

# ATPG by Fault Sampling

*Yueli Liu*

## Abstract

ATPG (Automatic Test-Pattern Generation) is the process of finding an input pattern that will sensitize a specific fault, and then propagate the effect of this fault to a primary output of the device. In this paper, the algorithms of ATPG for combinational circuit are briefly introduced. It includes D-algorithm, Basic ATPG Algorithm, PODEM algorithm and testability measurements, finally the technical challenge and recent development for ATPG is discussed.

## 1. Introduction

For a given set of faults in the circuit under test, how do we obtain a certain (small) number of test patterns which guarantees a certain (high) fault coverage? This is a VLSL testing problem. For large circuits, the number of faults can become very large. It is thus beneficial to minimize the number of faults.

ATPG (Automatic Test Pattern Generation) is the process of generation patterns to test a circuit, which is described strictly with a logic-level netlist. ATPG algorithms inject a fault into a circuit, and then use a variety of mechanisms to activate the fault and cause its effect to propagate through the hardware and manifest itself at a circuit output. The output signal changes from the value expected for the fault-free circuit, and this causes the fault to be detected. Combinational circuit ATPG is considered to be a matured science by forty years development.

In this paper, several algorithms of ATPG for combinational circuit are introduced. And then the relationship between testability measures and fault coverage are also briefly discussed, finally the technical challenge and recent development for ATPG is discussed.

## 2. Algorithms of ATPG for combinational circuit

Automatic Test Pattern Generation (ATPG) is the process of generating a set of input patterns for a digital device that are sufficient to detect all physical defects in the device that may be present in it after it is manufactured.

### 2.1 D-algorithm

D stands for discrepancy and is a composite signal. It simultaneously denotes the signal value on the good machine and faulty machine according to the table. The D-calculus was developed by Roth in 1966 together with an ATPG algorithm. The symbol D (for detect) indicates the value of a node is a logic '0' in the good circuit and a logic '1'

in the bad circuit. We can also write this as  $D = 0/1$ . In general we write  $g/b$ , a composite logic value, to indicate a node value in the good circuit is  $g$  and  $b$  in the bad circuit. The complement of  $D$  is  $\bar{D} = 1/0$  ( $\bar{D}$  is rarely written as  $D'$  since  $D$  is a logic value just like '1' and '0'). Notice that  $D$  does not mean *not* detected, but simply that we see a '0' in the good circuit and a '1' in the bad circuit.

If we wish to propagate a signal from one or more inputs of a logic cell to the logic cell output, we set the remaining inputs of that logic cell to what we call the enabling value. The enabling value is '1' for AND and NAND gates and '0' for OR and NOR gates. The controlling value of '0' for an AND gate justifies the output to '0' and for a NAND gate justifies the output to '1'. The controlling values of '1' justifies the output of an OR gate to '1' and justifies the output of a NOR gate to '0'. To find controlling and enabling values for more complex logic cells, such as AOI and OAI logic cells, we can use their simpler AND, OR, NAND, and NOR gate representations.

## 2.2 A Basic ATPG Algorithm

A basic algorithm can generate test vectors automatically. We detect a fault by first activating (or exciting the fault). To do this we must drive the faulty node to the opposite value of the fault. Next we work forward from the fault origin and sensitize a path to a PO. This propagates the fault effect to the PO so that it may be observed.

We can visualize fault propagation by supposing that we set all nodes in a circuit to unknown, 'X'. Then, as we successively propagate the fault effect toward the POs, we can imagine a wave of  $D$ 's and  $\bar{D}$ 's, called the  $D$ -frontier, which propagates from the fault origin toward the POs. As a value of  $D$  or  $\bar{D}$  reaches the inputs of a logic cell whose other inputs are 'X', we add that logic cell to the  $D$ -frontier. Then we find values for the other inputs to propagate the  $D$ -frontier through the logic cell to continue the process.

This basic algorithm of justifying and then propagating a fault works when we can justify nodes without interference from other nodes. This algorithm breaks down when we have reconvergent fanout.

## 2.3 The PODEM Algorithm

The path-oriented decision making (PODEM) algorithm solves the problem of reconvergent fanout and allows multipath sensitization [Goel, 1981]. The method is similar to the basic algorithm we have already described except PODEM will retry a step, reversing an incorrect decision. There are four basic steps that we label: objective, backtrack, implication, *and*  $D$ -frontier. These steps are as follows:

- 1 Pick an objective to set a node to a value. Start with the fault origin as an objective and all other nodes set to 'X'.
- 2 Backtrace to a PI and set it to a value that will help meet the objective.

- 3 Simulate the network to calculate the effect of fixing the value of the PI (this step is called implication). If there is no possibility of sensitizing a path to a PO, then retry by reversing the value of the PI that was set in step 2 and simulate again.
- 4 Update the *D-frontier* and return to step 1. Stop if the D-frontier reaches a PO.

We can use intelligent procedures, based on controllability and observability, to guide PODEM and reduce the number of incorrect decisions. PODEM is a development of the D-algorithm, and there are several other ATPG algorithms that are developments of PODEM. One of these is FAN (fanout-oriented test generation) that removes the need to backtrace all the way to a PI, reducing the search time [Fujiwara and Shimono, 1983; Schulz, Trischler, and Sarfert, 1988]. Algorithms based on the D-algorithm, PODEM, and FAN are the basis of many commercial ATPG systems.

