Combining Symbolic and Genetic Techniques for Efficient Sequential Circuit Test Generation

M. Boschini [#] X. Yu [†] F. Fummi [‡] E. M. Rudnick [†]

[#] ST Microelectronics
Agrate
Milano, ITALY

[‡] Università di Verona
DST Informatica
Verona, ITALY

[†] Center for Reliable & High-Performance Computing
University of Illinois
Urbana, IL, USA

Abstract

Symbolic and genetic techniques are combined in a new approach to sequential circuit test generation that uses circuit decomposition, rather than the algorithmic decomposition used in previous hybrid test generators. Symbolic techniques are used to generate test sequences for the control logic, and genetic algorithms are used to generate sequences for the datapath. The combined sequences provide higher fault coverages than those generated by existing deterministic and GA-based test generators, and execution times are significantly lower in many cases.

1 Introduction

Several different approaches have been proposed for sequential circuit test generation, including deterministic techniques, genetic techniques, symbolic techniques, and various combinations of the three. Deterministic algorithms tend to be highly complex and time consuming [1, 2, 3, 4]. As an example, the HITEC test generator [4] uses both forward and reverse time processing to generate a test for a target fault. During test generation, each target fault must be excited and the fault effects propagated to a primary output (PO), either in the same time frame in which the fault is excited or in a subsequent time frame. The required state must then be justified. If conflicts are found, backtracking is required, and alternative decisions must be made. Genetic algorithm (GA) based approaches simplify test generation by processing in the forward direction only [5, 6, 7, 8]. Populations of candidate test sequences are evolved over several generations, with more highly fit sequences propagating from one generation to the next. Fitness is determined based on fault detection capabilities, amount of circuit activity induced, and other factors related to detecting the target faults. GA-based techniques have been especially effective for data-dominant circuits. Symbolic techniques [9, 10] based on binary decision diagrams (BDDs) have also been used very effectively on control-dominant circuits. Very high fault coverages have been achieved, although the size of the circuit that can be handled is limited.

Deterministic and genetic techniques have been combined in an attempt to improve fault coverages and reduce execution time. Using GA's for state justification and deterministic techniques for fault excitation and propagation was beneficial for some circuits [11]. Alternating deterministic and genetic phases resulted in improvements in both fault coverage and execution time [12, 13]. Significant improvements in fault coverage were obtained by integrating a GA-based test generation framework with deterministic test generation techniques for fault propagation and backtracing of required values in a combinational logic block [14]. Combining a GA with symbolic techniques for fault propagation and accurate fault simulation gave good results for many circuits [15], and higher fault coverages were achieved for a multiple observation time test strategy. Symbolic algorithms were also useful when integrated with a deterministic test generation algorithm in providing an early backtrack indication, which resulted in higher fault coverages and lower execution times [16].

Combination of the various test generation techniques has benefited from the fact that the different techniques are best suited for different types of circuits. However, in all cases, the proposed approaches have relied upon decomposition of the test generation algorithm, with the different techniques being used to carry out particular tasks. For example, GA's were used for state justification in [11], deterministic algorithms were used for state space traversal in [12, 13], and symbolic algorithms were used for accurate evaluation of candidate sequences in [15].

We propose a new approach to test generation that combines symbolic and genetic techniques but which uses circuit decomposition rather than algorithmic decomposition in determining when to apply each of the two techniques. The circuit is decomposed into datapath and control logic portions, and then the test generation technique best suited for each portion is applied. Symbolic algorithms have proved to be the most effective for control dominant circuits, so they are used for the control logic. GA-based techniques have been most effective for data-dominant circuits, so a GA is used for the datapath.

The proposed testing methodology requires that the gate-level circuit description used as input be decomposed into a controller and datapath. This description can be obtained through register transfer level (RTL) synthesis starting from a description based on the Finite State Machine with Datapath (FSMD [17]) model. The FSMD model allows the behavior of an RTL description to be easily captured, since it separates the control part of a description from the computation part. Moreover, behavioral descriptions are typically translated into a target RTL architectural model

in the form of a FSMD. If this kind of description is automatically generated, the controller is clearly separated from the datapath, which is composed of library modules such as registers, operators, and multiplexers. On the contrary, a device description directly written at the register transfer level can be a mix of control and computation. Hardware description languages such as VHDL or Verilog allow modeling of state transitions to be combined with the computation of results. In this case, it is possible to apply the restructuring methodology described in [18], which translates a generic RTL VHDL description into a reference model in which the VHDL code representing the controller (FSM) and the code describing the datapath are clearly identified. As shown in [18], the restructuring methodology allows for better application of sequential optimization algorithms, resulting in more optimized gate-level circuits.

