MIX : A Test Generation System for Synchronous Sequential Circuits

Xijiang Lin Irith Pomeranz Sudhakar M. Reddy
Electrical and Computer Engineering Department
University of Iowa
Iowa City, IA 52242

Abstract

We describe a test generation system for synchronous sequential circuits described at the gate level. The test generation system, called MIX, combines several test generation approaches to derive test sequences exhibiting very high fault coverages at relatively low CPU times. It is known that different faults in a synchronous sequential circuit may be more amenable to different test generation approaches. The strength of MIX stems from the fact that a large number of different approaches is used to attack faults with different characteristics. Several new techniques are incorporated into MIX, including a new definition of an XD-frontier, storing a partial state transition graph to help in the derivation of justification sequences, utilization of sequences generated for aborted faults, consideration of multiple time frames simultaneously during state justification, and dynamic computation of dependencies among flip-flops. A simplified form of test generation under the restricted multiple observation times test strategy is also employed, based on state expansion. Restricted multiple observation times fault simulation is used in MIX to identify detected faults beyond those detected by conventional fault simulation.

1 Introduction

A large number of test generation procedures for synchronous sequential circuits described at the gate level have been developed in recent years. Deterministic branch-and-bound test generation procedures such as the ones reported in [1][2][3][4][5][6][7][8] are very successful in achieving high fault coverages; however, they have a high computational complexity. This complexity led to the development of several alternative approaches. The procedure of [9] uses a direct search to generate test sequences. The procedures of [10][11][12][13][14][15][16][17][18][19] are based on genetic optimization. In [11][13][14][17], genetic optimization is combined with deterministic test generation to improve the fault coverage that can be achieved in practical time. The test generation procedure LOC-STEP of [20] avoids fault oriented test generation and consideration of faulty circuits by generating input sequences that imitate properties of test sequences generated by deterministic test generation procedures.

Various techniques have been developed to improve the test generation process in every one of the classes above. These techniques include, for example, the use of static and dynamic unique sensitization[4], the use of previous good state knowledge and a targeted D frontier[7], the use of forward time processing only[8], the use of propagation sequences for faulty values stored in flip-flops [16][21], and the use of synchronizing sequences [19].

The test generation procedure MIX described in this work is unique in that it combines many of the approaches above with several new techniques for improving the effectiveness of test generation. The approaches are applied from the least computational intensive to the most computational intensive. Thus, each fault is exposed to a large number of fundamentally different test generation approaches, and will be detected by the least expensive one that can detect it. Test generation procedures that mix various test generation techniques can be found in [22]. However, unlike MIX that uses different test generation approaches, the procedure of [22] uses only deterministic test generation, and switches among various techniques within this class of procedures.

Test generation in MIX is done assuming that the circuit, when powered-up, is in an unknown state. Synchronizing sequences are used to specify the state variable values and reduce the loss of fault coverage due to unspecified values in the fault-free circuit. Symbolic techniques have been used in [23][24][25][26][27] to alleviate the loss of accuracy due to three value simulation and offer a more complete test generation process. These procedures are typically based on binary decision diagrams (BDDs) and are not applicable to circuits for which BDD representations cannot be derived in practical time. For full accuracy, one must use the multiple observation times test strategy [28]. Test generation under the multiple observation times test strategy was described in [24][28][29]. In this work, we use the restricted multiple observation times approach [28] based on a single response of the fault free circuit to identify faults that cannot otherwise be detected. The advantage of the restricted multiple observation times approach is that it does not impose any requirements on the test application process beyond those of the conventional approach (the full multiple observation times approach may require multiple fault free responses to be stored).

The paper is organized as follows. In Section 2,

1063-9667/97 $10.00 © 1997 IEEE

we give an overview of the MIX procedure and describe some of the techniques it uses. In Section 3, we describe each one of the main procedures of MIX. Experimental results are given in Section 4. Concluding remarks are given in Section 5.

2 Overview of MIX

The test generation procedure MIX applies several increasingly time consuming procedures to generate test sequences. These techniques are listed in this section. Figure 1 shows the flowchat of MIX.

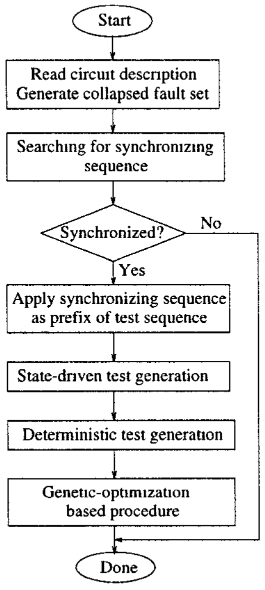

Start
Read circuit description
Generate collapsed fault set
Searching for synchronizing sequence
Synchronized?
No
Yes
Apply synchronizing sequence as prefix of test sequence
State-driven test generation
Deterministic test generation
Genetic-optimization based procedure
Done

Figure 1: MIX ATPG system

MIX first calls a procedure to find a synchronizing sequence for the fault free circuit. The goal of this step is to ensure that, once the inputs are specified, the output values obtained for the fault free circuit are fully specified, and no loss of fault coverage results from unspecified values in the fault free circuit.

