JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 16, 703-717 (2000)

Compact Test Generation Using a Frozen Clock Testing Strategy*

ELIZABETH M. RUDNICK AND MIRON ABRAMOVICI+
Center for Reliable and High-Performance Computing
University of Illinois
Urbana, IL, U.S.A.
+Bell Labs
Lucent Technologies
Murray Hill, NJ, U.S.A.

Test application time is an important factor in the overall cost of VLSI chip testing. We present a new ATPG approach to generating compact test sequences for sequential circuits. Our approach combines a conventional ATPG algorithm, a technique based on the frozen clock testing strategy, and a dynamic compaction method based on a genetic algorithm. The frozen clock strategy temporarily suspends the sequential behavior of the circuit by stopping its clock and applying several vectors to increase the number of faults detected without changing the circuit state. Results show that test sets generated using the new approach are more compact than those generated by previous approaches for many circuits.

Keywords: automatic test generation, compact test sets, frozen clock testing strategy, sequential circuit testing, test vector compaction

1. INTRODUCTION

Automatic test equipment is expensive; therefore, it is desirable to minimize the time required to test a VLSI chip. Of course, the quality of a test must be maintained while the test application time is reduced. Various approaches have been used in the past to reduce the test application time. Test set compaction is the most common approach, and both static and dynamic compaction approaches have been used. With static compaction, a test set is first generated, and attempts are then made to shorten it without reducing fault coverage. Greater reductions in test set length have been obtained using dynamic compaction [1-6]. With this approach, compaction is performed while tests are being generated.

In this work, we use the *frozen clock testing strategy* to reduce the length of the test sequences. This approach, first proposed by Abramovici et al. [7], temporarily suspends the sequential behavior of the circuit by stopping its clock and applying several combinational vectors to primary inputs (PIs) without changing the circuit state. This enables several additional faults to be detected in each clock cycle that would otherwise not be detected until much later in the test. Fault effects that may be present at several different flip-flops can be propagated to the primary outputs (POs) with additional vectors applied while the clock is frozen. We will refer to these additional vectors as *unclocked vectors*. We will

Received June 29, 1999; revised September 14, 1999; accepted March 7, 2000.
Communicated by Cheng-Wen Wu.
*This research was supported in part by DARPA under Contract DABT63-95-C-0069 and in part by Hewlett-Packard under an equipment grant.

use the term *clocked vector* to refer to a normal sequential vector, for which the clock is strobed at the end of the cycle after the vector is applied. While application of the clock could be controlled by automatic test equipment, this approach may be impractical. Alternatively, the clocking can be controlled by using a gated clock and one additional input pin on the chip. If a single clock is used on the chip, no clock skew problems are introduced with this method.

In order to use the frozen clock testing strategy effectively, a test generator that generates appropriate test sequences is needed. Our new test generator, FreezeFrame, which makes use of the HITEC deterministic test generator [8] and the PROOFS fault simulator [9], is one such test generator. HITEC generates test sequences for sequential circuits by targeting each fault individually, and FreezeFrame optimizes these sequences. Not all the bits in a sequence are necessarily specified, and each partially specified sequence is optimized by FreezeFrame using the procedures developed for the Squeeze dynamic compactor[5, 6]. After an optimized sequence is obtained, FreezeFrame generates unclocked vectors using a genetic algorithm (GA) and deterministic procedures to detect additional faults if possible. Finally, after a complete test set is generated, FreezeFrame performs static compaction using a new tool, called IcePick, to remove any redundant unclocked vectors.

The rest of this paper is organized as follows. The GA used to evolve test sequences and unclocked vectors is explained in section 2. Section 3 gives an overview of FreezeFrame, and details of the approach are given in section 4. Results for ISCAS89 sequential benchmark circuits are presented in section 5, and section 6 concludes the paper.

2. GENETIC ALGORITHMS

A simple GA is used to solve the test sequence and unclocked vector optimization problems. The GA contains a population of binary *strings* (also called *individuals*)[10], each of which represents a test sequence or unclocked vector. Each string has an associated *fitness*, which indicates the number of faults detected by the corresponding test sequence or unclocked vector. For test sequence compaction, the initial population is seeded with the test sequence generated by the deterministic test generator, and unspecified bits are filled randomly [5, 6]. For unclocked vector generation, the initial population may be seeded with a test vector generated using PODEM [11], and again, any unspecified bits are filled randomly. Alternatively, the entire vector may be seeded randomly. A highly fit population is evolved through several generations by *selecting* two individuals, *crossing* the two individuals, and *mutating* bits in the resulting individuals with a given probability. Tournament selection without replacement and uniform crossover [12, 13] are used. Distinct generations are evolved, and the processes of selection, crossover, and mutation are repeated until all the individuals in a new population are generated. Then the old generation is discarded. Since selection is biased towards more highly fit individuals, the fitness of the overall population is expected to improve in successive generations. However the best individual may appear in any generation, and this best individual is returned as the solution. An example illustrating the application of GAs to test generation is shown in [13, 14].

