Automatic Test Pattern Generation for Delay Defects Using Timed Characteristic Functions[*]

Shin-Yann Ho, Shuo-Ren Lin, Ko-Lung Yuan, Chien-Yen Kuo, Kuan-Yu Liao, Jie-Hong R. Jiang, and Chien-Mo Li
Department of Electrical Engineering / Graduate Institute of Electronics Engineering
National Taiwan University, Taipei 10617, Taiwan

ABSTRACT

Testing integrated circuits under delay defects becomes an essential quality control step in nanometer fabrication technologies, which encounter inevitable process variations. Prior methods on automatic test pattern generation (ATPG) for delay defects, however, are either overly simplified (e.g., timing unaware) or computationally too expensive. This paper proposes a viable ATPG method based on a satisfiability (SAT) formulation using timed characteristic functions (TCFs), which gained notable scalability enhancement very recently. The approach provides a balanced trade-off between accuracy and efficiency. Experimental results show promising runtime and fault coverage improvements over prior SAT-based timing-aware ATPG methods. Moreover, our method provides a nice complement to commercial tools in enhancing test quality.

1. INTRODUCTION

With feature sizes shrinking, defects causing timing faults by the intrinsic imperfection of manufacturing process have become one of the most important considerations in testing modern very large scale integration (VLSI) designs [2]. Such defects introduce into a circuit extra delays with certain delay sizes, and may potentially result in behavior deviating from correct operation. When the delay sizes are large, delay defects can be reasonably tested with the conventional (timing-unaware) automatic test pattern generation (ATPG) technique for stuck-at faults. However, when the delay sizes are small, delay defects can be hardly triggered in the test mode at a low frequency, and have to be tested at the normal operation speed with timing-aware ATPG methods. For high performance circuits, the influence of small delay defects (SDDs) cannot be neglected [13]. SDD testing intends to capture faults caused by unintended extra delays in certain small ranges, called *fault sizes*, and is an important and challenging issue in modern nanometer technologies.

Among various delay fault models, path delay fault (PDF) and transition delay fault (TDF) models are widely used in industry. The PDF model considers the paths of a circuit and tests to see if any path delay exceed some timing requirement. Hence the PDF model can be used in a less conservative statistical design methodology. However, since the number of paths of a circuit can be exponential in the number of gates, the number of faults to be considered can be formidable [2]. Also, PDFs may require more sophisticated algorithms for test generation. Therefore most of the timing-related defects in industry are modeled using TDFs, where the number of faults is linear in the number of gates.

ATPG techniques for TDFs can be classified into two categories: timing-unaware and timing-aware methods. Timing-unaware ATPG, commonly adopted in industry, assumes the fault sizes of TDFs are large enough such that the sensitization conditions used to propagate the standard stuck-at faults are applicable for TDF detection. Although timing-unaware ATPG can be computationally efficient, it may be too crude to provide accurate test solutions. In fact, timing-unaware ATPG tends to propagate faults through short paths. However, SDDs can be difficult to be detected in this regard because an SDD often needs to be propagated along a critical path (with a small timing slack) passing through the fault site.

To address the shortcoming of timing-unaware ATPG, N-detect TDF ATPG was provided as another solution for detecting SDDs [1]. The N-detect ATPG detects a fault N times by tracing different paths for test pattern generation. The intuition is that, when N is sufficiently large, an SDD can be detected with a high probability since some of the N detections may sensitize some long paths for fault propagation. The drawback of this method is the enlarged test lengths. As a remedy, a selection strategy for filtering useful test patterns from a timing-unaware N-detect ATPG test set was proposed in [10]. Nevertheless, it is still time consuming since path delay calculation for every fault is needed.

Timing-aware ATPG tools [15] [11] have been proposed to overcome the shortcomings of timing-unaware ATPG. However, the increases on CPU runtime and test length (number of test patterns) are substantial. These methods cannot handle large circuits in practice. There are recent efforts, e.g., [18, 19, 16], taking advantage of the advancement of satisfiability (SAT) solving for timing-aware ATPG. Because encoding circuit delay information into a conjunctive normal form (CNF) formula may result in variable blow-up under practical timing precision, PHAETON [18] and WaveSAT [16] impose expensive computation. Based on PHAETON and interpolation-based model checking, recent work [19] proposed a sequential ATPG technique for SDDs. Although the quality of generated test patterns was improved, the computational scalability is still an issue and can be further improved.

