Improved Sequential ATPG Using Functional Observation Information and New Justification Methods *

Jaehong Park, Chanhee Oh and M. Ray Mercer

Dept. of Electrical and Computer Engineering
The University of Texas at Austin
Austin TX 78712-1084

Abstract

Sequential ATPG (Automatic Test Pattern Generation) is a very desirable CAD tool, but to date, the size and complexity of circuits for which sequential ATPG could be performed has been limited. We have discovered a method for collecting functional information which makes fault observation significantly easier. We also propose a new method for state justification which is a combination of function-based methods and structure-based methods. Our sequential ATPG system deals with circuits without a reset state or a synchronizing sequence, and the experimental results show that the proposed method achieves significant improvements over existing sequential ATPG methods.

1 Introduction

Sequential Automatic Test Pattern Generation (ATPG) has been a goal of CAD developers for at least a decade. Various sequential test generators have been proposed in the past [GHOS] [NIER] [LEE] [CHO91] [POME92] [CHO93] [POME94]. Recent advances in the Boolean function manipulation techniques based on Ordered Binary Decision Diagrams (OBDDs) [BRYA] have enabled reachability analysis for the State Transition Graphs (STGs) of sequential circuits [COUD89], which dramatically improves the performance of sequential circuit test generators. However, ATPG techniques based on the reachability analysis can only be applied for the restricted set of circuits with a designated reset state [CHO91] [POME94] or circuits for which a synchronizing sequence can be found [CHO93]. Although other approaches do not have such limitations, they are far from efficient. Our research deals with the generation of test patterns for sequential circuits without the assumption of a reset state or a synchronizing sequence and based on single observation test time strategy in contrast to multiple observation test time strategy [POME92].

*This work was supported in part by SRC under contract 93-DP-142 and in part by ARPA under contract SFRC # DAAL01-93-K-3317

The process of sequential ATPG can be decomposed into three sub-problems. These are 1) combinational ATPG, 2) fault observation and 3) state justification. Of these three, combinational ATPG is by far the most mature. In this work, we propose enhancements for state justification based upon pre-image computation, and we also introduce a new method for fault observation based on the **Difference Propagation Diagram** . Experimental results show that the proposed ATPG methods achieve significant improvements over existing sequential ATPG algorithms.

2 Sequential Test Generation Algorithm Overview

The iterative array model [ABRA] used in most sequential test generators is shown in Figure 1. The circuit is duplicated to take multiple time frames into account, and thus the fault f exists in every array element. The array element at $t = 0$ has the activation state where the fault is activated for the first time. The fault effect is propagated to a primary output by a propagation sequence, and the activation state is justified from the initial state by a justification sequence.

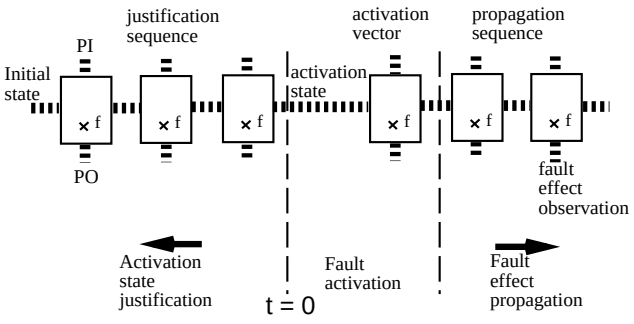

Figure 1: Iterative array model

The fault propagation is guided by the **Difference Propagation Diagram** which is a unique feature of our ATPG system. We construct the Difference Propagation Diagram for the good circuit. We use both functional

and structural methods to compute justification sequences. While structural methods process a cube at a time, functional methods deal with functions and process multiple cubes simultaneously. Hence, our methods either does depth first search or does breadth first search. Functional methods are based on the pre-image computation techniques [PIXL] [COUD91] and are performed for the good circuit. If the faulty circuit is driven to the activation state by the justification sequence, then the computed justification sequence can be used. If the justification sequence is not valid, we use structural methods [LEE] [NIER] to compute the justification sequence for the faulty circuit. The most significant difference between our approach and earlier work is in the fault propagation and state justification tasks. These are explained in detail below.

