Test Automation - Automatic Test Generation Technology and Its Applications

Kwang-Ting (Tim) Cheng and Angela Krstic
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106

1. Introduction

Test development has been known as a tedious and time-consuming process. For complex designs, the process can sometime stretch over several months and become a bottleneck for product time-to-market. In the past two decades, various test development automation tools have been introduced and gradually accepted by designers and test engineers to automate dozens of tasks that are essential for developing adequate tests. The test development automation tools can generally be classified into four categories: design-for-testability (DFT) tools, test-pattern-generation tools, pattern-grading tools and test program development and debugging tools.

DFT tools are used early in the design process and are often used by the design engineers instead of test engineers. Tools in this category include testability analysis tools, design-rule checkers and test-structure insertion tools. Testability analysis tools report areas of a design that could cause testability problems (e.g., areas that have poor controllability or poor observability) to guide the redesign for improved testability. The test structure insertion tools automatically augment and/ or modify the HDL design or netlist to include the boundary scan, internal scan, memory Built-In Self-Test (BIST) or logic BIST structures. Before the test structure is inserted, the design rule checker examines the design against some test-structure-specific testability design rules. Different test structures (boundary scan, internal scan, memory BIST and logic BIST) impose slightly different design rules. Violations of the testability design rules have to be fixed before test structure can be inserted.

Test pattern generation tools attempt to automatically generate high-quality tests for specified fault model(s). Automatic Test Pattern Generation (ATPG) is one of the most difficult problems for electronic design automation (EDA) and is a core technology for test. It has been a popular research topic for over 30 years for both theoreticians as well as practicing tool developers. The ATPG problem is a complex search problem as well as a non-linear optimization problem. The objectives of the research have been primarily in improving the scalability of the algorithm, in adapting the algorithms to handle various fault models, in taking into account some practical issues such as dealing with busses, dynamic logic, and others that do not have equivalent Boolean models, in extending the algorithms beyond Boolean domains to handle circuits and fault models

directly in higher (RTL or even behavior-level) or lower (transistor) levels of abstraction. The objectives of optimization could be length of test sequence, the power consumption caused by test sequence, etc. In Section 2, the ATPG technology will be reviewed in more details.

For a given netlist and a specified fault model (e.g., the stuck-at fault, the bridging fault, the transistor short/open faults, the transition fault or the path delay fault), the fault grading tool derives the fault coverage figure of a set of tests. The tool also reports the faults not detected by the tests such that new tests for undetected faults can be further developed. Although modern test pattern generators include fault simulators, free-standing fault simulators can still be useful. Such simulators grade functional vectors (developed by designers for design verification) that provide some fault coverage without additional test generation effort. Functional vectors, typically applied at the circuit's operational speed, often catch timing-related faults that may not be detected by scan-test vectors generated by ATPG tools. For complex designs without DFT, fault grading could be a very CPU-intensive task. Hardware accelerators have been developed and used to speed up this process.

As the design trends are moving towards nanometer technology, new ATPG problems are emerging. The impact of noise effects such as crosstalk and power supply noise on the design reliability and performance can no longer be ignored during design validation. Current modeling and vector generation techniques are oversimplified and cannot take noise effects into consideration. New techniques have to be developed such that they can include accurate timing information while at the same time capable of handling increasingly larger designs. These techniques need to validate new design corners that capture worst-case design scenarios, including noise effects. New fault modeling and ATPG techniques are also needed for manufacturing testing since, for nanometer technology, many of today's design validation problems will soon become manufacturing test problems as well.

The ATPG technology, in addition to generating high-quality tests for various fault models, offers efficient techniques for analyzing the logic networks. The technology has been successfully applied in several other areas of electronic design automation such as logic optimization, design verification and timing analysis. The problems of interest in these applications are transformed and modeled as a search problem for a test of a stuck-at fault or a delay fault. These applications will be discussed in this article.

In the rest of the paper, we first briefly review the ATPG technology. We then address some new issues of test generation and test automation for nanometer technology. Finally, we discuss other applications of ATPG technology in electronic design automation.

2. ATPG Technology

The objective of ATPG is finding an input sequence that detects all modeled faults, i.e., an input sequence which when applied to the circuit can distinguish between the correct circuit and

any circuit with a modeled fault. The effectiveness of the test sequence is measured by the achieved fault coverage for the given fault model and the number of generated vectors (test application time is directly proportional to the number of test vectors) [1]. In addition to high test effectiveness, an ideal ATPG process is also required to have a low cost of test generation. The cost of ATPG is affected by the fault model used (stuck-at, bridging, transistor open/shorts, functional, delay faults, etc.), type of the circuit under test (combinational/full scan circuit, synchronous sequential circuit or asynchronous sequential circuit), level of abstraction used to represent the circuit under test (gate-level, RT-level, switch-level) and the required test quality.

The ATPG process for a given target fault consists of three phases: fault initialization, fault activation and fault propagation. During fault initialization phase the signal values required for fault activation are specified. In the fault propagation phase, the fault effect is propagated to primary outputs. Two basic operations in ATPG process are justification and implication of signal values. Justification is a process of assigning values to gate inputs when the logic value of the gate output is known. It is a recursive process that ends when the primary inputs are reached. There might be many different ways to justify a given signal value. For example, consider a two-input AND gate with value 0 at the output. There are three different ways for justification: assigning a 0 to the first input and 1 to the second input, assigning a 1 to the first input and a 0 to the second input or assigning value 0 to both inputs. Therefore, justification involves a decision making process. The new values assigned during justification process can uniquely determine the values on some other signals. This process is called implication. Due to the presence of reconvergent fanouts in the circuit, the assignment of new signal values during implication can conflict with the values of these signals assigned in earlier steps of ATPG. If this happens, the effects of the last decision have to be reversed and a new decision has to be made to allow the ATPG process to continue. This process is called backtracking. The decision making and backtracking strategies have a strong impact on the efficiency of the ATPG process.

