Combinational Circuit Concurrent Test Generation: A Brief Overview

HILLARY GRIMES III, GRADUATE STUDENT, *AUBURN UNIVERSITY*

Abstract

The complexity of VLSI devices increases rapidly as technology increases, resulting in an increase in the difficulty of test generation. Traditional test generation algorithms target one fault of a fault set at a time to generate a test. This paper presents a brief overview of a few different approaches to concurrent test generation. Concurrent test generation is the process of generating tests by "concurrently" targeting multiple faults.

1. Introduction

Automatic Test Pattern Generation (ATPG) is the process of generating a test set (a set of input combinations and expected output responses) to test for all or most faults in a circuit. The complexity of both sequential and combinational ATPG is NP-complete, growing with circuit size. As VLSI technology advances, integrated circuits are becoming larger and more complex, resulting in the need to develop more efficient methods of generating test sets.

Many ATPG algorithms generate tests by targeting a single fault and finding an input vector to activate the fault and propagate its effect to a primary output. The input vector generated is a test that can detect the targeted fault. In concurrent test generation, the concurrent ATPG targets multiple faults, and tests are generated to detect as many targeted faults as possible. Tests are generated by "concurrently" targeting multiple faults.

In Section 2 of this paper, two similar directed search approaches to concurrent test generation are described: one phase of the three phase CONTEST algorithm and a phase in many genetic algorithm approaches to concurrent ATPG. Both approaches are simulation based methods, in which a fault simulator is used during the ATPG process. Before a vector is chosen to be added to the test set, it is simulated, providing timing analysis during test generation, eliminating the generation of hazardous tests. This gives simulation based methods a significant advantage over other tests generation methods [6].

The two concurrent ATPG approaches described in section 3 target groups of concurrently testable faults, and attempt to generate a single test that detects all faults in each targeted group. In both methods, concurrent-D algebra and concurrent ATPG using single-fault ATPG and simulation, the groups of faults targeted during test generation are grouped by a unique fault collapsing algorithm, which is also described in section 3.

2. Two Similar Directed Search Methods

The two concurrent test generation methods described in this section use a directed search approach. In the directed search approach, the test generator searches the input vector space in a "directed" manner for tests. The input vector space contains all possible input vectors and tests are usually clustered within the input vector space [1, 6]. By starting with any initial vector, a directed search algorithm directs the search for tests toward the test clusters.

The first concurrent test generation method described is phase 2 of the CONTEST algorithm. CONTEST consists of 3 phases: phase 1 generates an initialization vector sequence for sequential circuits, phase 2 is the concurrent test generation phase, and phase 3 generates tests by targeting single faults [1, 3, 6]. In phase 2, the search for tests is a directed search which is "directed" by the use of cost functions [1, 3].

The second concurrent test generation method described is a phase in most genetic algorithm approaches to test generation. This approach is similar to phase 2 of the CONTEST algorithm [6] and the search through the input vector space is directed though the use of fitness functions [4].

2.1 CONTEST: Phase 2

In phase 2 of the CONTEST algorithm, tests are generated by concurrently targeting all undetected faults. Beginning with an initial vector, the circuit is simulated using a concurrent

fault simulator. During simulation, cost functions are computed for each target fault. The cost function for a fault is the shortest distance that fault's effect is from a primary output. If the fault effect is observable on a primary output, the cost function is zero, and if the fault has no fault effect (the fault is not activated) the cost function is infinite. Trial vectors are generated by modifying the current input vector with a one bit change. Each trial vector is simulated, and the associated costs are computed. By comparing the costs from the current input vector simulation with the costs from the trial vector simulation, the algorithm either accepts or rejects the trial vector. The trial vector is accepted if the overall costs of undetected faults are found to decrease. If accepted, the trial vector becomes the current vector, and the process starts over. When the costs associated with all single bit changes in the current vector do not reduce total cost, phase 2 ends and CONTEST begins phase 3. Phase 3 targets single faults to continue the search for tests, and is needed because phase 2 only achieves between 65 and 85 percent fault coverage [1, 3, 6].

2.2 Concurrent ATPG using Genetic Algorithms

Many ATPGs based on genetic algorithms include a phase of concurrent test generation which is similar to phase 2 of the CONTEST algorithm. Both directed search approaches are "evolutionary", meaning they "evolve" tests by accepting or rejecting vectors based on how well they detect faults [4, 5, 6].