Compaction and test generation are different from other applications of GAs in that execution time is a critical factor. The GA is used repeatedly on a number of similar problems, and the time needed to evaluate the fitness of an individual can be long. Therefore, instead of using a large population and allowing the GA to run for a large number of generations to

ensure that an optimal solution is found, we use a small population of size 32 and limit the number of generations to 8 to reduce the execution time. The test sequence evolved may not be the best possible sequence, but results will show that very compact tests are indeed obtained.

3. OVERVIEW

FreezeFrame produces a test set containing sequences of clocked vectors and unclocked vectors, as illustrated in Fig. 1. Unclocked vectors are shown in an italic font, and clocked vectors are shown in a regular font. Each clocked sequential test is delineated by heavy black bars, and the unclocked vectors to be applied before a clocked vector is applied are shown to the left on the same line. After each clocked vector is applied, transitions propagate through the combinational logic of the circuit, and at the end of the clock cycle, the flip-flops are clocked to update the circuit state. However, after an unclocked vector is applied, the clock is not strobed. Therefore, no new fault effects are stored in the flip-flops. This allows these test vectors to propagate fault effects to the POs without changing the state of the circuit. Thus, fault effects present at the flip-flops after application of a previous clocked vector may be detected by a number of different unclocked vectors.

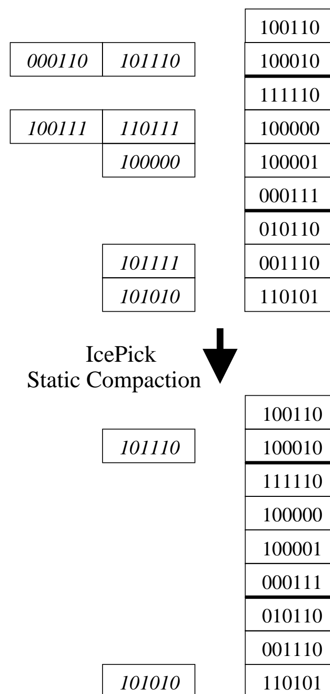

Fig. 1. Test set before and after compaction.

FreezeFrame uses HITEC [8] to produce each test sequence. HITEC is a deterministic test generator, and it targets one fault at a time by trying to activate the fault and propa-

gate the fault effects to a PO. HITEC makes several passes through the fault list, with increasing time limits per fault for successive passes. The next fault in the fault list is selected as the target fault, and test generation is attempted. If HITEC is successful, the partially specified test sequence is sent to FreezeFrame, as shown in Fig. 2. FreezeFrame uses the procedures developed for the Squeeze dynamic compactor [5, 6] to produce a fully specified test sequence. Any unnecessary vectors at the beginning and end of the sequence are first trimmed using a fault simulator. HITEC assumes the circuit starts from an unknown state for every fault targeted. However, the fault simulator may have some information about what state the circuit is in after the previous sequences have been applied. Thus, some of the initialization vectors generated by HITEC may not be necessary for detection of the target fault. Furthermore, some vectors at the end of the sequence may not be necessary if the target fault is incidentally detected by a combination of existing vectors and vectors in the new sequence. After the unnecessary vectors are removed, a GA is used to evolve the remaining vectors into a sequence that maximizes fault detection. The GA will randomly fill all unspecified bits with 1's and 0's in the first generation and optimize the sequence over a number of generations. Because the specified bits are allowed to change values, the original target fault may no longer be detected by the optimized sequence. However, it is likely that the fault will be detected in a later pass through the fault list if it is not covered by a sequence generated for a different target fault.

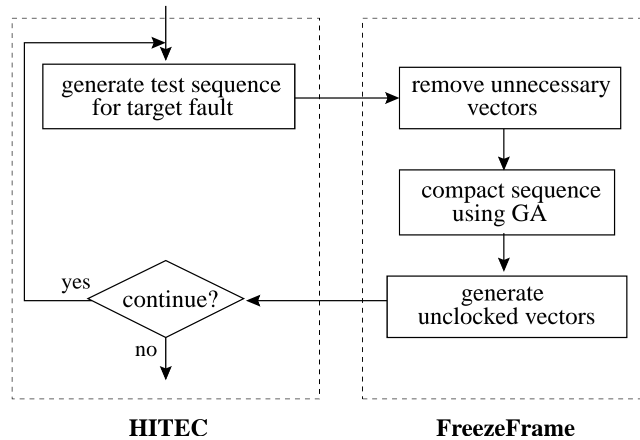

Fig. 2. Test generation with dynamic compaction.

