Simplifying Sequential Circuit Test Generation

MENG-LIEH SHEU
CHUNG LEN LEE
National Chiao Tung University

Design for testability reduces testing costs for sequential circuits. The authors present a parity checker DFT scheme they have incorporated into a finite-state machine synthesis system. Generating tests for circuits synthesized according to this scheme becomes extremely simple. The derived test sequence very efficiently detects faults.

SEQUENTIAL TEST generation poses a difficult problem for circuits implemented from finite-state machines. The flip-flops in sequential circuits synthesized from FSMs generally have intricate cyclic structures that complicate sequential test generation. We have developed a parity checker design-for-testability scheme that significantly enhances circuit testability, thus simplifying the testability problem.

Years ago, Hennie[1] proposed a checking experiment to test FSMs by deriving "distinguish" sequences that can differentiate good machines from faulty ones. The very long test sequence involved, plus the near impossibility of deriving the distinguish sequence sometimes makes this approach impractical.

To shorten the derived checking sequence, Fujiwara et al.[2] proposed making machines output-observable by adding a number of extra outputs. This method reduces the length of the checking sequence for the modified machine nearly to a minimum.

For practical circuit designs, manufacturers commonly use scan designs[3] to facilitate sequential circuit testing. To enhance circuit controllability and observability, full scan designs chain all the storage elements into a shift register. Such designs, though, involve relatively large hardware overhead and long test sequences.

Partial scan designs, by contrast, select only a subset of flip-flops for scanning. Though they significantly reduce the large hardware overhead and shorten the test sequence, several factors argue against their use.

- Sequential test generation is inevitable for the nonscanned flip-flops.
- All scan designs may degrade performance due to the extra delay introduced in the scan path.
- Additional input/output pins are necessary for the scan data I/O and scan clock control.
- The scan mode cannot operate at the speed of the circuit during normal operation.

Researchers have developed many efficient sequential test generation algorithms to generate test sequences for FSM circuits based on the stuck-at-fault model. Due to the heavy computation involved, these can generate test sequences only for moderately large circuits.

Recently, Cheng et al.[4] proposed a functional test generation method for FSMs based on their single-state transition fault model. The derived functional

0740-7475/94/$04.00 © 1994 IEEE

Useful definitions

DFT	Design for testability
DS	Distinguish sequence
DT	Distinguish table
FSM	Finite state machine
FTG	Functional test generation
PCDFT	Parity checker DFT
SPDS	State-pair distinguishing sequence
SSPDS	Shortest SPDS
SST	Single-state transition

test sequences have high stuck-at fault coverages, demonstrating the effectiveness of this functional fault model.

One solution for this testing problem is to design (or synthesize) the circuit of an FSM in a testable way. For example, Devadas et al.[5] proposed a procedure to synthesize highly testable FSMs that eliminates sequential redundancies of an FSM. Agrawal et al.[6] devised a general architecture of testable design for FSMs in which the FSM function merges with a test function during circuit synthesization. The same group[7] also developed a state assignment procedure for synthesizing testable designs that produces highly testable designs by reducing the cyclic structure of the flip-flops.

Saucier et al.[8] developed a method to synthesize concurrently checked controllers. It employs an embedded signature monitoring approach and signature justification method to verify signatures of the paths of a self-checking FSM. Recently, Fujiwara et al.[9] proposed a parity scan design to shorten the test application time. In their approach, the circuit is fully scanned and treated as a combinational circuit during testing. To eliminate the scan-out sequence of parity-testable faults, an added parity checker checks all flip-flops.

Our new parity checker DFT scheme for FSMs[10] uses a parity checker to monitor state changes after FSMs have made state transitions to detect state transition faults. For the circuit, which still works as a sequential circuit during testing, the parity checker directly observes the fault effect appearing at the state lines. This eliminates any propagation or scan-out sequence. Our approach significantly enhances circuit testability, making the testing problem much simpler.

Before proceeding to the parity checker DFT scheme for FSMs, readers should acquaint themselves with several definitions of distinguishability and theorems for distinguishable machines that we address in the accompanying box on the next page.

Parity checker DFT scheme

For an FSM to detect an SST fault, we need an input sequence that will propagate the fault effect of the faulty state pair to primary outputs. During the fault effect propagation, three possible cases can cause the fault effect to disappear such that the primary outputs cannot observe the fault effect:

1. The faulty state pair is equivalent.
2. The faulty state pair is partially equivalent.
3. The faulty state pair forms a loop, as shown in Figure 1a, where *A*, *B* are two states of a faulty state pair.

In Figure 1a, for the first case, states *A* and *B* transit to the same next states, giving the same outputs under any input. They are equivalent. This is the equivalent sequential redundancy reported by Devada, Ma, and Newton.[11]

For the second case, states *A* and *B* transit to the same next state *C* with the same output *O* under certain input *I*. They are partially equivalent. In this case, if the input *I* is accidentally chosen as the propagation sequence, the fault effect will disappear.

For the third case, states *A*, *B*, and *C* transit to each other to form a loop with the same output *O* under the same input *I*. In this case, states *A*, *B*, and *C* may be fully or partially equivalent. The case having more states in such a loop presents the same situation as the third case.

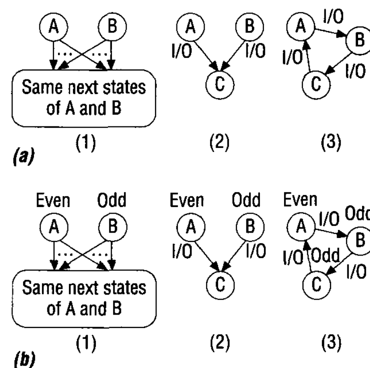


Figure 1. Three cases that can cause the fault effect to disappear (a), and ways to eliminate fault-effect disappearance (b). The faulty state pairs are equivalent (case 1), partially equivalent (case 2), and form a loop (case 3).