3 Fault activation and propagation

Figure 2 is an iterative array model of forward time processing. x is the fault site at each time frame, and the thick line is the sensitized path. At time frame 0, if possible, the fault effect is propagated to a primary output. Otherwise, the fault effect is propagated to a flip-flop. Once the fault effect is propagated to a flip-flop, the search and decision making is guided by the functional information represented by the **Difference Propagation Diagram**.

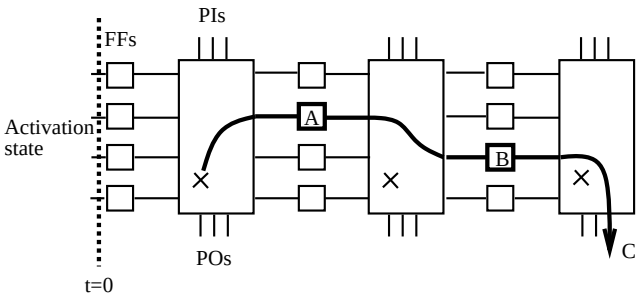

Figure 2: Fault activation and propagation

The Boolean difference of a function $f(x_1, x_2, ..., x_i, ..., x_n)$ with respect to x_i is written as $\partial f(x_1, x_2, ..., x_i, ..., x_n) \ / \ \partial x_i$, and is defined as the EXCLUSIVE-OR of f_{x_i} and $f_{\overline{x_i}}$, where $f_{x_i} = f(x_1, x_2, ..., 1, ..., x_n)$ and $f_{\overline{x_i}} = f(x_1, x_2, ..., 0, ..., x_n)$. The next state functions of sequential circuits are represented in terms of primary input variables and present state variables (variables which are associated with flip-flops at the present time frame). Suppose a represents the present state variable for flip-flop A, and b represents the next state function for flip-flop B in figure 2. Then, the function $\partial b \ / \ \partial a = 1$ represents the vector space where the difference at flip-flop A can propagate to flip-flop B. Similarly, if b represents the present state variable for flip-flop B, and c represents the primary output C, then the function $\partial c \ / \ \partial b = 1$ represents the vector space where the difference at flip-flop B can propagate to primary output C. Note that these Boolean differences are expressed in terms of (1) circuit primary input variables and (2) present state variables.

We define the cost of a cube in the Boolean difference as the number of flip-flop variable assignments. The cost of difference propagation of a flip-flop is defined as the cost of the minimum cost cube. If $\partial f \ / \ \partial x_i$ has a cube whose cost is 0 (vacuous in flip-flop variables), then we have found an input pattern by which we can propagate a difference from a flip-flop to a primary output or another flip-flop independent of the values of all other present state variables (i.e. from any state).

A **Difference Propagation Diagram** is a graph $G = (V, E)$ where V is the set of flip-flops and primary outputs, and E is the set of edges (v_i, v_j) such that (v_i, v_j) is in E if and only if a flip-flop or a primary output represented by v_j is structurally reachable from a flip-flop represented by v_i. For each edge in the graph, we compute the Boolean difference of the function for the destination vertex with respect to the variable for the source vertex. The minimum cost cube and the corresponding difference propagation cost are computed for the Boolean difference and stored on the edge. Since we use OBDDs to represent combinational functions of sequential circuits, the above information can be computed easily due to the efficiency of OBDDs for function manipulation.

Observation cost of a flip-flop is defined as the cost of the minimum cost path to propagate the difference from the flip-flop to any primary output. The cost of a path is the sum of the cost at each edge along the path. The observation cost is computed for each flip-flop. Experimental results for ISCAS benchmark circuits [BRGL] show that for most circuits, as many as 50% of flip-flops have observation cost less than 2. We call the collection of input patterns along the minimum cost path as **MAXSIPS (MAXimally State Independent Propagation Sequences)**. The observation costs are used as an observability measure for flip-flops.

