Copyright © 1988, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

A SYNTHESIS AND OPTIMIZATION PROCEDURE FOR FULLY TESTABLE SEQUENTIAL MACHINES

by

Srinivas Devadas, Hi-Keung Tony Ma,
A. Richard Newton, and
Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M88/14

12 February 1988

A SYNTHESIS AND OPTIMIZATION PROCEDURE FOR FULLY TESTABLE SEQUENTIAL MACHINES

by

Srinivas Devadas, Hi-Keung Tony Ma,
A. Richard Newton, and
Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M88/14

12 February 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A SYNTHESIS AND OPTIMIZATION PROCEDURE FOR FULLY TESTABLE SEQUENTIAL MACHINES

by

Srinivas Devadas, Hi-Keung Tony Ma,
A. Richard Newton, and
Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M88/14

12 February 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A Synthesis and Optimization Procedure for Fully Testable Sequential Machines

Srinivas Devadas, Hi-Keung Tony Ma,
A. Richard Newton and Alberto Sangiovanni-Vincentelli
Department of Electrical Engineering and Computer Sciences
Cory Hall
University of California, Berkeley, CA. 94720

Telephone
(415) 642-6633 or (415) 642-2967

Abstract

In this paper, we outline a synthesis procedure which beginning from a State Transition Graph description of a sequential machine produces an optimized **fully testable** logic implementation. This logic-level implementation is guaranteed to be testable for *all single stuck-at* faults and the *test sequences for these faults can be obtained using combinational test generation techniques alone.*

All single stuck-at faults in the synthesized logic-level automaton can be tested **without access to the memory elements** using these test sequences. Thus, the **testing time required is smaller than that using a Scan Design methodology.** The area penalty incurred due to the constraints on the optimization are small. The performance of the synthesized design is usually *better* than a unconstrained design optimized for area alone.

The relationship between combinational logic optimization and combinational test generation is well known. In this paper, we show that an intimate relationship exists between state assignment and the testability of a sequential machine. We propose a procedure of **constrained state assignment and logic optimization** which guarantees testability for **both Moore and Mealy machines.** We present results which illustrate the efficacy of this procedure – the area/performance penalties in return for 100% testability are negligible.

Acknowledgements

This work is supported in part by the Semiconductor Research Corporation, the Defense Advanced Research Projects Agency under contract N00039-86-R-0365 and a grant from AT&T Bell Laboratories. Their support is gratefully acknowledged.

1. INTRODUCTION

Test generation for sequential circuits has long been recognized as a difficult task [1]. A popular approach to solving the this problem is to make all the memory elements controllable and observable, i.e. Complete Scan Design [2] [3]. Scan Design approaches have been successfully used to reduce the complexity of the problem of test generation for sequential circuits by transforming it into a combinational one which is considerably less difficult. The design rules of Scan Design also constrain the sequential circuits to be synchronous so that the normal operation of the sequential circuit is free of races and hazards. However, there are situations where the cost in terms of area and performance of Complete Scan Design is unaffordable. Also, the testing time associated with Scan Design is very high because values have to be *sequentially* scanned into and out of the memory elements one clock cycle at a time.

Several approaches [4] [5] [6] [7] [8] [9] 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 [5] [8]. 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.

The relationship between combinational logic synthesis and test generation is well known. In [10], a synthesis procedure which guaranteed fully testable irredundant combinational logic circuits was proposed. Equally intimate relationships between the more complicated problems of sequential circuit synthesis and test generation have been envisioned.

In this paper, we outline a synthesis and optimization procedure which, beginning from a State Transition Graph description of a Moore or Mealy finite automaton, produces a 100% testable logic-level implementation of the machine. The test sequences for all single stuck-at faults in the machine can be derived using test generation algorithms on the combinational logic blocks of the machine. *All* single stuck-at faults in the synthesized logic-level automaton can be tested *without access to the memory elements* using these test sequences. Thus, the *testing time required is smaller than that using a Scan Design methodology.* The area penalty incurred due to the con-

straints on the optimization are small. The performance of the synthesized design is usually *better* than a unconstrained design optimized for area alone.