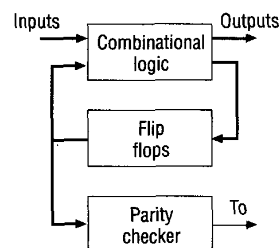


Figure 2. The parity checker DFT scheme eliminates the propagation or scan-out sequence required by other schemes.

We propose a new DFT scheme to further eliminate the propagation or scan-out sequence that other schemes require. In our scheme, as shown in Figure 2, we attach a parity checker to the state lines to monitor the fault effects appearing at the flip-flops.

Basically, we assign the states of the FSM with codes of a particular parity. The parity checker checks the parity of the flip-flops that code the states of the machine. Whenever an SST fault oc-

Definitions and theorems

In a state transition graph representing an FSM, the vertices stand for the states of the FSM and the directed edges represent state transitions. We assume that the FSM has a reset state. When implementing an FSM into circuit form, any nonredundant stuck-at fault of the circuit will corrupt some state transitions of the machine's state transition graph. By corrupt, we mean that for a state transition, either or both of its output and next states are changed.

Based on this assumption, Cheng et al.[1] proposed a single-state transition fault model. An SST fault exists for an FSM if one of its state transitions transits to an incorrect destination state. To detect an SST fault, the test sequence contains three subsequences:

- the initialization sequence, which drives the machine to the source state of the faulty state transition
- the activation vector, which activates the fault to generate a [good/faulty] state pair
- the propagation sequence, which gives the machine output that differentiates the [good/faulty] state pair.

Definition 1.[2] Let S_m and S_n be two states of an FSM F. S_m and S_n are equivalent, if the output responses of F are identical for any input sequence starting with S_m and S_n.

Definition 2. For an FSM F, S_m and S_n are partially equivalent if at least one input sequence drives two states S_m and S_n of F to a state S_c with an identical output response.

Definition 3.[2] S_m and S_n are distinguishable if they are not equivalent.

Definition 4.[3] An input sequence is a distinguishing sequence (DS) for an FSM F, if the output responses of F are not the same for each different initial states. If this sequence exists only for a specific state pair, we call it a state pair distinguishing sequence (SPDS) of this state pair. A state pair may have many different SPDSs. SSPDS denotes the shortest SPDS of the state pair.

There is no SPDS for two equivalent states, and the input sequence that makes two states partially equivalent is not an SPDS either. An SSPDS varies in length between 1 and $(N-1)$, where N is the number of states.

Definition 5. A state pair $[S_m, S_n]$ of an FSM F is a k-distinguishable state pair, if the length of the SSPDS of $[S_m, S_n]$ is k. A state S_m is a K-distinguishable state, if the longest length of all SSPDSs of $[S_m, S_n]$ is K, where S_n is another state of F. If the maximum K value of all states of F is κ, F is called a κ-distinguishable machine.

The SSPDS of a state pair is the shortest propagation sequence for an SST fault to differentiate a faulty state from a good state.

With the above definitions, we have the following theorem.

Theorem 1. For a κ-distinguishable FSM F with N states and M state transitions, the upper bound of the length of the test sequence for detecting all SST faults of F is $(\kappa+N)M(N-1)$.

Proof. For an SST fault, traversing from a state to the source state of the SST fault requires at most $(N-1)$ state transitions. After the machine reaches the source state, an input vector can activate the

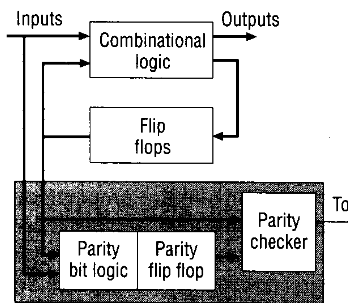


Figure 3. The parity checker DFT scheme with concurrent error detection. Adding one flip-flop and its associated logic to preserve the parity of the states will extend our approach to a concurrent error detection design.

curs, a faulty state pair arises, and the faulty state demonstrates an opposite parity that the parity-checker detects. Hence, by assigning different parities to states of the state pair as shown in Figure 1b, our DFT scheme can avoid all three cases of state undistinguishness just described.

For case 1 of Figure 1b, we assign different parity to states A and B, as they have the largest undistinguishability. The parity checker will directly detect the fault effect of the equivalent faulty state pair, thus rendering the propagation or scan-out sequence unnecessary. The same exercise can also apply to state pairs of cases 2 and 3. After assigning different parities, the equivalent, partially equivalent or loop states become totally distinguishable. The parity checker can immediately detect all fault effects propagated to the state lines, making the test sequence to detect faults shorter and removing some sequential redundant faults.[11] Test generation for circuits synthesized according to this scheme becomes very simple.

With one more flip-flop and its associated logic for preserving the parity of the states (see the shaded portion of Figure 3), this scheme not only helps detect the faults of an FSM during testing. It can also serve as a scheme for the

Definitions and theorems *continued*

faulty transition for the fault and generate a faulty state pair. Since the machine is κ-distinguishable, the propagation sequence of any faulty state pair is no longer than κ. The maximum length for the input sequence to detect this SST fault is $(N-1+1+\kappa) = (\kappa+N)$. Since there are M transitions, and each transition has at most $(N-1)$ wrong next states, the total number of SST faults is $M(N-1)$. Hence, the maximum length for the input sequence to detect all SST faults is $(\kappa+N)M(N-1)$. For the worst case, $\kappa = N$, the upper bound becomes $2NM(N-1)$.