Combination of symbolic test generation algorithms for the controller with GA-based test generation algorithms for the datapath provides high fault coverages without the need for a significant amount of design-for-testability (DFT) hardware. The required DFT hardware consists of a partial scan chain that includes only those registers providing status information from the datapath to the controller. In summary, the proposed testing methodology can be applied to those circuits that can be described as shown in Figure 1, in which the scan chain provides complete controllability and observability of the status signals.

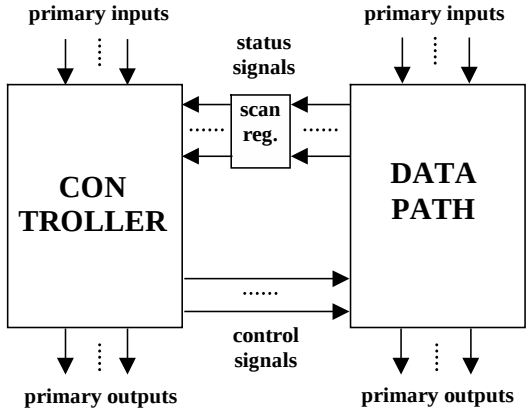

Figure 1: **Reference circuit model.**

The remainder of the paper is organized as follows. The overall testing methodology is presented in Section 2, where the roles of symbolic techniques and GA's and their interaction are clarified. The use of BDD-based techniques is described in Section 3, while the application of GA's is presented in Section 4. Section 5 presents experimental results that confirm the effectiveness of the proposed test generation approach, while Section 6 gives some concluding remarks and discusses future work.

2 Testing Methodology

The core problem of the proposed methodology concerns the interaction between BDD-based and GA-based techniques for test generation. Symbolic techniques allow the generation of groups of test sequences for the same cost as the generation of a single test sequence [10]. Moreover, convergence of GA's depends upon the accuracy and size of the initial population (in this case, the initial set of test patterns). For these reasons, the proposed methodology uses BDD-based algorithms to generate sets of partially specified patterns, which are tuned by GA's.

Two main phases compose the proposed methodology: (1) `phase 1`, when faults in the datapath are targeted, and (2) `phase 2`, when faults in the controller are targeted. The two phases are described next.

2.1 Phase 1

The main objective of this phase is generation of test sequences that detect faults in the datapath, starting from sequences that traverse the finite state machine (FSM) that represents the controller. In fact, the isolated datapath portion would be easily tested by using GA's alone, but the datapath is driven by a controller in the cases examined here. The application of GA's to the entire FSMD would likely produce unacceptable results, due to the difficulty of FSM traversal by GA's. Thus, BDD's are used to deterministically traverse the FSM, and completion of these sequences in order to test faults in the datapath is left to the GA.

`Step 1` **Initialization Sequence:** An initialization sequence is necessary to drive the fault-free FSM into a specified initial state. In this way, two-valued BDD's can be used for FSM traversal instead of a more complex three-valued representation [9]. A random initialization sequence is used; if necessary, more complex techniques could be applied (e.g., [9, 19]), which do not need an initialization sequence. In any case, the use of a reset signal in the FSM synthesis is a reasonable assumption.

`Step 2` **FSM Traversal:** The FSM must be traversed by exciting a maximum number of paths to maximize the variations of sequences on the control signals, which affects the detection of faults in the datapath. No general criteria can be identified to guide this traversal; thus, we use a test generation method to perform this task. Each fault of the FSM is analyzed to identify a test sequence that activates it and propagates its fault effects to the outputs of the FSM. Then, the corresponding output sequence is derived by performing an image operation on the FSM relation (see Section 3). This strategy guarantees excitation of a majority of behaviors of the FSM, since it tries to distinguish the fault-free behavior from all faulty behaviors.

`Step 3` **Datapath Completion:** Output sequences generated in the previous step represent partially specified input sequences for the datapath, since primary inputs (PI's) are left unspecified. A GA is used to complete the unspecified values by targeting the detection of stuck-at faults in the datapath (see Section 4).