Test generation using genetic algorithms begins with a set of vectors, which initially is usually generated randomly, called a population. Each vector has an associated fitness which is evaluated through simulation [5, 6]. The higher the fitness function, the more "fit" a vector is to detect faults. The population is improved iteratively, with each iteration producing a new population, and each population produced is called a generation. Each generation is "evolved" from the previous generations through three evolutionary operators called selection, mutation, and crossover [4, 5, 6].

In selection, two of the higher fit individuals (vectors) are selected from the population for reproduction in generating the next generation [4, 5]. The crossover evolutionary operator exchanges bits between two selected vectors of the population to produce two vectors for the next generation. Mutation is the process of modifying bits from a selected vector in the population to produce a new vector for the next generation [4, 6]. The generation of vectors for a new generation is biased toward higher fitness [5, 6], and the overall fitness of the population generally increases in successive generations [5].

As the population is evolved, the best test is stored separately. With each new generation, new vectors are compared to the best test, and the best test is updated if a better test is found. The best test is added to the test set after the last generation is produced. This process is repeated, and tests are generated until fault coverage does not improve [5].

3. Methods Targeting Groups of Concurrently Testable Faults

The two approaches to concurrent test generation discussed in this section use a unique independence fault collapsing algorithm prior to concurrent test generation. The purpose of this algorithm is to group faults into subsets where most faults in each group (subset) are likely to have a concurrent test [7,8,9]. Both methods discussed, concurrent-D algebra and concurrent ATPG using single-fault ATPG and simulation, attempt to generate a concurrent test for all or most faults in each group.

3.1 Independence Fault Collapsing Algorithm

Independent faults are faults that have no common test. One example would be a sa0 and a sa1 fault on the same line. Because both faults cannot be active for the same test vector, they cannot have a common test. Concurrently-testable faults are defined as two faults that are not independent and do not have a dominance relationship. Two concurrently testable faults have both a concurrent test, which is a test for both faults, and a unique test, which is a test for one fault but not the other [7, 8, 9].

The independence fault collapsing algorithm begins with a dominance collapsed fault set, and an independence graph and independence matrix is generated. In an independence graph, each node represents a fault, and an edge between two faults (nodes) means that the two faults are independent. In an independence matrix, each fault has a row and a column, and each position in the matrix has a "0" or a "1". A "1" is placed at that position if the two faults are independent, and a 0 is placed if

they are not independent (there exists a common test for both faults) [7, 8].

Three methods to determine independence relations between faults in the dominance collapsed fault set are described in [7, 8]. In the first method, structural independences are found for boolean gates, shown in figure 1 [7, 8].

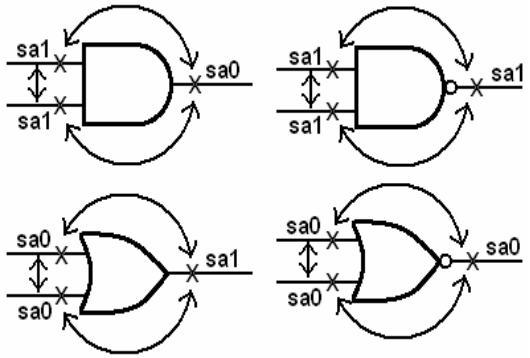

Figure 1: Structural Independence Relationships for Boolean Gates

The second method uses the structural independence relationships to find implied fault independences between subnetworks in a hierarchally described circuit. This method is based on hierarchal fault collapsing [7, 8], which collapses fault sets of smaller subnetworks and stores the resulting data in a library in the form of a dominance matrix. The dominance matrix contains information about equivalence and dominance relationships between the subnetwork's inputs or outputs and faults in the fault list [2]. If the two subnetworks are connected as shown in figure 2, and fault a is equivalent to fault b, then an independence relationship between faults c and d would imply an independence relationship between faults a and d. Also, if fault b dominates fault a, then independence between faults c and d would imply independence between faults a and d [7, 8].