Definition 6. If the two states of a state pair have different output responses, no matter what applied input sequences, they are a totally distinguishable state pair.

Definition 7. A state S_m of an FSM F is a totally distinguishable state if $[S_m,S_n]$ is a totally distinguishable state pair for any state S_n, where S_n is another state of F.

Detecting an SST fault that creates a totally distinguishable faulty state pair is very easy. All the input patterns can detect such a fault. Detecting the SST fault that causes faulty state transitions whose next states are totally distinguishable is also very easy. Applying any input at the next clock quickly reveals such a fault. Hence, activating the transition whose next state is a totally distinguishable state will reveal an SST fault.

Definition 8. For a machine F, if all of its states are totally distinguishable states, F is a totally distinguishable machine.

Theorem 2. To detect all the SST faults of a totally distinguishable machine, the input sequence needs to traverse all the state transitions; the upper bound of such traversing is $MN+1$, where M is the number of transitions and N is the number of states.

Proof. For a totally distinguishable machine, all states are totally distinguishable—all state transitions have totally distinguishable next states, or all state pairs are totally distinguishable. Hence, while a state transition activates $(N-1)$ SST faults, the current transition detects other $(N-1)$ SST faults that a previous state transition activated. That is, the $(N-1)$ SST faults activated by the current state transition will be detected at the next clock under any applied input.

The machine needs only to traverse all the M-state transitions to activate all $M(N-1)$ SST faults; one more state transition will reveal the $(N-1)$ faults activated by the last state transition. For each state transition, it will take at most $(N-1)$ transitions to initialize the present state that will be the source state of a faulty transition, plus one more input to activate the $(N-1)$ SST faults. Any input following the faulty transition can then detect the $(N-1)$ SST faults and activate another $(N-1)$ SST fault. The upper bound of such traversing is $M(N-1+1)+1 = MN+1$.

References

1. K.T. Cheng and J.Y. Jou, "A Functional Fault Model for Sequential Machines," *IEEE Trans. Computer-Aided Design*, Vol. 11, 1992, pp. 1065-1073.
2. S.C. Lee, *Digital Circuits and Logic Design*, Prentice-Hall, Englewood Cliffs, N.J., 1976.
3. K.T. Cheng and J.Y. Jou, "Functional Test Generation for Finite State Machines," *Proc. Int'l Test Conf.*, CS Press, 1990, pp. 162-168.

fault-tolerant design of an FSM for concurrent error detection. In such a case, the output of the parity checker (labeled TO) serves as an error indicator of transient faults appearing at flip-flops during normal operation.

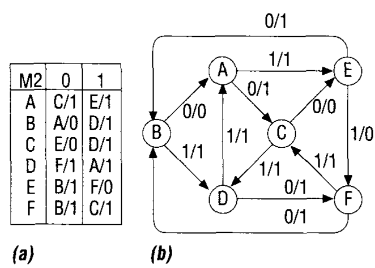
To reflect the testability of an FSM, we use the value κ and the number of κ-distinguishable states as parameters. For a κ-distinguishable FSM described by a state transition graph, we can also derive the number κ as well as the number of the κ-distinguishable states. The larger the value of κ and the number of κ-distinguishable states, the longer a test sequence must be to test this machine. Also, the more difficult this test sequence will be to generate. We want, therefore, to design a machine having only totally distinguishable states. Such a machine makes test generation very easy and greatly shortens the obtained test sequence.

The value κ and the number of κ-distinguishable states are generally greater than one for an FSM. To reduce κ—to make a machine totally distinguishable—our scheme uses a parity checker to design the FSM and makes all SST faults having a lower level of testability to be observed immediately at the output of the parity checker by assigning different state parity to the SST faults.

State parity assignment

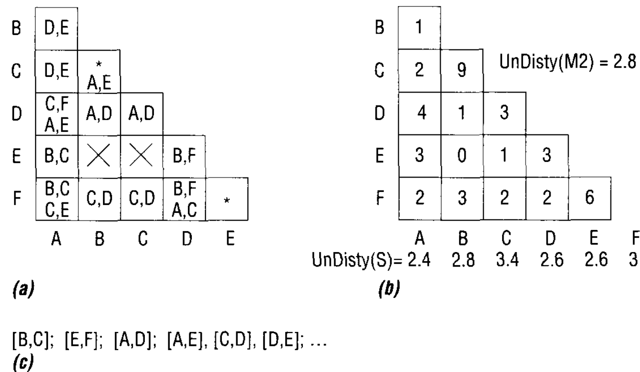
The testability feature of our approach rests on the fact that different parities on the states of the FSM will occur as the machine becomes faulty. By checking the parity, our scheme can differentiate the faulty state pair. However, for an FSM, seldom will all states or state pairs be totally distinguishable. In our proposed DFT scheme, we have developed a state assignment procedure that assigns state pairs of the FSM to maximize the number of totally distinguishable states.

To better understand the state parity assignment procedure, let's look at the example of FSM M2.[4] Figure 4 (next page) shows the state transition table



M2	0	1
A	C/1	E/1
B	A/0	D/1
C	E/0	D/1
D	F/1	A/1
E	B/1	F/0
F	B/1	C/1

Figure 4. A sample FSM M2: state transition table (a), state transition graph (b).