The second procedure included in MIX is referred to as *state-driven* test generation. It is based on the observation from [20] and [30] that an input sequence causing the fault free circuit to traverse a large number of different states is effective for fault detection. The unique feature of the state-driven procedure in MIX is that at every time unit, it performs test generation for faults that are likely to be detected in that time unit in order to select the input pattern. It thus maximizes the number of faults detected by every pattern.

The third procedure in MIX consists of deterministic test generation. This procedure is preceded by a procedure that determines values that cannot be obtained on single flip-flops and on pairs of flip-flops. These values are used to avoid unreachable states during the activation, propagation and state justification phases of the test generation procedure.

The fourth procedure included in MIX is a genetic-optimization based procedure.

Each phase of the procedure is followed by conventional fault simulation and by fault simulation under the restricted multiple observation times approach to identify detected faults. Two fault simulation procedures are used for the restricted multiple observation times approach. The first procedure is the one from [31] which is based on the identification of conflicting output values between the fault free and the faulty circuits. The second procedure is based on state expansion in the faulty circuit, similar to the procedure from [32]. The use of two different procedures helps to ensure that the maximal number of detected faults can be identified.

Several new techniques are used throughout the MIX procedure to enhance its effectiveness. These are described next.

The concept of *XD-frontier* is introduced in MIX for two purposes. The first is to identify lines carrying unknown values due to conventional three value implications, when more accurate implications can show that the line can only carry one value, 0 or 1. The second purpose is to help in the determination of the number of time frames that need to be considered before a fault can be activated.

To define the XD-frontier, we first need the following definitions. A pair of values for the good (i.e., fault free) and faulty circuits is denoted by V_g/V_f, where V_g and V_f are 0, 1 or X.

Definition 1: An X/X path is a path where an X value is assigned to each line in it, in both the good and the faulty circuits.

Definition 2: A line belongs to the XD-frontier if there is an X/0 or X/1 value pair on the line and there exists at least one X/X path from the line to any primary output or pseudo-output of the circuit.

An XD-frontier can exist in time frames where the primary inputs and/or pseudo-inputs are unspecified. Since XD-frontiers potentially carry fault effects, propagating the XD-frontier is likely to propagate D values.

During fault propagation, the XD-frontier is used as follows. As long as the D-frontier is not empty, it has higher priority for propagation than the XD-frontier, and fault effects are propagated using the D-frontier. When the D-frontier becomes empty, propagation of values on the XD-frontier is performed. Propagation of XD values (X/0 and X/1) is similar to propagation of D values, except that values need to be assigned only in the faulty circuit. For example, to propagate an X/0 value from an input of an AND gate to its output, it is necessary to set all the other gate inputs to 1 in the faulty circuit.

The X/0 and X/1 values are included in the nine valued logic proposed earlier in [37]. However, no attempt to propagate these signal values in order to potentially propagate fault effects was made in earlier works. Additionally, their use here is in the context of the restricted multiple observation time test strategy.

Once an XD value reaches a primary output (i.e., a value X/1 or X/0 appears on a primary output), the output value in the fault free circuit is set to the opposite of the faulty value. Implications are then performed. The following results are possible. (1) If a conflict arises, the output value is reversed and kept at its new specified value. The implications of the new assignment are also determined. The result is that additional values are specified and may be used to detect faults more easily. (2) If the output value does not cause any conflicts, an attempt is made to justify the assignment. If justification succeeds, the fault has been propagated to a primary output. If the justification procedure verifies that the assignment is impossible, the assigned value is reversed and the implications of the reversed assingment are performed. The same procedure can be used if 1/X or 0/X is observed at a primary output except that the circuit under analysis is the faulty circuit.