We show that a strong relationship exists between state assignment, logic optimization and testability of a sequential machine. We outline a procedure of constrained state assignment and combinational logic optimization which ensures 100% testability for both Moore and Mealy finite state machines. Results obtained on benchmark examples show that the area penalties incurred due to the constraints imposed during state coding and logic optimization are small. The performance of the resulting circuits is *better* than that of unconstrained designs optimized for minimum area (This is because one of the constraints imposed requires combinational logic partitioning in the machine).

Basic definitions and terminologies used are given in Section 2. In Section 3, we state the necessary conditions required for a fully testable Moore machine. Extensions to Mealy machines are made in Section 4. In Section 5, we discuss how an existing state assignment algorithm can be modified to produce a constrained encoding satisfying the testability criterion. Results obtained thus far are presented in Section 6.

2. PRELIMINARIES

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 $\bar{a}$). A **cube** is a set C of literals such that $x \in C$ implies $\bar{x} \notin C$ (e.g., $\{a, b, \bar{c}\}$ is a cube, and $\{a, \bar{a}\}$ 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, $\{\{a\}, \{b, \bar{c}\}\}$ is an expression consisting of the two cubes $\{a\}$ and $\{b, \bar{c}\}$. 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.

The **distance** between two minterms is defined to be the number of bit positions they differ in.

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.

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 to correspond to a **completely specified machine.** An **incompletely specified machine** is one where at least one transition edge from some state is not specified.

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 for every possible state, q, in the STG an input sequence exists which when applied to the machine, initially at the reset state, places the machine in q. Thus every state is **reachable** from the reset state. Note that requiring a machine to be R-reachable is a less stringent condition than requiring it to be strongly connected – a strongly connected machine is R-reachable but not vice versa.

The fault model assumed is **single stuck-at.** A finite state machine is assumed to be implemented by combinational logic and feedback registers. Tests are generated for stuck-at faults in the combinational logic part.

A combinational logic network is said to be **irredundant** if all the faults in the network are testable.

To detect a fault in a sequential machine, the machine has to be placed in a state which can then excite and propagate the effect of the fault to the primary outputs. The first step of reaching

the state in question is called **state justification**. The second step is called **fault excitation-and-propagation**.

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 the fault. A path in a State Transition Graph is said to be corrupted if at least one edge in the path has been corrupted.

3. FULLY TESTABLE MOORE MACHINES

A general model for a Moore finite state machine is shown in Figure 1. It is realized by two logic blocks, the Output Logic (*OL*) block and the Next State Logic (*NSL*) block, and feedback registers. In a Moore machine, the outputs depend only on the present state of the machine.

Given n latches in the machine, the machine has 2^n possible states. However, the number of states in a State Transition Graph (STG) description of a machine need not necessarily be an integer power of 2.

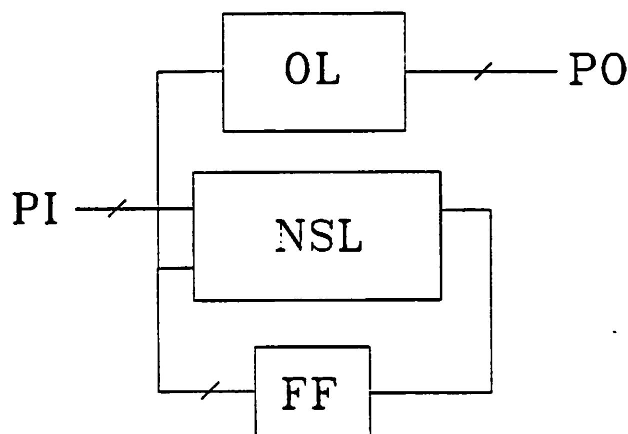

Fig. 1: General Moore Machine Model

We first prove the following result.