Once a fully specified sequence is obtained, unclocked vectors are generated to propagate as many fault effects on the state lines to the POs as possible. For each clocked vector, each state line is considered individually, and an attempt is made to generate an unclocked vector that will propagate any fault effects on that state line. Then faults within the combinational logic that were not activated by the previous clocked vector are targeted, and additional unclocked vectors are generated if possible. Unclocked vectors are applied when the clock is frozen, taking advantage of the internal state from the previous clocked vector to detect additional faults. Faults may be detected by later test vectors specifically generated to detect them, but if the unclocked vectors are used, they can be detected sooner with fewer vectors. The earlier the fault is detected, the shorter the test generation time will be.

Nevertheless, some of the unclocked vectors generated are redundant and can be removed, as shown in Fig. 1. The redundant vectors detect faults that are detected by later clocked vectors or unclocked vectors. After test generation is completed, the IcePick tool performs fault simulation and set covering to generate the final optimized test set.

4. FREEZEFRAME

Fig. 3 illustrates the basic idea behind testing using a frozen clock strategy. Note that state lines that contain fault effects are shown on the left with a value 1/0 or 0/1. In this example, if the clocked test vector is applied by itself (Fig. 3a), then only one fault effect can be propagated to the PO. At this point, the circuit is clocked, and the state lines are changed; thus, the opportunity to detect the 0/1 fault effect is lost. If unclocked vectors can be applied to the PIs before the clocked test vector is applied (Fig. 3b and 3c), two additional fault effects can be propagated to the POs. In this way, the fault effects propagated to state lines are observed and faults are detected earlier.

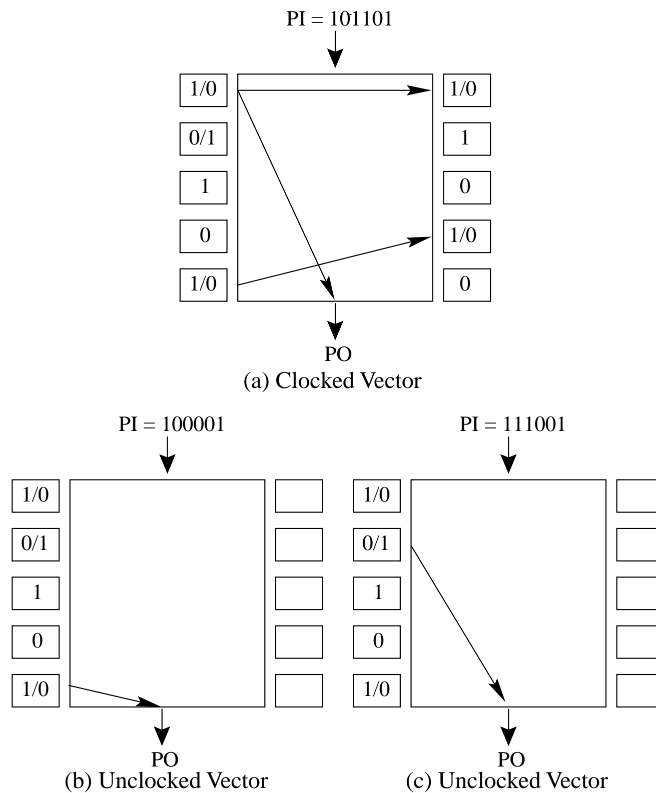

Fig. 3. Unclocked vectors propagate fault effects to POs.

Aside from the activated faults, some faults may not have been activated by the last clocked vector applied, but the current fault-free circuit state may be required to detect

them; i.e., the fault-free state provides a sensitized path for detection of some faults. These faults are also targeted by the unclocked vectors. In the following subsections, test generation and static compaction of unclocked vectors will be described.

4.1 Test Generation

After each clocked test vector sequence is generated by HITEC and compacted by the Squeeze algorithms, the faults detected by that sequence are eliminated from the collapsed list of undetected faults. The modified fault list is then used to generate unclocked vectors for every vector in the original clocked sequence. The generation of unclocked vectors is outlined in Fig. 4. The purpose of generating unclocked vectors is to exploit the state reached after the previous clocked vectors are applied. The most obvious faults that should be targeted are those whose effects have been propagated to the flip-flops in the previous time frame, i.e., activated faults. In addition, there may be faults in the combinational logic which have not been activated but which can be detected in the current time frame, possibly due to specific values being set on the state lines. Methods to generate unclocked vectors for these two types of faults will be described next.

```
For each test sequence s Do
  Compact and optimize s;
  Drop faults detected by s from fault list;
  For each vector v in s Do
    /* Part A: PODEM for activated faults */
    For each state line f with fault effects Do
      Reset GA;
      Run PODEM to produce unclocked
      vector seed;
      If successful Then
        Run GA;
        If the most-fit unclocked vector, ω,
        detects at least 1 fault Then
          Add ω to the test set;
          Drop faults detected by ω;
    /* Part B: PODEM for unactivated faults */
    Identify candidate unactivated faults using
    critical path tracing;
    For each candidate fault g Do
      Reset GA;
      Run PODEM to produce unclocked vector seed;
      If successful Then
        Run GA;
        If the most-fit unclocked vector, ω,
        detects at least 1 fault Then
          Add ω to the test set;
          Drop faults detected by ω;
    /* Part C: Random GA seeds */
    Success ← true;
    While Success
      Seed GA randomly;
      Run GA;
      If the most-fit unclocked vector, ω,
      detects at least 1 fault Then
        Add ω to the test set;
        Drop faults detected by ω;
      Else
        Success ← false;
```

Fig. 4. Unclocked vector generation.

