14th IEEE VLSI Test Symposium, April 1996, pp. 216-223

Automatic Test Generation Using Genetically-Engineered Distinguishing Sequences *

Michael S. Hsiao, Elizabeth M. Rudnick, and Janak H. Patel
Center for Reliable and High-Performance Computing
University of Illinois, Urbana, IL 61801

Abstract

A fault-oriented sequential circuit test generator is described in which various types of distinguishing sequences are derived, both statically and dynamically, to aid the test generation process. A two-phase algorithm is used during test generation. The first phase activates the target fault, and the second phase propagates the fault effects (FE's) from the flip-flops with assistance from the distinguishing sequences. This strategy improves the propagation of FE's to the primary outputs, and the overall fault coverage is greatly increased. In our new test generator, DIGATE, genetic algorithms are used to derive both activating and distinguishing sequences during test generation. Our results show very high fault coverages for the ISCAS89 sequential benchmark circuits and several synthesized circuits.

I Introduction

The task of test generation is to find a sequence which is capable of distinguishing the fault-free machine from the faulty machine resulting from the presence of a fault. Deterministic test generators for sequential circuits attempt to do this but are prone to large numbers of backtracks and complex scheduling algorithms [1-11]. Simulation-based test generators, on the other hand, avoid the complexity of backtracking altogether by processing in the forward direction only. Several novel approaches to test generation using genetic algorithms (GA's) have been proposed in the past [12-19]. Fitness functions were used to guide the GA to find a test vector or sequence that maximizes given objectives for a single fault or group of faults. In GATEST [15], the objective of the fitness function was to maximize the number of faults detected and the number of fault effects (FE's) propagated to flip-flops, and in CRIS [12] and GATTO [16], increasing the circuit activity was a dominant objective. The objectives of propagating FE's to flip-flops and increasing circuit activity have been shown to increase the probability of detecting faults at the primary outputs (PO's). Although the fault detection probability improves, propagating a FE from a flip-flop to a PO remains a difficult problem. Without the knowledge of distinguishing sequences, propagation of FE's to the flip-flops is usually done indiscriminately, resulting in much wasted effort, since propagation of FE's from certain flip-flops may not be possible. As a result, many faults are unable to reach the PO's, yielding a lower fault coverage. This phenomenon suggests that the fitness functions used in the past do not exploit the knowledge of fault propagation. For example, the fitness function objectives fail to avoid fault propagation to hard-to-observe flip-flops. In addition, once the FE has reached a flip-flop, the fitness function does not have any specific knowledge that helps to propagate the FE to a PO. Even when a sequence is found for a given flip-flop to propagate the FE's, information about the sequence often becomes unavailable in the future when a similar situation arises again.

The presence of a fault creates a faulty machine (circuit structure) which differs from the fault-free machine. In order to detect the fault, these two machines have to be distinguished. The principal approach taken in this paper is to use distinguishing sequences in as many places as possible to reduce the work of rediscovering such sequences. However, several questions still remain. How many distinguishing sequences should be generated? By what procedure should such distinguishing sequences be generated? What types of sequences should be generated (e.g., sequences that distinguish two states, two sets of states, the fault-free machine and faulty machine, etc.)? Since any procedure for deriving distinguishing sequences is complex, we cannot indiscriminately generate many sequences. In addition, a distinguishing sequence derived for a fault-free machine may not be valid for distinguishing a faulty machine from the fault-free machine, or it may be valid for distinguishing some faulty machines but not for other faulty machines.

In this paper, we pre-generate a class of distinguishing sequences statically for the fault-free machine and also dynamically capture distinguishing sequences for the fault-free and faulty machines during the test generation process. These sequences are then used as seeds for the GA during fault propagation. If these seeds are valid for the present situation, no further processing is required. Otherwise, we genetically engineer valid sequences from the seeds. In addition to this, the difficulty of deriving a distinguishing sequence is also taken into account at run time in the computation of flip-flop observability. This measure of observability is much more accurate than the conventional observability metric and helps to guide the test generator much more effectively.

Previously, homing, synchronizing, and distinguishing

*This research was supported in part by the Semiconductor Research Corporation under contract SRC 95-DP-109, in part by ARPA under contract DABT63-95-C-0069, and by Hewlett-Packard under an equipment grant.

sequences have been used to aid the test generator in finding a test sequence [6, 9, 10, 11]. In [6, 9, 11], symbolic and state-table-based techniques were used to derive these sequences in the fault-free machine. In [6], cube intersections of ON/OFF-set representations were used to derive distinguishing sequences. Binary decision diagrams (BDD's) and implicit state enumeration were used in [9] to derive synchronizing sequences. In the work by Park et al., [11], functional information was used to pre-generate sequences which simplified propagation of FE's from the flip-flops to the PO's, and state justification was done by using BDD's. Since these sequences are generated using the fault-free machine only, they become insufficient when a faulty machine is encountered. Homing sequences composed of specifying and distinguishing portions were used to aid ATPG in [10]. Computations of the specifying and distinguishing portions of the sequence are done repeatedly for each fault; thus, no knowledge of distinguishing sequences is stored.

