	6.1s
planet	97	91.72	93s	104	95.67	59.5s

Table 2: Synthesis for Testability Results

The number of test sequences required varied between 70-300 for these examples. The average length of each sequence was 5. Since the test vectors only access the primary inputs and only the primary outputs are observed, each vector can be applied in one clock cycle.

6 Conclusions

Previous approaches to synthesizing easily testable sequential machines from State Transition Graph descriptions have concentrated on the stuck-at fault model. For PLAs, an extended fault model called the crosspoint fault model is used. We have proposed a procedure of constrained state assignment and logic optimization which guarantees testability for all combinationally irredundant crosspoint faults in a PLA-based finite state machine. No direct access to the flip-flops is required. The test sequences to detect these faults can be obtained using PLA test pattern generation technique and breadth-first search on the State Transition Graph. This procedure thus represents an alternative to a Scan Design methodology. Preliminary results indicate that the area/performance penalties in return for easy testability are small.

7 Acknowledgements

We would like to thank Mr. Tony Ma for many interesting discussions on synthesizing testable sequential machines. This work was supported in part by the Semiconductor Research Corporation, and in part by the Defense Advanced Research Projects Agency under contract N00039-86-R-0365. Their support is gratefully acknowledged.

References

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

[2] M. A. Breuer. A random and an algorithmic technique for fault detection and test generation. In *IEEE Transactions on Computers*, November 1971.

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

[4] C. W. Cha. A testing strategy for plas. In *Proc. 15th Design Automation Conference*, June 1978.

[5] S. Devadas, H-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli. Mustang: state assignment of finite state machines for optimal multi-level logic implementations. In *Int'l Conference on Computer-Aided Design (ICCAD)*, November 1987.

[6] S. Devadas, H-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli. Synthesis and optimization procedures for fully and easily testable sequential machines. In *Proc. of International Test Conference*, September 1988.

[7] E. B. Eichelberger and E. Lindbloom. A heuristic test-pattern generator for programmable logic arrays. In *IBM J. Res. Develop.*, Jan 1980.

[8] E. B. Eichelberger and T. W. Williams. A logic design structure for lsi testability. In *Proc. 14th Design Automation Conference*, June 1977.

[9] G. De Micheli et. al. Optimal state assignment of finite state machines. In *IEEE Transactions on CAD*, July 1985.

[10] K. Bartlett et. al. Multi-level logic minimization using implicit don't cares. In *IEEE Transactions on CAD*, June 1988.

[11] H. Fujiwara and K. Kinoshita. A design of programmable logic arrays with universal tests. In *IEEE Transactions on Computers*, November 1981.

[12] S. J. Hong and D. L. Ostapko. Fault analysis and test generation for programmable logic arrays(pla's). In *IEEE Transactions on Computers*, September 1979.

[13] S. J. Hong and D. L. Ostapko. Fitpla: a programmable logic array for function independent testing. In *Dig. 10th Int. Symp. FTC*, 1980.

[14] H-K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli. Test generation for sequential finite state machines. In *Proc. of Int'l Conference on Computer-Aided Design (ICCAD)*, November 1987.

[15] S. Mallela and S. Wu. A sequential test generation system. In *Proc. of International Test Conference*, October 1985.

[16] R. Marlett. A comprehensive test generation system for highly sequential circuits. In *Proc. of 15th Design Automation Conference*, June 1978.

[17] S. Nitta, M. Kawamura, and K. Hirabayashi. Test generation by activation and defect-drive (tegad). In *INTEGRATION Journal*, 1985.

[18] H. D. Schnurmann, E. Lindbloom, and R. G. Carpenter. The weighted random test-pattern generator. In *IEEE Transactions on Computers*, July 1975.

[19] S. Shteingart, A. W. Nagle, and J. Grason. Rtg: automatic register level test generator. In *Proc. of 22nd Design Automation Conference*, June 1985.

[20] R. S. Wei and A. L. Sangiovanni-Vincentelli. Platypus: a pla test pattern generation tool. In *IEEE Transactions on CAD*, October 1986.