ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

COMBINATIONAL CIRCUIT TEST GENERATION

"In order for the . . . operation of a test . . . to guarantee that a computing system has no faulty components, the test conditions . . . should be devised at the level of the components themselves, rather than at the level of programmed orders . . . This is the only way in which all conditions of operation of each logical function can be uniquely . . . defined and all logical components within each logical function can be made to perform the task to which they are assigned . . . thereby producing a minimum program which tests and detects failure . . ." — Richard D. Eldred, in 1959 paper, "Test Routines Based on Symbolic Logical Statements" [215].

With the above words, Eldred began the era of structural logic circuit testing at Datamatic. Roth's work at IBM marked the true beginning of systematic generation of tests for hardware faults in digital computers and laid the mathematical basis for test-pattern generation. Automatic test-pattern generation (ATPG) is the process of generating patterns to test a circuit, which is described strictly with a logic-level net list (schematic.) These algorithms usually operate with a fault generator program, which creates the minimal collapsed fault list (see Chapter 4) so that the designer need not be concerned with fault generation. In a certain sense, ATPG algorithms are multi-purpose, in that they can generate circuit test-patterns, they can find redundant or unnecessary circuit logic, and they can prove whether one circuit implementation matches another circuit implementation [409, 410, 717].

We first describe algorithms and representations needed by ATPG. We then introduce redundancy identification (RID), a very important benefit of ATPG algorithms.Controllability and observability testability measures (see Chapter 6) are used in all major ATPG algorithms. Finally, we present several key combinational ATPG algorithms, and show their behavior with examples.

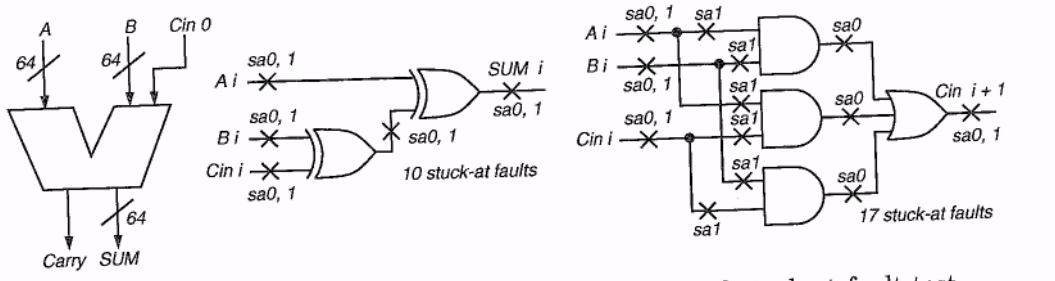


Figure 7.1: 64-bit adder: functional test vs. structural stuck-at fault test.

7.1 Algorithms and

We now discuss the broad categories of ATPG algorithms and the search space representations used by these algorithms.

7.1.1 Structural vs. Functional Test

First, we explain the difference between structural and functional test. El-dred [215] is credited with switching this field from functional to structural test generation. However, the

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

first publication of the stuck-at logic 0 (saO) or 1 (sal) fault for test generation was by Ga[illegible] Norby, and Roth in 1961 [237]. Later, Seshu and Freeman mentioned the stuck-at fault model for parallel fault simulation [587]. In 1963, Poage presented a theoretical analysis of stuck-at faults [523]. The first structural test method was used to test the Honeywell Datamatic 1000} a vacuum tube/diode second-generation mainframe computer.

Functional ATPG programs generate a complete set of test-patterns to completely exercise the circuit function. Figure 7.1 shows a 64-bit ripple-carry adder, and gives a very naive (and inefficient) logic design for one bit slice of the adder, which is sufficient to make our point. From a functional point of view, the adder has 129 inputs and 65 outputs. Therefore, to completely exercise its function, we need 2129 = 680,564,733,841,876,926,926,749, 214,863,536,422,912 input patterns, producing 265 = 36,893,488,147,419,103,232 output responses. The fastest automatic test equipment (ATE), at present, operates at 1 GHz. This ATE would take 2.1580566142 x 1022 years to apply all of these patterns to the circuit-under-test (CUT), assuming that the tester and circuit can operate at 1 GHz. Thus, we see that an exhaustive functional test is impractical, except for small circuits, and today most circuits tend to be huge.