Theorem 3.1: Given a n-latch logic-level implementation of a Moore machine (shown in Figure 1), if (1) the combinational logic blocks OL and NSL are irredundant (2) the machine is R-reachable i.e. all 2^n states are reachable from the reset state and (3) all the 2^n states have distinct outputs, the machine is fully testable for all stuck-at faults in OL and NSL.

Proof: Consider a fault, F, in the OL block. Since the block is irredundant (Condition 1), a state, s, exists which detects F. This state, s, can be reached from the reset state, R, of the machine via an input sequence, I, because the machine is R-reachable (Condition 2). State s will be reached on applying I from R regardless of F since F is in the OL block. Therefore, a sequence exists, namely I, which can detect F.

Now consider a fault, F in the NSL block. Again, since NSL is irredundant, a state, s, and an input i exist which propagate the effect of this fault to the next state lines. Instead of obtaining the true next state, q, we obtain a faulty next state q^F. q and q^F have distinct outputs (Condition 3). Therefore, at the next clock cycle the effect of F is propagated to the primary outputs. We however, have to reach s from R. A path exists from s to R (Condition 2). However, this path may or may not have been corrupted by F. If the path has not been corrupted, we can detect F after reaching s and applying input i. If the path has been corrupted, it means that for some edge in the path, the next state reached was different due to F. In this case, the fault is detected even *before* reaching s, since two different states were reached in the faulty and fault-free machine. Q.E.D.

We now analyze the implications of each of the conditions of Theorem 3.1. (1) is an essential condition. Obviously, a redundant fault in NSL or OL cannot be detected in the sequential machine. Redundancies are sometimes introduced for performance reasons, but mostly they are due to unoptimized logic [10]. An irredundant logic network would have minimum area. With recent advances in multi-level logic optimization, large networks can be made irredundant. If

redundancies are required in the combinational logic for performance reasons, the proposed procedure will still guarantee testability and produce tests for all combinationally irredundant faults.

In general, State Transition Graph specifications of machines have reset states and are R-reachable. However, as mentioned previously, a STG specification of a machine need not necessarily have $N_s = 2^k$ states, $k = 1, 2...$ 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 STG's encountered in practical design have a large number of transitions that are not specified.

Condition 3 is obviously unacceptable, since if the STG specification does not satisfy it, it cannot be made to do so without changing the functionality of the machine. This condition is now relaxed.

Consider the logic-level implementation of the Moore machine shown in Figure 2. The *NSL* block has been realized as n distinct single-output circuits or *partitions*. The following theorem shows that a *constrained state assignment* can ensure a fully testable circuit.

Theorem 3.2: Given a n-latch logic-level implementation of a Moore machine (shown in Figure 2), if (1) the combinational logic blocks OL and NSL_i, $i = 1, 2 .. n$, are irredundant (2) the machine is R-reachable and (3) if the state encoding of the machine is such that each pair of states asserting the same output has codes of distance-2 from each other, the machine is fully testable.

Proof: The faults in the OL block are detected as before in Theorem 3.1. Consider a fault F in the NSL block. Without loss of generality, assume that F is in the first partition. The effect of the fault when detected is to produce a 0 (1) instead of a 1 (0) at the NSL_1. In either case, the faulty next state produced, q^F, will differ from the true state, q, in at most one bit. Since state assign-

ment has guaranteed that all states asserting the same outputs have been assigned distance-2 codes, q and q^F assert different outputs. This means that F is detected in the next clock cycle. Q.E.D.

A realization of a machine like the one shown in Figure 2 implies that logic cannot be shared between next state lines. Thus, a certain area penalty may be associated with such an implementation. The performance of the circuit does *not* suffer due to logic partitioning (and in fact may be improve). However, the implementation shown is an extreme case and can be generalized. A partition may contain more than one NSL_i. This means that the logic between these lines *can* be shared.

The number of *NSL* partitions required relates to the number of states asserting the same output in the original STG. We first show that the state assignment constraint (Condition 3 of Theorem 3.2) can be satisfied quite easily.