Finally, input sequences for the controller are merged with the sequences generated for the datapath. The combined sequences detect:

- all faults of the datapath that have been successfully analyzed in `step 3`;

- all faults of the FSM that can be observed on the PO's of the FSM;
- all faults of the FSM that can be observed only on the control lines, but have been serendipitously propagated to the PO's of the datapath during `step 3`.

Once sequences have been generated for all faults in the controller and datapath completion has been performed, the test generator proceeds to `phase 2`.

2.2 Phase 2

The main objective of this phase is the generation of test sequences that detect the faults that remain undetected in the controller. Such faults are targeted again as in `phase 1`, but the test generator for the datapath is targeted to propagate fault effects on the control lines to the PO's of the datapath.

`Step 1` **FSM Test Generation:** All test sequences shorter than a specified limit are generated for the target faults by using the implicit algorithm described in Section 3. Such sequences are generated starting from the fault-free-state/faulty-state pair reached by the fault-free circuit and the faulty circuit after application of the test sequence identified in the previous steps. Each test sequence is transformed into two output sequences on the control lines, corresponding to the fault-free and faulty responses. This operation is implicitly performed for all test sequences simultaneously. Then, each pair of output sequences is separately analyzed in `step 2`.

`Step 2` **Fault Propagation in Datapath:** Pairs of output sequences generated in the previous step represent partially specified pairs of input sequences for the datapath, since PI's of the datapath are left unspecified. The GA is employed to complete unspecified values in order to propagate the differences between the fault-free and faulty sequences to the PO's of the datapath (see Section 4).

If the operation in `step 2` is able to propagate the fault effects, the corresponding input sequence of the controller is merged with the input sequence of the datapath to produce a test sequence for the circuit that is able to detect the current target fault in the FSM. These two steps are repeated until no other faults remain in the fault list of the controller.

The next two sections describe in depth the BDD- and GA-based techniques for generation of test sequences.

3 BDD-Based Analysis

Let us model the controller of the FSMD as an FSM M, which is described as a 5-tuple $M = (X, Z, S, S^0, R)$, where X is the input alphabet, Z is the output alphabet, S is the finite set of states, S^0 is the reset state, and $R \subseteq S \times X \times S \times Z \rightarrow \{0,1\}$ is the *global relation*. The characteristic function $R(x,z,s,t) = 1$ if and only if, under input $x \in X$, the FSM makes a transition from present state $s \in S$ to next state $t \in S$ outputting $z \in Z$. This characteristic function is represented by using a BDD-based description. An FSM can be represented by a *state transition graph* (STG), whose vertices are elements of $s \in S$ and edges are labeled with pairs $(x,z) \in X \times Z$. Input symbols X are coded by I input variables $i_1, \cdots, i_I$, and output symbols Z are coded by O output variables $o_1, \cdots, o_O$.

Let M_f be the FSM representing the behavior of M affected by the fault f. It is represented by the faulty global relation $R_f(x,z,s,t)$. A *test sequence* for fault f is a sequence of input vectors such that, when applied to machines M and M_f (both starting from their reset states), produces two different output vectors for M and M_f. By concurrently traversing both M and M_f starting from their reset states, it is possible to generate (if any) a test sequence.

The assumption of the existence of a reset state is reasonable for the control logic in the class of circuits analyzed here; however, the functional test generation approach works also if a pair of *initial states* is used, i.e., if an initialization sequence is used to drive the fault-free and faulty FSM's from the initial unknown state into a pair of (different) known states. Faults that prevent initialization of the faulty FSM are not considered.

For each stuck-at fault, the corresponding faulty FSM M_f is extracted from the gate-level circuit description. To improve efficiency, clusters of faults are concurrently examined.

3.1 Test Sequence Representation

Given a fault, f, one way of representing all test sequences for f shorter than a limit is through a directed acyclic graph (DAG), $SEG(f)$, called in the sequel the *sequence expansion graph* [10]. Each node of $SEG(f)$ is labeled s/s_f, where s and s_f are states of the fault-free and faulty STGs, respectively. An edge connecting nodes s/s_f and t/t_f in $SEG(f)$ and labeled x represents the pair of transitions which allow the two STGs to move, under input x, from states s and s_f to states t and t_f, respectively.