Structural test, on the other hand, only exercises the minimal set of stuck-at faults on each line of the circuit, after discarding equivalent faults. If we use fault equivalence (see Chapter 4), then each bit-slice in the adder would only have 27 faults (see Figure 7.1), ignoring fault equivalence along the carry lines. This adder has no redundant hardware and the total structural fault list will have no more than ;

64 x 27 = 1,728 faults. So we need, at most, 1,728 test-patterns. The 1 GHz ATE would apply these patterns in 0.000001728 s, and since this test-pattern set covers j all possible structural stuck-at faults in the adder, it achieves exactly the same fault : coverage as the intractable functional test-pattern set described above. Frequently, the circuit designer will provide a limited subset of the functional test-patterns for the circuit, but those typically cover only 70 to 75% of the total number of faults. Testing for only 75% of the modeled failures is of limited value - it will catch only the most severe defects. Thus, we see the importance of ATPG algorithms. The vectors they produce supplement the functional test vectors from the designer to raise the stuck-at fault coverage to 98% or higher levels.

7.1.2 Definition of Automatic Test-Pattern Generator

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. Fault effects are propagated from an AND/NAND gate input to its output by setting other inputs to 1, a non-controlling value for AND/NAND. Fault effects are propagated from an OR/NOR gate input to its output by setting other inputs to 0, a non-controlling value for OR/NOR. Fault effects are propagated from an XOR/XNOR gate input to its output by setting all other inputs to 0 or 1 as is convenient.

E-beam testing [672] allows observation of internal circuit signals by “developing” a picture of the circuit that shows the internal nodes charged to logic 0 in one color and those charged to logic 1 in a different color. This eliminates the need to propagate fault effects to primary outputs (POs.) However, this method is impractically expensive, is only used for very specialized applications, and in a sense converts an intractable testing problem into another intractable image processing problem, since some mechanism must now look at a VLSI chip image and determine whether all signals are “colored” correctly. ATPG algorithms are extremely valuable, in that they propagate an abnormal voltage reading from the internals of the circuit to a PO, where an ATE can examine the voltage and determine whether it is correct.

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

Scan-Design for Microprocessor Testing. At present, the preferred method for testing at le parts of Intel Pentium™ and AMD K6™ microprocessors uses combinational ATPG. A scan-chain inserter adds special-purpose MUX and clocking hardware to every circuit flip-flop for testing, so that in scan mode, the flip-flops are converted into a giant shift register, and the entire state of the microprocessor can be shifted out through a special test-mode port called scan-out (see Chapter 14).

Similarly, a desired initial flip-flop state can be serially shifted into flip-flops through a special test-mode port called scan-in. This approach converts a difficult sequential circuit ATPG problem into a more tractable combinational circuit ATPG problem, at the expense of:

1. Using 5 to 20% of the chip area for the scan chain hardware in large chips.
2. Slowing down all flip-flops because of the added scan chain MUX delays.
3. Reserving one or more additional pins for scan chain control.

4. Lengthening the test-pattern sequence. This occurs because to set the machine having n flip-flops to any desired initial state requires n clocks of the scan chain. The application of the desired test-pattern requires 1 additional clock, followed by n additional clocks to read out the flip-flop state (in the event that the fault effect is captured in a flip-flop, rather than propagated to the circuit output.) For multiple tests, these can be overlapped (see Chapter 14.)

However, scan design coupled with combinational ATPG is the most popular test method with microprocessor and other VLSI chip designers, because it is very likely to generate a test set with close to 100% fault coverage. Besides, the test development time is predictable and can be accounted for in the new product introduction schedule. The state-of-the-art of sequential ATPG frequently causes major design delays, due to algorithm problems and uneatable hardware, and can delay a prod-uct introduction. In Chapters 14 and 16 we will cover scan design in greater detail, but we see that, at least for now, the combinational ATPG programs are extremely important.

7.1.3 Search Space Abstractions

All ATPG programs need a data structure describing the search space for test patterns.

Binary Search Trees. Consider the binary tree in Figure 7.2(b) for the circuit primary inputs (Pis) in Figure 7.2(a). Goel [256, 258] first used these trees in the combinational ATPG literature. The tree represents all eight choices for circuit input patterns. At the topmost node, if the left branch is selected, then signal A is set to 0 (the A branch), but if the right one is selected, then A is 1. At the second and third levels in the tree, subsequent values are selected for other circuit inputs, first for B and then for C. This covers all possible input patterns. The leaf nodes of the tree are labeled with the good machine output that the corresponding input values will cause. All ATPG algorithms implicitly search this tree to find test-patterns, and in the worst case, must examine the entire tree to prove that a fault is untestable. We wish to avoid a complete examination, because the number of tree leaves is:

nnumber-primary-inputs

and rises exponentially.