As in the deterministic approach, our new test generator, DIstinguishing sequences GA-based TEst generator, DIGATE, targets one fault at a time and divides the test generation process into two phases: **fault activation** and **fault propagation**. Fault activation excites the fault and propagates the FE's to a PO or at least one flip-flop. Fault propagation propagates the FE's from one or more flip-flops to a PO, possibly through several time frames. Both phases are performed using the GA framework, with appropriate distinguishing sequences seeded in the GA during the fault propagation phase. FE's may reach multiple flip-flops at the end of the activation phase, which will require the use of several distinguishing sequences in the propagation phase; therefore, the list of distinguishing sequences is pruned adaptively over time to increase the power and accuracy in distinguishing the states. When a sequence is found that successfully propagates the FE's to the PO's, a fault simulator is invoked to remove any additional faults detected by the sequence. Flip-flops which do not have distinguishing sequences are identified during the process, and propagating FE's to these hard-to-observe flip-flops is avoided. DIGATE targets all faults until little or no more improvement is made. Results of DIGATE on the ISCAS89 sequential benchmark circuits and several synthesized circuits show very high fault coverages.

The remainder of the paper is organized as follows. Section II briefly describes the genetic algorithm and simulation frameworks used in this work. Section III gives details about the DIGATE algorithm, including generation and pruning of distinguishing sequences, as well as selection of target faults and fitness evaluation. Experimental results are given in Section IV, showing the effectiveness of DIGATE, and Section V concludes the paper.

II Genetic Algorithms for DIGATE

DIGATE uses a GA framework similar to the simple GA described by Goldberg [20]. The GA contains a population of *strings*, or individuals, in which each individual represents a sequence of test vectors. The population size used is a function of the string length, which depends on both the number of primary inputs (PI's) and the test sequence length. During the first stage of DIGATE, the population size is set to $4 * \mathit{square\ root}(\mathit{sequence\ length})$ when the number of PI's is less than 16 and $16 * \mathit{square\ root}(\mathit{squence\ length})$ when the the number of PI's is greater than or equal to 16.

Each individual has an associated *fitness*, which measures the test sequence quality in terms of fault detection, distinguishing power, and other factors. The fitness function used in this work depends on the phase of test generation and will be explained in the next section. The population is initialized with random strings, and if any distinguishing sequences exist during the fault propagation phase, these sequences are used as seeds as well. A fault simulator is used to compute the fitness of each individual. Then the evolutionary processes of *selection*, *crossover*, and *mutation* are used to generate an entirely new population from the existing population. Evolution from one generation to the next is continued until a sequence is found to detect the target fault or a maximum number of generations is reached. To generate a new population from the existing one, two individuals are selected, with selection biased toward more highly fit individuals. The two individuals are crossed to create two entirely new individuals, and each character in a new string is mutated with some small mutation probability. A mutation probability of 0.01 is used in this work, and since a binary coding is used, mutation is done by simply flipping the bit. The two new individuals are then placed in the new population, and this process continues until the new generation is entirely filled. At this point, the previous generation can be discarded. In our work, we use tournament selection without replacement and uniform crossover. In *tournament selection without replacement,* two individuals are randomly chosen and removed from the population, and the best is selected; the two individuals are not replaced into the original population until all other individuals have also been removed. Thus, it takes two passes through the parent population to completely fill the new population. In *uniform crossover,* characters from the two parents are swapped with probability 1/2 at each string position in generating the two offspring. A crossover probability of 1 is used; i.e., the two parents are always crossed in generating the two offspring. Because selection is biased toward more highly fit individuals, the average fitness is expected to increase from one generation to the next. However, the best individual may appear in any generation.

Because one fault is targeted at a time and the majority of time spent by the GA is in the fitness evaluation, parallelism among the individuals can be exploited. Therefore, parallel-pattern simulation [21] is used to speed up the process. In DIGATE, 32 sequences are simulated simultaneously, with values bit-packed into 32-bit words during simulation. A fault-free simulation is run, followed by insertion of the fault, and faulty circuit evaluation, in which events start exclusively from the faulty gate.

III The DIGATE Algorithm

DIGATE is comprised of three stages; each stage involves several passes through the fault list, and a stage is finished when little or no improvement in fault coverage is achieved. Faults are targeted individually within each stage, and GA's are used to activate a fault and propagate the FE's to the PO's. Each stage has a different test sequence length for individuals in the

GA population. A multiple of the structural sequential depth of the circuit is used as the test sequence length, where the sequential depth is defined as the minimum number of flip-flops in a path between the PI's and the furthest gate. The sequence length of an individual is set equal to the sequential depth in the first stage, two times the sequential depth in the second stage, and four times the sequential depth in the third and final stage. The longer sequences may be required for hard-to-detect faults, but the time required for the fitness evaluation is directly proportional to the test sequence length. Therefore, the shorter sequences are tried first, and faults are removed from the fault list once they are detected.