This paper proposes a timing-aware ATPG technique for SDDs based on SAT solving of timed characteristic functions (TCFs) [14, 5, 6]. Under the TCF formulation, delay paths are implicitly enumerated and the induced formulas are more manageable in preventing variable blow-up and easier to solve compared to prior timing-aware ATPG methods. Practical techniques are also proposed to enhance computational scalability and test pattern quality. Prelimi-

[*]This work was supported in part by National Science Council under grants NSC 99-2221-E-002-214-MY3, 99-2923-E-002-005-MY3, and 102-2221-E-002-232.

nary experimental results show substantial runtime improvement over prior timing-aware ATPG methods and promising test coverage enhancement over commercial timing-unaware ATPG tools.

The rest of this paper is organized as follows. Section 2 provides essential preliminaries. Section 3 introduces our approach to SAT-based formulation of timing-aware ATPG. Section 4 presents the overall ATPG procedure. Implementation issues are discussed in Section 5. Experimental results are evaluated in Section 6. Section 7 finally concludes this paper.

2. PRELIMINARIES

2.1 Circuit and Delay Models

A combinational circuit $C(V, E)$ consists of a set V of nodes and a set $E \subseteq V \times V$ of directed edges. The set of nodes, according to their attributes, can be partitioned into three subsets, including primary inputs (PIs), primary outputs (POs), and function gates. A function gate can be of simple gate types such as buffer, invertor, AND, OR, NAND, NOR, XOR, XNOR, etc., or complex gate types such as multiplexer, AOI, etc. We assume a pin-to-pin delay model and delay values may vary between rise and fall time. Interconnect delays are assumed to be combined into gate delays.

For a node g in a circuit, its sets of *fanin* and *fanout* nodes are denoted as $FI(g)$ and $FO(g)$, respectively. A path in a circuit consists of a consecutive nodes (often from some PI to some PO) connected by edges. The *transitive fanin* (respectively *fanout*) *cone* of a gate g is the set of gates that can reach g (respectively can be reached from g) through some paths.

For some cared gate $g \in FI(f)$, the fanins other than g to f are called the *side inputs*. If g's value v_g can completely determine the value of f regardless of the values of other inputs, then v_g is a controlling value of f with respect to g. Otherwise v_g is a non-controlling value. E.g., logic value 0 and 1 are the controlling value of an AND and OR gate, respectively. The notion of the controlling value can be generalized to a *controlling cube*, which is a partial assignment to $FI(f)$ that determines the value of f without referring to the variables in $FI(f)$ not present in the cube. Let C_1 and C_0 be the sets of prime implicants of onset and offset of f, respectively. Then the set of all controlling cubes can be obtained by the union of C_1 and C_0 excluding the minterm primes. The reason of excluding minterm primes is that the output value of f can be determined only when all inputs of f are determined.

2.2 Path Sensitization Criteria

Two circuit operation modes are commonly used in different contexts. One is the so-called *floating mode operation*, widely assumed in logic synthesis. It considers all signals in a circuit are of an unknown initial value not until they stablize to their final values induced by some PI assignment. Under this mode, the *exact sensitization criterion* can be defined as follows. A path π is sensitizable if, for every node $g \in FI(f)$ for g, f on π, either g has the earliest arrival controlling value to f or g has the latest arrival noncontrolling value provided other side inputs are all of noncontrolling values to f.

The other is the well-known two-pattern operation, intensively applied in delay testing. It requires the first pattern to initialize the circuit, and the second one to trigger the signal transition effects. A signal is of value either 0 or 1, without in an unknown value. Under this mode of operation, a path π is sensitizable if, for every node g along π, node g has a transition and it further triggers the corresponding transition at g's fanout f on the path under the current assignments to the side inputs of f.

In the floating mode operation, there is only one transition on every signal, from unknown to 0 or from unknown to 1. In the two-pattern mode, there can be more than one transitions on a signal. This difference hints that floating mode timing simulation can be much faster than two-pattern simulation. On the other hand, floating-mode analysis is conservative since intermediate transitions in the two-pattern operation are abstracted as an unknown value in the floating mode. The path delay reported by floating-mode analysis on timing is greater than or equal to that by two-pattern simulation. Nevertheless floating mode analysis is able to identify false paths and can be much more accurate than static timing analysis.