Binary Decision Diagrams. Any switching function can be completely described by the binary decision diagram (BDD), which was invented by Lee in 1959 [385].

The present discussion is based on the work of Akers who applied BDDs to solve the problem of testing [43, 44]. Figure 7.2© shows the BDD for the circuit of Figure 7.2(a). In order to read the diagram, we start at the topmost root node, and follow a path from that node to one of the two bottommost nodes, 0 or 1, which gives the circuit output value. The product of the Boolean literals along the path

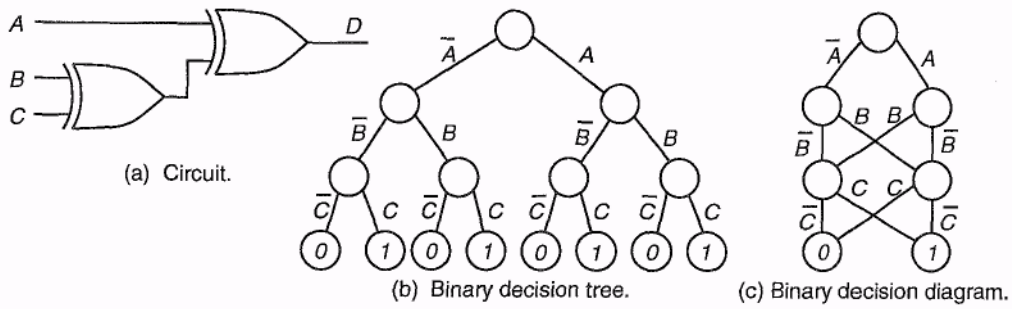


Figure 7.2: Different representations of a circuit.

gives a circuit maxterm or minterm, depending on whether we end up at the 0 or 1 output node. For example, the leftmost path in the BDD is A B C, which produces the circuit output 0, and

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

this is consistent with the circuit function. The rightmost path in the BDD is A B C, which produces the circuit output 1, also consistent with the circuit function. We can verify that all BDD paths are consistent with the logic function. BDDs have been used for ATPG [234, 635], but suffer from the problems of computational intractability, particularly for multiplier circuits. One observes vast changes in compute time, depending on the order in which circuit Pis are expanded in the BDD [104].

7.1.4 Algorithm Completeness

The notion of ATPG algorithm completeness means that in order to generate a test-pattern, the algorithm must ultimately be able to search the entire binary decision tree, if necessary, to generate a test-pattern even for a hard-to-test fault. If the fault is untestable, then after searching the entire tree no test is found. This means that the circuit behaves correctly even in the presence of that fault. It is important for an ATPG algorithm to be complete, or it may not attain the required fault coverage.

7.1.5 ATPG Algebras

The ATPG algebra is a higher-order Boolean set notation with the purpose of representing both the "good" and the "failing" circuit (or machine) values simultaneously. This has the advantage of requiring only one pass of ATPG to determine signal values for both machines. Since a test vector requires that a difference be maintained between the two machines, it is computationally fastest to represent both machines in the algebra, rather than maintaining them separately Roth [551] showed how multiple-path sensitization, required to test certain combinational circuits, could be done with his five-valued algebra given in Table 7.1.

Later, Muth [481] showed that in order to test finite state machines, the X symbol must be expanded to cover the cases where one among the good or failing machine values may be known, but the other machine value is unknown. Table 7.1 shows Muth's nine-valued algebra, which frequently benefits combinational circuit

Table 7.1: Roth's five-valued and Muth's nine-valued algebras.

Symbol	Meaning	Roth's 5-valued algebra		Muth's 9-valued algebra	
		Good machine	Failing machine	Good machine	Failing machine
D	(1/0)	1	0	1	0
$\overline{D}$	(0/1)	0	1	0	1
0	(0/0)	0	0	0	0
1	(1/1)	1	1	1	1
X	(X/X)	X	X	X	X
$G0$	(0/X)	–	–	0	X
$G1$	(1/X)	–	–	1	X
$F0$	(X/0)	–	–	X	0
$F1$	(X/1)	–	–	X	1

ATPG, as well [116]. For computing the response of a logic gate to input symbols of the algebra, we expand the symbols into the good machine/bad machine values given in column 2 of Table 7.1. We then independently compute the logic gate response for both machines and combine the output values back into the algebra.