Using GA's to target untestable faults is a waste of time, since untestable faults cannot be identified using our approach. Thus, the HITEC deterministic test generator [7] is used after the first GA stage in order to identify and remove many of the untestable faults. A small time limit of 0.4 seconds per fault is used in an initial HITEC pass through the fault list to minimize the execution time. If a large number of untestable faults are identified or if only a small number of faults remain in the fault list, a second HITEC pass with a time limit of 4 seconds per fault is used. Any test sequences generated by HITEC are discarded.

Within each GA stage of DIGATE, a two-phase strategy, similar to that used in deterministic test generators, is taken, and a single fault is targeted at a time. The two-phase structure is illustrated in Figure 1. Both phases use the GA to find test sequences. The first phase focuses on activating the target fault, while the second phase tries to propagate the FE's to the PO's. A target fault is selected from the fault list in the fault activation phase, and the GA is used to try to excite the fault and propagate the FE's to a PO or to the flip-flops. If the FE's have successfully propagated to one or more flip-flops, any distinguishing sequences corresponding to those flip-flops are seeded into the GA, and an attempt is made to engineer a valid distinguishing sequence for propagating the FE's to a PO in the fault propagation phase.

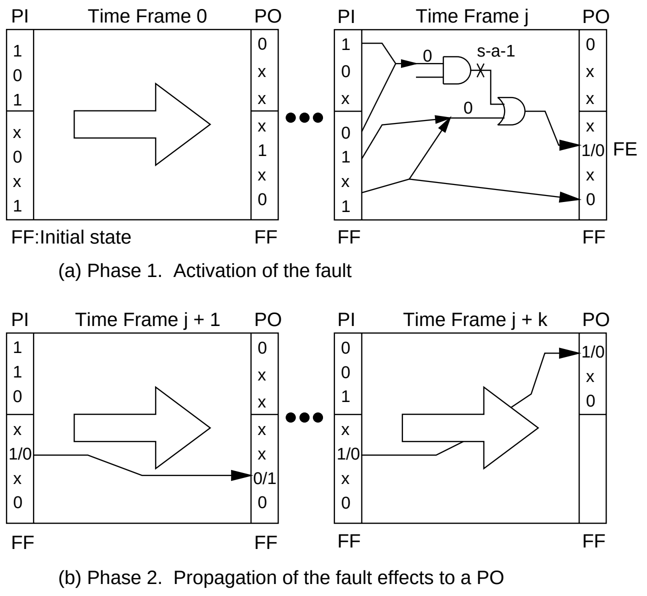
PI
Time Frame 0
PO
Time Frame j
s-a-1
FE
FF:Initial state
FF
(a) Phase 1. Activation of the fault
Time Frame j + 1
Time Frame j + k
(b) Phase 2. Propagation of the fault effects to a PO

Figure 1: Two-phase strategy.

Figure 2 illustrates the different types of distinguishing sequences. A distinguishing sequence of type A for flip-flop i is defined as a sequence which produces two distinct output responses when applied to the fault-free machine for two initial states, and the initial states differ in the i^{th} position and are independent of all other flip-flop values. A type-B distinguishing sequence for flip-flop i is a sequence which, when applied to the fault-free machine with i^{th} flip-flop = 0 (or 1) and applied to the faulty machine with the same flip-flop = 1 (or 0), produces two distinct output responses independent of the values of all other flip-flops. A type-C distinguishing sequence is similar to type B except that a subset of flip-flops are assigned to specific logic values.

The value **x** in the state denotes an unknown or more precisely a *don't care* value, and the character **S** in a state represents a string of known values (e.g., **1** or **0**). The distinguishing sequences of type A are pre-generated statically for the fault-free machine only, while sequences of types B and C are derived dynamically for both the fault-free and faulty machines during test generation. Note that the distinguishing sequences of type C may depend on a partial state of the machine, so they cannot necessarily be applied directly. With these distinguishing sequences of various types seeded, the GA is used to evolve a valid distinguishing sequence to propagate the FE's from the flip-flops to the PO's. The sequences generated in [11] are similar to the type-A distinguishing sequences, except that they were generated using BDD's; no pruning of sequences was done, and sequences of types B and C were absent. When the sequences fail to distinguish the states for specific faulty machines, no procedure was given to modify the sequences. In contrast, we use a variety of distinguishing sequences and modify them to get valid sequences for each fault. The following subsections explain our genetically-engineered distinguishing sequences in greater detail.

A Generation of distinguishing sequences

A distinguishing sequence associated with a flip-flop is guaranteed to propagate a FE from the given flip-flop to the PO's in fault-free machines. The most general case of generating such a sequence is as follows. A $\mathbf{D} = (\mathbf{1/0})$ is placed at the output of a flip-flop while all other flip-flops in the circuit are set to unknown values. If a sequence is generated that makes the **D** observable at the PO's, the sequence is a distinguishing sequence of type A for the given flip-flop in the fault-free machine. This type of sequence is able to distinguish $2^{2(N-1)}$ pairs of states in the fault-free machine, where N is the total number of flip-flops in the circuit. In most circuits, however, the number of type-A distinguishing sequences is very small; sometimes none exist at all. In addition, this type of sequence may not successfully distinguish the states in the fault-free machine from those in the faulty machine, which is required when generating a test sequence for a target fault. Fortunately, FE's are often propagated to many flip-flops before detection at a PO during the course of test generation. This gives rise to the distinguishing sequences of types B and C.