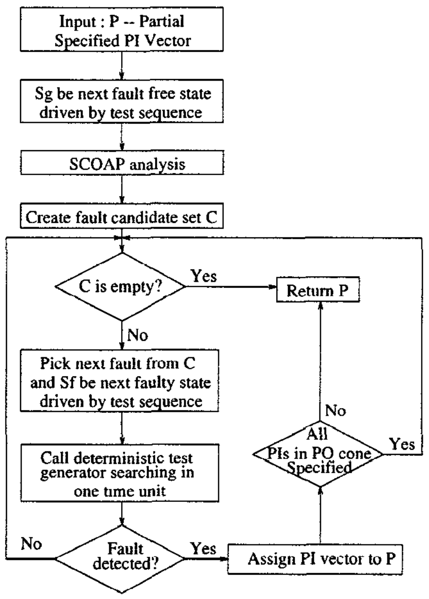
The storage of a partial state transition graph (PSTG) is another technique that contributes to the effectiveness of MIX. The use of a PSTG for test generation was also considered in [33]. The partial state transition graph is developed in MIX during the test generation process. Every time a test sequence is generated, the state transitions traversed by the fault free circuit are added to the partial state transition graph. Since the fault free circuit is synchronized in the first phase of MIX, test sequences typically start from a fully specified state, and all the states reached during the test sequence can be used to update the state transition graph. The deterministic test generation phase uses the partial state transition graph to derive justification test sequences. If state justification needs to take the fault free circuit from a state S_0 to a state S_1, and if both S_0 and S_1 are in the state transition graph, then the shortest path from S_0 to S_1 in the state transition graph is computed. If a path is found, it is used to define the justification sequence. In [7] and [18], specific justification sequences are stored for future use. Storing a partial state transition graph offers more flexibility, as it may represent a large number of transfer sequences using a small number of edges.

Typically, when the effort in deterministic test generation for a target fault f reaches a limit, the fault is aborted and the effort expended in test generation is lost. In MIX, the input sequence generated by deterministic test generation before a fault is aborted is not discarded. Input sequences generated for aborted faults are used for three purposes, described next.

The sequences generated for aborted faults are used to seed the genetic optimization process. Since these sequences typically activate and at least partially propagate and justify fault effects, they are likely to be useful for the detection of the same faults and other faults with similar activation, propagation and justification requirements.

The second use for these sequences is in the generation of the partial state transition graph of the fault free circuit. This is done similar to the way successful test sequences are used. However, if any of the state variables at the beginning of the sequence are unspecified, they are randomly filled before the state transitions along the sequence are added to the partial state transition graph. The initial state may be unreachable from the synchronization state, but it is possible that it will be reached later as the partial state transition graph is updated dynamically.

The third use of input sequences generated for aborted faults is as starting points for test generation for additional faults. A special procedure is used to fill in the unspecified input values in these sequences so as to detect additional target faults. This is similar to the heuristics used for combinational test generation in [35]. The procedure for setting unspecified input values, described next, is used during several other phases of the test generation procedure.



Input : P -- Partial Specified PI Vector
Sg be next fault free state driven by test sequence
SCOAP analysis
Create fault candidate set C
C is empty?
Yes
Return P
No
Pick next fault from C and Sf be next faulty state driven by test sequence
Call deterministic test generator searching in one time unit
No
All PIs in PO cone Specified
Yes
No
Fault detected?
Yes
Assign PI vector to P

Figure 2: *fill_PI_values*(P)

Consider a partially specified input pattern P of a test sequence T'. Let P be applied at time unit u when the fault free circuit is in state S_g. The SCOAP testability measures are computed for time unit u, and a set C of candidate faults that may potentially be detected at time unit u is derived. The faults in C are such that their lines have high controllability and observability values, and are thus likely to be detectable at time unit u. For each candidate fault f in C, the following procedure is then applied. First, the state S_f of the faulty circuit in the presence of f is derived. Deterministic test generation is then carried out at time unit u to propagate f to the primary outputs. If the fault is detected, the input values required to detect the fault are added to P. Additional faults in C are considered in the same way until either all the primary outputs of the fault free circuit are fully specified, or all the faults in C have been tried. The procedure is referred to as *fill_PI_values*() and it is shown in Figure 2. In the case where an input sequence generated for an aborted fault detects new faults after calling *fill_PI_values*(), the useful part of the input sequence is added to the test sequence. Otherwise, this sequence is used as a seed for the genetic-optimization based procedure.

A deterministic test generation procedure using a single time frame is called in *fill_PI_values*() uses the XD-frontier as well as the D-frontier during fault propagation. The XD-frontier helps in determining additional specified values. If a fault f is detected at time unit u by specifying the value of a state variable y, the value of y is complemented and deterministic test generation continues using the complemented value of y. If the fault is detected for both values of y, then under the restricted multiple observation times approach, the fault can be declared as detected.

3 The Procedures of MIX

In this section, we describe the main procedures of MIX in more detail.

3.1 State-Driven Test Generation

We start with a description of previously proposed state-driven procedures. We then describe the new features of the state-driven procedure of MIX. The test generation procedure LOCSTEP [20] generates a sequence T of a fixed length L by selecting the input patterns of T one at a time, starting from the input pattern at time unit 0. At every time unit, it generates N_p random patterns and simulates them on the fault free circuit. It then selects the one that best matches a set of properties related to the sequence of states traversed by the fault free circuit. One of the properties requires that a new state (a state that was not reached before) would be traversed by the fault free circuit whenever possible. Other properties are related to the selection of the next state when a new state cannot be reached, and a state traversed before must be repeated.