### 3. Testability and fault sampling

A testability measure can reveal the testability of a circuit before test generation. It can also identify the areas of poor testability in a circuit and to estimate the fault coverage of a circuit on a randomly generated test vectors. The controllability and observability measures are two useful concepts for testability analysis. Controllability gives us a measure of how easy or difficult it is to force the logic level in a node to either **1** or **0**. The observability of a node is defined to give a measure of how easy or difficult it is to propagate the logic level in a node to a directly observable output.

In order for an ATPG system to provide a test for a fault on a node, it must be possible to both control and observe the behavior of the node. A software program that measures the controllability and observability of nodes in a circuit is useful in conjunction with ATPG software. There are several different measures for controllability and observability [Butler and Mercer, 1992]. We shall describe one of the first such systems called SCOAP (Sandia Controllability/Observability Analysis Program) [Goldstein, 1979]. These measures are also used by ATPG algorithms.

Fault Sampling is to select a randomly subset (sample) of faults for simulation. Measured coverage in the sample is used to estimate fault coverage in the entire circuit. The advantage of fault sampling is saving in computing resources (CPU time and memory.) The disadvantage is the limited data on undetected faults.

Fault coverage is the percentage of faults covered by test vectors. Generally this coverage is over the set of all single stuck-at faults after it has been reduced by fault collapsing. A statistical theory of digital circuit testability is developed in Sharad c. et al's paper. A relationship between the average fault coverage and circuit testability is introduced and a method is presented to estimate circuit testability from fault simulation data collected in the normal course of test generation. The estimated testability take account of both the circuit topology and the characteristics of test vectors.

### 4. Technical Challenges

The general process of ATPG is a combinatorial search of the  $2^n$  search space (where  $n$  is the number of inputs to the [combinational] device) for a test pattern for a specific fault. For a sequential device, the search space is much larger:  $2^{(n+m)}$  (where  $n$  is the number of inputs, and  $m$  is the number of state variables, or flip-flops, in the device). A number of heuristics based on the topology of the gate-level circuit description can help the search process. These heuristics are well developed enough for combinational circuits that the test community deems the combinational circuit ATPG problem to be solved. ATPG for sequential circuits is another situation completely, however, and that is where the bulk of our efforts are centered.

As far as parallelization of the ATPG problem is concerned, there are five basic techniques: search space parallelization, fault partitioning, topological partitioning, heuristic parallelization, and algorithmic partitioning. For sequential circuit test generation, both processing power and memory space (across all processors) is a limitation. To perform ATPG on a 1 million gate + microprocessor, a machine of the class of the SP2 in Hawaii is most likely too small. One factor that we must keep in mind in the test community is that there is a real bias against results that are produced on those classes of large parallel machines because not every design engineer has access to them. We must try to develop algorithms that fit will on networks of workstations as well as large parallel machines.

Memory space is as important as processing power. One without the other is useless. Therefore, memory space must expand at least linearly with processing power. Most algorithms we are developing now are more latency limited than bandwidth limited. Since we are targeting networks of workstations as well as parallel machines, we try to reduce the effect of latency as much as possible, but the fact remains that lower latency makes us look better. The ideal machine for us would be able to pass medium-sized packets of data (approximately 1--10 kilobytes) with very low latencies regardless of the number of processors.

## 5. Recent Development

As the number of devices on integrated circuits continues to double every 18 to 24 months, balancing cost and quality has become increasingly difficult. Design-for-test and reducing the cost of manufacturing test are key to this balancing act.

Reducing cost-of-test without compromising product quality means reducing the volume of data needed to test each device to the highest possible coverage. ATPG can specifically target structural faults leading to much more compact test sets to arrive at equivalent test coverage. Furthermore, pattern compression and optimization techniques have advanced significantly in the last several years so that more efficient patterns and, thus, more compact test sets, are created. Two basic techniques are used by ATPG tools to compress manufacturing test vectors. Static compression techniques fault simulate a given pattern set to determine which patterns are redundant—they do not detect new

faults. Dynamic compression is a technique where multiple faults are targeted during test pattern generation itself. This can be an effective technique at reducing pattern count, but comes at the cost of increased test generation effort and CPU time. Both of these compression techniques have proven effective at significantly reducing the total number of test vectors needed to achieve equivalent test coverage. Combinations of both techniques can be used to optimize both run times as well as pattern count. Recent advances in test pattern compression continue to advance the capabilities of ATPG pattern compression and continue to reduce overall cost-of-test.

Pattern Optimization capability is a new approach to delivering the most effective test patterns. Pattern optimization takes a given pattern set and orders it from most effective to least effective pattern, ensuring that whether the pattern set has to be truncated or not, the patterns are delivered in the most effective manner. Pattern optimization ensures that if pattern sets must be truncated, the highest possible coverage is preserved. In cases where pattern sets do not need to be Truncated, pattern optimization can still ensure the lowest test cost since defective parts will tend to fail more quickly thus reducing the test time for failing devices and improving overall tester throughput. As gate counts continue to rise, Automatic test pattern generation and design-for-test techniques will play a key role in improving overall test quality as well as in reducing cost of- test. Automatic test pattern generation significantly reduces both the manual effort, as well as test set sizes, compared to testing with functional test patterns alone. Advanced pattern compression and Optimization techniques available with leading-edge ATPG tools also help to further reduce test costs as well as ensure optimal coverage and efficiency.

## References

- [1] <http://www-ee.eng.hawaii.edu/~msmith/ASICs/HTML/Book/CH14/CH14.5.htm>
- [2] [http://csserver.evansville.edu/~mr56/ece553/Lecture\\_9.pdf](http://csserver.evansville.edu/~mr56/ece553/Lecture_9.pdf)