A distinguishing sequence of type B or C for a flip-flop is specific to a target faulty machine (machine resulting from presence of the target fault). Given a fault-free machine **G** and

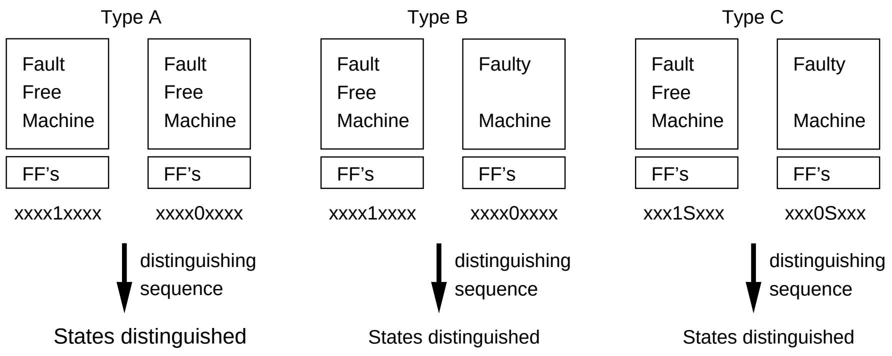

Figure 2: Types of distinguishing sequence.

a corresponding faulty machine **F**, a type-B sequence would be able to distinguish $2^{2(N-1)}$ pairs of states between machines **G** and **F**. On the other hand, a type-C sequence would distinguish only $2^{2(N-M-1)}$ pairs of states, where M denotes the number of flip-flops with specified values. It should be noted that distinguishing sequences of types A and B are more powerful than those of type C, since more pairs of states can be distinguished by the sequences of types A and B.

While distinguishing sequences of type A are capable of distinguishing two different fault-free states, they may not necessarily be able to distinguish the same two states in fault-free machine **G** and faulty machine **F**. Nevertheless, a type-A sequence may be very similar to a sequence that is able to distinguish the two states in machines **G** and **F**. Therefore, it is helpful to seed the GA with distinguishing sequences of type A in searching for a successful distinguishing sequence. Carrying this idea further, when a distinguishing sequence of type B or C is found for a certain fault **f1**, that sequence may not be directly applicable under a different fault **f2**. A similar argument applies in this case: the previously derived distinguishing sequence may be used as a seed for the GA to help find a valid sequence.

A distinguishing sequence of type C requires a subset of the flip-flops to have specific values in order to successfully propagate a **D** from the given flip-flop to a PO. Under this restriction, many flip-flops often have distinguishing sequences of type C when those of types A and B do not exist. In many cases, a type-C distinguishing sequence works as well as one of type A or B, namely, those cases in which the current state is contained within the required sets of states for the distinguishing sequence.

Storing the type-C distinguishing sequences, however, poses a problem. Including the specific values of the required flip-flops for the sequences may adversely affect both the execution time and memory storage. Furthermore, when a distinguishing sequence of type C is derived dynamically during test generation, it is difficult to identify the flip-flops which require pre-assigned values. Thus, instead of storing the values for various subsets of flip-flops, a distinguishing power is associated with each distinguishing sequence to indicate how well the sequence distinguishes two states. As a consequence, the distinguishing power also indirectly indicates how well a FE will propagate from the corresponding flip-flop to a PO. Although the state-containment information is missing for these distinguishing sequences, they are still useful as seeds for the GA to evolve an effective distinguishing sequence. The distinguishing power of every corresponding sequence is updated for each successful and unsuccessful GA application.

All three types of distinguishing sequence are generated by the GA. Before test generation begins, the GA is set in a preprocessing stage to compute any distinguishing sequences of type A. During the test generation process, derivation and pruning of distinguishing sequences of types B and C are done concurrently and adaptively. The GA is initialized with random sequences, and any distinguishing sequences of the flip-flops to which FE's have propagated are used as seeds in place of some of the random sequences in the fault propagation phase. Since small population sizes are used for the GA in order to reduce execution time, the number of distinguishing sequences stored per flip-flop is limited to five.

For flip-flops that do not have an associated distinguishing sequence of any type, an observability value is used to indicate how observable the flip-flop is in the GA framework. Initially, all flip-flops in the circuit are set to a certain observability value. As time progresses, these observabilities for the flip-flops will only decrease if no distinguishing sequence can be obtained for them. The lower the observability value, the harder it is to generate a distinguishing sequence for that flip-flop. This measure of observability is much more accurate than conventional observability values, and it enables propagation of FE's to hard-to-observe flip-flops to be avoided during test generation. Pseudo-code for the two-phase test generation algorithm is shown in Figure 3.

B Pruning the distinguishing sequences

The distinguishing power associated with a distinguishing sequence should indicate how well the sequence can propagate a FE from the corresponding flip-flop to the PO's. By definition, a smaller subset of specified flip-flops necessary for a distinguishing sequence to propagate the FE indicates a higher distinguishing power. Furthermore, a distinguishing sequence capable of propagating a FE under a different faulty machine would be given a higher power than a sequence which is not.