B	D,E				
C	D,E	* A,E			
D	C,F A,E	A,D	A,D		
E	B,C	X	X	B,F	
F	B,C C,E	C,D	C,D	B,F A,C	*
	A	B	C	D	E

(a)

B	1					
C	2	9				
D	4	1	3			
E	3	0	1	3		
F	2	3	2	2	6	
	A	B	C	D	E	F
UnDisty(S)=	2.4	2.8	3.4	2.6	2.6	3

UnDisty(M2) = 2.8

(b)

[B,C]; [E,F]; [A,D]; [A,E], [C,D], [D,E]; ...
(c)

Figure 5. The distinguish table constructed from the state transition table of M2: the distinguish table (a), undistinguishability table (b), and the order list (c).

and the associated state transition graph of M2. At first, we construct a distinguish table (DT), as shown in Figure 5a, from the state transition table. The distinguish table compiles the distinguishability of each state. For this FSM, states B and E are totally distinguishable because they transit to different states with different outputs under any input. Also, states C and E are totally distinguishable.

Hence, in the distinguish table, we mark the table entries of state pairs $[B,E]$ and $[C,E]$ with an X. States B and C transit to the same next state D with same output 1 under input 1, but transit to states A and E with output 0 under input 0. States B and C thus are partially equivalent, or may be equivalent if states A and E are equivalent. We mark the table entry of state pair $[B,C]$ with an asterisk and an $[A,E]$. States A and B transit to different next states and have different outputs under input 0 but the same output, that is 1, under input 1. We also mark the table entry of state pair $[A,B]$ with $[D,E]$, which are the next states of states A and B under input 1.

The accompanying Procedure for Building a distinguish table box shows the procedure we derive to construct the distinguish table from the state transition table for an FSM.

Procedure for building a distinguish table

{
For each State-Pair $[S_i,S_j]$ **do**
  {
  For each input k transition T_k do
    {
    if outputs of transitions (S_i,T_k) and (S_j,T_k) are the same **then**
      if Next states of transitions (S_i,T_k) and (S_j,T_k) are the same **then**
        mark an "*" in $[S_i,S_j]$ entry of DT; /**Partially_Equivalent* */
      else
        mark a Next_State_Pair$[(S_i,T_k),(S_j,T_k)]$ in $[S_i,S_j]$ entry of DT;
    }
  if (No. of Partially_Equivalent =
    No. of inputs) in $[S_i,S_j]$ entry of DT **then** states S_i and S_j are Equivalent;
  if $[S_i,S_j]$ entry of DT is empty **then**
    mark an "X" in $[S_i,S_j]$ entry of DT; /**Totally_Distinguishable**/
    }
}

In constructing the distinguish table, the procedure also calculates undistinguishability to reflect the distinguishability. For an FSM F with N states and each state with L outgoing edges, we define the undistinguishability of a state pair and a state as follows:

For a state pair $[S_i,S_j]$, its undistinguishability is UnDisty($[S_i,S_j]$) = Number of next state pair in $[S_i,S_j]$ entry + number of $[S_i,S_j]$ in distinguish table + K, where $K = pN$, if S_i and S_j have p partially equivalent transitions, where $1 \le p \le L$; $K = 0$,otherwise.

For a state S_i, its undistinguishability is

$$\text{UnDisty}(S_j) = \frac{1}{(N-1)} \sum_{j=1, j \ne i}^{N} \text{UnDisty}([S_i, S_j])$$

With the above definitions, we construct an undistinguishability table of state pairs for the example FSM as shown in Figure 5b. For example, UnDisty($[A,D]$) = 2+1+1 = 4, because there are two next state pairs in the $[A,D]$ entry, and $[A,D]$ appears once as the next state pair in the $[B,D]$ and

State parity assignment procedure

```
{

/* Input : Order list of state-pair, <SP-list> *

/* Output: Parity list of states, <Even-list>, <Odd-list> */
For each state-pair SP in <SP-list> do
   {
   if both states of SP are assigned then next state-pair;
   if only one state of SP is assigned then
      {
      assign another state of SP to opposite parity list;
      next state-pair;
      }
   choose the larger undistinguishability state S from SP;
   calculate E-cost = Σ UnDisty ([S,<Odd-list>]) and
          O-cost = Σ Undisty ([S,<Even-list>]);
   if E-cost is larger than O-cost then
      assign S to <Even-list> and the other state to <Odd-list>;
   else
      assign S to <Odd-list> and the other state to <Even-list>;
   next state-pair;
   }
}
```

[C,D] entries. Since there is one * in the [B,C] entry, UnDisty([B,C]) = 6+1+1+1 = 9. Figure 5b also shows the undistinguishability of every state. For state C, UnDisty(C) = (2+9+3+1+2)/5 = 3.4. Here, we can define the undistinguishability of an FSM as

$$\text{UnDisty}(F) = \frac{1}{N}\sum_{j=1}^{N}\text{UnDisty}\left(S_j\right)$$

The undistinguishability of an FSM can serve as a testability measure for the machine. The larger the undistinguishability for a machine, the harder the test generation will be. For the example FSM shown in Figures 4 and 5, this value is UnDisty(M2) = (2.4+2.8+3.4+2.6 +2.6+3)/6 = 2.8.

With the undistinguishability value computed for each state pair, we derive an order list of state pairs for the state parity assignment. Figure 5c shows such a list derived from Figure 5b. In the list, the state pair [B,C] should get assigned to a different parity first because it has the highest undistinguishability value. Pair [E,F] should come next, and so forth.

For parity assignment, as an even or odd parity is assigned to one state of a state pair, the scheme assigns the opposite parity to the other state of the pair. With this assignment made, the states of this state pair—originally having high undistinguishability—become totally distinguishable. Those state pairs that are not totally distinguishable will also become much less k-distinguishable.

Generally, we can easily generate the test sequence of those k-distinguishable state pairs. The State parity assignment box describes the procedure used to assign state parity for an FSM.