7.1.6 Algorithm Types

We classify various types of ATPG algorithms, and present their complexity.

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

Exhaustive. In this approach, for an n-input circuit, we generate all 2" input patterns. For the reasons discussed above, this is infeasible unless the circuit is partitioned into cones of logic, each with 15 or fewer inputs. We can then perform exhaustive test-pattern generation for each cone. However, those faults that require multiple cones to be activated in a synergistic way during testing may not be tested.

Random - Used With Algorithmic Methods. In 1972 at the University of Illinois, Agrawal and Agrawal [26] suggested the use of random pattern generation (RPG) for testing. An essential part of the RPG scheme is a fault simulator that selects useful patterns (Figure 7.3.) While generating tests for the boards of the ILLIAC IV parallel computer they reported that the coverage of random patterns would often saturate between 60-80% and that switching to a D-algorithm based program at that point proved beneficial. This limitation of RPG, which relates to the testability of the circuit, was observed by Eichelberger et al. [210] for programmable logic arrays (PLAs.) It has been realized that patterns with equally likely Os and Is, as is used to start the RPG in Figure 7.3, may not be the best choice [15, 16, 20]. When the probabilities of Os and Is are different from 0.5, the patterns are called weighted random patterns (WRP.) Such patterns have been used by Schnurmann et al. [571], Waicukauski and Lindbloom [705], and several others. A method for finding an

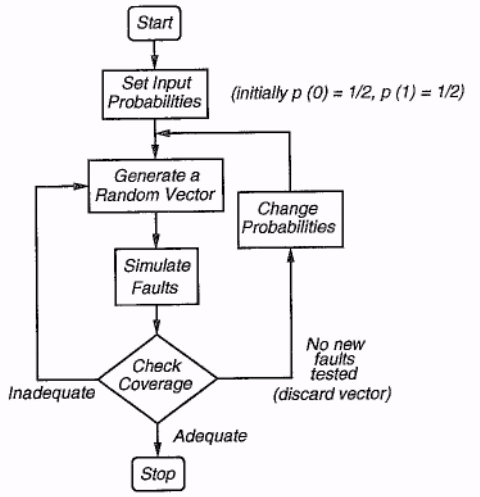


Figure 7.3: Random pattern generation method.

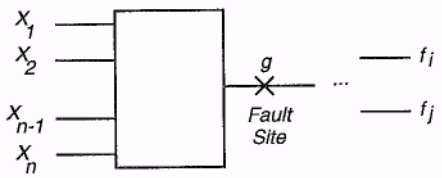


Figure 7.4: Boolean difference.

optimum set of probabilities for inputs is given by Wunderlich [739]. For application of RPG to sequential circuits, see the discussion of the simulation-based methods in Chapter 8. David's recent book [187] comprehensively covers this subject.

Symbolic - Boolean Difference. Sellers et al. [582, 583] use Shannon's Expansion Theorem to characterize Boolean circuits. For example, an arbitrary Boolean function F(X1,X2..., Xn) can be expanded about any variable, say X%, as:

$$F(X_1, X_2, \ldots, X_n) = X_2 \cdot F(X_1, 1, \ldots, X_n) + \overline{X_2} \cdot F(X_1, 0, \ldots, X_n)$$

Assuming a logic function: g = G(Xi,X2, • • •, Xn) for the fault site shown in Fig-ure 7.4, we express the outputs as:

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

$$f_j = F_j(g, X_1, X_2, \ldots, X_n) \quad ; \quad 1 \le j \le m \qquad (7.1)$$
$$X_i = 0 \ \ or \ \ 1 \ \ for \ \ 1 \le i \le n$$

Notice that these output equations express the circuit outputs in terms of the pri-many inputs Xi and the fault site signal g. Sellers et al. define the Boolean difference, or Boolean partial derivative, of a circuit as:

$$\frac{\partial F_j}{\partial g} = F_j(1, X_1, X_2, \ldots, X_n) \oplus F_j(0, X_1, \ldots, X_n) \qquad (7.2)$$