The above process can be directly applied for combinational circuits or full-scan circuits in which all inputs of the combinational part of the circuits (primary inputs and state variables) can be assigned arbitrary values and the fault effect can be observed on any output (circuit outputs and state variables). On the other hand, test generation for sequential circuits without full scan is more involved due to the fact that the state lines cannot be directly controlled and observed. Figure 1(a) illustrates the Hoffman model of a sequential circuit. Symbols PI, PO, PS and NS are used to represent primary inputs, primary outputs, present state lines and next state lines, respectively. Symbol CLK represents the clock. In any clock cycle c_i, arbitrary test values can only be assigned to the primary inputs of the circuit while the values on the next state lines depend on the values on the present state lines at the end of the clock cycle c_{i-1}. Therefore, the operation of a sequential circuit can be represented using an iterative array of combinational logic, as shown in Figure 1(b). Each copy of a combinational logic is called a time-frame. The present state values in time-frame *k* correspond to the next state values in time-frame *k-1*. Due to time-frame unfolding, a single fault in the original sequential circuit is represented by a multiple fault in the iterative logic array

model. While in the combinational/full-scan circuits all three phases of the test generation process can be performed with a single input vector (or a single input vector pair for some fault models), testing a fault in a sequential circuit requires a vector sequence. This is because, fault initialization and fault propagation may require several clock cycles, i.e., the circuit may have to go through several time-frames. If activating the fault requires assignment of signal values on some present state lines, the fault initialization has to be carried out such that these present state lines are justified by assigning signal values in the previous time-frame. This process continues until no signal assignment on the present state lines is required. Similarly, after the fault has been activated, it may not be possible to propagate the fault effect to the primary outputs in the same time-frame, i.e., it may take several time-frames for the fault effect to become observable on the primary outputs.

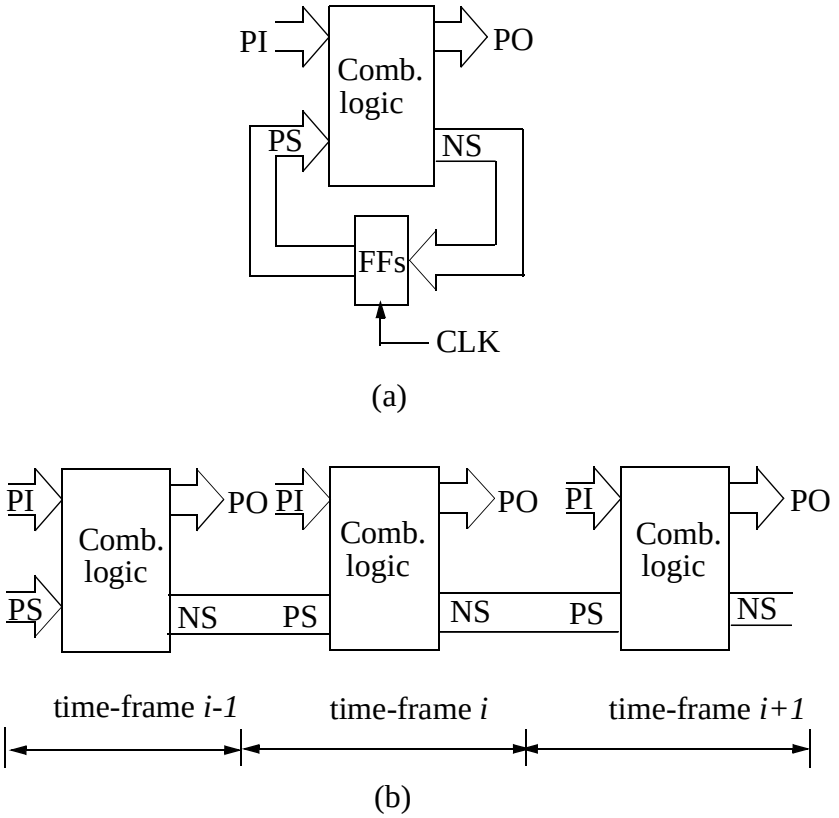


Figure 1. Models for sequential circuits: (a) Huffman model and (b) iterative array model.

There are several issues affecting the complexity of sequential ATPG. (1) Handling highly sequential circuits might not be possible due to the large number of time-frames required. Consider a 20-bit counter. Detecting the stuck-at-0 fault at the most-significant-bit (MSB)would require a sequence of 2^{19} vectors (to set the MSB to 1) and thus this many time frames would be needed in ATPG. Such a small but highly sequential circuit could hardly be handled by any existing logic-level ATPG tool. Therefore, sequential ATPG is by no means a complete solution

for highly sequential circuits. However, sequential ATPG combined with lower-cost design-for-testability techniques such as partial scan design offers an attractive, alternative test solution. Interested readers can refer to [4] for details. (2) Propagation delay cannot be accurately modeled and incorporated in the ATPG process and that may cause problems for circuits with combinational asynchronous feedback loops and for multiple-phase or multiple-clock designs. Even though an iterative array model can still be constructed for such circuits, the produced test sequences may cause races and hazards and become invalid. A practical engineering solution to this problem is to use a simulator (which can model delay and timing accurately) to verify the generated tests and to filter out the tests causing races and hazards. (3) ATPG tools need to ensure that the produced test sequence will not cause bus contention. ATPG-produced tests are non-functional and may cause undesired properties such as bus contention if special care is not taken. ATPG tools usually modify the logic model for test generation to ensure that the produced test can not only detect the target fault but also will not cause bus contention. This point will be further elaborated in Section 4.