Figure 6 shows the new distinguish table after completion of the parity assignment for the example. In the figure,

B	D,E				
C	D,E	X			
D	C,F A,E	A,D	A,D		
E	B,C	X	X	B,F	
F	B,C C,E	C,D	C,D	B,F A,C	X
	A	B	C	D	E

Even: A,C,F
Odd: B,D,E

Figure 6. *The modified distinguish table of example M2. All state pairs are totally distinguishable except [A,C], and this machine becomes a 1-distinguishable machines, with an undistinguishability that approaches 0.*

states A, C, and F are assigned even parity codes; states B, D, and E are assigned odd parity codes. Since A and B have different parity, the machine can differentiate them at the next clock. An X marked on the [A,B] entry indicates that it is a totally distinguishable state pair. Also, the state pairs [A,D], [A,E], [B,C], ... are totally distinguishable.

The state pair [A,F] has two next state pairs, [B,C] and [C,E], that are both totally distinguishable. State pair [A,F] then is totally distinguishable after one clock. Hence, in the figure, all state pairs are totally distinguishable except [A,C]. This machine becomes a 1-distinguishable machine, with an undistinguishability approaching 0. The machine can record state pair [A,C] for the later test generation.

Test generation

Adding the parity checker and adopting the state parity assignment greatly simplifies test generation of FSMs. We have developed two methods for generating test sequences with 100% fault coverage for different fault models. A functional test generation method applied before synthesis of FSMs works for the single-state transition fault model. A deterministic test generation method applied after the circuit implementation works for the single stuck-at fault model.

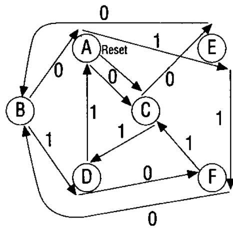


Figure 7. Functional test generation of example M2.

Table 1. Two state assignments for M2.

M2	a	Parity	b
A	000	Even	000
B	010	Odd	010
C	110	Even	001
D	100	Odd	100
E	001	Odd	011
F	101	Even	101

Functional test generation. As we discussed earlier, after state parities are assigned, the machine becomes a minimal κ-distinguishable device. For a κ-distinguishable FSM to detect all SST faults, the test generation problem basically is a postman traveling problem[17] with a constraint that the postman must consecutively pass some specified streets. Starting from the post-office (reset state), the postman, however, should traverse all streets at least once, while passing the minimal number of streets. (In the postman traveling problem, a postman picks up mail at the post office and delivers it to each block in the territory. The postman wishes to choose a route that minimizes the distance traveled. The vertices and edges of a graph G modeling this situation correspond to the street corners and connecting blocks of the postman's territory. The solution, then, to the postman traveling problem is a closed walk of minimum length G that uses every edge at least once.)

Table 2. The fault coverage percentages of the two implementations shown in Table 1.

Test	M2a (%)	M2b (%)
0	28	7
0	44	29
0	57	42
0	63	49
1	70	49
1	76	60
0	80	68
1	85	80
0	90	85
1	92	85
1	96	85
1	98	87
0	99	91

For the state transition graph, directed edges (state transitions) are like one-way streets, and the vertices (states) are like crossing streets, with the constraint that some edges must be consecutively traversed at least once. For a totally distinguishable machine, the constraint disappears. In this special case, the functional test generation resembles the checking experiments Fujiwara and Kinoshita present.[2] If the state transition graph is an eulerian, the problem simplifies to a one-stroke one, and the test length becomes the minimum test length—the number of transitions plus one. (An eularian graph is one that lets us walk all edges once and only once; that is, a eulerian graph can be drawn in one stroke, with no edge repeated.)

The information obtained in the state assignment procedure can assist during functional test generation. For the distinguish table of example M2 shown in Figure 6, the machine is now a 1-distinguishable machine after state parity assignment. One remaining 1-distinguishable state pair, $[A,C]$, has an undistinguishable next state pair $[D,E]$ under input 1. So the input 1 state transitions of states A and C is a constraint during test generation.

Since $[D,E]$ is totally distinguishable after one additional clock, making any additional transition after the constrained transitions will remove the constraints. Since the state transition graph is an eulerian, one stroke will generate the test sequence, as Figure 7 shows. Here, the reset state is A. After traversing the states, the derived test sequence is 0000110101110 and the final state is C.

For comparison, we performed two implementations of different state assignments with the M2 machine we have used as an example. Table 1 presents the results. For the M2a implementation, we adopted the state-parity assignment, and for the M2b implementation, we assigned the states without considering parity. We then synthesized these two assignments into logic circuits with the aid of MisII.[12] We then applied the test sequence generated in Figure 7 to these two logic circuits to detect stuck-at faults.

From theorem 2, for the M2a implementation, the SST fault coverage is guaranteed to be 100%. Table 2 shows the fault coverage of stuck-at fault versus the test sequence applied for these two implementations. Clearly, the fault coverage rises much faster, reaching 99% for the M2a implementation than with M2b. Fault coverage reaches only 91% under the applied test sequence for the M2b implementation.

Deterministic test generation. In general, FSMs are not completely specified for their state transition tables. Unspecified states, state transitions, and "don't care" terms remain for the tables. During functional test generation, calculation of the undistinguishability does not reflect these unspecified terms. Also, for the traversing sequences containing "don't care" terms, we randomly assign logic 0 or 1 to form the input sequence. These reduce the effectiveness of the derived functional

test sequences for detecting the stuck-at faults of the implemented circuits. For example, in the M2a implementation just discussed, the derived functional test sequence reach only a 99% stuck-at fault coverage.