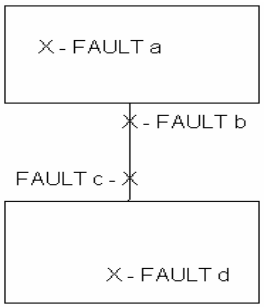

Figure 2: Independence Relations between Subnetworks

The third method determines functional independence relationships using an ATPG that checks for redundant faults and three copies of the circuit shown in figure 3. Each fault is inserted in C, and the ATPG detects faults in A. If the finds a redundant fault in A, then that fault is independent of the fault currently inserted in C [7, 8].

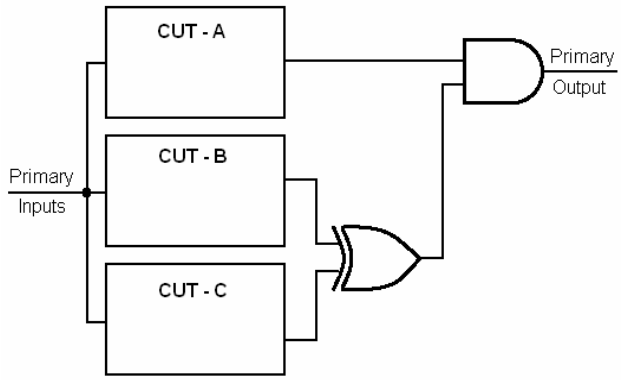

Figure 3: Using ATPG to find Functional Independence Relationships

Once independence relationships have been determined, the independence matrix is generated, and the algorithm computes the degree of independence for each fault. The degree of independence is computed by adding all positions in a fault's row or column in the independence matrix. Faults are then rearranged in the independence matrix by decreasing order of their degree of independence, and a similarity metric is computed for each pair of faults. The similarity metric is a measure of how similar two faults are in concurrent testability with respect to all faults, and its value ranges between 0 and the total number of faults. If N=number of faults and x is a position in the independence matrix, the similarity index between two faults is [7, 8]:

$$SIM(f_i, f_j) = Nx_{ij} + (1 - x_{ij}) \sum_{k=1}^{N} | x_{ik} - x_{jk} |$$

If two faults are equivalent, their similarity metric is 0, and if they are independent, their similarity metric is the maximum value (indicating they are not similar) [7, 8].

The algorithm then begins grouping faults by adding each fault to an initially empty graph in decreasing order of degree of independence. In this graph, each node represents a group of faults. The first fault added simply creates the first node. Before each

remaining fault is added, a similarity index is computed for each node already in the graph. The similarity index for a node is the maximum similarity metric between a fault already in that node and the fault that is to be added. The fault is then placed in the node with the smallest similarity index. If the similarity index of every node is the maximum, then there is a fault in every node that is independent of the fault being added, so a new node is created. After adding all faults, each node contains a set of faults that are likely to have a concurrent test [7, 8].

This independence fault collapsing algorithm begins with a dominance collapsed fault set, so functional dominance collapsing must be preformed before generating the independence matrix. Since both functional dominance collapsing and independence matrix generation are based on ATPG and complex, a second method to generate the independence matrix is presented in [7]. This method uses a fault simulator that simulates without fault dropping to generate the independence matrix starting with an equivalence collapsed fault set. First, an independence graph is generated assuming all faults are independent (every node has an edge to every other node). Then the fault simulator is used to simulate random vectors and find which faults are detected by each vector. When more than one fault is found to be detected by the same vector, the edge between these faults in the independence graph is removed. When a large number of random vectors are simulated without removing an edge in the independence graph, the process stops, and the independence matrix is generated without the complexity of functional dominance collapsing or functional independence identification [7].

3.2 Concurrent ATPG using Concurrent-D Algebra

Test generation using concurrent-D algebra, proposed in [7], targets all faults in each collapsed set to generate a concurrent test. In concurrent-D algebra, a D has the same meaning as a D in D algebra. During single-fault ATPG, if a signal is affected by the targeted fault, then it is assigned the value D, then that signal would be a "1" if the fault is not present, and a "0" if the targeted fault is present. The main difference in concurrent-D algebra approach is that the ATPG is targeting multiple faults concurrently. When a signal in the circuit is affected by a single fault i, its value is set to D_i or $\overline{D}_i$, and when multiple faults i and j affect a line, it's value is set to D_{ij} or $\overline{D}_{ij}$ [7]. Concurrent-D algebra for a 2-input AND gate [7] and a 2-input OR gate is shown in table 1 below:

INPUT 2 \ AND	Input 1: 0	1	X	D_i	D_j	$\overline{D}_i$	$\overline{D}_j$	D_{ij}	$\overline{D}_{ij}$
0	0	0	0	0	0	0	0	0	0
1	0	1	X	D_i	D_j	$\overline{D}_i$	$\overline{D}_j$	D_{ij}	$\overline{D}_{ij}$
X	0	X	X	X	X	X	X	X	X
D_i	0	D_i	X	D_i	D_{ij}	0	$\overline{D}_j$	D_{ij}	$\overline{D}_j$
D_j	0	D_j	X	D_{ij}	D_j	$\overline{D}_i$	0	D_{ij}	$\overline{D}_i$
$\overline{D}_i$	0	$\overline{D}_i$	X	0	$\overline{D}_i$	$\overline{D}_i$	0	0	$\overline{D}_i$
$\overline{D}_j$	0	$\overline{D}_j$	X	$\overline{D}_j$	0	0	$\overline{D}_j$	0	$\overline{D}_j$
D_{ij}	0	D_{ij}	X	D_{ij}	D_{ij}	0	0	D_{ij}	0
$\overline{D}_{ij}$	0	$\overline{D}_{ij}$	X	$\overline{D}_j$	$\overline{D}_i$	$\overline{D}_i$	$\overline{D}_j$	0	$\overline{D}_{ij}$

INPUT 2 \ OR	Input 1: 0	1	X	D_i	D_j	$\overline{D}_i$	$\overline{D}_j$	D_{ij}	$\overline{D}_{ij}$
0	0	1	X	D_i	D_j	$\overline{D}_i$	$\overline{D}_j$	D_{ij}	$\overline{D}_{ij}$
1	1	1	1	1	1	1	1	1	1
X	X	1	X	X	X	X	X	X	X
D_i	D_i	1	X	D_i	1	1	D_i	D_i	1
D_j	D_j	1	X	1	D_j	D_j	1	D_j	1
$\overline{D}_i$	$\overline{D}_i$	1	X	1	D_j	$\overline{D}_i$	$\overline{D}_{ij}$	D_j	$\overline{D}_{ij}$
$\overline{D}_j$	$\overline{D}_j$	1	X	D_i	1	$\overline{D}_{ij}$	$\overline{D}_j$	D_i	$\overline{D}_{ij}$
D_{ij}	D_{ij}	1	X	D_i	D_j	D_j	D_i	D_{ij}	1
$\overline{D}_{ij}$	$\overline{D}_{ij}$	1	X	1	1	$\overline{D}_{ij}$	$\overline{D}_{ij}$	1	$\overline{D}_{ij}$

Table 1: Concurrent-D Algebra for 2-input AND [7] and OR Gate

A concurrent ATPG program using concurrent-D algebra has not been implemented [7, 9], but a different approach to concurrent ATPG is proposed in [7] that uses a single fault ATPG and a fault simulator, which is discussed in the next section.

Results obtained by manual application of concurrent-D algebra on the c17 benchmark circuit to generate a concurrent test are reported in [7], and are shown here in figure 4. There are 11 sa1 faults in the dominance collapsed fault set for c17, which are grouped into 4 groups after applying the independence fault collapsing algorithm, shown in table 2 [7, 9]. The three faults in fault group 2 are targeted concurrently, and a single concurrent test, "01111", that detects all faults in the group is generated [7].