In search of high defect coverage, many different fault models have been proposed and automatic test pattern generation techniques tailored to these fault models have been developed. These ATPG techniques operate on different design description levels and some techniques operate on more than one level of abstraction. Next, we describe some of the representative ATPG techniques for different fault models.

2.1. ATPG for single stuck-at faults

The D-algorithm [12] is one of the earliest ATPG algorithms for single stuck-at faults in combinational or full-scan circuits. The D-algorithm uses five-value algebra for test generation: 0, 1, X, D and $\overline{D}$. Values 0, 1 and X are used to represent logic 0, logic 1 and don't care values, respectively. Value D is used for representing value 1 in the fault-free circuit and value 0 in the faulty circuit (D=1/0). Similarly, $\overline{D}$ = 0/1. Values D and $\overline{D}$ behave like Boolean variables and can be used for activating and propagating the fault effect. In D-algorithm, justification of signal values involves assignment of values on internal circuit signals (signals other than primary inputs) on which many available choices might exist. Therefore, if conflicts occur, the backtracking process can be inefficient for large circuits. A more efficient decision making process was proposed almost 20 years ago in the PODEM algorithm [7]. PODEM only allows assignment of values at primary inputs (PIs) and these values are then propagated towards internal lines using implications. After the initial assignment of values on some primary inputs, and implication of these values, the process continues by assigning additional PI values and checking if the fault effect has been propagated to the primary outputs. If at any stage, the fault cannot be excited or the error cannot be propagated further, the ATPG process backtracks to the most recent PI assignment and changes it. Testability analysis is used to guide the choice of the PIs and values assigned to them in order to heuristically maximize the chance for a successful search. Various further accelerations of combinational ATPG have been introduced in the past 20 years making it

a very mature research topic [6]. These improved algorithms reduce the number of backtracks and the processing between backtracking.

Most of the work on sequential ATPG has been done on the gate level, under the assumption that the initial state of the circuit is unknown. Most of the approaches used in practice are based on the time-frame expansion technique as discussed above. Another class of ATPG approaches, named simulation-based approaches, is enhanced from a logic simulator or a fault simulator and has gained increasing interests. These approaches are combinations of simulation and cost function guidance. The basic concept of these approaches is as follows: Suppose we wish to generate a test for a given fault or set of faults. Based upon the previous tests, a new trial vector or a trial sequence is generated. Logic simulation or fault simulation is then performed for this trial vector/sequence. From the simulation result, a cost function, that, in some way, determines how *far* the state of the circuit is from the required state for the detection of the fault(s), is computed. If the cost is reduced, the new trial vector/sequence will be included in the final test sequence, otherwise it will be discarded. A new trial vector is then generated and the process repeats until the fault is detected or the number of trial vectors reaches a pre-specified limit. The differences between various methods in this class of approaches are in (1) the ways to generate new trial vectors and (2) the cost functions for guiding the search. Most simulation-based methods consist of multiple phases and different phases use different cost functions to achieve different objectives. The main problem with these approaches is that they are not complete, i.e., even though a test for some fault exists, these approaches might not be able to find it, even with an infinite amount of ATPG time. Also, simulation approaches cannot identify untestable faults for which there exists no test.

2.2. ATPG for bridging and transistor open/short faults

Even though it has been shown that tests for single stuck-at faults can detect many other types of faults, test data have demonstrated that testing only these faults is not sufficient to cover some defects in VLSI designs. Therefore, ATPG techniques have been developed for testing other fault models. Bridging and transistor open/short fault models fall into the category of technology dependent, switch-level fault models. Due to a very large number of possible faults usually only a subset of most likely faults is selected for testing. Bridging faults model shorting of adjacent lines in the design layout. The faulty value is identical on shorted lines and it represents AND or OR function of shorted signals. Some bridging faults can introduce feedback loops into a combinational circuit and result in a sequential behavior. Transistor open/short fault models assume that a transistor is permanently non-conducting/conducting. Transistor open faults in combinational circuits can also result in a sequential behavior. Detection of transistor open faults in CMOS requires application of a test vector pair. With appropriate fault behavior modeling, tests for bridging faults and transistor open/short faults can be generated using adapted stuck-at fault test strategies. Most of the ATPG techniques for testing bridging faults and transistor open/short faults can handle only combinational and full-scan circuits.

2.3. ATPG for delay faults

The increasing circuit operating frequencies and demands for low cost and high quality require that the temporal correctness of the circuit can be guaranteed. For high performance circuits with aggressive timing requirements, small process variations can lead to failures at the design clock rate. These defects can stay undetected after at-speed or stuck-at fault testing but can be detected using delay tests. Testing delay defects in combinational circuits requires application of a vector pair.