We also store the Boolean differences of output/flip-flop functions with respect to each fanin flip-flop variable when their OBDD size fits in the available memory. We use the stored Boolean differences when the observation cost is not 0 and the stored cube contradicts the present state value. Suppose the fault effect has been propagated to a flip-flop whose observation cost is 0. Then, MAXSIPS provides a propagation sequence which guarantees observation of the fault effect at a primary output. If a fault effect has been propagated to a flip-flop whose observation cost is greater than 0, then we compare the values of flip-flop variables in the minimum cost cube with the present state variable values. If their values match, we use them, and if they contradict, we use the entire Boolean difference function.

Although MAXSIPS and the Difference Propagation Diagram are computed for good circuits, it has been shown that most propagation sequences for the fault-free circuit are also valid for faulty circuits [GHOS]. Hence, chances are very high that the information in the Difference Propagation Diagram is also valid for faulty circuits.

Figure 3 is an example circuit and its Difference Propagation Diagram. For each edge, the difference propagation cost is shown, but the corresponding cube is not shown for brevity. When the fault effect is propagated to FF 1, the propagation sequence can be computed immediately (via FF2 and FF3) from MAXSIPS. Note that primary input cube $(X1, X2, X3, X4) = (d, d, 1, 1)$, where d =don't care, is a cube which propagates a difference from FF2 to FF3, and it can be computed easily because we use OBDDs to represent combinational functions. If we use conventional constant (1 and 0) based ATPG, we need to set a value at FF4, and extend time frames backward to justify the value at FF4.

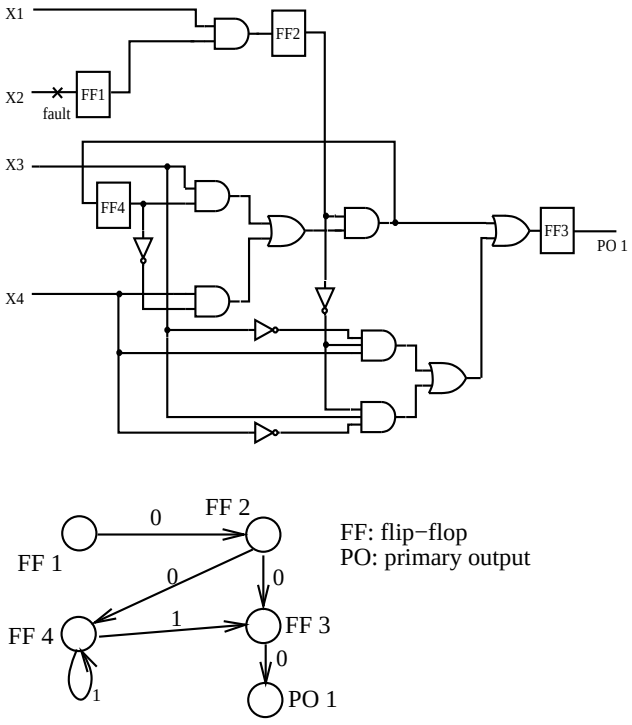

Figure 3: Example circuit and its Difference Propagation Diagram

If the fault effect cannot be propagated to a primary output, we propagate the fault effect to a flip-flop with the lowest observation cost. After propagating to a flip-flop, we try to set primary inputs according to the cube and Boolean Difference functions stored in the Difference Propagation Diagram. If the fault effect is not propagated as desired, backtrack is performed to maintain completeness. Experimental results show that this heuristic is very efficient for finding propagation sequences. In the following two sections, we will show the state justification methods and the entire ATPG system structure.

4 State justification

After the fault effect is propagated to a primary output, the activation state is justified through backward time processing.