The state-driven procedure of MIX uses test generation for target faults to guide the selection of the input pattern at every time unit. Two procedures are used for determining the input pattern at every time unit. First, procedure *fill_PI_values*() described above is called to detect as many faults as possible. Then, a second procedure is called to drive the fault free circuit into a new state. This procedure uses a branch-and-bound approach to determine the unspecified input values. The procedure terminates when a new state is reached, no additional backtracking can be made (indicating that no new state can be reached), or when a backtrack limit is reached.

When a primary input vector leading to a new state cannot be found, the following heuristic is used to select a next state that has been visited before. During the state-driven procedure, the partial state transition graph of the circuit is updated every time a new state transition is explored. For every state in the state transition graph, there is a counter called *n_times_visited*. This counter stores the number of times the state has been reached during the state-driven phase. When a new state cannot be found, the primary input vector leading to a state with the lowest value of *n_times_visited* is selected.

If the generated test vector fails to detect new faults for several time units, the undetected faults are sampled and deterministic test generation is used to find a vector which propagates the sampled faults into the pseudo-outputs of the current time frame. This increases the likelihood of detecting faults in the next time unit. The state-driven procedure is shown in Figure 3. In the figure, *curr_window_size* is the length of the test sequence since the last detection, and the detection ratio is defined as the number of new faults detected over this sequence divided by its length *curr_window_size*. The state-driven procedure is terminated when a preselected number of consecutive test patterns, *max_window_size*, fails to detect a preselected number of additional faults.

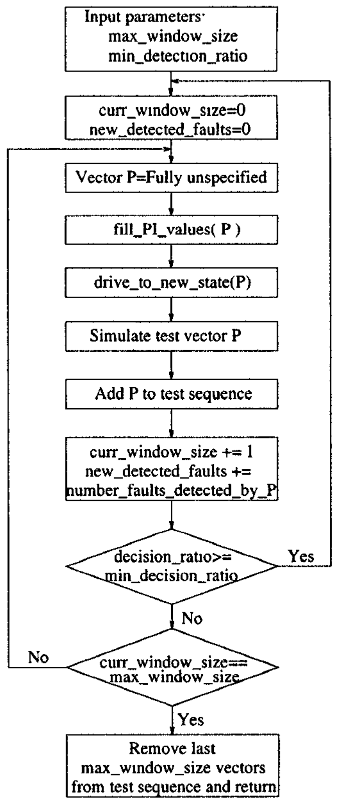


Figure 3: State-driven test generation

Once the state-driven test generation phase is done, fault simulation is performed for the input sequence T made up of the synchronizing sequence followed by the state-driven test sequence. Faults that are not detected are considered for test generation under the restricted multiple observation times approach. The purpose of this step is twofold. (1) To detect additional faults. (2) To synchronize each faulty circuit as much as possible to help detect the fault during deterministic test generation. The restricted multiple observation times test generation procedure is based on state expansion, as follows. The input sequence T is simulated starting the faulty circuit from the all-unspecified state. As long as the number of specified state variables increases, the simulation continues. Otherwise, an unspecified state variable y is selected, the state is duplicated, and y is set to 0 in one copy and to 1 in the other copy. The state variable y is selected such that it affects the largest number of unspecified next state variables. Simulation is repeated with the new states. If the fault is detected for any of the expanded states, the state is dropped from further consideration. The expansion process is terminated if one of the following conditions is met. (1) The fault is detected for all the expanded states. (2) All the next states are fully specified. (3) The next states are specified on more state variables than the expanded present states. (4) Preselected bounds on the number of expansions and the number of expanded states are reached. This process continues for the succeeding time frames.

If all the expanded states at the end of this process are fully specified, deterministic test generation is invoked for each state to distinguish it from the fault free state. Test generation is done forward in time. If all the states are distinguished from the fault-free state, the fault is marked as detected and the distinguishing sequence is added to the test sequence.

3.2 Deterministic Test Generation

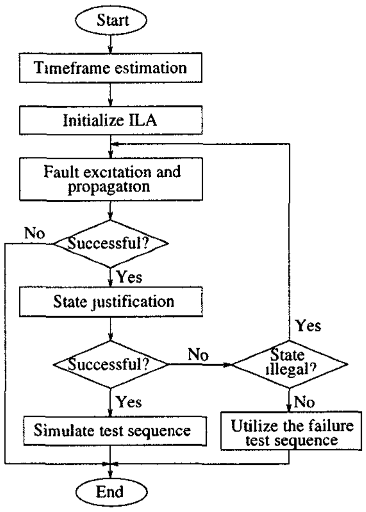


Figure 4: Deterministic test generation

The deterministic test generation algorithm used in MIX is shown in Figure 4. It is similar to the Lee-Reddy procedure [6] and HITEC [7] in that fault propagation is done forward in time, and state justification is done backward in time. The new features of the MIX deterministic test generation procedure are described next.