Unlike ATPG for stuck-at faults, ATPG for delay faults is closely tied to the test application strategy. This means that before tests for delay faults are derived, it is necessary to know how these tests will be applied to the circuit. The testing strategy depends on the type of the circuit (combinational, scan, non-scan or partial scan sequential circuits) as well as on the speed of the test equipment. Ordinarily, testing delay defects requires that the test vectors be applied to the circuit at its intended operating speed. However, since high speed testers require huge investments, testers currently used in test facilities could be slower than the new designs that need to be tested on them. Testing high speed designs on slower testers requires special test application and test generation strategies. This topic is now being investigated by several research groups.

Since application of an arbitrary vector pair to the combinational part of a circuit is not possible for a non-scan or even full scan sequential circuit, ATPG for delay faults in sequential circuits is significantly more difficult than ATPG for combinational circuits. Various testing strategies for sequential circuits have been proposed but it still remains a topic needing breakthroughs. A survey of combinational and sequential ATPG techniques for delay faults can be found in [11]. It is generally believed that some form of design-for-testability is required in order to achieve high quality delay testing for sequential circuits.

3. New test generation issues for nanometer technologies

The trend of moving toward the nanometer technology introduces new failure modes and a new set of design and test problems. Continuous shrinking of device feature size, increased number of interconnection layers and gate density, increased current density and higher voltage drop along the power nets give rise to the “noise faults” such as power supply noise or crosstalk. These faults can cause logic errors or excessive propagation delays, i.e., circuit performance degradation. Detection of errors caused by noise effects needs to be addressed in both design verification and manufacturing testing phases.

When coupled with process variations, noise effects have been shown to result in new worst-case design corners that need to be identified and checked in the design validation phase. However, detection of these worst-case design corners is extremely difficult due to the fact that noise effects are highly input pattern and timing sensitive. Simulation of designer-derived functional vectors and identification of problem areas using timing analysis that cannot consider

the impact of noise effects on the propagation delays are not sufficient for the purpose of design validation. The new methodologies for generating validation vectors that can excite the worst-case design corners require integration of accurate timing information into the process of vector derivation [9]. For manufacturing testing, ATPG techniques also have to be augmented and adapted to new failure conditions introduced by the nanometer technology. These failures obviously cannot be detected by tests targeting the conventional fault models (e.g., stuck-at faults, logic-level delay faults). New test methodologies have to be capable of checking for the worst-case noise scenarios. In other words, new test vectors are required to, in addition to sensitizing the faults and propagating the fault effects to the primary outputs, also produce the worst-case noise effects. Also, they have to be capable of handling increasingly larger designs.

ATPG for maximum power supply noise. For a highly integrated system-on-a-chip, more devices are switching simultaneously and this results in increased power supply noise. The power supply noise consists of two components: inductive ΔI noise and power net IR voltage drop. The inductive ΔI noise is caused by the change of instantaneous current on either the package lead inductance or on wire/substrate inductance and it is proportional to $L \cdot \frac{di}{dt}$. The IR voltage drop is caused by instantaneous current through the resistive power and ground lines. The ΔI noise can cause a voltage glitch on the power/ground lines and result in timing and/or logic errors. Similarly, large voltage drops through the power supply lines resulting from high instantaneous current and high current density can lead to short or open circuits due to electromigration. Activating these defects and propagating them to the primary outputs requires carefully selected test vectors.

In addition to affecting the reliability of designs, power supply noise can also affect the performance. The power supply noise reduces the actual voltage level reaching a device, which in turn, leads to increased cell and interconnect propagation delays. This results in increased signal arrival times at the primary outputs and the next state lines and causes performance degradation. SPICE simulations show that a 10-15% voltage drop during the cell transition period can cause a 20-30% increase in cell propagation delay. If the cell is a clock buffer or is in the timing-critical path, this delay deviation could cause serious clock-skew problems or a non-trivial increase in the critical path delay. To guarantee the performance of the designs these effects have to be detected by, for example, applying delay tests. However, most of the existing delay techniques are based on simplified, logic-level delay fault models and cannot be directly used to model and test timing defects in high-speed designs based on deep submicron technologies. New delay testing strategies are needed to close the gap between the logic-level delay fault models and the physical defects. The new tests have to be derived such that they produce the worst-case power supply noise along the sensitized paths and therefore, cause the worst-case propagation delays on these paths.

ATPG for worst-case crosstalk effects. The increase in design density in deep submicron designs leads to more significant interference between the signals due to capacitive coupling, i.e., crosstalk. Crosstalk can induce Boolean errors as well as delay faults. Therefore, ATPG for

worst-case crosstalk effects needs to produce vectors that can create and propagate crosstalk pulses as well as crosstalk induced delays.

Crosstalk induced pulses are likely to cause errors on hazard sensitive lines such as inputs to dynamic gates, clock, set/reset and data inputs to flip-flops, etc. The effects of crosstalk pulses can be logic errors or degradation of the voltage levels which further result in longer propagation delays. The goal of ATPG for worst-case crosstalk pulse is to generate a pulse of maximum amplitude and width at the fault site and propagate its effects to primary outputs with minimal attenuation [3].