To reach 100% stuck-at fault coverage, we use a deterministic test generation method to generate test for a circuit synthesized employing the parity checker DFT scheme as follows:

1. Find a sequence that traverses all states at least once as an initial test sequence.
2. Run sequential fault simulation for the test sequence; drop the detected faults and stop if no fault remains; and record the last reached state as the current state.
3. Select a target fault.
4. Run the combinational test generation procedure for the target fault. The fault effect may appear in primary outputs, or appear in odd number of flip-flops. Record the state that detects the fault as the target state and the input that detects the fault as the activation pattern.
5. Find a sequence that justifies the current state to the target state as the new test sequence, add the activation pattern into the test sequence, and go to step 2.

Experiments

We built an automatic synthesis system on a Sun workstation to synthesize and generate functional test sequences for FSMs by incorporating our DFT scheme. Upon receiving the transition table of an FSM, the system builds the distinguish table, calculates the undistinguishability, and suggests a parity state group partition, as well as deriving a set of functional test sequences. With the suggested state group partition, it uses MUSTANG,[13] a state-assignment tool developed at UC Berkeley, to make the state assignments and MisII to perform logic optimization and circuit realization for the combinational logic system then automatically adds the flip-flops and parity checker to make a sequential circuit. Finally, a single state transition fault simulator SSTFS and a stuck-at fault simulator SEESIM[14] evaluate the fault coverage.

Table 3. Results of functional test generation PCDFT-FTG.

FSM name	PCDFT-FTG CPU speed (sec.)	PCDFT-FTG Test length (patterns)	SST fault coverage With PCDFT (%)	SST fault coverage Without PCDFT (%)	SA fault coverage With PCDFT (%)	SA fault coverage Without PCDFT (%)
bbsra	0.02	92	100	90.7	100	77.2
bbsse	0.03	150	100	83.6	98.2	87.7
bbtas	0.02	36	100	94.2	100	72.6
beecount	0.02	53	100	82.4	100	83.5
cse	0.05	212	100	75.4	99.6	96.5
dk14	0.02	83	100	91.7	100	97.6
dk15	0.02	43	100	85.4	100	97.2
dk16	0.13	195	100	98.7	100	99.5
dk17	0.02	65	100	91.5	100	96.4
dk27	0.02	22	100	91.7	100	86.3
dk512	0.03	56	100	96.0	100	93.1
ex3	0.02	93	100	55.6	99.3	67.3
ex7	0.02	101	100	38.3	98.6	53.9
lion	0.02	14	100	87.9	100	94.2
lion9	0.02	33	100	80.1	99.6	91.2
opus	0.02	82	100	92.8	100	97.8
sand	0.20	334	100	99.3	99.8	94.5
sse	0.03	169	100	84.2	98.7	85.9
styr	0.18	389	100	89.4	99.8	91.6
train4	0.02	14	100	81.0	100	95.4
train11	0.02	42	100	80.8	100	98.2
Average	**0.04**	**108**	**100**	**84.3**	*	**88.5**

***99.69524**

As Tables 3 and 4 show, the system has synthesized a number of MCNC (Microelectronic Center at North Carolina) benchmark FSMs.[15] In Table 3, the second and third columns are the test generation times and the test lengths of the functional test sequences generated during synthesis. The fourth column is the SST fault coverage for the FSMs designed with the parity checker DFT scheme (PCDFT). All the SST fault coverage are 100% for these FSMs. The fifth column is the SST fault coverage for the FSMs with a parity checker added to FSMs to help detect the transition fault, but without applying the state parity assignment. For this case, the SST fault coverage averages 15% lower. The last two columns are the single stuck-at (SA) fault coverage of each implemented logic circuit with and without employing the PCDFT scheme.

Table 3 shows that the stuck-at fault coverages under the derived test sequence for the circuits employing the PCDFT scheme are generally over 99%, while the average SA fault coverage for the circuits without employing the

Table 4. Comparison of FTG and our PCDFT-FTG.

FSM name	FTG CPU* speed (sec.)	FTG Test length (patterns)	FTG Fault coverage (%)	PCDFT-FTG CPU* speed (sec.)	PCDFT-FTG Test length (patterns)	PCDFT-FTG Fault coverage (%)
dk14	1.0	228	100.0	0.02	83	100.0
dk15	0.2	146	100.0	0.02	43	100.0
dk16	10.0	406	100.0	0.13	195	100.0
dk17	0.4	86	100.0	0.02	65	100.0
dk512	0.5	89	100.0	0.03	56	100.0
ex7	2.0	158	99.3	0.02	101	98.6
opus	0.2	96	98.5	0.02	82	100.0
styr	204.0	964	97.2	0.18	389	99.8
bbara	2.0	241	100.0	0.02	92	100.0
cse	45.0	880	97.9	0.05	212	99.6
sand	202.0	809	97.7	0.20	334	99.8
Average	**42.5**	**373**	**99.1**	**0.06**	**150**	**99.8**
Normalized	**235**			**1.00**		

**CPU seconds: FTG on Sun 4/260 (10 MIPS), PCDF-FTG on Sun Sparc 2 (28 MIPS).*

Table 5. Results of our deterministic test generation PCDFT-Det TG.