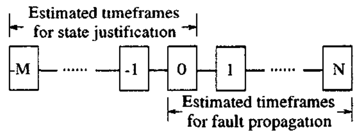

Estimated timeframes for state justification
-M
-1
0
1
N
Estimated timeframes for fault propagation

Figure 5: Initial ILA

MIX estimates the number of time frames required for state justification and for fault propagation by using controllability and observability measures. Estimation of the number of time frames required to detect a fault was also used in FASTEST [8]. If the total number of time frames does not exceed a preselected bound, then an iterative logic array of the appropriate length is created. Figure 5 shows the initial ILA. The fault activation time frame is referred to as time frame 0. Time frames 1 to N are reserved for fault propagation, and time frames -1 to $-M$ are reserved for state justification. This iterative logic array is used during the fault activation and propagation phase, and may be extended by adding time frames $N+1$, $N+2$, ... as required to detect the fault. Time frames -1 to $-M$ are maintained in order to increase the likelihood that the present state at the activation time can be justified, as follows. During the fault activation and propagation phase, if a present state value is assigned at time frame 0, it is implied into time frames -1, -2, ..., $-M$. As a result, the values of additional state variables at time frame 0 may be determined. For example, suppose that a state variable y_1 is set to 0 at time frame 0. At time frame -1, the corresponding next state variable Y_1 is set to 0, and this value is implied. Suppose that as a result of setting Y_1 to 0, another next state variable Y_2 is set to 1. Then the value of y_2 at time frame 0 is also set to 1, and states that assign the combination (00) to y_1 and y_2 will be avoided. In this case, test generation procedures that allow any state at time frame 0 may select to assign 0 to both y_1 and y_2, and they will fail during the state justification phase. When that happens, fault activation and propagation will have to be repeated. MIX uses time frames -1, -2, ..., $-M$ to determine the relationships among the values of the state variables at time frame 0 in order to reduce the likelihood of obtaining an invalid state that cannot be justified. The effectiveness of this type of implications was also observed in [36].

State justification is done backward in time. However, instead of considering one time frame at a time as in other procedures [6][7], several time frames are considered simultaneously. These include a time frame i which is decremented by one at the end of every iteration, and in addition time frames $i-1$, $i-2$, ... The reason is that conflicting assignments may be identified faster by performing implications through several preceding time frames, similar to the reason for using time frames -1, -2, ..., $-M$ during fault activation and propagation. Moreover, dynamic learning at the current state lines of the time frame under consideration is done before state justification begins at that time frame in order to compute the dependencies among the flip-flop values. The learning will reduce the likelihood of generating an illegal state. This technique is called *over-expanding* in MIX.

Multiple backtrace similar to FAN [38] is used in MIX during state justification in one time frame. Instead of making decisions at internal stems, all decisions are made at primary inputs and current state lines. The contradiction ratio defined as

$$\mid n_0 - n_1 \mid / (n_0 + n_1)^2$$

determines the decision point and its value. Here, n_0 and n_1 are the numbers of zero and one requirements at a signal line. The line with the minimal contradiction ratio is selected as the decision point and the decision value is decided by the maximal value of n_0

and n_1. Furthermore, the decision value in the faulty circuit is set to be the same as that in the fault free circuit if the value in the fault free circuit has already been assigned.

The partial state transition graph is used to guide the decision process during the justification phase as follows. Let the first time frame of the iterative logic array be i. Let the state at time frame i be S_j. We search the partial state transition graph for a state S which is at the smallest possible Hamming distance from S_j, and has the shortest transfer sequence from either the final state reached under the existing test sequence, S_f, or the synchronization state, S_s. We search for two transfer sequences starting from S_f and S_s, and pick the shorter one. Let the state preceding S in the transfer sequence from S_f or S_s to S be S_p. We use S_p to guide state justification during time frame i. When a decision is made as to the values of the state variables at time frame i, the values closest to S_p are selected.

When state justification fails due to limits imposed on computation time (e.g., the backtrack limit), the targeted states at each time frame are checked to find a fault-free state S_k at time frame m that covers any state reachable from S_s in the partial state transition graph. If S_k is found, a shortest transfer sequence starting from S_f or S_s is selected. An iterative logic array with a number of time frames equal to the transfer sequence length is created, and the states in the transfer sequence are assigned to the corresponding time frames of the fault free circuit. The states at the first time frame of the iterative logic array are set according to the sequence used to define the fault free states. An input sequence is searched for in this iterative logic array to satisfy both the fault free and faulty states. If an input sequence is found, it is concatenated to the partial justification sequence after time unit m to construct the justification sequence. If the initial state is S_s, the synchronizing sequence is added as a prefix of the justification sequence.