Studies have shown that increased coupling effects between signals can cause a significant increase or decrease of signal delays. As an illustration, consider the simulation model shown in Figure 2[3]. The affecting line (A) and the victim line (V) are capacitively cross-coupled. SPICE

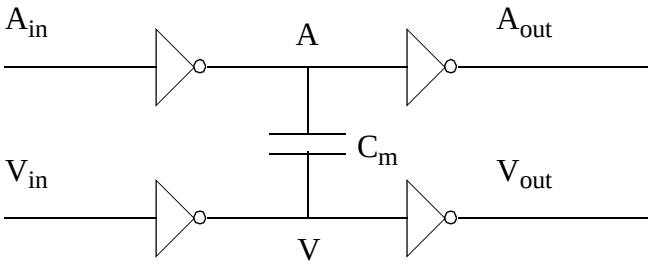


Figure 2. Simulation model for crosstalk.

simulations show that when A_{in} and V_{in} have transitions in the same direction at the same time or within a small time interval, a speedup occurs on both lines. Also, when A_{in} and V_{in} have opposite transitions at the same time or within a small timing interval, a slowdown occurs on both lines. Speedup and slowdown are measured with respect to nominal delay that corresponds to the situation in which one of the lines is static and the other has a transition. Both speedup and slowdown can cause errors. Signal slowdown can cause delay faults if a transition is propagated along paths with small slacks while signal speedup can cause race conditions if transitions are propagated along short paths. To be able to guarantee the performance of the designs, ATPG techniques have to consider the worst-case crosstalk impact on the propagation delays.

4. Other Applications of ATPG engine

The ATPG technology, undoubtedly offering efficient techniques for analyzing the logic networks, has successfully been applied in several other non-test areas of IC design automation. In the following, we give a summary of its applications in logic optimization, logic equivalence checking, design property checking and timing analysis.

4.1. Logic Optimization

Redundancy Removal. The ATPG concepts and techniques have been successfully applied for logic optimization. The main link between the test and the logic optimization worlds is the concept of redundancy. The existence of undetectable faults implies the existence of redundant logic. If the fault effect cannot be observed at a primary output, then the behavior of the fault-free circuit is equivalent to the behavior of the faulty circuit. Thus, the faulty circuit is a valid implementation of the fault-free circuit. For a stuck-at fault, the faulty circuit is obtained by substituting the faulty wire by a constant value (0 or 1). This operation, called redundancy removal, involves the removal of the faulty wire along with all the logic driving it. It removes redundant logic by using ATPG techniques to efficiently identify redundant faults. Because this method only removes logic from the circuits, the size of the circuit reduces after redundancy removal, the topological delay of the longest paths will be shorter than or at most equal to that of the original circuit and the power dissipation of the optimized circuit will also be lower.

Consider the circuit in Figure 3(a). The stuck-at-0 fault at the output of gate *G* is undetectable

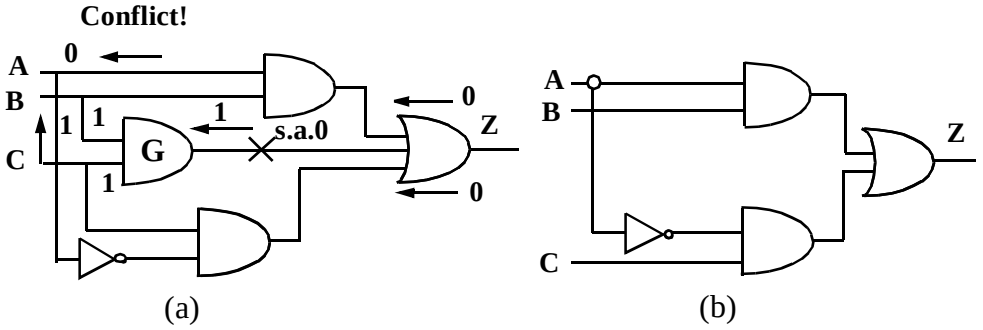
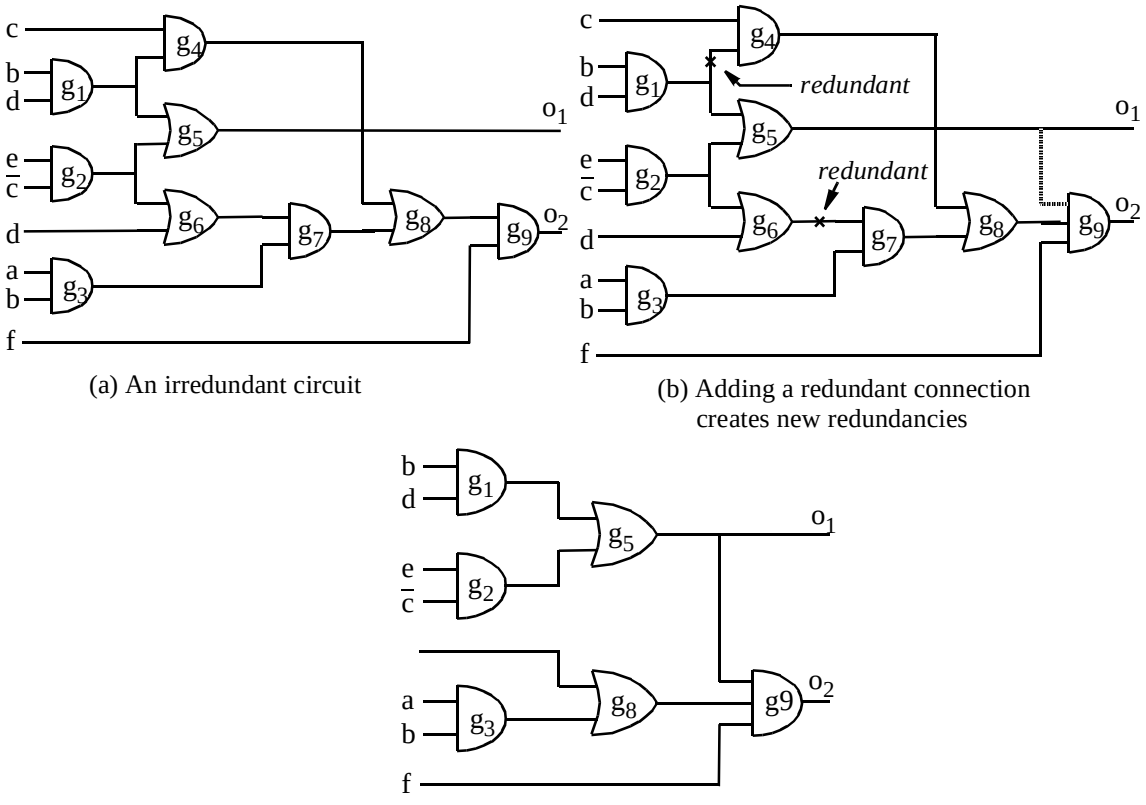