This concept is incorporated into DIGATE. When a distinguishing sequence S_0 is obtained, a minimal distinguishing power is given to the sequence. When another FE reaches the same flip-flop at a later time, S_0 is applied again to guide the GA. If a sequence is found and is the same as S_0, the distinguishing power for S_0 is incremented. If the sequence found differs from S_0, an additional sequence S_1 is added to the set of distinguishing sequences for the flip-flop, and distinguishing

```
Generate for each flip-flop a dist. sequence of type A
For all undetected faults do
    Pick a target fault
    /* fault activation phase */
    Call GA to generate a sequence that activates the
        target fault
    Drop all other faults detected by the sequence
    /* fault propagation phase */
    If target fault not detected then
        Initialize GA with random sequences
        Seed GA with the dist. sequences corresponding
            to flip-flops with the FE's
        Call GA to propagate the FE's to a PO
        If sequence $S_1$ derived by GA is successful then
            If dist. sequence $S_0$ already exists for the FF then
                Add $S_1$ to the table if different from $S_0$
                Increment the dist. power for $S_0$ (and $S_1$)
            Else
                Add the dist. sequence $S_1$ to table, giving
                    it minimal dist. power
            Drop all other faults detected by the sequence
        Else /* dist. seq. unsuccessful */
            If dist. sequences exist for the FF then
                Decrement the dist. power of the sequences
            Else
                Decrement the observability of the FF
```

Figure 3: Test generation algorithm

powers for both S_0 and S_1 are incremented. On the other hand, if no sequence is found, the distinguishing power of S_0 is decremented. If the distinguishing power drops below the minimal value, S_0 is removed from future consideration.

When starting with a set of flip-flops where none has a distinguishing sequence, the GA is initialized with random seeds. If a distinguishing sequence is derived, this derived sequence becomes a candidate distinguishing sequence for every flip-flop in the set. It should be noted that this sequence may be suitable for certain flip-flops only, while unfit for the rest of the flip-flops in the set, but as the distinguishing sequences are further pruned, the unfit ones are eventually weeded out.

C Fitness functions

Since the two phases of DIGATE target different goals, their corresponding fitness functions will differ. The parameters that affect the fitness of an individual in the GA are as follows:

P1: Fault detection by the individual
P2: Sum of distinguishing powers for the distinguishing sequences of the flip-flops with FE's
P3: Sum of observabilities for the flip-flops with FE's
P4: Weighted faulty circuit activity induced
P5: No. of hard-to-control flip-flops set to specific values
P6: No. of new states visited by the individual

Parameter *P1* is self-explanatory, in particular during the fault propagation phase. It is included in the activation phase to cover faults that propagate directly to the PO's in the time frame in which they are excited. *P2* measures the quality of the set of flip-flops reached by the FE's. Maximizing *P2* increases the probability that the FE's reach flip-flops having more powerful distinguishing sequences. If none or few flip-flops have distinguishing sequences, maximizing *P3* will avoid propagating FE's to the hard-to-observe flip-flops for which obtaining distinguishing sequences is difficult. *P4* measures the number of weighted events generated in the faulty circuit by the sequence. Partial cones are computed and set up for the PO's and the flip-flops associated with either more powerful distinguishing sequences or low observability values. Figure 4 illustrates the setup of partial cones; each partial cone has

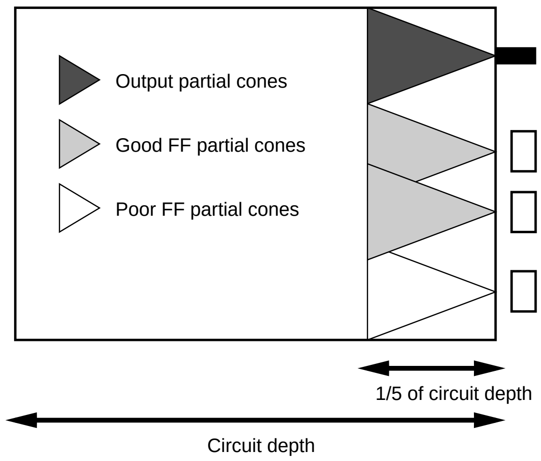

Figure 4: Output and flip-flop partial cones.

a depth of one-fifth the circuit's depth. Events are weighted more heavily if they are inside partial cones of the PO's or flip-flops with more powerful distinguishing sequences; events inside the partial cones of the hard-to-observe flip-flops are weighted more lightly. Events on any nodes outside these good and poor FF partial cones are assigned weights of one. The partial cones are recomputed at the beginning of each of the three GA stages in order to include cones of flip-flops for which new distinguishing sequences have been obtained or observability values have decreased. *P5* is used to bias the search toward finding sequences that can set the hard-to-control flip-flops. For instance, a sequence that sets a flip-flop with a high 0-controllability value (difficult to set the flip-flop to 0) to 0 is given a greater fitness value. The 0 and 1-controllability values are computed a priori by HITEC [7]. By maximizing *P5*, the GA is likely to bring the circuit to previously unexplored state spaces. *P6* is used also to expand the search space. Pomeranz [22] and Marchok [23] have suggested that visiting as many different states as possible helps to detect more faults. The fitness functions in DIGATE favor visiting more states when the fault detection count drops very low. Thus, *P6* is considered in the final stage of DIGATE only.