The sequence expansion graph can be built by first considering the node associated with the reset states of M and M_f, i.e., the node labeled S^0/S^0_f and called, in the sequel, the *root* of the sequence expansion graph. Then, for each input x, the pair of next states t/t_f is computed such that $R(x,s,t,z) = 1$ and $R_f(x,s_f,t_f,z_f) = 1$. If outputs z and z_f are equal, then the root is connected to node t/t_f by an edge labeled x. Otherwise (z and z_f are different), the root is not connected to any node, for input x, and an edge labeled x (called a *distinguishing edge*) is attached to the node.

All newly created nodes are processed following a breadth-first search scheme and using the expansion technique just outlined. Therefore, $SEG(f)$ is levelized, and all nodes belonging to the i-th level are reachable from the root by an input sequence of length i.

We associate two relations with each level i of the sequence expansion graph. The first one, named *potentially distinguishing sequence* $PDS[i]$, stores all transitions for which M and M_f are characterized by the same output vector:

$$PDS[i] \subseteq X \times S \times S_f \times S \times S_f \rightarrow \{0,1\}$$

$PDS[i](x,s,s_f,t,t_f) = 1$ if and only if, under input vector $x \in X$, the fault-free machine M makes a transition from present state $s \in S$ to next state $t \in S$,

the faulty machine M_f makes a transition from present state $s_f \in S_f$ to next state $t_f \in S_f$, and both M and M_f generate the same output vector.

The second relation, named *sequentially distinguishing sequence SDS*[i], stores all transitions for which M and M_f produce a different output vector (distinguishing edges):

$$SDS[i] \subset X \times S \times S_f \rightarrow \{0,1\}$$

Similarly to $PDS[i]$, $SDS[i](x,s,s_f) = 1$ if and only if, under input vector $x \in X$, the output vector generated by M with a transition out-going from state $s \in S$ differs from the output vector generated by M_f with the transition out-going from state $s_f \in S_f$.

To summarize, representing the sequence expansion graph as a whole requires two arrays of relations, whose sizes are bounded by the sequential depth of the product machine $M \times M_f$. This representation allows us to implicitly describe multiple sequences without mixing transitions belonging to distinct sequences. Algorithms for implicit computation of the sequence expansion graph can be found in [10], where they are implemented in the *IFSMTest* testability environment for sets of FSM's. The approach can be efficiently applied to FSM's with a sequential depth of thousands and is thus adequate for the class of controllers analyzed here.

3.2 Output Sequence Generation

Step 2 of `phase 1` and step 1 of `phase 2` require the generation of output sequences of the FSM corresponding to the test sequences identified. Such output sequences are represented by using two other arrays of relations, which are derived from $PDS[i]$ and $SDS[i]$ by performing *image* operations. The $OPDS[i]$ relation (O means *output*) stores all transitions for which M and M_f produce the same output:

$$OPDS[i] \subset S \times S_f \times Z \times S \times S_f \rightarrow \{0,1\}$$

$OPDS[i](s,s_f,z,t,t_f) = 1$ if and only if M has a transition from $s \in S$ to $t \in T$, and M_f has a transition from $s_f \in S$ to $t_f \in T$, but produces the same output value $z \in Z$. On the contrary, the other relation, $ODS[i]$, stores all transitions for which M and M_f produce a different output:

$$ODS[i] \subset S \times S_f \times Z \times Z_f \rightarrow \{0,1\}$$

$ODS[i](s,s_f,z,z_f) = 1$ if and only if M and M_f generate different output values ($z \neq z_f$) respectively starting from $s \in S$ and $s_f \in S$.

All output sequences are concurrently generated for a cluster of faults during the test generation process by performing one traversal of M and M_f. After that, different output sequences can be obtained by repeatedly scanning levels of $OPDS[i]$ and $ODS[i]$, thus generating sets of vectors, which are analyzed by the GA's described in the next section.

4 GA-Based Analysis

COTE, the GA-based test sequence generator for the datapath, accepts a partially specified sequence as input and generates a fully specified sequence as output. Only the description of the datapath is provided to COTE, complete with control signals, datapath PI's, datapath PO's, and status signals. Note that the status signals are considered to be PO's, since they will be included in the partial scan chain. The specified inputs of the sequence are the values of control signals provided by the controller, and the unspecified inputs are the values of datapath PI's. The objective of the GA is to detect faults by setting appropriate datapath PI values, with the constraint that the control signals are set to values which can be generated by the controller. As described in Section 2, test generation is carried out in two phases. In the first phase, COTE uses the sequences provided by IFSMTest, the test generator for the controller, to target faults in the datapath. The GA will try to maximize the number of faults detected by each sequence. In the second phase, faults in the controller are targeted. The GA will therefore attempt to propagate fault effects from specific control signals to the PO's.