Figure 3. An introductory example of redundancy removal.

(this can be easily identified by simple implication as shown in the figure). Therefore, this wire can be fixed to logic 0 without changing the function of output *Z*. Therefore, it can be removed and the resulting circuit is shown in Figure 3(b). Notice that output *Z* of the circuit in Figure 3(a) is hazard-free but the one in Figure 3(b) may have glitches. Redundancy may be required for glitch-free designs. Redundancy can be removed only if having glitches is not a concern (e.g., synchronous design).

When a stuck-at-1(0) fault is identified as redundant, we simply set the faulty net to logic 1(0) and then recursively apply the following rules: (1) If an input of a gate has a constant non-controlling value, remove the input. (2) If an input of a gate has a constant controlling value, remove the gate and its inputs, propagate the constant to each of the gate's fanouts. (3) If a gate becomes floating and has no fanout, remove the gate and its inputs.

Logic Restructuring and Redundancy Addition and Removal. After removing a redundant fault, some other faults that were redundant in the original circuit may become irredundant and faults that were not redundant originally may now become redundant. While this fact complicates



(a) An irredundant circuit

(b) Adding a redundant connection creates new redundancies

(c) The final optimized circuit

Figure 4. An example of redundancy addition and removal.

the process of Redundancy Removal, it can be exploited to obtain stronger optimization methods. Even for an irredundant circuit, redundancies may be added to create new redundancies elsewhere in the circuit. By removing the created new redundancies, an optimized circuit may be obtained. Such a technique is named as Redundancy Addition and Removal or logic restructuring. For example, consider the circuit given in Figure 4(a). This circuit is irredundant. If we add a connection from the output of gate g_5 to the input of gate g_9 (shown as a dashed line in Figure 4(b)), the functionality of the circuit does not change. In other words, the added connection is redundant. However, the addition of the connection causes two originally irredundant wires to become redundant as shown in Figure 4(b). After removing these two wires and associated gates that either become floating (g_6) or have a single fanin (g_4 and g_7), the circuit can be greatly optimized as shown in Figure 4(c).

Efficient algorithms for finding effective Redundancy Addition and Removal transformations (e.g., [5]) have been proposed in the past few years that demonstrate the potential of this approach. By properly orienting the search for redundancy, these techniques can be adapted to target several optimization goals.

Just as combinational test generation extends to sequential test generation with the use of

time-frame expansion, ATPG-based logic optimization methods can naturally be extended to perform sequential transformations. The sequential transformations offered by Redundancy Removal and Redundancy Addition and Removal can be derived directly at the structure level without constructing the state transition information as required by most other approaches. The capabilities provided by these methods introduce a breath of fresh air into an area that, in many aspects, is widely considered as mature.

4.2. Design verification

Logic equivalence checking. During the design process, it is necessary to check the equivalence of two designs described at the same or different levels of abstractions. For example, checking the functional equivalence of the optimized implementation against the RTL specification is critically important in order to guarantee that no error is introduced during the logic synthesis and optimization process, especially when human effort is involved in the process. Similarly, checking the equivalence of the gate-level implementation versus the gate-level model extracted from the layout can assure that no error is made during the physical design process.

Traditionally, checking the functional equivalence of two Boolean functions is accomplished by constructing their canonical representations, e.g., truth-tables, or BDDs. Two circuits are equivalent if and only if their canonical representations are isomorphic.

A joint network of two Boolean networks under comparison can be formed by connecting the corresponding primary input pairs of the two networks together and connecting the corresponding primary output pairs to XOR gates as shown in Figure 5. The outputs of these XOR-gates are the

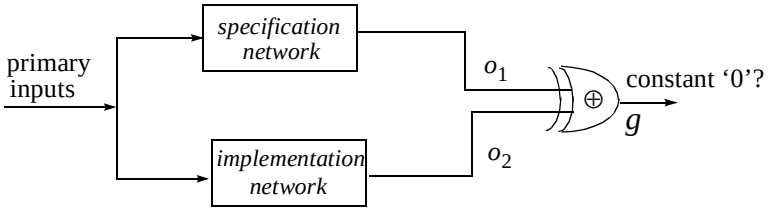
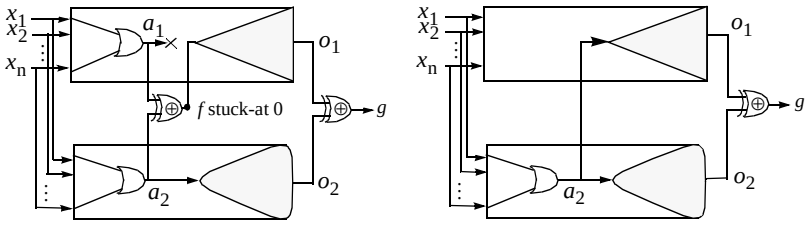