Different weights are given to each parameter in the fitness computation during the two phases of DIGATE:

Fault activation phase:
$$fitness = 0.2P1 + 0.7(P2 + P3) + 0.1(P4 + P5 + P6^{\dagger})$$
Fault propagation phase:
$$fitness = 0.8P1 + 0.1(P2 + P3) + 0.1(P4 + P5 + P6^{\dagger})$$
†: evaluated only in the final GA stage

In the fault activation phase, the aim is to excite the fault and propagate the FE's to as many good flip-flops as possible with short sequences and minimal time, where good flip-flops are those with more powerful distinguishing sequences. In the fault propagation phase, the goal is to find a sequence that will propagate the FE's to a PO, so a heavier weight is given to fault detection.

D Selection of the target fault

In the traditional deterministic approach to test generation, the target fault selected is the first fault among the remaining undetected faults, since during state justification, the state has to be backtraced to an entirely unknown state using reverse time processing. In DIGATE, however, since only forward propagation is involved, the subsequent fault to be targeted can be selected more intelligently.

In [16], choosing the target fault is done iteratively by maximizing an evaluation function which indicates the amount of circuit activity generated by a particular fault. Instead of measuring circuit activity, DIGATE selects the fault that has its FE's propagated to a flip-flop having a distinguishing sequence of maximum distinguishing power. The activation phase of DIGATE is omitted by the selection of this target fault, since the effects of the targeted fault have already reached at least one flip-flop. Thus, DIGATE can enter the propagation phase immediately. The target fault selected in this manner is likely to have a much higher probability of detection than a randomly selected fault.

If no fault has reached any flip-flop having a distinguishing sequence, the selection of the target fault is biased toward the fault that has reached the greatest number of flip-flops. However, the activation phase of DIGATE is not omitted in this case.

IV Experimental Results

DIGATE was implemented in C++, and experiments were conducted on the ISCAS89 sequential benchmark circuits [24], as well as several synthesized circuits, to evaluate its performance. All circuits were evaluated on an HP 9000 J200 with 256 MB RAM. Table 1 displays the characteristics of the synthesized circuits. Sequential depth (structural), number

Table 1: Characteristics of Synthesized Circuits

Circuit	Seq. Depth	FF's	PI's	PO's	Faults
am2910	4	87	20	16	2391
mult16	9	55	18	33	1708
div16	19	50	33	34	2147
pcont2	3	24	9	8	11300
piir8o	5	56	9	8	19920
piir8	5	56	9	8	29689

of flip-flops, number of PI's, number of PO's, and the total number of collapsed faults for each circuit are given in the table. The functions of these circuits are as follows: am2910 is a 12-bit microprogram sequencer [25]; mult16 is a 16-bit two's complement multiplier using a shift-and-add algorithm; div16 is a 16-bit divider using repeated subtraction to perform division; pcont2 is an 8-bit parallel controller for DSP applications; and both piir8o and piir8 are 8-point infinite impulse response DSP filters.

Results are given in Table 2 for DIGATE and various other test generators. For each circuit, the total number of collapsed faults is given, followed by the number of faults detected and the test set length for each test generator. The number of distinguishing sequences generated by DIGATE is also reported for each circuit. The first test generator is HITEC [7], a deterministic test generator, followed by the GA-based test generators GATEST [15, 26], CRIS [12], GATTO [16], and finally our new test generator, DIGATE.

Fault coverage is defined as the percentage of faults detected. From the table, the fault coverages achieved by DIGATE are significantly higher than those obtained by the other GA-based test generators for most of the circuits. For the larger circuits, DIGATE performs better than both HITEC and the GA-based test generators. For the hard-to-test ISCAS89 circuits, such as s400, s444, s526, s1423, s5378, and s35932, where long execution times are required by HITEC, the fault coverages achieved by DIGATE are significantly higher. DIGATE outperforms both HITEC and GATEST for all the synthesized circuits. The circuits that DIGATE does not perform as well on are s820, s832, s1488, and s1494. These circuits contain faults that require specific and often long sequences for fault activation. None of the GA-based test generators could match the results of HITEC for these circuits, since only HITEC was able to generate the exact sequences required to excite the faults.

The test sets obtained by DIGATE are shorter than those obtained by HITEC, even when higher fault coverages are achieved by DIGATE. The test sets are shorter than those obtained by CRIS and GATTO for most of the circuits. The test set lengths are comparable to those for GATEST, although sometimes longer due to the two-phase strategy of activating and propagating the fault. A small number of vectors at the beginning of the test set is able to detect a large fraction of the faults, while the remainder of the test set targets a few difficult faults at the expense of long sequences. This phenomenon is illustrated in Table 3 for several large circuits, which indicates that DIGATE is able to achieve very high fault coverages using a small number of vectors in a short time.