After the activation, propagation and justification phases are done, we concatenate the input sequences generated (regardless of whether the target fault was detected or not). We then use procedure *fill_PI_values*() to determine input values that remain unspecified. The resulting sequence is simulated and the faults it detects are dropped from further consideration. If no new faults are detected by the resulting sequence, the resulting test sequence is kept as a seed for the genetic optimization based procedure.

3.3 Genetic-Optimization Based Phase

After the deterministic test generation phase, some faults are marked as aborted due to failure of the justification phase. For each such aborted fault, an iterative logic array consisting of a preselected number of time frames is created. By assuming that the state variables in the first time frame are fully controllable, the target state of the justification phase is justified. The justification sequence generated in this way is called a pseudo-justification sequence.

We concatenate the pseudo-justification sequence with the propagation sequence of the fault as shown

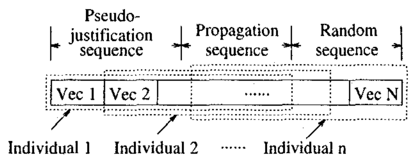


Figure 6: Generate the initial individuals

in Figure 6. We then use the sequence to seed the genetic-optimization based test generation procedure. The genetic-optimization procedure is then called to create the best possible individual.

4 Experimental Results

MIX was implemented in C++ and run on an HP 9000-715 workstation. It was applied to ISCAS-89 and ADDENDUM-93 benchmark circuits. The fault simulator PROOFS [39] was used to simulate the generated test sequences in addition to the restricted multiple observation times simulators embedded in MIX. Table 1 shows the test generation results. For each circuit, the total number of collapsed faults is given under column Total Faults. The number of faults detected and the number of untestable faults are reported under columns Detected and Untestable, respectively. The number of detected faults corresponds to the restricted multiple observation times strategy. In parentheses under the Detected column we also show the number of faults detected by conventional fault simulation using PROOFS. Column Vectors shows the test sequence length, and column CPU shows the CPU time in seconds taken by MIX. The number of faults detected by state-driven test generation are given under the State-Driven column.

We point out that some faults that can be detected under the conventional testing approach are detected by MIX under the restricted multiple observation times approach. Once a fault is detected under the latter approach, it will not be considered again under the conventional one. Consequently, the total number of detected faults should be considered when studying the effectiveness of MIX.

The results of MIX are compared to the results of STRATEGATE [18], a GA-based test generator. The results obtained by STRATEGATE are shown in Table 1 as well. STRATEGATE was run on an HP 9000-J200 workstation. As can be seen from the table, MIX detects at least as many faults as STRATEGATE for all the circuits. It also identifies a large portion of the untestable faults. Moreover, for circuits s510 and s953 that cannot be synchronized using three-value simulation, MIX achieves the same fault coverage as the symbolic-based test generator reported in [25].

For circuits s641 and s713, MIX identifies 0 and 36 untestable fault, respectively. This is due to the fact that the deterministic test generation phase estimates the number of time frames needed for fault excitation and propagation. When this number exceeds a preselected constant, the fault is not targeted. None of the faults left in circuits s641 and s713 were targeted by the deterministic test generator. However, when they were considered separately, MIX verified that all of these faults are untestable.

MIX generates shorter test sequences for 12 of 17 circuits than that by STRATEGATE. Additionally, the CPU times are shorter in general.

5 Concluding Remarks

We presented a test generation system, MIX, for synchronous sequential circuits. It mixes deterministic test generation, state-driven and genetic optimization based test generation in order to achieve very high fault coverages. MIX is composed of four main procedures, circuit synchronization, state-driven test generation, deterministic test generation, and genetic optimization base test generation. The test generation works from the least computational intensive to the most computational intensive approach such that the faults go through several test generation strategies before they are marked as aborted.

With the aid of several new techniques, including the XD-frontier, storing of a partial state transition graph, utilization of the sequences generated for aborted faults, and computing flip-flop dependencies dynamically, test generation was shown to be efficient. By applying the restricted multiple observation times approach, fault coverages were improved significantly for circuits for which the conventional single observation time strategy achieves very low fault coverages.

References

[1] G. R. Puzolu and J. P. Roth, "A Heuristic Algorithm for the Testing of Asynchronous Circuits," *IEEE Trans. on Computers,* pp. 639-647, June 1971.

[2] R. Marlett, "An Efficient Test Generation System for Sequential Circuits," *23rd Design Automation Conference,* June 1986, pp.250-256.

[3] H-K. T. Ma, et. al., "Test Generation for Sequential Circuits," *IEEE Trans. on Computer-Aided Design,* Vol. 7, No. 10, pp. 1081-1093, Oct. 1988.

[4] M. Schulz and E. Auth, "ESSENTIAL : An Efficient Self-Learning Test Pattern Generation Algorithm For Sequential Circuits," *1989 Internation Test Conference,* pp. 28-37.