Lemma 3.1: Given a State Transition Graph, if at most half the number of states assert the same output, a state assignment satisfying a distance-2 constraint between states with the same outputs

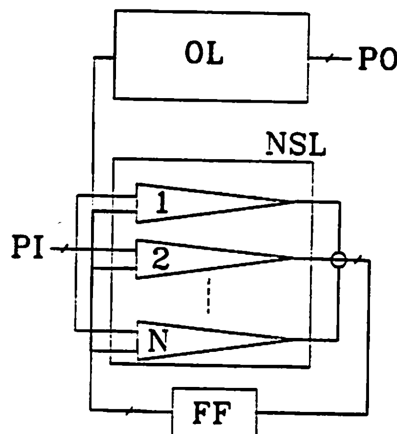

Fig. 2: Partitioned Moore Machine Model

can be found.

Proof: Given k bits, we have 2^k possible codes. These codes can be split into 2 sets each of cardinality 2^{k-1}, such that codes within each set are of distance-2. Given a STG with N_s states, we add states to raise the number of states to $2^{\lceil log(N_s) \rceil}$. The number of states in a distance-2 set is $2^{\lceil log(N_s) \rceil - 1} \geq \frac{N_s}{2}$.

We now prove the following result which gives us the required number of partitions of the *NSL* lines as a function of the number of states with the same output.

Theorem 3.3: If at most k states exist in a State Transition Graph which produce the same outputs, $\lceil log(k) \rceil + 1$ separate partitions suffice to obtain a fully testable machine.

Proof: In the worst possible case, if we have 2^n states in the machine, we have a situation where $\left\lfloor \frac{2^n}{k} \right\rfloor$ sets of states exist, the states within each set asserting the same output.

We need to ensure for each set that no two of these k states are ever produced as a fault-free faulty pair due to a fault in *NSL*. This means that the codes assigned to any two of these states must differ in at least two next state lines belonging to two distinct partitions. By Lemma 3.1, the number of bits required to generate 2 sets of 2^{p-1} distance-2 codes is p. To generate 2 sets of k distance-2 codes, we require $\lceil log(k) \rceil + 1$ partitions. We now have $n - (\lceil log(k) \rceil + 1)$ bits remaining. This means we can have

$$2^{n - \lceil log(k) \rceil - 1} \times 2 = \frac{2^{n-1}}{2^{\lceil log(k) \rceil}} \times 2 = \left\lfloor \frac{2^n}{k} \right\rfloor$$

sets each with k codes which differ in two next state lines belonging to two distinct partitions. Q.E.D.

There are thus three steps in producing combinational logic specifications for *OL* and *NSL* blocks from a State Transition Graph description. 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 outputs and (3) state assignment obeying the constraint relations generated. A straightforward solution exists for Steps 1 and 2, however the optimality of the eventual implementation depend on the choices made during these steps. 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. Similarly, if the number of required partitions is less than the number of next state lines, choices exist as to which next state lines to group together. Next state lines which can share logic maximally should be placed in the same partition. In Step 3, an optimal state assignment which minimizes combinational logic while meeting the distance constraints has to be found. This step is further discussed in Section 5.

After obtaining the combinational logic specifications, logic optimization algorithms which can ensure an irredundant logic network (e.g. [10]) can be applied. If redundancies are required in the logic, this synthesis procedure ensures that all combinationally irredundant faults are sequentially irredundant as well.

To generate tests for the sequential machine, tests vectors are generated for all stuck-at faults in the *OL* and *NSL* combinational circuits. Then, justification paths are obtained from the STG using simple breadth-first search. It is guaranteed (by the theorems proved in this section) that these paths concatenated with the test vectors applied to the primary inputs of the sequential machine will detect all possible faults in the machine so as to be observable at the primary outputs.

4. FULLY TESTABLE MEALY MACHINES

A general model for a Mealy finite state machine is shown in Figure 3. It is realized by a single logic block and feedback registers. The output logic and the next state logic are both realized by one block. In a Mealy machine, the outputs depend on both the present state as well as the primary inputs. A model for a Mealy machine with each next state line realized as a separate circuit and with the output and next state logic separated is shown in Figure 4.