Table 3 also shows the effectiveness of DIGATE in terms of execution time for the large circuits. The best results of the other test generators in Table 2 are listed beside the results of DIGATE. The run times at three checkpoints are displayed for DIGATE. These checkpoints were placed at the end of each GA stage of DIGATE. The fault coverages at the end of the first GA stage are already higher than the final fault coverages of the other test generators for many circuits. Similar phenomena were also observed for the smaller circuits, although the results are not shown in the table. The total number of flip-flops and the number of flip-flops having distinguishing sequences are given for each checkpoint. Note that a small portion of flip-flops having distinguishing sequences is sufficient to improve the overall fault coverage, and for circuits having distinguishing sequences for a significant portion of flip-flops, the fault coverages improved greatly.

Table 2: Comparison of Fault Coverages

Circuit	**Total faults**	**HITEC [7]**		**GATEST [15, 26]**		**CRIS [12]**		**GATTO [16]**		**DIGATE**		
		Det	Vec	Det	Vec	Det	Vec	Det	Vec	Det	Vec	Dist
s298	308	**265**	306	264	161	253	476	-	-	264	239	7
s344	342	328	142	**329**	95	328	115	-	-	**329**	109	8
s382	399	**363**	4931	347	281	273	246	-	-	**363**	581	6
s400	426	**383**	4309	365	280	357	758	-	-	382	3369	11
s444	474	414	2240	405	275	397	519	-	-	**420**	1393	9
s526	555	365	2232	417	281	428	692	-	-	**446**	2867	9
s641	467	**404**	216	**404**	139	398	628	-	-	**404**	180	17
s713	581	**476**	194	**476**	128	475	1124	-	-	**476**	147	17
s820	850	**813**	984	517	146	451	1381	-	-	621	465	9
s832	870	**817**	981	539	150	370	1328	-	-	606	703	9
s1196	1242	**1239**	453	1232	347	1180	2744	1226	5202	1236	549	13
s1238	1355	**1283**	478	1274	383	1229	4313	1274	4672	1281	504	12
s1423	1515	776	177	1222	663	1167	2696	1265	3394	**1393**	4044	30
s1488	1486	**1444**	1294	1392	243	1355	1960	1344	631	1378	542	5
s1494	1506	**1453**	1407	1416	245	1357	1928	1277	912	1354	581	6
s5378	4603	3238	941	3175	511	3029	1255	3277	1132	**3447**	10500	94
s35932	39094	34902	240	35009	197	34481	1525	32943	563	**35100**	386	301
am2910	2391	2164	874	2163	745	-	-	-	-	**2195**	2206	71
mult16	1708	1640	273	1653	204	-	-	-	-	**1664**	915	32
div16	2147	1665	189	1739	634	-	-	-	-	**1802**	4481	3
pcont2	11300	3354	7	6826	272	-	-	-	-	**6837**	3452	0
piir8o	19920	14221	347	15013	531	-	-	-	-	**15072**	506	0
piir8	29689	11131	31	-	-	-	-	-	-	**18140**	603	0

Det: number of faults detected Vec: test set length Dist: number of distinguishing sequences generated
Highest numbers of detections are highlighted

V Conclusions

A test generation framework which utilizes genetically-engineered distinguishing sequences was presented. DIGATE is composed of three stages involving GA's, and several passes through the fault list are made in each stage. Test generation for a targeted fault is carried out in two phases. The first phase tries to excite a fault and propagate the FE's to the flip-flops, and the second phase attempts to drive the FE's from the flip-flops to the PO's. Various types of distinguishing sequence are computed in the preprocessing step and during the test generation process. These distinguishing sequences are seeded in the GA to evolve valid distinguishing sequences for the target fault in the fault propagation phase. Pruning of distinguishing sequences is done adaptively during test generation to improve their effectiveness, quantified by the distinguishing power index. The GA uses fitness functions that maximize the number of detected faults, propagation of FE's to flip-flops with powerful distinguishing sequences, faulty circuit events, and reachable state-space. With the aid of the distinguishing sequences, DIGATE achieves high fault coverages in short execution times, and the overall fault coverage is improved significantly compared to results obtained from other test generators.

References

[1] S. Seshu and D. N. Freeman, "The diagnosis of asynchronous sequential switching systems," *IRE Trans. Electronic Computing,* vol. 11, pp. 459-465, August 1962.

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

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

[4] 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, October 1988.

[5] M. H. Schulz and E. Auth, "Essential: An efficient self-learning test pattern generation algorithm for sequential circuits," *Proc. Int. Test Conf.*, pp. 28-37, 1989.

[6] A. Ghosh, S. Devadas, and A. R. Newton, "Test generation for highly sequential circuits," *Proc. Int. Conf. Computer-Aided Design,* pp. 362-365, 1989.

[7] T. M. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," *Proc. European Conf. Design Automation,* pp. 214-218, 1991.

[8] D. H. Lee and S. M. Reddy, "A new test generation method for sequential circuits," *Proc. Int. Conf. Computer-Aided Design,* pp. 446-449, 1991.