The **pre-image** of a set of states A under $\vec{\delta}$ is defined as

$$pre\text{–}image(\vec{\delta}, A) \quad = \quad \{\vec{x} \mid \exists \vec{y}, \vec{y} \in A, \vec{\delta}(\vec{x}) = \vec{y}\}$$

where $\vec{x}$ consists of present state variables and primary input variables, and $\vec{y}$ consists of next state variables. [COUD91] and [PIXL] proposed a pre-image computation techniques using OBDD. We use these techniques in the justification sequence computation process for good circuits whose overall algorithms are shown in Figure 4. In figure 4, S_a is the activation state and S_p is the last state resulting from the test vectors for previous faults. Pre-image computation is continued until S_p is reached, there is no assignments on flip-flop variables, or no new state is visited. Though no assignments on flip-flop variables implies S_p is reached, we consider the two cases independent for clarity of explanation. ExtractStates performs universal abstraction on primary input variables. When the ATPG process starts for the first fault, we have an unknown state as S_p, and the algorithms in figure 4 are modified so that pimg[i] share the same input pattern.

```
Compute Good Circuit Justification Sequence
      by Function-based Methods (Sa, Sp)
{
  i=0
  pimg[i]=start=totalVisited = Sa
  while(1)
    i=i+1
    pimg[i] = preImage(start)
    if pimg[i] has a cube which consists of only
            primary inputs or is included in Sp
      break
    else
      if pimg[i] is included in totalVisited
        return FAIL
      else
        start = extractStates(pimg[i])
        totalVisited = totalVisited + start
  compute justification sequence from pimg[]
  return SUCCESS
}
```

Figure 4: Function-based justification sequence computation algorithms

Suppose the result of fault activation and propagation stage is the activation state S_{gf}. From S_{gf}, we extract the good circuit state S_g. For example, if S_{gf} is $(1/x, 0/0, x/1, 1/1, x/x)$, then S_g is $(1, 0, x, 1, x)$. We compute the justification sequence of S_g for the good circuit using the above algorithms. If S_g is unreachable from the initial state, then S_{gf} is also unreachable from the initial state. Therefore, the complexity of identifying unreachability of an activation state decreases dramatically when the activation state is unreachable in the good circuit do-

main, because the good and faulty circuit domain complexity is $O(N^2)$ when the good circuit domain complexity is $O(N)$. Since if a state is unreachable from a specific state, then the state is also unreachable from the unknown state, the completeness of the ATPG algorithm is not affected in case the initial state is not the unknown state, but a specific state resulting from previous test vectors.

If the activation state S_{gf} is reachable from the initial state and the activation state requires no assignment on the faulty circuit, the justification sequence for the good circuit is also valid for the faulty circuit. For example, if the activation state is $(1/x, 0/x, x/x, 1/x)$, then the justification sequence of S_g, $(1, 0, x, 1)$, is also a justification sequence for S_{gf}. Otherwise, the justification sequence of S_g is concatenated with a propagation sequence and simulated against the faulty circuit to check if the fault is detected. If the sequence cannot detect the fault, we use structural methods [LEE] to compute a justification sequence of S_{gf}.

While structure-based conventional methods deal with a cube at a time, function-based methods deal with functions, which are collections of cubes. Therefore, the ways of performing search are breadth first search for function-based methods and depth first search for structure-based methods. Theorem 1 shows the advantage of function-based methods due to the breadth first property of function-based methods. In theorem 1, for simplicity, we assume state justification is computed for the good circuit.