Activated Faults The effects of a fault may propagate to one or more state lines in a given time frame. The fault may then be detected in the next time frame. It is possible for the effects of several faults to propagate to the same state line, and the effects of a single fault may propagate to multiple state lines. To simplify unclocked vector generation, the faults themselves are not considered individually; rather, each state line that holds fault effects is considered, as indicated in Part A of Fig. 4. A deterministic algorithm is used to generate an unclocked vector to propagate fault effects from a given state line to the POs. However, the fault effects may be due to a single fault only, and fault effects for this fault may have propagated to another state line. The fault effects on the second state line could mask the fault effects from the first state line so that the fault is not detected. Therefore, we use a fault simulator combined with a GA to guarantee that the unclocked vector generated does in fact detect at least one fault. The purpose of the GA is to try to detect other faults in addition to the faults whose effects have propagated to the targeted state line.

To facilitate the generation of unclocked vectors, a list of state lines that contain fault effects is built. Only state lines having known fault-free values are considered, and each flip-flop in the list is processed individually. Nine-valued logic is used to specify completely the exact value of each node at each step. For each state line in the list, FreezeFrame uses X-path check [15] to determine the feasibility of generating an unclocked vector. If X-path check does not fail, the PODEM algorithm [11] is used in an attempt to generate a partially specified vector that propagates fault effects from a particular flip-flop to the POs. A 0.5-second-per-fault time limit is used in PODEM for efficiency.

Next, a GA is used to generate a fully specified unclocked vector. The partially specified vector generated by PODEM is used as a seed for the initial GA population. In order for the GA to work effectively, a variety of bit patterns is needed for the unspecified bits in the unclocked vector. Therefore, each unspecified bit in each copy of the unclocked vector is filled at random with a 1 or 0. The fitness value for an unclocked vector was chosen as the number of faults detected by the vector for a given fault sample. A small fault sample is used to speed up the computation of the fitness value since the entire fault list may be very large. Faults associated with the fault effects on the targeted state line must be included in the fault sample. A sample of additional faults is also included. To minimize the execution time, the GA is evolved beyond the first generation only if the best individual detects at least one fault. After the population has evolved over 8 generations, the vector with the highest fitness value is added to the existing unclocked vectors for the particular clocked vector if it detects any additional faults, and the detected faults are dropped from the fault list. Unclocked vector generation continues for the next state line that contains fault effects. After all the state lines that contain fault effects are processed, attempts are made to generate unclocked vectors for unactivated faults. Then the clocked vector is applied, which propagates a new set of fault effects to the state lines, and the unclocked vector generation process begins again. Once all the clocked vectors in the sequence have been processed in this manner, HITEC is called to generate a new clocked sequence to detect a fault in the remaining fault list.

Unactivated Faults For faults which were not activated in the previous time frame, the current state may still be useful for fault excitation or fault effect propagation. After generation of unclocked vectors to detect activated faults is attempted, the unactivated faults are targeted by FreezeFrame, as indicated in Fig. 4. Deterministic algorithms are first used to specifically target faults that may be detectable (Part B in Fig. 4). Finally, a simulation-based approach is used to target the faults (Part C in Fig. 4.)

The deterministic algorithm begins with critical path tracing [15] from the primary outputs in an attempt to identify faults that may be detected in the current time frame if appropriate inputs are applied for fault excitation and fault effect propagation. Next, PODEM is used to generate an unclocked vector for each fault in the target fault list. PODEM may fail due to conflicting requirements, but if it is successful, the vector generated is used as a seed in the GA in an attempt to extend the vector to detect additional faults.

Critical path tracing is first used to find faults that are on a sensitized path, based on values specified in the current state, but which have not yet been detected. Thus, backtracing is done from primary outputs having known binary values and proceeds along (potentially) sensitized paths having known values. When a fanout branch is reached, processing continues on the fanout stem. For gates along the path having specified values on some inputs and unspecified values on other inputs, the detectable faults on the specified inputs are added to the fault list if they have not yet been detected. If all the specified inputs have the noncontrolling value *v*, the $s-a-\overline{v}$ faults on all the specified inputs are added. If only one specified input has the controlling value ω, then only the $s-a-\overline{\omega}$ fault on that input is added. If multiple inputs have the controlling value ω, then no faults are added. Consider the circuit shown in Fig. 5. The stuck-at-0 (s-a-0) fault on node *c* is detectable if node *d* can be set to the noncontrolling value. This fault is, therefore, added to the target fault list if it has not yet been detected, and backtracing continues along sensitized paths with specified values until the state lines are reached. This procedure is repeated for all other primary outputs having known binary values. A flag is set for each gate traversed to prevent the gates from being processed repeatedly for different primary outputs. Furthermore, since FreezeFrame uses a collapsed fault list, only the representative fault for a set of equivalent faults is targeted.