FSM name	PCDFT-DetTG CPU speed (sec.)	PCDFT-DetTG Test length (patterns)	SA fault coverage With PCDFT (%)	SA fault coverage Without PCDFT (%)
bbara	1.1	51	100	81.2
bbsse	2.3	104	100	96.4
bbtas	0.4	16	100	58.3
beecount	0.3	42	100	93.8
cse	5.9	158	100	84.1
dk14	2.3	61	100	100
dk15	1.0	29	100	99.2
dk16	9.1	172	100	93.4
dk17	0.8	45	100	96.8
dk27	0.2	19	100	91.8
dk512	1.0	58	100	94.7
ex3	1.1	36	100	90.3
ex7	1.5	44	100	88.1
lion	0.1	11	100	88.9
lion9	0.1	21	100	92.2
opus	1.1	54	100	94.4
sand	24.5	214	100	80.0
sse	2.4	90	100	97.8
styr	23.4	287	100	81.6
train4	0.1	14	100	90.2
train11	0.7	38	100	89.3
Average	**3.8**	**74**	**100**	**89.6**

Table 6. Comparison of STG3 and PCDFT-DetTG.

FSM name	STG3 CPU* speed (sec.)	STG3 Test length (patterns)	STG3 Fault coverage (%)	PCDFT-DetTG CPU* speed (sec.)	PCDFT-DetTG Test length (patterns)	PCDFT-DetTG Fault coverage (%)
dk14	12	90	100.0	2.3	61	100
dk15	4	44	100.0	1.0	29	100
dk16	7,176	322	97.6	9.1	172	100
dk17	9	66	100.0	0.8	45	100
dk512	92	85	100.0	1.0	58	100
ex7	314	58	99.2	1.5	44	100
opus	25	80	100.0	1.1	54	100
styr	20,613	515	94.3	23.4	287	100
bbara	36	133	100.0	1.1	51	100
cse	9,657	344	97.8	5.9	158	100
sand	9,106	376	97.6	24.5	214	100
Average	**4,277**	**192**	**98.8**	**6.5**	**107**	**100**
Normalized	**218**			**1.0**		

**CPU seconds: FTG on Sun 4/260 (10 MIPS), PCDF-FTG on Sun Sparc 2 (28 MIPS).*

PCDFT scheme is only 88.5%. For the PCDFT circuits whose SA fault coverage are not 100%, the undetected faults mainly come from the unspecified terms of the FSMs. All these non-fully-testable circuits have very low connectivity, indicating that functional test sequences will not effectively detect SA faults when the machines have many unspecified terms. The test generation times are almost negligible for most circuits.

Table 4 compares the test generation times of Table 3 with those found in Cheng and Jou.[4] Here, the test generation times for our PCDFT scheme circuits represent the CPU times of a Sun Sparc 2 workstation, a 28 MIPS machine, while those of the functional test generation (FTG) circuits[4] represent CPU times of a Sun 4/260 workstation, a 10 MIPS machine. After normalizing CPU times, the test generation times for PCDFT circuits are 235 times shorter than those of FTG circuits. The table also

lists the test length and fault coverage of each sequence. Our PCDFT scheme circuits have higher fault coverages, nearly 100% on average, and shorter test lengths than those of FTG circuits.

Table 5 shows the results for the test sequences generated by the deterministic test generation method for the implemented PCDFT circuits. In the table, the second and the third columns are the test generation times and lengths of the test sequences generated by the deterministic test generator (PCDFT-DetTG). The fourth column is the stuck-at fault coverages for the test sequences for the PCDFT circuits, which are all 100%. The fifth column shows the fault coverages for the nonPCDFT circuits if they are applied to the same test sequences. The average fault coverage is only 90% for this case. Also, note that for the deterministic test sequences, their test lengths are shorter than those of the functional test sequences of Table 3.

To show the efficiency of this test generation, Table 6 compares our results with those of STG3.[4] Our test lengths are shorter than those of STG3; we achieve 100% fault coverage for all circuits with much less test generation time. We gain all this effectiveness from the small penalty of the parity checker.

THE EXTRA OVERHEAD on logic gates added in this scheme is very small, requiring only $\log_2$(number of states)−1 numbers of XOR gates. This figure is less than the general extra overhead in the conventional scan designs. Table 7 compares the scan designs, the parity-scan design,[9] and our parity checker DFT.

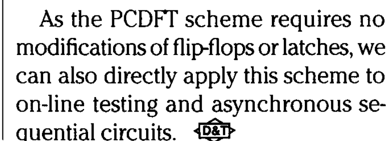
Table 7. Comparison of our DFT scheme (PCDFT) with other approaches.

Approach	Hardware overhead	Extra I/O pins	Performance degradation	Test generation	Test length	CED capability
PCDFT	Low	1	No	Simple	Short	Yes
Parity-scan[13]	High	≥3	Yes	Easy	Long	Yes
Full scan	High	≥3	Yes	Easy	Long	No
Partial scan	Low	≥3	Yes	Harder	Short	No

PCDFT needs no modification of the flip-flops, requiring only an XOR tree for the parity checker and an extra output pin. The chip area and extra I/O pins are lower than those of scan designs having at least three extra I/O pins. Test generation for PCDFT is as simple as combinational test generation, and the generated test sequence is more effective than other approaches. Also, like the partial scan design, the PCDFT scheme can apply to partial flip-flops, which have low controllability/observability. This adaptability further saves on hardware overhead while maintaining the advantage that no circuit performance degradation is introduced as PCDFT needs no modification of the flip-flops. Also, with the full PCDFT scheme, we can consider the added parity checker as a concurrent error detection scheme for fault-tolerant designs.

Our next step is to extend this PCDFT scheme to general (non-FSM) sequential circuits. For such circuits, we will incoporate the PCDFT scheme into sequential test generation to alleviate the test generation effort. We will adapt a partial PCDFT scheme to reduce the size of the added parity checker. Our goal is to achieve a fully and easily testable design with a minimal length of test sequences by adding less hardware overhead.