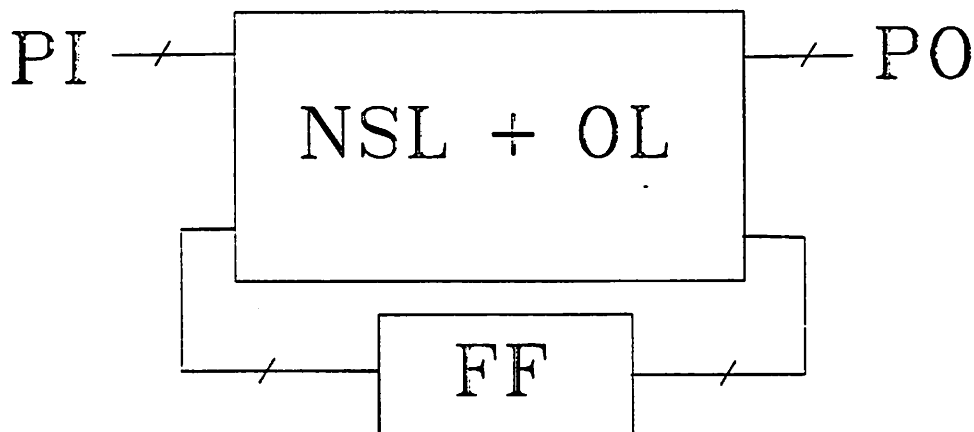

Fig. 3: General Mealy Machine Model

We prove a theorem in direct correspondence to Theorem 3.2 for Mealy machines. First, we define the notion of O-equivalence between two states in a Mealy machine.

Definition 4.1: Two states in a Mealy machine are said to be **O-equivalent** if each pair of fanout edges on the same input from these states produces the same output.

Theorem 4.1: Given a n-latch logic-level implementation of a Mealy machine (shown in Figure 4), if (1) the combinational logic blocks OL and NSL_i, $i = 1,2 .. n$, are irredundant (2) the machine is R-reachable i.e. all 2^n states are reachable from the reset state and (3) if the codes of states of the machine are such that each pair of O-equivalent states have codes of distance-2 from each other, the machine is fully testable.

Proof: Consider a fault in the OL block. There exists a state, s and input i which detects this fault by Condition 1. R-reachability and the fact that F is in the OL block imply that state s can be reached from R. F can thus be detected.

Consider a fault F in the NSL block. Without loss of generality, assume that F is in the first partition. Since this partition is irredundant, a state s and an input i_1 exist which can propagate the

effect of the fault to the next state line. The effect of the fault when detected is to produce a 0 (1) instead of a 1 (0) at the NSL_1. In either case, the faulty next state produced, q^F, will differ from the true state, q, in at most one bit. Condition 3 guarantees that q and q^F are *not* O-equivalent since all O-equivalent states have distance-2 codes. This means that an input, i_2, exists which will produce a different output in the faulty machine (which is in q^F) from the fault-free machine (which is in q). We, however, have to reach s from R. A path exists from s to R (Condition 2). However, this path may or may not have been corrupted by F. If the path has not been corrupted, we can detect F after reaching s and applying input i_1 followed by i_2. If the path has been corrupted, it means that for some edge in the path, the next state reached was different due to F. We have a fault-free/faulty pair (q', q'^F) By the argument above, an input i_3 which produces a different output for q' and q'^F exists, thus detecting F. Q.E.D.

The synthesis procedure for obtaining a fully testable Mealy machine is the same as the procedure outlined for the Moore machine in Section 3. To generate tests for the machine, as before, all the combinational logic tests for the *OL* and *NSL* blocks are generated. The justification path to the state detecting the fault concatenated with the primary input part of the combinational test vector and the differentiating input vector (for the fault-free/faulty next state pair) constitutes the test sequence for a given fault.