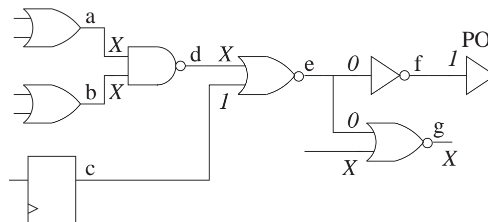

Fig. 5. Critical path tracing from nodes with binary values.

Next, faults that may be on a sensitized path, if appropriate values are applied to the primary inputs, are targeted. Thus, backtracing starts from primary outputs having unknown logic values, proceeding along paths with unknown values at gate outputs until the primary inputs are reached. Consider the example circuit with logic values shown in Fig. 6. The following target faults are identified using critical path tracing beginning from the primary output: node *f* s-a-0 and s-a-1, node *e* s-a-0 and s-a-1, node *d* s-a-0 and s-a-1, node *c* s-a-1, node *b* s-a-0 and s-a-1, and node *a* s-a-0 and s-a-1. Note that if a node has a known logic value *v*, then the $s-a-\overline{v}$ fault is targeted. Again, only undetected faults are added to the target fault list, a flag is set for each gate traversed to prevent gates from being traversed multiple times, and only the representative fault for a set of equivalent faults is targeted.

Any unclocked vectors that are generated by PODEM are simply used as seeds for the GA. Therefore, the targeted faults must be included in the fault sample to guarantee that the fitness value will be at least 1, and that the vector will be added to the test set. Thus, faults identified using critical path tracing which are not included in the fault sample are removed from the target fault list.

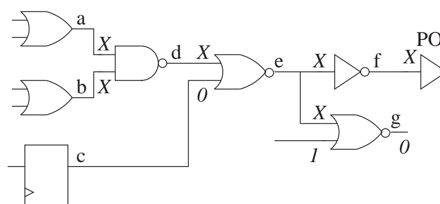

Fig. 6. Critical path tracing from nodes with unknown values.

Seeding the GA randomly, as indicated in Part C of Fig. 4, is often sufficient for detection of faults. Therefore, the final phase of unclocked vector generation uses random seeds to target faults. Test generation terminates as soon as a vector is generated that does not detect any faults.

Fault Sampling The execution time can become very high if each candidate unclocked vector is evaluated using the entire list of undetected faults that remain. For efficiency, fault samples are needed. All faults that will be targeted deterministically must be included in the fault sample. Rather than obtaining separate fault samples for each clocked vector in a sequence, one fault sample is selected per sequence. Any faults that are detected are dropped, and faults that cannot be detected in a single clock cycle are excluded. This includes faults that are determined by HITEC to be untestable and those on nodes that are not in the input cone of logic of a primary output. If the total number of faults in the sample is less than 500 after activated faults are added, then additional faults are added randomly. Again, only faults that can be detected within one clock cycle are included. A flag is set for each fault included in the fault sample so that inclusion in the fault sample can be checked very quickly.

4.2 Static Compaction

Unclocked vectors are generated without knowledge of the test sequences that are produced afterward. Therefore, there may be redundant unclocked vectors in the test set, and test compaction is needed. An unclocked vector will become redundant if all of the faults it propagates to POs are incidentally detected by other necessary clocked vectors.

Static compaction for unclocked vectors is done using a new tool called IcePick, as shown in Fig. 1. The first step in compacting the unclocked vectors eliminates unclocked vectors that detect only faults covered by clocked vectors. From the remaining unclocked vectors, the smallest set that covers the maximum number of undetected faults is selected. In general, the set cover problem is NP-complete. In practice, a greedy cover algorithm is often used that effectively approximates the NP-complete solution. Fault simulation is performed to construct a list of faults detected by each unclocked vector, and all essential

unclocked vectors are chosen first. An essential unclocked vector detects a fault that no other unclocked vector detects. Those faults detected by the essential vectors are then dropped from the fault list. Finally, a greedy algorithm is employed to select a subset of the remaining vectors. The unclocked vector that covers the most undetected faults is chosen first, and those faults are dropped. The cycle is repeated by choosing an unclocked vector that detects the most undetected faults until there are no unclocked vectors remaining that cover any undetected faults. In general, essential selection followed by greedy selection yields a very good approximation of the absolute smallest set cover [16].