A simple GA similar to the one described in [7] is used in both phases of test generation in COTE. The following GA parameters are used: a population size of 32, 8 generations, a mutation rate of 1/64, and a crossover rate of 1. Tournament selection without replacement and uniform crossover are used.

In the first phase of test generation, the sequence length of individuals in the GA population is set equal to the length of the sequence provided by IFSMTest. GA individuals correspond to the values to be applied to the PI's of the datapath in the same time frames in which the given controller values are applied. The GA fitness function is primarily a measure of the number of faults detected for a random sample of 100 faults. The fraction of fault effects propagated to flip-flops is also included in the fitness function, however, since activated faults are more likely to be detected in the next time frame. A fault sample is used rather than the complete list of undetected faults in order to minimize execution time, since most of the time required by the test generator is used for fault simulation of candidate sequences. The GA is invoked repeatedly, once for each sequence provided by IFSMTest, until a signal is sent to COTE to switch to `phase 2`. Only sequences that detect faults in the datapath are returned to IFSMTest and added to the test set.

In the second phase of test generation, IFSMTest sends information about which control signals carry fault effects, along with the fault-free signal values. Fault effects can appear on multiple signals and in multiple time frames. The test generator for the datapath uses the PODEM algorithm [20] in an attempt to propagate fault effects from the control signals to the datapath PO's. If PODEM is successful, then the GA is used in an attempt to justify the state required by PODEM for fault detection. If the fault effects cannot be propagated to the PO's in the same time frame in which they arrive at the control inputs, an attempt is made to propagate the fault effects to the datapath flip-flops using PODEM. The procedure is then repeated for the next time frame.

In trying to justify the state required by PODEM, the datapath PI's are set to random values for all individuals in the GA. The fitness function measures the

number of required flip-flop values that are correctly justified by each candidate sequence. If a sequence is successfully found that propagates the effects of the target fault to the PO's, the fully specified sequence is sent to IFSMTest, where the complete test sequence for the entire circuit is assembled and added to the test set.

5 Experimental Results

The implicit test generation algorithms have been implemented in the `fsm_dp_test` command of the *IFSMTest* environment. This program is composed of approximately 10,000 lines of C code; it uses the CUDD BDD library [21] for BDD manipulation. The GA's and PODEM have been implemented in C++ in the *COTE* program in 9000 lines of C++ code. *IFSMTest* and *COTE* were run on a Sun UltraSparc 30/248 with 1 Gbyte RAM. None of the experiments required more than 100 Mbytes RAM. The two programs were interfaced through sockets.

Table 1: Benchmark Characteristics

Circuit	PI's	PO's	Memory Elements	VHDL Lines
Write	14	13	18	1506
Tx	23	24	23	2514
Alpha	25	25	16	2614
Write_e	41	37	45	1707
Alpha_e	25	115	110	2830
Cidgen	133	37	104	6798
Ellipf	257	256	549	866

Table 2: Datapath and Control Logic Characteristics

Circuit	Controller			Datapath		
	In	Out	Mem	In	Out	Mem
Write	5	7	10	16	13	8
Tx	14	17	11	26	24	12
Alpha	19	23	10	29	25	6
Write_e	5	7	13	43	37	32
Alpha_e	19	23	14	29	115	96
Cidgen	6	16	13	139	33	91
Ellipf	1	39	5	295	256	544

The proposed methodology has been verified on industrial RTL benchmarks representing telecommunication devices described using VHDL. The restructuring methodology described in [18] was applied to the benchmarks to separate the control logic from the datapaths. In addition, the `Ellipf` high-level synthesis benchmark was also analyzed. It was synthesized starting from a high-level VHDL description; thus, the RTL description was already partitioned into an FSM and a datapath, and no restructuring technique was applied. Characteristics of the circuits are given in Table 1 in terms of inputs, outputs, memory elements, and number of VHDL lines. Testability of the unrestructured circuits is too poor to make simple sequential test generation a feasible testing solution [18]. Characteristics of the datapaths and control logic for the benchmark circuits are given in Table 2, including number of inputs, outputs, and memory elements of the two parts. The circuits were synthesized by using a commercial synthesis tool to obtain gate-level representations.