Theorem 1 *Suppose s is an activation state, and we are trying to find a justification sequence from s to the initial unknown state. If the set of justification sequences computed by structure-based methods is S_s and the set of justification sequences computed by function-based methods is S_f, then $S_s \subseteq S_f$ and $S_s \not\supset S_f$.*

Proof: *Suppose the justification sequence $t = t_0, t_1, ...t_{n-1}$ is computed by structure-based methods, t is applied to the circuit at the initial state, and the state of the circuit changes in the order of s_0, s_1, s_2, ..., s_n, where s_0 is the initial state and s_n is the activation state. Pre-image computation for state s_n can identify all combinations of states and input vectors whose successor state is s_n. Hence, if structure-based methods can identify s_{n-1} and t_{n-1}, function-based methods can also identify them. The same holds for s_{n-2} and t_{n-2}, ..., s_0 and t_0. Therefore, function-based methods can compute t.*

Suppose figure 5 is the state transition graph of the circuit, and s_5 is the activation state. If we use function-based methods, $\{s_3, s_4, s_5\}$ are identified as the pre-image of s_5 with input 1. Pre-image of $\{s_3, s_4, s_5\}$ with input 0 is computed as the whole state set, $\{s_0, s_1, s_2, s_3, s_4, s_5\}$. Hence, 01 is a justification sequence of s_5 starting from the initial unknown state. But, if we compute a cube at a time as in structure-based methods, computing the justification sequence 01 is impossible unless s_3, s_4 and s_5 can be represented as one cube. Therefore, structure-based methods may not find a justification sequence which function-based methods can find. □

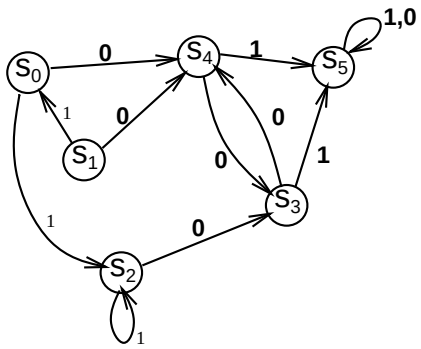

Figure 5: Example state transition diagram

While function-based methods have the advantages of breadth first search, structure-based methods have advantages of depth first search. Depth first search can find a niche in cases when the depth of search is deep, or the amount of information needed for breadth first search is too large. Since we use both breadth and depth first search, our algorithms result in significant improvements over previous approaches.

5 Experiments and Results

Our ATPG algorithms have been implemented with the C language and has been run on ISCAS sequential benchmark circuits [BRGL] with a SPARC station 2. We do not assume that circuits have reset states or synchronizing sequences. CPU time and fault coverage are compared in table 1 between two existing methods [NIER] [LEE] and our new method. The CPU times are measured in seconds for all the faults including untestable faults and aborted faults. HITEC was run on a SPARC Station 1, and [LEE] was run on a SPARC station 2. The more recent HITEC results reported in [LEE] are used instead of those in [NIER]. As one can see, our new methods give comparable fault coverage in significantly less CPU time.

We also found that our methods give shorter test length for most circuits, because of breadth first search characteristics of function-based justification methods.

6 Conclusion

We have presented two innovations for sequential ATPG. We have demonstrated the effectiveness of the innovations by showing the experimental results. Our sequential ATPG system deals with circuits without a reset state or a synchronizing sequence.

The use of functional information by Difference Propagation Diagram has been shown to be very effective for fault observation. Once a fault effect is propagated to a flip-flop, we could propagate the fault effect to a primary output without much effort. Since the Difference Propagation Diagram is constructed for only the good circuit, the expense of computing the diagram is negligible.

For state justification, we used both function-based techniques and structure-based techniques, which gives significant improvements over existing methods. Therefore,