They express the fault

$$G(X_1, X_2, \ldots, X_n) = 1 \qquad (7.3)$$
$$\frac{\partial F_j}{\partial g} = F_j(1, X_1, X_2, \ldots, X_n) \oplus F_j(0, X_1, \ldots, X_n) = 1 \qquad (7.4)$$

detection requirements for g s-a-0 at output fj as:

Equation 7.3 says that to test a stuck-at-0 fault at g, the logic gate G must sensitize the fault site by driving it to logic 1. Equation 7.4 says that in order to detect the fault, the Boolean difference of some output with respect to the fault site g must be 1 (i.e., the output must change its signal value when the fault site signal switches from 1 to 0.) Unfortunately, due to high complexity the Boolean difference is not an efficient way to compute test patterns for large circuits.

Path Sensitlzation Methods. Path sensitization at the logic gate level of representation is currently the preferred ATPG method. The approach consists of three steps [550]:

I. Fault sensitization, in which a stuck-at fault is activated by forcing the signal driving it to an opposite value from the fault value. This is necessary to ensure a behavioral difference between the good circuit and the faulty circuit. Fault sensitization is also known as fault activation or fault excitation.

2. Fault propagation, in which the fault effect is propagated through one or more paths to a PO of the circuit. For some faults, it is necessary to simultaneously propagate the fault effect over multiple paths to test it. In general, the number of paths may rise exponentially in the number of logic gates in the circuit. Fault propagation is also known as path sensitization.
3. Line justification, in which the internal signal assignments previously made to sensitize a fault or propagate its effect are justified by setting Pis of the circuit.

In the second and third steps, we may find a conflict, where a necessary signal assignment contradicts some previously-made assignment. This forces the ATPG algorithm to backtrack or backup, i.e., discard a previously-made signal assignment and make an alternative assignment.

Consider the example in Figure 7.5 [550]. In all examples in this chapter, we will label Pis and POs with capital letters, and every other signal line in the circuit with a lower-case letter. Note that PI B (a fanout stem] fans out to two AND gates, whose outputs are h and i. The fanout branches from B to the inputs of the two AND gates are labeled f and g. It frequently happens that tests for faults on B are different from tests for faults on f, which are also different from tests for faults on g. That is why we must label every distinct line or wire of a signal net. We generate a test for B stuck-at-0. For fault sensitization, we set 5 to 1, and this leads to the signal assignments f = D and g — D (see Table 7.1.)

Fault propagation requires us to select among three scenarios: ,

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

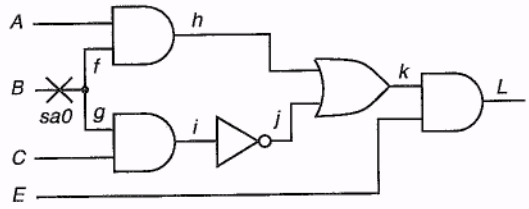


Figure 7.5: A combinational circuit example for path sensitization.

1. Propagation along the path f — h — k — L, or
2. Propagation along the path g — i — j — k — L, or
3. Simultaneous propagation along both paths f — h — k — L and g — i—j — k — L.

We choose path f — h — k — L for propagation. This means that for every AND gate along the path, the off-path inputs should be set to non-controlling values (1), and similarly for every OR gate along the path the off-path inputs should be set to 0. This results in the signal assignments A = 1, j = 0, and E = 1. Line justification now requires us to justify any internal signals where we assumed a value assignment. In this case, the only one is j. We can assign i = 1 to justify j — 0 by backwards logic simulation of inverter j. However, AND gate i then needs to have an output of 1, but it already has input g = D. Backwards logic simulation, using the 5-valued algebra (see Table 7.1), reveals that there is no way to get i = I when an input already was set to D. Therefore, we backtrack and retract the assignment j = 0 and try the alternative assignment j =1. However, this immediately blocks fault propagation along path f — h — k — L. We conclude that the only viable options may be scenarios 2 and 3 listed above.