5. 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. [11] [12]). The program MUSTANG [12] produces a state assignment that heuristically minimizes the number of literals in the combinational logic *after* multiple-level logic optimization.

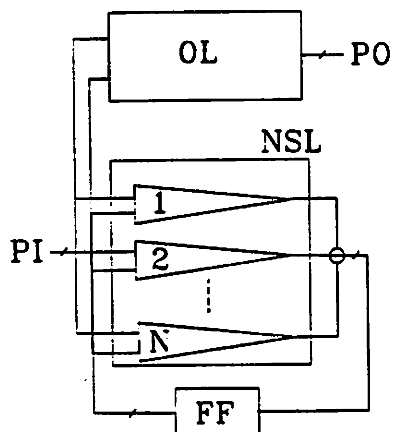

Fig. 4: Partitioned Mealy Machine Model

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 distance-2 constraints. A minimally-distant set of codes satisfying these constraints is assigned to the states.

6. RESULTS

Results obtained on four State Transition Graphs from the MCNC 1987 Logic Synthesis Workshop benchmark set are given in Table 1. First, the machines were encoded and optimized disregarding testability. The number of gates in the machine, the fault coverage obtained and the

was accomplished using an efficient test generation algorithm that was recently proposed [13]. Then, each of the machines were synthesized using the procedure described in Sections 3 & 4. Again, the number of gates, fault coverage obtained and the test generation time are given. Sequential test generation for these circuits was faster because combinational test generation and breadth-first search suffice to produce the test sequences. The example *scf* is a Moore machine, the others Mealy machines.

The area penalties incurred are due to three reasons : (1) the constraints imposed during state assignment (2) the addition of extra edges to the STG to obtain R-reachability and (3) logic partitioning constraints. Empirical evidence has shown that (3) is easily the most significant factor – the next state lines may have to be realized as separate circuits. Additionally, for a Mealy machine, unlike in an unconstrained design, the next state and the output logic have to be separated.

Logic partitioning is extensively used to gain higher performance. A Mealy machine with separate next state and output logic blocks can be clocked faster than a machine with a single lumped block of logic. This is the case in the example designs of Table 1 as well. Thus, the fully testable machines produced by logic partitioning may well represent a more desirable point in the area/performance trade-off curve.

The number of gates in a circuit is, in general, indicative of the area required to implement the circuit. However, in some cases, this measure of area may not be very accurate. To obtain accurate estimates of circuit areas, the synthesized examples of Table 1 were placed and routed using the TimberWolf standard cell placement and routing package [14]. The areas of the resulting designs after place and route for the unconstrained and constrained cases are given in Table 2. For each example, the areas of the designs have been normalized to that of the unconstrained design.

In Table 2, some constrained designs are about the same size or smaller than the corresponding unconstrained ones. Logic partitioning, in these cases, has decreased routing complexity in the circuit to the extent of nullifying the increase in the number of logic gates. The cost function used

EX	#inp	#out	#states	#lat	I - OPTIMIZE			II - TESTABLE		
					#gates	fault cov.	tpg time	#gates	fault cov.	tpg time
sse	7	7	13	4	91	84.57	69.9s	129	100.0	5.2s
tbk	6	3	16	4	181	98.57*	72.1s	231	98.57*	4.1s
scf	27	54	97	7	502	96.14	83.1m	541	100.0	71s
dfile	2	1	24	5	124	96.94	104s	144	100.0	2.0s
planet	7	19	48	6	417	98.82	373s	449	100.0	14s

s is CPU-seconds, m is CPU-minutes on a VAX 11/8650 running ULTRIX
* OL block was not combinationally irredundant

Table 1: Synthesis for Testability Results

EXAMPLE	I - OPTIMIZE		II - TESTABLE	
	#gates	area	#gates	area
sse	91	1.0	129	1.34
tbk	181	1.0	231	1.10
scf	502	1.0	541	1.01
dfile	124	1.0	144	0.98
planet	417	1.0	449	0.86

Table 2: Areas of Standard Cell Designs

in multi-level logic optimization is the number of literals (transistors) in the circuit [15], and is sometimes is a poor estimate of the circuit area.