As the PCDFT scheme requires no modifications of flip-flops or latches, we can also directly apply this scheme to on-line testing and asynchronous sequential circuits.



Acknowledgments

The authors wish to express their gratitude for helpful comments from the reviewers. The National Science Council, Taiwan, Republic of China, supported this work under contracts NSC-81-0404-E009-136 and NSC-82-0404-E009-183.

References

1. F.C. Hennie, "Fault-Detecting Experiments for Sequential Circuits," *Proc. Fifth Ann. Symp. Switching Circuit Theory and Logical Design*, Princeton, N.J., 1964, pp. 95-110.
2. H. Fujiwara and K. Kinoshita, "Design of Diagnosable Sequential Machines Utilizing Extra Outputs," *IEEE Trans. Computers*, Vol. 23-2, 1974, pp. 138-145.
3. M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, Oxford, England, 1990.
4. K.T. Cheng and J.Y. Jou, "Functional Test Generation for Finite State Machines," *Proc. Int'l Test Conference*, CS Press, 1990, pp. 162-168.
5. S. Devadas and K. Keutzer, "A Unified Approach to the Synthesis of Fully Testable Sequential Machines," *IEEE Trans. Computer-Aided Design*, Vol. 10, 1991, pp.3 9-50.
6. V.D. Agrawal and K.T. Cheng, "Test Function Specification in Synthesis," *Proc. 27th Design Automation Conf.*, CS Press, 1990, pp. 235-240.
7. K.T. Cheng and V.D. Agrawal, "State Assignment for Testable Design," *Int'l J. Computer-Aided VLSI Design*, Vol. 3, March 1991, pp. 291-308.
8. R. Leveugle and G. Saucier, "Optimized

JOIN US AT ICCAD

for an interesting panel discussion:

CAD Profession: Facts, Trends, and Illusions

A 10X growth in semiconductor capacity every six years has forced designers to rely on CAD tools to deliver 10X productivity improvements. However, with the pace in semiconductor processing showing no sign of slowing down, many believe current CAD tools no longer support designers' needs.

Moderator Daniel Gajski (UC Irvine) will lead panelists Ron Collett (Collett Int'l), James Duley (Hewlett-Packard), Carlos Dangelo (LSI Logic), and Richard Newton (UC Berkeley) as they consider this challenge and discuss:

- ✓ The underlying semiconductor revolution
- ✓ Its impact on the marketplace and CAD in particular
- ✓ The EDA industry—Can it deliver solutions to these problems and still stay profitable?
- ✓ The role venture capital and start-ups should play
- ✓ The role of academic research in meeting these challenges

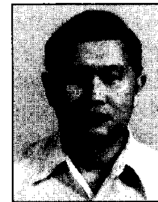
Plan to attend the panel discussion on Wednesday, November 9. The IEEE/ACM International Conference on Computer-Aided Design meets November 7-10, 1994, in San Jose, California.

IEEE Design&Test of Computers

Synthesis of Concurrently Checked Controllers," *IEEE Trans. Computers*, Vol. 39, 1990, pp. 419-425.

9. H. Fujiwara and A. Yamamoto, "Parity-Scan Design to Reduce the Cost of Test Application," *Proc. Int'l Test Conf.*, CS Press, 1992, pp. 283-292,.
10. M.L. Sheu and C.L. Lee, "A Parity Checker Design for Testability Scheme for Finite State Machines," *Proc. Asia-Pacific Conf. Circuits and Systems*, IEEE Circuits and Systems Society, Piscataway, N.J., 1992, pp. 53-58.
11. S. Devadas, H.K. Ma, and A.R. Newton, "Redundancies and Don't Cares in Sequential Logic Synthesis," *J. Electronic Testing: Theory and Applications*, Jan. 1990, pp.15-30.
12. R. Brayton et al., "MIS: Multiple-Level Interactive Logic Optimization Systems," *IEEE Trans. Computer-Aided Design*, Vol. 6, 1987, pp. 1062-1081.
13. S. Devadas et al., "MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations," *IEEE Trans. Computer-Aided Design*, Vol.7, 1988, pp. 1290-1300.
14. C.P. Wu, C.L. Lee, and W.Z. Shen, "SEESIM—A Fast Synchronous Sequential Circuit Fault Simulator with Single Event Equivalence," *Proc. European Design Automation Conf.*, CS Press, 1992, pp. 446-449.
15. B. Lisanke, "Logic Synthesis and Optimization Benchmarks," tech. report, MCNC, Research Triangle Park, N.C., Dec. 1988.

Meng-Lieh Sheu is currently working toward his PhD in the Department of Electronics Engineering at the National Chiao Tung University, where he earlier received BS and MS degrees in electronics engineering. His research interests include VLSI testing, logic and high-level synthesis, and computer-aided design. He is a student member of the IEEE Computer Society, Circuits and Systems Society, and Communications Society.



Chung Len Lee currently is a professor in the Department of Electronics Engineering, National Chiao Tung University, where his teaching and research interests focus on integrated circuits and testing. He has supervised more than 90 MS and PhD candidates, and has published more than 140 papers in technical journals. He received his BS from the National Taiwan University and MS and PhD degrees from Carnegie Mellon University. Presently, he serves on the editorial board of the *Journal of Electronic Testing: Theory and Applications*, and is a member of the IEEE Asian Test Technology Committee. He is a senior member of the IEEE Circuits and Systems Society and the IEEE Computer Society.

Direct questions concerning this article to Cheng Len Lee, Department of Electronics Engineering, National Chiao Tung University, 30050, Hsin-Chu, Taiwan; cllee@ cc.nctu.edu.tw.