Figure 5. Joint network for equivalence checking.

new primary outputs of the joint network. The functional equivalence of the two networks can be asserted if the primary output response of the joint network is '0' for any input vector. Therefore, proving the equivalence of two circuits is reduced to proving that no input vector produces '1' at this model's output signal *g*. Instead of constructing a canonical representation of this joint network, equivalence checking can also be formulated as a search problem that searches for a *distinguishing vector*, for which the two circuits under verification produce different output responses. If no distinguishing vector can be found after the entire space is searched, the two circuits are proven equivalent. Otherwise, a counter-example is generated to disprove the equivalence. Since a distinguishing vector is also a test vector for the joint network's output *g* stuck-at-0 fault, equivalence checking becomes a test generation process for *g* stuck-at-0 fault. However, directly applying ATPG to check the output equivalence (i.e., finding a test for output *g*

stuck-at-0 fault) could be CPU-intensive for large designs. The complexity can be substantially reduced by exploring the internal functional similarity between the two circuits under verification [2]. By using naming information or structure analysis, a set of potentially equivalent internal signal pairs can first be identified. Then a model like the one shown in Figure 6(a) can be constructed, where signals a_1 and a_2 are candidate internal equivalent signals. Checking



(a) model for checking if (a_1, a_2) is equivalent.

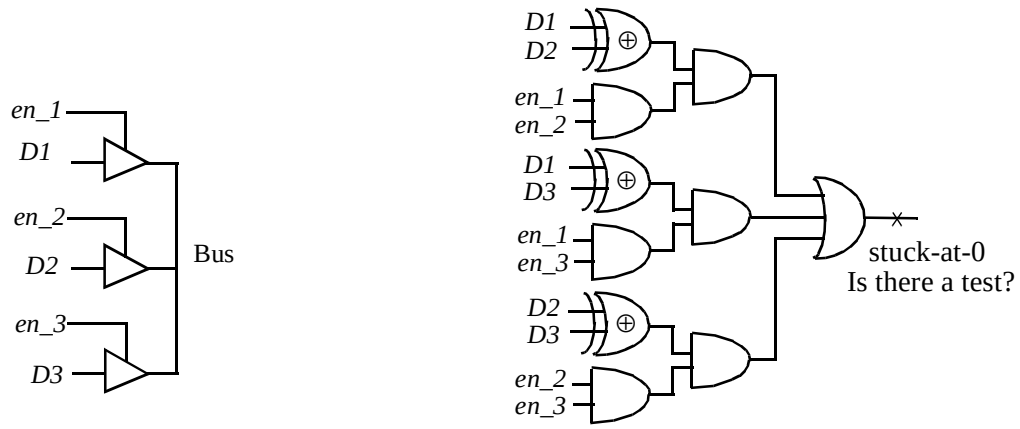
(b) replace a_1 by a_2.

Figure 6. Pruning the joint network by finding an internal equivalent pair.

equivalence between a_1 and a_2 is then performed by running ATPG for the f stuck-at-0 fault. If ATPG concludes that no test exists for the target fault in this model, the joint network can then be simplified to the one shown in Figure 6(b), where signal a_1 has been replaced by signal a_2. With the simplified model, the complexity of ATPG for the output g stuck-at 0 fault will be reduced. If the two circuits under verification have some internal equivalent signal pairs and if the above process identifies these pairs sequentially from primary inputs toward primary outputs, then when the process reaches the output of the joint network, the joint network could become substantially smaller and ATPG for g stuck-at 0 fault will be quite trivial. Various heuristics for further enhancing the idea and combining it with Binary Decision Diagram (BDD) techniques have been developed in the past few years (a survey of this topic can be found in [8]). Commercial tools for equivalence checking can now handle circuit modules of size close to a million gates within tens of minutes of CPU time.

Property checking. An ATPG engine can be used to find an example for proving that the circuit violates certain properties or, after exhausting the search space, to prove that no such example exists and thus prove that the circuit meets certain properties. Specific examples include checking for tri-state bus contention in a design, checking for asynchronous feedback loops, checking if a single clock can simultaneously capture data into more than one port of a latch, and checking for potential races [10][13].

A bus contention occurs when multiple tri-state drivers of the bus are enabled and their data are not consistent. Figure 7(b) shows the ATPG model for checking if there is a possibility of bus contention for the bus in Figure 7(a). If the ATPG engine finds a test for the output stuck-at-0 fault, the test found will be the vector causing a bus contention. If no test exists, the bus can never



(a) A bus

(b) An ATPG model for checking bus contention

Figure 7. Application to checking bus contention

have a contention. In a similar manner, ATPG can be used to check whether there is a possibility of bus floating. ATPG simply needs to check if there is a vector setting all enable lines in an inactive state.

Races are referred to as the situation when data travel through two levels of latches in one clock cycle. Races occurs if (1) there is a sensitized path from output of one latch to the input of another latch, (2) the clocks of both latches are active at the sample time and (3) the duration of the active clock is longer than the propagation delay of the sensitized data path. ATPG engine can be used to identify the test cases that could potentially cause races. The first two of the above requirements for races can be easily checked by an ATPG engine. In Figure 8, ATPG engine will try to find a test that sets both Clk_A and Clk_B high and at the same time sensitizes a path from Q_A to D_B. If such a test is found, the test can then be used for simulation (which takes timing/delays

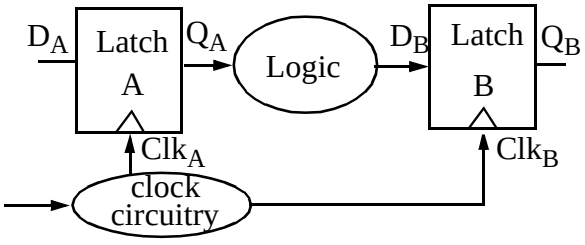


Figure 8. Illustration for application in checking potential race

into account) and condition (3) can then be checked.