The number of test sequences required varied between 30-70 for these examples. The number of test sequences can be reduced by applying combinational test compaction strategies after generating all the test vectors for the combinational logic blocks. The average length of each sequence was ~5. Since the test vectors only access the primary inputs and only the primary out-

puts are observed, each vector can be applied in one clock cycle.

7. CONCLUSIONS

We have described a synthesis procedure that produces an optimized fully testable logic implementation of a sequential machine from a State Transition Graph description of the machine. This logic-level implementation is guaranteed to be testable for all single stuck-at faults. No access to the memory elements is required. The test sequences for these faults can be obtained using combinational test generation techniques alone.

We have shown that an intimate relationship exists between state assignment and the testability of a sequential machine. A procedure of constrained state assignment and logic optimization can guarantee a fully testable machine.

The testing time required in this method is smaller than that using a Scan Design methodology. Experimental results have shown that the area penalty incurred due to the constraints on the optimization are small. The performance of the synthesized design is usually better than a unconstrained design optimized for area alone.

REFERENCES

1. M. A. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, 1986.
2. E. B. Eichelberger and T. W. Williams, "A logic design structure for LSI testability," *Proc. 14th Design Automation Conference*, June 1977, 462-468.
3. V. D. Agarwal, S. K. Jain and D. M. Singer, "Automation in Design for Testability," *Proc. of Custom Integrated Circuits Conference*, Rochester, NY, May 21-23 1984.
4. M. A. Breuer, "A Random and an Algorithmic technique for fault detection and Test generation for sequential circuits," *IEEE Transactions on Computers*, Vol. C-20(November 1971), 1366-1370.
5. H. D. Schnurmann, E. Lindbloom and R. G. Carpenter, "The Weighted Random Test-Pattern Generator," *IEEE Transactions on Computers*, Vol. C-24(July 1975), 695-700.
6. R. Marlett, "EBT: A Comprehensive Test Generation Technique for highly sequential circuits," *Proc. of 15th Design Automation Conference*, Las Vegas, June 1978, 332-338.
7. S. Mallela and S. Wu, "A Sequential Test Generation System," *Proc. of International Test Conference*, Philadelphia, PA, October 1983, 57-61.
8. S. Nitta, M. Kawamura and K. Hirabayashi, "Test Generation by Activation and Defect-Drive (TEGAD)," *INTEGRATION, the VLSI Journal*, Vol. 3 (1985)(1985), 2-12.
9. S. Shteingart, A. W. Nagle and J. Grason, "RTG: Automatic Register Level Test Generator," *Proc. of 22nd Design Automation Conference*, Las Vegas, June 1985, 803-807.
10. K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. L. Sangiovanni-Vincentelli and A. R. Wang, "Multi-level logic minimization using implicit don't cares," *submitted IEEE Transactions on CAD*, August 1986.
11. G. D. Micheli, R. K. Brayton and A. Sangiovanni-Vincentelli, "Optimal state assignment of finite state machines," *IEEE Transactions on CAD*, July 1985, 269-285.
12. S. Devadas, H. T. Ma, A. R. Newton and A. Sangiovanni-Vincentelli, "MUSTANG: State Assignment of Finite State Machines for Optimal Multi-Level Logic Implementations," *Proc. of Int'l Conference on Computer-Aided Design*, Santa Clara, November 1987.
13. H. K. T. Ma, S. Devadas, A. R. Newton and A. L. Sangiovanni-Vincentelli, "Test Generation for Sequential Finite State Machines," *Proc. of Int'l Conference on Computer-Aided Design (ICCAD)*, Santa Clara, November 1987.
14. C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *IEEE Transactions on Circuits and Systems*, April 1985.
15. R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A. Wang, "MIS: A Multiple Level Logic Optimization System," *IEEE Transactions on CAD*, November 1987.