[5] W. Cheng and T. Chakraborty, "Gentest – An Automatic Test Generation System for Sequential Circuits," *IEEE Computer,* pp. 43-49, April 1989.

[6] D. H. Lee and S. M. Reddy, "A New Test Generation Method for Sequential circuits," *Proc. of International Conference on Computer Aided Design,* pp. 446-449, 1991.

[7] T. Niermann and J. Patel, "HITEC: A Test Generation Package for Sequential Circuits," *European Conf. on Design Automation 1991,* pp. 214-218.

[8] T. Kelsey, K. Saluja, and S. Lee, "An Efficient Algorithm for Sequential Circuit Test Generation," *IEEE Trans. on Computer,* vol. 42, pp. 1361-1371, November 1993.

[9] V. D. Agrawal, K.-T. Cheng, and Prathima Agrawal, "CONTEST: A Concurrent Test Generator for Sequential Circuits," *25th ACM/IEEE Design Automation Conference,* 1988, pp.84-89.

[10] D. G. Saab, Y. G. Saab and J. A. Abraham, "CRIS: A Test Cultivation Program for Sequential VLSI Circuits," *Intl. Conf. Computer-Aided Design,* Nov. 1992, pp. 216-219.

[11] D. G. Saab, Y. G. Saab and J. A. Abraham, "Iterative [Simulation-Based Genetics + Deterministic Techniques] = Complete ATPG," *in Proc. 1994 Intl. Conf. on Computer-Aided Design,* Nov. 1993, pp. 40-43.

[12] P. Prinetto, M. Rebaudengo, and M. S. Reorda, "An Automatic Test Pattern Generator for Large Sequential Circuits based on Genetic Algorithm", *International Test Conference,* 1994, pp. 240-249.

[13] E. M. Rudnick, J. H. Patel, G. S. Greenstein and T. M. Niermann, "Sequential Circuit Test Generation in a Genetic Algorithm Framework, *in Proc. Design Autom. Conf.,* June 1994, pp. 698-704.

[14] E. M. Rudnick and J. H. Patel, "Combining Deterministic and Genetic Approaches for Sequential Circuit Test Generation," *in Proc. 32rd Design Autom. Conf.,* June 1995, pp. 183-188.

[15] E. M. Rudnick, J. G. Holm, D.G. Saab, and J. H. Patel, "Application of Simple Genetic Algorithms to Sequential Circuit Test Generation," *Proc. European Design & Test Conference,* 1994, pp. 40-45.

[16] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Automatic test Generation Using Genetically-engineered Distinguishing Sequences," *IEEE VLSI Test Symposium,* April 1996, pp.216-223.

[17] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Alternating Strategies for Sequential Circuit ATPG," *Proc. European Design and Test Conf.,* 1996.

[18] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Sequential Circuit Test Generation Using Dynamic State Traversal," *European Design and Test Conference,* pp. 22-28, 1996.

[19] J. A. Wehbeh and D. G. Saab, "Initialization of Sequential circuits and its Application to ATPG," *14th VLSI Test Symposium,* pp. 246-251, 1996.

[20] I. Pomeranz and S. M. Reddy, "LOCSTEP: A Logic Simulation Based Test Generation Procedure," *25th Fault-tolerant Computing Symp.,* pp. 110-119, June 1995.

[21] A. Ghosh, S. Devadas, and A. R. Newton, "Test Generation for Highly Sequential Circuits," *Proc. Int. conference on computer Aided Automation,* pp. 362-365, 1989.

[22] A. Dargelas, C. Gauthron and Y. Bertrand, "MOSAIC: A Multiple-Strategy Oriented Sequential ATPG for Integrated Circuits," *in Proc. Europ. Design & Test Conf.,* March 1997, pp. 29-36.

[23] H. Cho, G. D. Hatchel, F. Somenzi, "Redundancy Identification/Removal and Test Generation for Sequential circuits Using Implicit State Enumeration," *IEEE Trans. on Computer Aided Design,* Vol. CAD-12, No. 7, pp. 935-945, July 1993.

[24] H. Cho, S. W. Jeong, and F. Somenzi, "Synchronizing Sequences and Symbolic Traversal Techniques in Test Generation," *Journal of Electronic Testing: Theory and Applications,* 4, pp. 19-31, 1993.

[25] G. Cabodi, et. al., "Full Symbolic ATPG for Large Circuits," *1994 International Test Conference,* pp. 980-986.

[26] F. Corno, et. al., "Improving Topological ATPG with Symbolic Techniques," *Proc. 13th VLSI Test Symposium,* pp. 338-343, 1995.