5. RESULTS

FreezeFrame was implemented, and experiments were carried out on an HP 9000 J200 with 256 MB RAM. Faults detected by the unclocked vectors at the beginning of the test set are usually easy-to-detect faults that will also be detected by the later clocked vectors. Even if many unclocked vectors are added near the beginning of the test set, most will be eliminated after static compaction is performed. Therefore, unclocked vectors are not generated for the first test sequence obtained from HITEC/Squeeze. When FreezeFrame reaches test sequences later in the test set, fewer unclocked vectors will be generated because there will be fewer propagated fault effects due to the reduced fault list and because most of the faults remaining will be hard-to-detect faults. In order to achieve efficiency in both space and time, we allowed only 8 unclocked vectors to be added for each clocked vector. The starting population size for the GA was 32, and the test generation ran in 3 passes through the fault list. The HITEC time limits for the three passes were 0.5, 5, and 50 seconds per fault.

Results for FreezeFrame are shown in Table 1 and compared to those obtained using Squeeze for ISCAS89 sequential benchmark circuits that are initializable using 3-valued logic. FreezeFrame was run both with and without PODEM to target fault effects on state lines. When PODEM was not used, the GA was simply seeded randomly and run repeatedly to target both activated and unactivated faults until no more faults were detected; i.e., only Part C of Fig. 4 was executed. When PODEM was used, the unactivated faults were not targeted deterministically. Rather, a GA seeded randomly was used to target the unactivated faults; i.e., only Parts A and C of Fig. 4 were executed. The total number of faults and the number of untestable faults (Unt) are shown for each circuit, along with the number of detected faults (Det), the number of vectors generated (Vec), and execution time for both Squeeze and FreezeFrame. The execution time includes the time required for HITEC, test sequence optimization, unclocked vector generation, and static compaction. The overall FreezeFrame execution time was slightly longer than that for Squeeze for some circuits due to the additional operations performed. For the largest circuit, the execution time overhead was 20%. Circuits in the table are separated into three groups. The first group includes circuits for which no unclocked vectors were generated: s298, s344, s382, s400, and s444. The second group includes circuits for which some unclocked vectors were generated, the unclocked vectors were removed by IcePick, and the final test set included only clocked vectors: s349, s526, s1423, s3271, s3384, s6669, and s35932. The last group includes circuits for which some unclocked vectors were generated, and unclocked vectors remained in the test set after static compaction with IcePick.

Table 1. FreezeFrame vs. squeeze.

	Faults		HITEC+ Squeeze [5] [6]			HITEC+ Freeze Frame					
						No PODEM			PODEM		
Circuit	Total	Unt	Det	Vec	Time	Det	Vec	Time	Det	Vec	Time
s298	308	26	265	133	16.7m	265	**119**	16.6m	265	133	16.8m
s344	342	11	**329**	**62**	3.94m	328	63	3.94m	328	63	3.95m
s382	399	9	**359**	485	38.7m	350	300	46.1m	347	298	48.1m
s400	426	17	375	454	42.5m	**380**	633	48.0m	367	335	48.4m
s444	474	24	**418**	598	43.0m	381	377	1.23h	412	548	1.11h
s349	350	13	334	53	3.67m	334	63	3.25m	**335**	62	2.96m
s526	555	23	359	**84**	5.05h	359	**84**	5.06h	359	**84**	5.06h
s1423	1515	14	1047	137	8.99h	1045	134	8.33h	**1059**	**135**	8.17h
s3271	3270	5	**3253**	**521**	33.9m	3248	546	34.8m	3239	552	45.7m
s3384	3380	1	**3066**	118	5.29h	3043	80	5.32h	3042	95	5.36h
s6669	6684	0	6663	135	29.2m	**6671**	165	23.6m	6668	140	29.7m
s35932	39,094	3984	35,084	178	2.44h	**35,091**	180	2.93h	35,075	127	2.94h
s386	384	70	314	128	22.2s	314	**101**	28.8s	314	106	30.7s
s641	467	63	404	72	32.3s	404	69	40.7s	404	**55**	36.2s
s713	581	105	476	**56**	33.5s	476	66	51.2s	476	65	50.3s
s820	850	36	814	462	4.63m	814	446	4.99m	814	**419**	4.83m
s832	870	52	818	438	5.93m	818	505	6.50m	818	424	6.04m
s1196	1242	3	1239	219	1.69m	1239	200	2.20m	1239	**197**	2.13m
s1238	1355	72	1283	232	1.84m	1283	208	2.41m	1283	**197**	2.36m
s1488	1486	41	1444	408	17.7m	1444	435	16.6m	1444	**367**	15.5m
s1494	1506	52	1453	359	11.2m	1453	**345**	11.1m	1453	353	11.1m
s3330	2870	123	2117	258	10.4h	**2120**	296	10.4h	2113	307	10.5h
s4863	4764	22	4629	249	2.23h	4627	248	2.33h	**4630**	260	2.34h
s5378	4603	224	**3245**	198	18.2h	3239	173	18.4h	3236	181	18.5h

*Highest fault coverage or most compact test set is highlighted in bold font.