We choose scenario 3. We change our fault propagation approach, and now make the assignments A = 1, C = 1, and E = 1 to ensure fault propagation. Forward logic simulation from these assignments yields i — D, h = D, j = D, and k = I, since D OR nD = 1. This is obtained by computing 1f0 OR 0f1 = 1f1. The D-frontier is the cut-set separating the circuit portion labeled with Xs from the portion labeled with D or D, where we include only the D or D signals closest to the outputs in the frontier. Finally, L = 1 and the D-frontier has disappeared at OR gate k, which means that the fault is untestable via these multiple paths. It only remains to return and try fault propagation along path g — i — j — k — L. We if set C = 1, h = 0, and E = 1 to propagate the fault. Forward logic simulation gives i=D,j=nD, k = nD, and L = nD. It remains to justify h = 0. This is achieved by backwards logic simulation for AND gate h, with input f = D, by setting input A= 0. The only test for B stuck-at-0 is ABCE = 0111, and this produces the output L = 0 in the good machine, and L = 1 in the failing machine.

This ATPG procedure is only correct for acyclic combinational circuits. We will discuss procedures for sequential circuits in Chapter 8. In particular, any circuit with feedback paths, flip-flops, or implicit latches expressed as combinational logic will ftequentry force this procedure into an infinite loop. Fault propagation and line

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

justification during ATPG consist of an intermixing of signal assignment operations, forwa logic gate simulation, backwards logic gate simulation, and backtracks.

Boolean Satisfiability and implication Graph Methods. The Boolean satisfiability problem means satisfying a Boolean expression or equation. An n-bit Boolean vector consists of a set of n binary variables, Xi, i = 1,2,. . .,n. Symbolically, a variable Xi or its complement Xi is referred to as a literal. The two-satisfiability (2-SAT) problem refers to finding a set of values for rEj's that will satisfy an equation of the tvpe:

$$\sum \alpha_k \beta_k = 0 \quad (\text{non} - \text{tautology}) \quad \text{or} \quad \prod (\alpha_k + \beta_k) = 1 \quad (\text{satisfiability}) \qquad (7.5)$$

where «& and f?& are any two literals, and summation and product are Boolean

OR and AND operations, respectively. A term or a factor in Equation 7.5 is called a Boolean clause or simply a clause. The 2-SAT problem is characterized by each clause having just two literals. The 2-SAT problem is solvable in polynomial time [190]. When the clauses in the Boolean expression contain three literals, the problem is known as the three-satisfiability (3-SAT) problem. The solution of 3-SAT has exponential time complexity.

Chakradhar et al. [121, 124, 130] and Larrabee [383, 384] have derived Boolean satisfiability formulations for the ATPG problem. Given a target fault, one derives an energy function of a neural network or a Boolean product of sums expression in terms of signal variables of the circuit such that any test for the target fault will minimize the energy function or satisfy the Boolean expression. The energy minimization is shown to be equivalent to a Boolean sum of products expression. The rest depends on finding efficient ways for solving the satisfiability problem.

These methods have been extended by others [291, 605, 648], and are now the fastest known ATPG algorithms for huge circuits. In these methods, the Boolean function of every logic gate is captured in equations that relate the input and output signals of the gate. Consider the signal relationships of the AND gate in Figure 7.6:

(7.6)

$$\begin{aligned} \text{If } a = 0, &\text{ then } z = 0 \\ \text{If } b = 0, &\text{ then } z = 0 \\ \text{If } z = 1, &\text{ then } a = 1 \text{ AND } b = 1 \\ \text{If } a = 1 \text{ AND } b = 1, &\text{ then } z = 1 \end{aligned}$$

For each constraint a cube is designed so that if signals are consistently labeled, that

cube will become 0. If any signal value around the logic gate is inconsistent with the gate function, then some cube will become 1. We simply sum up the cubes to obtain the following Boolean equation, where cubes are shown in order of the above conditions:

$$\bar{a}\, z + \bar{b}\, z + z\, \overline{ab} + a\, b\, \bar{z} = 0 \qquad (7.7)$$

which simplifies to:

$$\bar{a}z + \bar{b}z + ab\bar{z} = 0 \qquad (7.8)$$

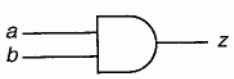


Figure 7.6: A two-input AND gate.

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

This equation is satisfied only when a, 6, and c assume values that are consistent with a function of the AND gate. The first two terms are 2-SAT terms and the third is a 3-SAT term. An alternative representation is called the pseudo-Boolean equation [124]. We convert a Boolean expression into a pseudo-Boolean form by replacing the Boolean OR and AND operators by arithmetic addition and multiplication, and treating signals as "real" variables that can assume values 0.0 and 1.0. Complementation x is replaced by 1.0 —x. For the AND gate, this form is:

$$F_{pseudo-Bool} = 2z + ab - az - bz - abz = 0.0 \qquad (7.9)$$

derived from Equation 7.8. A third representation is the energy function [126], obtained by letting the variables assume any real value in the range (0,1).