Circuit	CPU time			test length			fault coverage		
	HITEC	LEE	**NEW**	HITEC	LEE	**NEW**	HITEC	LEE	**NEW**
s208	29	49	**16**	184	194	**160**	63.7	63.7	**63.7**
s298	15969	3250	**1570**	306	203	**207**	86.0	85.7	**86.0**
s344	4785	354	**34**	142	64	**78**	95.9	96.1	**96.1**
s349	3235	407	**27**	137	84	**83**	95.7	95.7	**95.7**
s382	43197	3278	**538**	4931	1286	**1109**	90.9	91.2	**91.2**
s386	82	82	**23**	311	292	**232**	81.7	81.7	**81.7**
s400	43587	3817	**515**	4309	1098	**1001**	89.9	90.2	**90.2**
s420	17015	1034	**207**	120	172	**167**	41.3	41.6	**41.6**
s444	58131	3252	**1276**	2240	840	**897**	87.3	89.0	**89.0**
s526	168710	13434	**7651**	2232	1919	**1900**	65.7	81.2	**81.2**
s641	1083	28	**9**	216	185	**174**	86.5	86.3	**86.3**
s713	95	242	**39**	194	175	**149**	81.9	81.7	**81.7**
s820	5806	4096	**882**	984	903	**902**	95.6	95.5	**95.8**
s832	6350	3911	**1303**	981	905	**905**	93.6	94.0	**94.0**
s1196	102	149	**44**	453	376	**352**	99.7	99.7	**99.7**
s1238	144	195	**59**	478	372	**362**	94.6	94.6	**94.6**
s1488	13348	2749	**529**	1294	1270	**979**	97.1	97.1	**97.1**
s1494	6981	2345	**368**	1407	1163	**850**	96.4	96.4	**96.4**

Table 1: Test generation results comparison

we could take advantage of both breadth first and depth first search methods. The test sequences were shorter compared with existing methods, mainly due to the breadth first characteristics of our methods.

Acknowledgments

The authors wish to thank Dr. D. H. Lee at Kyong Book University, Korea and Dr. S. M. Reddy at the University of Iowa for providing structural state justification code and a fault simulator which was part of the implementation used to evaluate the concepts of this paper.

References

[ABRA] M. Abramovici, M. Breuer and A. Friedman, *Digital systems testing and testable design*, Computer Science Press (1990).

[BRGL] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. of International Symposium on Circuits and Systems,* 1989.

[BRYA] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," in *IEEE Trans. on Computers*, vol C-35, Aug. 1986.

[CHO91] H. Cho, G. D. Hachtel and F. Somenzi "Fast sequential ATPG based on implicit state enumeration," in *Proc. of International Test Conference,* 1991.

[CHO93] H. Cho, S.-W. Jeong, F. Somenzi and C. Pixley, "Synchronizing Sequences and Symbolic Traversal Techniques in Test Generation," in *Journal of Electronic Testing: Theory and Applications*, vol. 4, no. 1, Feb. 1993.

[COUD89] O. Coudert, C. Berthet and J. C. Madre, "Verification of sequential machines using boolean functional vectors," in L. Claesen, editor, *Proc. of IFIP International Workshop on Applied Formal Methods for Correct VLSI Design,* 1989.

[COUD91] O. Coudert, J. C. Madre, and C. Berthet, "Verifying temporal properties of sequential machines without building their state diagrams," in E. M. Clarke and R.P. Kurshan, editors, *Computer-Aided Verification '90*, American Mathematical Society - Association for Computing Machinery, 1991.

[GHOS] A. Ghosh, S. Devadas and A. R. Newton, "Test generation and verification for highly sequential circuits," in *IEEE Trans. on Computer-Aided Design,* vol. 10, May 1991.

[LEE] D. H. Lee and S. M. Reddy, "A new test generation method for sequential circuits," in *Proc. of International Conference on Computer Aided Design,* 1991.

[NIER] T. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *Proc. of European Design Automation Conference*, 1991.

[PIXL] C. Pixley, S.-W. Jeong and G. D. Hachtel, "Exact calculation of synchronization sequences based on binary decision diagrams," in *IEEE Trans. on Computer-Aided Design,* Aug. 1994.

[POME92] I. Pomeranz and S. M. Reddy, "The multiple Observation time test strategy," in *IEEE Trans. on Computer-Aided Design,* May, 1992.

[POME94] I. Pomeranz and S. M. Reddy, "On achieving complete fault coverage for sequential machines", in *IEEE Trans. on Computer-Aided Design,* Mar. 1994.