Table 1. Test Generation Results

	Total	MIX					STRATEGATE		
Circuit	Faults	Detected	Untestable	Vectors	CPU(sec)	State-Driven	Detected	Vectors	CPU(sec)
s208	215	150 (137)	65	128	3.1	114	-	-	-
s298	308	273 (263)	26	206	30.1	251	265	306	-
s344	342	335 (328)	5	76	6.4	286	329	85	-
s349	350	341 (334)	7	104	4.2	290	-	-	-
s382	399	375 (363)	4	834	493	342	364	1486	486
s386	384	314 (314)	70	226	11.1	278	-	-	-
s400	424	396 (382)	12	1170	467	345	384	2424	-
s420	430	204 (179)	226	100	15.6	159	-	-	-
s444	474	435 (423)	16	870	750	401	424	1945	1206
s510	564	564 (0)	0	587	80	234	-	-	-
s526	555	462 (451)	48	1545	1342	373	454	2642	3270
s641	467	408 (404)	0	131	4.3	303	404	166	-
s713	581	480 (476)	36	130	7.3	416	476	176	80
s820	850	815 (814)	35	838	67	443	814	590	218
s832	870	819 (817)	51	881	93	479	818	701	-
s838	857	303 (253)	501	109	223	238	-	-	-
s953	1079	1069 (90)	6	563	327	888	-	-	-
s967	1066	1047 (79)	13	530	444	866	-	-	-
s1196	1242	1239 (1239)	3	306	14.5	981	1239	574	90
s1238	1355	1283 (1283)	72	347	18.9	1079	1282	624	-
s1269	1343	1339 (240)	3	341	49.5	1300	-	-	-
s1423	1515	1455 (1403)	18	1658	3412	586	1414	3943	4572
s1488	1486	1446 (1441)	40	918	184	986	1444	593	-
s1494	1506	1455 (1452)	50	759	176	1084	1453	540	456
s1512	1357	827 (66)	7	679	7021	582	-	-	-
s3271	3270	3246 (3246)	0	1394	1870	3078	-	-	-
s3330	2870	2124 (2124)	707	756	997	1901	-	-	-
s3384	3380	3334 (3312)	15	1406	2059	2933	-	-	-
s4863	4763	4638 (4638)	125	612	1190	4118	-	-	-
s5378	4603	3643 (3621)	749	1766	7601	2849	3639	11571	136080
s6669	6684	6684 (6675)	0	582	224	6563	-	-	-
s35932	39094	35109 (35099)	3984	296	25248	33403	35100	257	39240
prolog	3305	2325 (2325)	694	958	6250	2148	-	-	-

[27] J. Park, C. Oh, and M. R. Mercer, "Improved Sequential ATPG Using Function Observation Information and New Justification Methods," *Proc. European Design and Test Conference*, pp. 262-266, 1995.

[28] I. Pomeranz and S. M. Reddy, "The Multiple Observation Time Test Strategy," *IEEE Tran. on Computer*, pp. 627-637, May 1992.

[29] I. Pomeranz and S. M. Reddy, "Application of Homing Sequence to Synchronous Sequential Circuit Testing," *IEEE Trans. on Computers*, Vol. 43, No. 5, May 1994.

[30] T. E. Marchok, A. El-Maleh, W. Maly and J. Rajski, "Complexity of Sequential ATPG," *in Proc. Europ. Design and Test Conf.*, March 1995, pp. 252-261.

[31] I. Pomeranz and S. M. Reddy, "Low-Complexity Fault Simulation under the Multiple Observation time Testing Approach," *International Test Conference*, 1995, pp. 272-281.

[32] I. Pomeranz and S. M. Reddy, "On Fault Simulation for Synchronous Sequential Circuits," *IEEE Trans. on Computers*, Feb. 1995, pp. 335-340.

[33] H. Cho, G. D. Hachtel, E. Macii, B. Plessier and F. Somenzi, "Algorithms for Approximate FSM Traversal," *in Proc. 30th Design Autom. Conf.*, June 1993, pp. 25-30.

[34] X. Chen and M. L. Bushnell, "Generalization of Search State Equivalence for Automatic Test Pattern Generation", *Proc. of 8th International Conference on VLSI Design*, 1995.

[35] I. Pomeranz, L. N. Reddy and S. M. Reddy, "COMPACTEST: A Method To Generate Compact Test Sets for Combinational Circuits", *IEEE Transactions on Computer-Aided Design*, July 1993, pp. 1040-1049.

[36] I. Pomeranz and S. M. Reddy, "Fault Simulation under the Multiple Observation Time Approach using Backward Implications", *in Proc. 34th Design Autom. Conf.*, June 1997.

[37] P. Muth, "A Nine-Valued Circuit Model for Test Generation," *IEEE Trans. on Computers*, pp. 630-636, June 1976.

[38] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithm," *IEEE Trans. on computers*, Vol. C-32, No. 12, pp. 1137-1144, December, 1983.

[39] T. M. Niermann, W. Cheng, and J. Patel, "PROOFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator," *IEEE Trans. on Computer-Aided Design*, pp. 198-207, February 1992.