For the first and second groups of circuits, FreezeFrame performed no better than Squeeze. In fact, higher fault coverages were often obtained when Squeeze was used. This is probably due to the nondeterminism of the algorithms used and in particular, the fault order dependency of HITEC. HITEC is unable to generate tests to detect some faults when they are specifically targeted. FreezeFrame is also not able to generate tests for some of the hard-to-detect faults if the states needed for their detection are never reached. For the third group of circuits, FreezeFrame with PODEM outperformed Squeeze in nine out of twelve

circuits. FreezeFrame without PODEM was better than Squeeze for seven of twelve circuits, and Squeeze performed best for just two circuits. These results show that FreezeFrame is indeed effective in reducing the test application time, but faults must be targeted deterministically to obtain the most compact test sets.

Additional information about the test vectors generated for circuits in the third group is provided in Table 2, including the number of clocked (Clk) vectors, the original number of unclocked (Unclk) vectors, the number of compacted unclocked (Comp) vectors, and the total number of vectors after compaction. For a few circuits, the unclocked vectors make up a significant fraction of the total vectors, but in most cases, very few unclocked vectors remain in the compacted test set.

Table 2. Unclocked vector generation and compaction in FreezeFrame.

Circuit	No PODEM					PODEM				
	Det	Vectors				Det	Vectors			
		Clk	Unclk	Comp	Total		Clk	Unclk	Comp	Total
s386	314	89	31	12	101	314	95	28	11	106
s641	404	53	27	16	69	404	40	25	15	55
s713	476	48	33	18	66	476	51	29	14	65
s820	814	443	38	3	446	814	413	45	6	419
s832	818	500	39	5	505	818	418	37	6	424
s1196	1239	106	124	94	200	1239	90	129	107	197
s1238	1283	112	123	96	208	1283	89	138	108	197
s1488	1444	428	49	7	435	1444	356	54	11	367
s1494	1453	336	51	9	345	1453	344	53	9	353
s3330	2120	294	22	2	296	2113	306	19	1	307
s4863	4627	239	46	9	248	4630	250	47	10	260
s5378	3239	159	53	14	173	3236	161	56	20	181

Clk: Number of clocked vectors **Unclk:** Original number of unclocked vectors
Comp: Number of compacted unclocked vectors

Additional experiments were performed to evaluate various strategies for targeting unactivated faults. Detailed results of the experiments are available in [17]. For all the experiments, the activated faults were targeted deterministically according to Part A of Fig. 4. In the first experiment, unactivated faults were targeted only randomly by using random seeds in the GA. Results for this experiment are shown in Table 1. In the second experiment, unactivated faults were targeted deterministically using critical path tracing first, followed by a random phase in which the GA was seeded randomly. Parts A, B, and C of Fig. 4 were carried out in sequence for this case. In the third experiment, random seeding of the GA was performed before the unactivated faults were targeted deterministically; i.e., Part C of Fig. 4 was executed before Part B.

When unactivated faults are targeted deterministically, the fault coverage improves slightly for several circuits. In particular, using the deterministic algorithm for targeting unactivated faults before seeding the GA randomly gives the best results. Furthermore, improvement in the test set size is often obtained. However, this improvement comes at the cost of additional execution time required for the extra operations performed. For the largest circuit, the execution time overhead was 40%, and the fault coverage was also lower.

Table 3. Comparison with previous approaches.

	SEQCOM [1]		COINS(*) [2]		[3]		BRF (2) [4]		Freeze Frame	
Circuit	Det	Vec	Det	Vec	Det	Vec	Det	Vec	Det	Vec
s386	314	131	314	154	306	380	–	–	314	106
s641	404	80	404	136	401	121	404	74	404	55
s713	–	–	–	–	467	142	476	71	476	65
s820	–	–	722	235	–	–	409	87	814	419
s832	–	–	–	–	–	–	494	101	818	424
s1196	1232	238	1235	307	1239	332	1239	228	1239	197
s1238	–	–	–	–	1283	348	1283	244	1283	197
s1423	–	–	–	–	–	–	1298	397	1059	135
s1488	1444	358	1444	566	–	–	1182	88	1444	367
s1494	–	–	–	–	–	–	1075	84	1453	353
s5378	–	–	–	–	–	–	3368	294	3236	181

FreezeFrame is compared to previous approaches in Table 3. The results shown for FreezeFrame were obtained using only random seeds in the GA to target unactivated faults. FreezeFrame's shorter test set achieves comparable or higher fault coverage for most of the circuits studied.

6. CONCLUSIONS

Since test application time is an important factor in reducing the VLSI chip testing cost, shorter test sets are preferred. At the same time, high fault coverage should be maintained. The traditional approach to producing shorter test sets is to use compaction. In our work, we developed a package called FreezeFrame that uses an approach introduced in [7], which involves freezing the clock. In FreezeFrame, we use the HITEC test generator and Squeeze dynamic compaction procedures to produce the initial test set, use a new algorithm to generate unclocked vectors to be applied while the clock is frozen, and IcePick to statically compact the unclocked vectors. FreezeFrame produces more compact test sets than other approaches yet maintains a comparable fault coverage for several of the ISCAS89 sequential benchmark circuits.