Fault Group	Faults
1	1, 8
2	2, 3, 9
3	4, 6, 10
4	5, 7, 11

Table 2: Fault Groups for c17

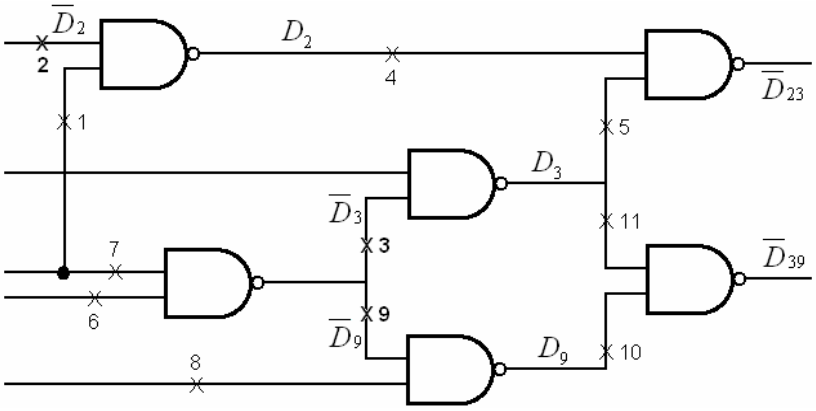

Figure 4: Results of Concurrent-D Algebra Targeting Faults 2, 3, and 9

3.3 Concurrent ATPG Using Single-Fault ATPG and Simulation

After the independence fault collapsing algorithm, faults are grouped into sets of faults that are likely to have a concurrent test. This approach to concurrent ATPG is presented in [7], and attempts to generate a single test to detect most faults in each collapsed group using a single fault ATPG and a fault simulator. Since a smaller set of tests is likely for a fault with a higher degree of independence as opposed to a fault with a lower degree of independence, the generation of a single test for the group begins by targeting the fault with the largest degree of independence using a single fault APTG. All test vectors are generated for the targeted fault, and simulated using a fault simulator to choose a vector as a single test for the group. The chosen vector is the one that is found to detect the highest number of faults in the group, and the highest number of faults outside the group if multiple vectors detect the same number of faults in the group [7-9].

Results of concurrent ATPG using single-fault ATPG and simulation on 18 various combinational circuits are reported in [7]. Table 3 is taken from these results and shows a comparison of the number of tests generated and the number of groups produced by the independent fault collapsing algorithm. One circuit (4-b ALU) shows 12 tests generated for 13 groups, and in 7 circuits the number of tests generated is equal to the number of groups. For the remaining 11 circuits, the number of tests generated is only slightly larger than the number of groups, showing the efficiency of this concurrent ATPG approach in generating a single test per group.

Circuit	# of Groups	# of Generated Tests	Circuit	# of Groups	# of Generated Tests
1-b adder	5	5	c499	52	52
2-b adder	5	5	c880	24	29
4-b adder	5	5	c1355	84	84
8-b adder	7	7	c1908	106	111
16-b adder	7	9	c2670	81	92
32-b adder	7	11	c3540	107	130
4-b ALU	13	12	c5315	92	104
c17	4	4	c6288	23	25
c432	30	34	c7552	190	198

Table 3: Results of Concurrent ATPG using Single-Fault ATPG and Simulation

4. Conclusion

Because ATPG is NP-complete, many algorithms have been developed that attempt to reduce the complexity in deriving a set of tests to adequately test modern complex devices. As seen from good results in [1, 3, 4, 5, 7, 8, 9], the concurrent ATPG algorithm approach to reducing difficulties in test generation shows promise for the future of ATPG.

5. References

[1] V. D. Agrawal, K. T. Cheng, and P. Agrawal, "A Directed Search Method for Test Generation Using a Concurrent Fault Simulator", *IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems*, Vol. 8, pp. 131-138, February 1989.

[2] A. V. S. S. Prasad, V. D. Agrawal, and M. V. Atre, "A New Algorithm for Global Fault Collapsing into Equivalence and Dominance Sets", Proc. International Test Conf., pp. 391-397, October 2002.

[3] V. D. Agrawal, K. T. Cheng, and P. Agrawal, "CONTEST: A Concurrent Test Generator for Sequential Circuits", *Proc. Des. Auto. Conf.*, pp. 84-89, June 1988.

[4] M. Srinivas and L. M. Patnaik, "A Simulation-Based Test Generation Scheme Using Genetic Algorithms", *Proc. Int. Conf. VLSI Design*, pp. 132-135, January 1993.

[5] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, "A Genetic Algorithm Framework for Test Generation", *IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, pp. 1034-1044, September 1997.

[6] M. L. Bushnell & V. D. Agrawal, "Essentials of Electronic Testing", Textbook, Kluwer academic publishers.

[7] A. S. Doshi, "Independence Fault Collapsing and Concurrent Test Generation", MS Thesis. Auburn University, Department of ECE, Auburn, AL, USA, 2006.

[8] A. S. Doshi and V. D. Agrawal, "Independence Fault Collapsing", *Proc. 9th VLSI Design and Test Symp.*, pp. 357-366, August 2005.

[9] V. D. Agrawal and A. S. Doshi, "Concurrent Test Generation", *Proc. 14th IEEE Asian Test Symp.*, December 2005.