To measure the efficiency of the proposed methodology, the new test generator *IFSMTest+COTE* is compared to three other test generators: (1) HITEC [4], a deterministic test pattern generator that does not distinguish between the controller and datapath, (2) Strategate [14], a GA-based test generator that incorporates deterministic techniques for fault activation, and (3) one of the most effective commercial automatic test pattern generators (ATPG). All ATPG's are applied to the same gate-level descriptions obtained after insertion of the status signals into a scan chain (see Figure 1). Results are reported in Table 3.

The fault coverage (*%F.C.*) achieved using the proposed approach is the same as that of HITEC for one circuit and significantly higher for the other circuits. For similar fault coverages, fewer vectors are generated by the proposed approach. Execution times are lower, and sometimes significantly lower, for all but one circuit, but in that case, a higher fault coverage is achieved. Compared to Strategate, the proposed approach achieves a higher fault coverage for all but two circuits with far fewer test vectors. Strategate was run on an HP 9000 J200 with 256 MB RAM and required significantly greater execution time overall. The times shown do not include the time to run HITEC, which is invoked by Strategate. Total times are about twice those shown. Results for the `Ellipf` benchmark were still not available after one week of execution. Compared to the commercial ATPG, the proposed approach obtains the same fault coverage for one circuit and higher fault coverages for all other circuits. Test set sizes for the two test generators are comparable, as are execution times. Fault coverages obtainable by previously proposed deterministic/GA hybrids are expected to be between those of HITEC and Strategate, based on results reported for gate-level benchmarks. This supposition has been confirmed using GA-HITEC [11] on the *Write*, *Tx*, and *Alpha* benchmarks.

IFSMTest+COTE is able to achieve higher fault coverages for many of the circuits because of the sequential interaction between the controller and the datapath. This complexity cannot be efficiently explored by using a traditional topological approach, such as the one used in HITEC or in the commercial ATPG, or by using GA's only, such as in Strategate. The `Ellipf` benchmark emphasizes this difference.

6 Concluding Remarks

A novel approach to sequential circuit test generation is proposed that combines the advantages of symbolic and genetic techniques, using circuit partitioning. Symbolic algorithms, which have been shown to be the most effective for control dominant circuits, are used for the control logic, while GA's, which are most effective for data-dominant circuits, are used for the datapath. Experimental results show that the proposed test generator is significantly faster and produces shorter

Table 3: **Proposed Methodology Compared to HITEC [4], Strategate [14], and a Commercial ATPG**

		HITEC				IFSMTest+COTE			
Circuit	**Faults**	**Det**	**%F.C.**	**Vec**	**CPU sec**	**Det**	**%F.C.**	**Vec**	**CPU sec**
Write	408	312	76.5	3173	9509	312	76.5	1796	847
Tx	724	480	66.3	4079	19,463	542	74.9	3340	2749
Alpha	727	164	22.5	1504	40,877	367	50.5	2219	5094
Write_e	850	113	13.3	68	16,905	635	74.7	5090	5224
Alpha_e	1697	68	4.1	12	41,584	820	48.3	9082	45,805
Cidgen	2009	1488	74.1	7864	58,234	1940	96.6	8040	25,319
Ellipf	14,922	768	5.2	2	134,000	10,816	72.5	87	68
		Strategate				Commercial ATPG			
Circuit	**Faults**	**Det**	**%F.C.**	**Vec**	**CPU sec**	**Det**	**%F.C.**	**Vec**	**CPU sec**
Write	408	324	79.0	12,641	3479	295	72.4	3388	840
Tx	724	573	79.0	27,078	17,210	542	74.9	5426	1705
Alpha	727	366	50.5	9704	13,228	350	48.1	2963	1127
Write_e	850	113	13.0	3015	96,988	483	56.8	6017	15,072
Alpha_e	1697	68	4.0	3147	91,549	489	28.8	5797	23,432
Cidgen	2009	1397	70.0	12,816	327,857	1382	68.8	6797	41,477
Ellipf	14,922	-	-	-	-	9461	63.4	547	4649

test sequences for comparable fault coverages, and in most cases, the fault coverages obtained are higher than those of existing deterministic and GA-based test generators.