Due to logic sharing, it is quite possible that there exist, at least topologically, asynchronous feedback loops (i.e., feedback loops do not contain any latch or flip-flop) in a pure synchronous circuit. ATPG engine can be used to check whether a topological, asynchronous feedback loop will cause a memory effect or an oscillation [10]. For each asynchronous loop starting and ending at a signal *S*, ATPG engine simply checks whether it is possible to find a test to sensitize this loop.

If such a test exists, then the loop will either cause a memory effect (the parity from *S* to *S* is even) or cause an oscillation (the parity is odd).

Timing verification and analysis. Test vectors, generated by path delay fault ATPG tools, that sensitize selected long paths are often used for simulation to verify the timing of the circuits. Furthermore, ATPG technology has also been used for identification of timing false paths. A path is false if it cannot support the propagation of a switching event. In determining the clock period of a circuit, the designer would like to find the slowest true path. Therefore, identifying long false paths is desirable for accurate timing analysis. Various sensitization criteria (either approximate or exact) have been developed for determining a true path. Such criteria set requirements (in terms of logic values and the arrival times of the logic values) at side inputs of the gates along the path. These requirements are somewhat similar to those for deriving tests for path delay faults. Thus, ATPG engine can be used directly for this application.

5. Summary and Looking Ahead

Test tools evolved beyond just test generation and fault simulation. The ATPG technology has been successful used for generating tests not only to screen out chips with manufacturing defects but also to identify design errors and timing problems in the design verification phase. It has also been used as a powerful logic analysis engine for the applications of logic optimization, false path identification and design property checking.

ATPG remains an active research topic in the CAD and test community. The emphasis of recent ATPG research has been in moving the ATPG operations toward higher levels of abstraction, in adapting the ATPG procedures to support core-based designs, and in handling new deep submicron faults. The vast majority of design work now takes place at the RTL level and above, using behavioral and/or logic synthesis as integral part of design. Test methods and tools must continue to move to these levels in order to be effectively coupled with the design process. Furthermore, test pattern generation procedures need to accommodate the widespread use of reusable embedded cores and to address circuits with heterogeneous components such as asynchronous and mixed-signal components.

Many of today's deep submicron signal integrity problems (which are targets of design validation) are becoming test problems as well. New “noise” faults, such as distributed delay variations, crosstalk induced delay and logic errors, excessive voltage drop and/or swing on power nets, and substrate and thermal noise, soon need to be targeted for manufacturing testing. In addition to the problem of modeling the behaviors of these noise, parametric faults to the levels of abstraction higher than the electrical-, circuit-, and transistor-levels for the purpose of efficient fault grading, another essential research topic is automatic generation of test vectors that can achieve a high coverage of these faults.

References

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1990.

[2] D. Brand, "Verification of Large Synthesized Designs", Proc. Int'l Conf. on Computer-Aided Design, pp. 534-537, Nov. 1993.

[3] W. Chen, S. K. Gupta and M. A. Breuer, "Analytic Models for Crosstalk Delay and Pulse Analysis Under Non-Ideal Inputs," *Proc. of International Test Conference*, pp. 809-818, Nov. 1997.

[4] K.-T. Cheng, "Gate-Level Test Generation for Sequential Circuits," ACM Trans. on Design Automation of Electronic Systems, vol. 1, no. 4, pp. 405-442, Oct. 1996.

[5] L. A. Entrena and K.-T. Cheng, "Combinational and Sequential Logic Optimization by Redundancy Addition and Removal", IEEE Trans. on Computer-Aided Design, vol. 14, no. 7, pp. 909-916, July 1995.

[6] Final report, *National Science Foundation Workshop on Future Research Directions in Testing of Electronic Circuits and Systems*, `http://yellowstone.ece.ucsb.edu/NSF_WORKSHOP/`, May 1998.

[7] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational logic Circuits," *Trans. on Computers*, vol. C-30, no. 3, pp. 215-222, March 1981.

[8] S.-Y. Huang and K.-T. Cheng, *Formal Equivalence Checking and Design Debugging*, Kluwer Academic Publishers, Boston, 1998.

[9] Y.-M. Jiang, K.-T. Cheng and A.-C. Deng, "Estimation of Maximum Power Supply Noise for Deep Sub-Micron Designs," *Proc. Int'l Symp. Low Power Electronics and Design*, pp. 233-238, August 1998.

[10] B. Keller et al, "ATPG in Practical and Non-Traditional Applications", Proc. IEEE International Test Conf., pp. 632-640, Oct. 1998.

[11] A. Krstic and K.-T. Cheng, *Delay Fault Testing for VLSI Circuits*, Kluwer Academic Publishers, Boston, 1998.

[12] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM Journal*. pp. 278-291, 1966.

[13] P. Wohl and J. Waicukauski, "Using ATPG for Clock Rules Checking in Complex Scan Designs", Proc. IEEE VLSI Test Symp., pp. 130-136, April 1997.