[9] H. Cho, S.-W. Jeong, and F. Somenzi, "Synchronizing sequences and symbolic traversal techniques in test generation," *Journal of Electronic Testing: Theory and Application,* Vol 4, no. 1, pp. 19-31, 1993.

[10] I. Pomeranz and S. M. Reddy, "Application of homing sequences to synchronous sequential circuit testing," *IEEE Trans. Computers,* vol. 43, no. 5, pp. 569-580, 1994.

[11] J. Park, C. Oh, and M. R. Mercer, "Improved sequential ATPG based on functional observation information and new justification methods," *Proc. European Design and Test Conf.*, 1995.

[12] D. G. Saab, Y. G. Saab, and J. A. Abraham, "CRIS: A test cultivation program for sequential VLSI circuits," *Proc. Int. Conf. Computer-Aided Design*, pp. 216-219, 1992.

Table 3: DIGATE Results for large circuits at various checkpoints

Circuit	**Best of other Test Gen's**			**DIGATE**						
	Det	Vec	TestGen	Chkpt	Det	Δ Det	Vec	Time	FF's	FF's w dist
s526				1	431	+3	556	3.28 min	21	8
				2	439	+11	907	10.9 min		8
	428	692	CRIS[12]	3	446	+18	2867	35.0 min		10
s1423				1	1381	+116	3760	23.5 min	74	29
				2	1393	+128	4044	1.42 hr		29
	1265	3394	GATTO[16]	3	1393	+128	4044	3.66 hr		29
s5378				1	3425	+148	2010	1.33 hr	179	74
				2	3436	+159	3170	4.00 hr		75
	3277	1132	GATTO[16]	3	3447	+170	10500	13.7 hr		75
s35932				1	35099	+90	225	17.4 hr	1728	288
				2	35100	+91	386	22.2 hr		288
	35009	197	GATEST[15]	3	35100	+91	386	58.1 hr		288
am2910				1	2118	-48	512	8.80 min	87	39
				2	2167	+3	1190	39.9 min		43
	2164	874	HITEC[7]	3	2194	+30	2206	1.67 hr		45
mult16				1	1662	+9	397	11.9 min	55	32
				2	1664	+11	915	25.0 min		32
	1653	204	GATEST[15]	3	1664	+11	915	2.14 hr		32
div16				1	1713	-26	725	1.84 hr	50	4
				2	1770	+31	1530	8.89 hr		5
	1739	634	GATEST[15]	3	1802	+63	4481	15.0 hr		5
pcont2				1	6828	+2	120	56.7 min	24	0
				2	6829	+3	380	8.33 hr		0
	6826	272	GATEST[15]	3	6837	+11	3452	24.4 hr		0
piir8o				1	15046	+33	292	2.67 hr	56	0
				2	15072	+59	506	6.40 hr		0
	15013	531	GATEST[15]	3	15072	+59	506	12.1 hr		0
piir8				1	18106	+6975	338	2.51 hr	56	0
				2	18139	+7008	512	7.45 hr		0
	11131	31	HITEC[7]	3	18140	+7009	603	11.2 hr		0

Performed on an HP 9000 J200 with 256 MB RAM
Check Point k: end of DIGATE GA stage k
Δ Det: improvement over the best coverage among other test generators
FF's w dist: number of FF's having at least one distinguishing sequence at the end of each checkpoint

[13] M. Srinivas and L. M. Patnaik, "A simulation-based test generation scheme using genetic algorithms," *Proc. Int. Conf. VLSI Design*, pp. 132-135, 1993.

[14] 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 and Test Conf.*, pp. 40-45, 1994.

[15] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, "Sequential circuit test generation in a genetic algorithm framework," *Proc. Design Automation Conf.*, pp. 698-704, 1994.

[16] P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, "An automatic test pattern generator for large sequential circuits based on genetic algorithms," *Proc. Int. Test Conf.*, pp. 240-249, 1994.

[17] 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, 1994.

[18] E. M. Rudnick and J. H. Patel, "Combining deterministic and genetic approaches for sequential circuit test generation," *Proc. Design Automation Conf.*, 1995.

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

[20] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning,* Reading, MA: Addison-Wesley, 1989.

[21] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design.* New York, NY: Computer Science Press, 1990.

[22] I. Pomeranz and S. M. Reddy, "LOCSTEP: A logic simulation based test generation procedure," *Proc. Fault Tolerant Computing Symp.*, June, 1995.

[23] T. E. Marchok, Aiman El-Maleh, W. Maly and J. Rajski, "Complexity of sequential ATPG," *Proc. European Design and Test Conf.*, March 1995.

[24] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," *Int. Symposium on Circuits and Systems*, pp. 1929-1934, 1989.

[25] Advanced Micro Devices, "The AM2910, a complete 12-bit microprogram sequence controller," in *AMD Data Book*, AMD Inc., Sunnyvale, CA, 1978.

[26] E. M. Rudnick, "Simulation-based techniques for sequential circuit testing," Ph.D. dissertation, Dept. Electrical and Computer Engineering, Tech. Report CRHC-94-14/UILU-ENG-94-2229, University of Illinois, August 1994.