ACKNOWLEDGMENT

The authors would like to thank Yanti Santoso, Matthew Merten, and Xiaoming Yu for assisting with the implementation.

REFERENCES

1. I. Pomeranz and S. M. Reddy, "On generating compact test sequences for synchronous sequential circuits," in *Proceedings of European Design Automation Conference* (*EURO-DAC*), 1995, pp. 105-110.
2. I. Pomeranz and S. M. Reddy, "Dynamic test compaction for synchronous sequential circuits using static compaction techniques," in *Proceedings of International Symposium on Fault-Tolerant Computing*, 1996, pp. 53-61.
3. A. Raghunathan and S. T. Chakradhar, "Dynamic test sequence compaction for sequential circuits," in *Proceedings of International Conference on VLSI Design*, 1996, pp. 170-173.
4. T. J. Lambert and K. K. Saluja, "Methods for dynamic test vector compaction in sequential test generation," in *Proceedings of International Conference on VLSI Design*, 1996, pp. 166-169.
5. E. M. Rudnick and J. H. Patel, "Putting the squeeze on test sequences," in *Proceedings of International Test Conference*, 1997, pp. 723-732.
6. E. M. Rudnick and J. H. Patel, "Efficient techniques for dynamic test sequence compaction," *IEEE Transactions on Computers*, Vol. 48, No. 3, 1999, pp. 323-330.
7. M. Abramovici, K. B. Rajan, and D. T. Miller, "FREEZE!: A new approach for testing sequential circuits," in *Proceedings of Design Automation Conference*, 1992, pp. 22-25.
8. T. M. Niermann and J. H. Patel "HITEC: A test generation package for sequential circuits," in *Proceedings of European Conference on Design Automation* (*EDAC*), 1991, pp. 214-218.
9. T. M. Niermann, W. -T. Cheng, and J. H. Patel, "PROOFS: A fast, memory-efficient sequential circuit fault simulator," *IEEE Transactions on Computer-Aided Design*, Vol. 11, No. 2, 1992, pp. 198-207.
10. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addison-Wesley, 1989.
11. P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Transactions on Computers*, Vol. C-30, No. 3, 1981, pp. 215-222.
12. E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, "A genetic algorithm framework for test generation," *IEEE Transactions on Computer-Aided Design*, Vol. 16, No. 9, 1997, pp. 1034-1044.
13. P. Mazumder and E. M. Rudnick, *Genetic Algorithms for VLSI Design, Layout, and Test Automation*, Prentice Hall, 1999.
14. Genetic Algorithm Example, www.crhc.uiuc.edu/IGATE/.
15. M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, The Institute of Electrical and Electronics Engineers, Inc., New York, 1990.

16. R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, 1984.
17. Y. I. Santoso, "Compact test generation based on a frozen clock testing strategy," M. S. thesis, Dept. of Electrical and Computer Engineering, University of Illinois, Urbana, IL, 1999.

Elizabeth M. Rudnick is an assistant professor in the Department of Electrical and Computer Engineering and a research assistant professor in the Coordinated Science Laboratory at the University of Illinois. Her research interests include test generation, design for testability, and design verification. She received the BS degree in chemical engineering and the MS and PhD degrees in electrical engineering from the University of Illinois in 1983, 1990, and 1994, respectively. She has worked at Motorola, Sunrise Test Systems, and Advanced Micro Devices in the areas of design verification, test generation, electronic design automation, and yield enhancement. She is a member of the IEEE and a coauthor of "Genetic Algorithms for VLSI Design, Layout, and Test Automation" (Prentice Hall, 1999).

Miron Abramovici is a Distinguished Member of the Technical Staff in the Design Research Principles Dept. at Bell Labs in Murray Hill, NJ, where he has worked in the areas of testing, diagnosis, reliable design, synthesis, design verification, hardware accelerators, and reconfigurable computing. He has been with Bell Labs since 1980. From 1971 to 1976 he was with the Department of Applied Mathematics of the Weizmann Institute of Science, Rehovot, Israel. He received the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, in 1980.

Dr. Abramovici is a Fellow of IEEE. He received the Bell Labs Distinguished Technical Staff Award and two AT&T Exceptional Contribution Awards. He served on the editorial board of IEEE Design and Test of Computers and is now on the editorial board of Journal of Electronic Testing. He was an adjunct Professor of Computer Engineering at the Illinois Institute of Technology in Chicago. He is a coauthor of "Digital Systems Testing and Testable Designs" (Computer Science Press, 1990 and IEEE Press, 1994), adopted world-wide as the standard text in testing.