An alternative and easier way to derive the Boolean equation representation is by the Boolean false expression [125] defined as:

$$f_{AND}\,(a, b, z) = z \oplus (ab) = \bar{a}z + \bar{b}z + ab\bar{z} \qquad (7.10)$$

The expression on the right hand side in Equation 7.10 evaluates to logic 0 only when a, b, and z take values that are consistent with the AND function. The complement of fAJVD is called the truth expression or the satisfiability expression, which evaluates to logic 1 only when values of a, 6, and z are consistent with the AND function. These expressions can be derived for any complex Boolean function using the exclusive-OR definition of Equation 7.10. The Boolean false expression can be directly converted into the energy function of a neural network by replacing the Boolean operators with arithmetic operators. The variables then represent the states of neurons, which can either assume continuous values or discrete 0 and 1 values as in the Hopfield model [304]. Applications of the Hopfield model of neural networks to testing problems have been described in a book [127].

The non-intuitive aspect of these formulations is that logic gate outputs, as well as inputs, appear in the expressions. The advantage of this approach is that we can write an energy function for every logic gate (or Boolean function module) in a circuit, and then sum all of those functions into a single energy function for the entire circuit. If the function value is 0, then all signals are consistently labeled; otherwise, they are not.

A really efficient way to minimize the energy function or find satisfying variable assignrnents for the false or truth functions is the implication graph. This graph has a node for every literal. Thus, a Boolean variable x is represented by two nodes, x and nx. A node can be "true" or "false." For x = I, the x node assumes the true state. For x = 0, the x node becomes true. A two-variable "if ... then" clause is represented as a directed edge from the literal representing the "if" condition to the literal representing the "then" clause. The graph can then be transformed into

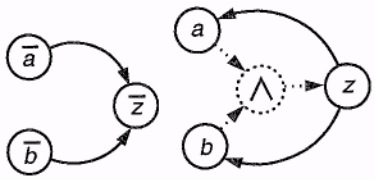


Figure 7.7: Implication graph (solid) and 3-SAT term (dotted) for 2-input AND.

a transitive closure [122] graph so that when a node is set to true, all reachable nodes are also set to true. This allows very efficient analysis of signal implications, because the transitive closure can determine more global signal relationships in the graph than other branch-and-bound search methods. Figure 7.7 (solid lines) shows the implication graph for the "if ... then" clauses of Equation 7.6. Note that only binary implications (those involving two literals) can be represented by an edge. The node with A (dotted lines) denotes an ANDing operator, and represents the 3-SAT term of Equation 7.8 (last clause of Equation 7.6) [291]. We cover these algorithms briefly in advanced topics (see Section 7.5.4.)

Computational Complexity. Ibarra and Sahni [317] analyzed the computational complexity of ATPG. They found that it is an NP-Complete problem, which means that no polynomial expression for the compute time function was found, and the problem is presumed to have exponential complexity. We will informally discuss how this arises. In the worst case, with no-pi inputs, there are 2no-pi different input combinations to try in depth-first fashion in the binary decision tree. When no_ff flip-flops are present in the circuit, there are potentially 4n0-ff different initial flip-flop states for ATPG to consider. This is because a flip-flop can be in either 0 or 1 state in the fault-free circuit and also in 0 or 1 state in the faulty circuit. Thus, the state-space of a flip-flop contains four elements (also see Section 8.2.) Finally, the work to forward simulate or reverse simulate all logic gates, as appropriate, rises proportionately to n, the number of logic gates. In the worst case, this work has to be done for all potential combinations of Pis and initial flip-flop states. The complete expression for the worst-case ATPG computational complexity is:

$$O(n \times 2^{no\_pi} \times 4^{no\_ff})$$

The above proof considers ATPG to be mathematically equivalent to the problem of Boolean satisfiability.

The entire history of ATPG algorithms has been a process of improving heuristic algorithms and procedures to (1) find all necessary signal assignments for a test as early as possible, and (2) search as little of the above decision space as possible.