Future work will concern the extension of the proposed testing methodology to general case FSMDs with global feedback loops between the control and datapath parts.

References

[1] W. -T. Cheng, "The BACK algorithm for sequential test generation," *Proc. Int. Conf. Computer Design*, pp. 66–69, Oct. 1988.

[2] H. -K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli, "Test generation for sequential circuits," *IEEE Trans. Computer-Aided Design*, vol. 7, no. 10, pp. 1081–1093, Oct. 1988.

[3] R. Marlett, "An effective test generation system for sequential circuits," *Proc. Design Automation Conf.*, pp. 250–256, June 1986.

[4] T. M. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," *Proc. European Conf. Design Automation (EDAC)*, pp. 214–218, Feb. 1991.

[5] D. G. Saab, Y. G. Saab, and J. A. Abraham, "Automatic test vector cultivation for sequential VLSI circuits using genetic algorithms," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 10, pp. 1278–1285, Oct. 1996.

[6] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, "A genetic algorithm for automatic test pattern generation for large synchronous sequential circuits," *IEEE Trans. Computer Aided Design*, vol. 15, no. 8, pp. 991–1000, August 1996.

[7] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, "A genetic algorithm framework for test generation," *IEEE Trans. Computer-Aided Design*, vol. 16, no. 9, pp. 1034–1044, Sept. 1997.

[8] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Application of genetically engineered finite-state-machine sequences to sequential circuit ATPG," *IEEE Trans. Computer-Aided Design*, vol. 17, no. 3, pp. 239–254, March 1998.

[9] H. Cho, S. Jeong, F. Somenzi, and C. Pixley, "Synchronizing sequences and symbolic traversal techniques in test generation," *Journal of Electronic Testing: Theory and Applications*, vol. 10, no. 4, pp. 19–31, 1993.

[10] F. Ferrandi, F. Fummi, M. Poncino, E. Macii, D. Sciuto. "Symbolic optimization of interacting controllers based on redundancy identification and removal," *IEEE Trans. Computer-Aided Design*, to appear, 2000.

[11] E. M. Rudnick and J. H. Patel, "Combining deterministic and genetic approaches for sequential circuit test generation," *Proc. Design Automation Conf.*, pp. 183–188, June 1995.

[12] D. G. Saab, Y. G. Saab, and J. A. Abraham, "Iterative [simulation-based genetics + deterministic techniques] = complete ATPG," *Proc. Int. Conf. Computer-Aided Design*, pp. 40–43, Nov. 1994.

[13] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Alternating strategies for sequential circuit ATPG," *Proc. European Design and Test Conf.*, pp. 368–374, 1996.

[14] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Sequential circuit test generation using dynamic state traversal," *Proc. European Design and Test Conf.*, pp. 22–28, 1997.

[15] M. Keim, N. Drechsler, and B. Becker, "Combining GAs and symbolic methods for high quality tests of sequential circuits," *ASP Design Automation Conf.*, Jan. 1999.

[16] F. Corno, J. H. Patel, E. M. Rudnick, M. Sonza Reorda, and R. Vietti, "Enhancing topological ATPG with high-level information and symbolic techniques," *Proc. Int. Conf. Computer Design*, Oct. 1998.

[17] D. D. Gajski, N. D. Dutt, and A. C-H. Wu, *High-Level Synthesis, Introduction to Chip and System Design.* Kluwer Academic Publishers, 1992.

[18] D. Corvino, I. Epicoco, F. Ferrandi, F. Fummi, and D. Sciuto. "Automatic VHDL restructuring for RTL synthesis optimization and testability improvement," *Proc. IEEE Int. Conf. Computer Design*, pp. 587–596, 1998.

[19] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, and G. Squillero, "A new aproach for initialization sequences computation for synchronous sequential circuits," *Proc. IEEE Int. Conf. Computer Design*, pp. 381–386, 1997.

[20] P. Goel, "An implicit enumeration algorithm to generate tests for combinationa l logic circuits," *IEEE Trans. Computers*, vol. C-30, no. 3, pp. 215–222, March 1981.

[21] F. Somenzi. *CUDD: CU decision diagram package, version 2.2.0.* Department of Electrical and Computer Engineering, University of Colorado at Boulder, 1998.