The worst-case decision space is 2no-pl x ^no-ff. For logic simulation, the computational complexity is O(n). For combinational fault simulation, the complexity os 0(n2)[277], and for sequential fault simulation, the complexity is estimated to be between O(n2) and O(n3), based on empirical measurements. This means that, whenever possible, we will use fault simulation to avoid ATPG computations. For

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

Table 7.2: History of algorithm speedups.

Algorithm	Estimated speedup over D-Algorithm (normalized to D-ALG CPU time)	Year
D-ALG [551]	1	1966
PODEM [258]	7	1981
FAN [229, 232, 233]	23	1983
TOPS [360]	292	1987
SOCRATES [576]	1574† ATPG System	1988
Waicukauski *et al.* [708]	2189† ATPG System	1990
EST [110, 253]	8765† ATPG System	1991
TRAN [122]	3005† ATPG System	1993
Recursive learning [376]	485	1995
Tafertshofer *et al.* [648]	25057	1997

instance, we use RPG and fault simulation to get tests. When that fails, we use ATPG for hard-to-test faults. If we find a pattern for a fault, then we simulate that pattern against all remaining undetected faults, in the hope that we will "accidentally" test additional faults.

VLSI designers have become accustomed to analog circuit simulators (e.g., SPICE [163, 484, 486]), which let them model the actual defects in circuit behavior and see analog signal aberrations. The problems of analog modeling for ATPG are:

1. The huge number of different faults possible in large circuits.
2. The exponential complexity of the algorithm (i.e., for a sequential circuit with only 20 flip-flops, sequential ATPG may take days of computing.) Since sequential ATPG is slow at the Boolean level of representation, it will be even slower at the analog level of representation.
3. ATPG for transistor structures must model bidirectional and tri-state behavior (see Chapter 4.) Fault models and ATPG algorithms exist but are more complex than their logic gate counterparts [105, 287, 538]. Though there are test generators that can operate at the transistor level [173, 212, 250, 386, 389], the prevailing test methodology continues to rely upon gate-level stuck-at faults.

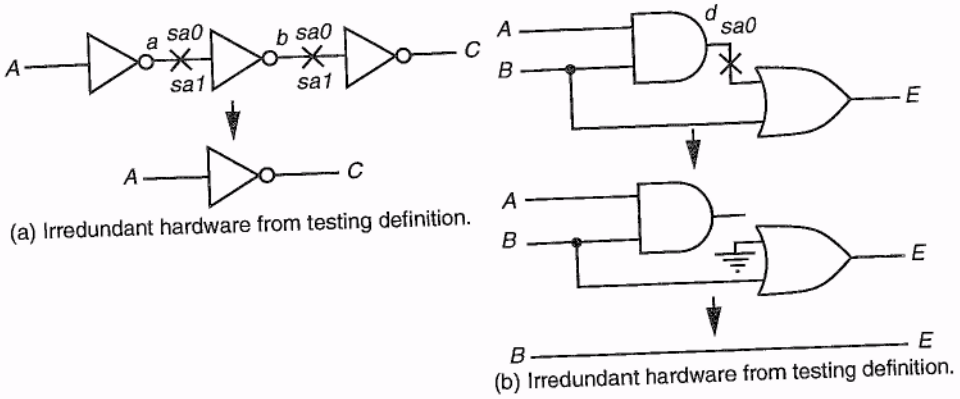
Table 7.2 shows the history of accelerating combinational ATPG. The speedups in the table are very approximate, and should be treated as order-of-magnitude estimates, due to the difficulty in normalizing the CPU times of the older CPUs on which the earlier algorithm experiments were run, and due to implementation differences. The earliest and last two table entries are not ATPG systems (see Section 7.6), but the middle entries are. The ATPG system, using random-pattern generation, has an unfair advantage over the pure algorithm execution experiments.

Also, the TRAN algorithm and the algorithm of Tafertshofer et al. perform much

better than the other systems on particularly huge circuits, which is not reflected

Table 7.3: Test vectors for all circuit faults.

Fault	Test	Response (good/failing)
a sa0	$A = 0$	D
a sa1	$A = 1$	$\overline{D}$
b sa0	$A = 1$	$\overline{D}$
b sa1	$A = 0$	D



(a) Irredundant hardware from testing definition.

(b) Irredundant hardware from testing definition.

Figure 7.8: Redundancy definition for testing purposes.

by these benchmark results. We see that improvements come slowly, and that they have barely kept pace with Moore's Law [477, 478].