2.3 Transition Delay Faults and their Testing

The faults caused by unintended extra rising and falling signal delays are known as the *transition delay faults* (TDF). The transition fault model commonly assumes that only one delay fault is present in the circuit. There are two types of TDFs: slow-to-rise (STR) and slow-to-fall (STF). An STR (STF) fault introduces extra 0-to-1 (1-to-0) transition delay at the fault site. The main merit of using the transition fault model is that the number of faults in the circuit is linear in circuit gate count. To test a delay fault, two patterns are needed to demonstrate a discrepancy. The first pattern gives a proper initial value (0 for STR and 1 for STF) at the fault site. The second pattern provides an intended final value at fault site and propagates the fault to some observable output. Conventionally the extra delay caused by a TDF is assumed to be large without quantifying the amount of extra delay, called the *fault size*. For the fault model without specifies the fault size, stuck-at fault test generation tools are often used. However, defects which can be modeled using small delays are much more than those modeled using large delays. It is not realistic to model defects by a large delay model in some cases.

This paper assumes that a given circuit operates at some target speed determined by the clock period, and the fault size can be small. We assume that, for a fault-free circuit, the clock period is greater than all possible arrival times at POs. For a faulty circuit, there may exist some arrival times exceed the clock period. We say that a fault is detected by two patterns if the transition fault can be activated and there exists at least one transition event after the clock period under fault simulation. In our test pattern generation, a fault size is considered as a lower bound of the delay's range that the generated test patterns can detect. That is, we use a specified fault size for test pattern generation to detect transition faults with delay greater than or equal to the fault size.

2.4 Propositional Satisfiability

The *satisfiability* (SAT) problem asks whether a propositional formula represented in the *conjunction normal form* (CNF) can be satisfied. A CNF formula is a conjunction of *clauses*, each of which is a disjunction of literals. A *literal* is a Boolean variable or its negation. The corresponding literal of a variable x in a cube c is denoted as $lit(x) \in c$, and the polarity of $lit(x)$, denoted as $pol(lit(x))$ is set to 1 if $x = lit(x)$ and to 0 if $x = \neg lit(x)$. A SAT solver can be applied to find a truth assignment of a CNF formula if it is satisfiable, or report the formula unsatisfiable. Modern SAT solvers have reached their practicality in solving industrial challenging problems. This paper relies on SAT solving for test pattern generation of delay defects.

To translate a circuit constraint into a CNF formula for SAT solving, Tseitin transformation [20], which allows linear time translation, is widely used. For example, a NAND gate a with two fanins b and c can be translated as $(\neg a \vee \neg b \vee \neg c)(a \vee b)(a \vee c)$. For a given circuit, the translation encodes every gate into several clauses, whose conjunction corresponds to the CNF formula of the circuit. In the sequel, we use φ_C to represent the CNF formula translated from a circuit C.

2.5 Timed Characteristic Functions

Under the floating mode assumption, a *timed characteristic function* (TCF) $\chi^{f,\geq t}$ of a node f characterizes a set of PI assignments that make the output value of f change from its initial unknown value to a final stabilized value no-earlier than time t [14]. A 0/1-specified TCF $\chi^{f=v,\geq t}$, for $v \in \{0,1\}$, further constrains the final stabilized value of f to be logic value 0 or 1. Unlike the 0/1-unspecified TCF, the 0/1-specified TCF, though more complex, allows different rise and fall times in the timing model and is more accurate.

A TCF formula can be constructed recursively from the POs to PIs of a circuit, and converted into a CNF formula for SAT solving. This paper adopts the following formulations proposed in [5, 6]. Let S be the set of controlling cubes of a gate f in a circuit. The 0/1-unspecified TCF can be expressed as follows.

$$\chi^{f,\geq t} = \bigvee_{g_i \in FI(f)} \chi^{g_i,\geq t-d_i} \wedge \bigwedge_{c \in S} \bigvee_{lit(g_i) \in c} (\chi^{g_i,\geq t-d_i} \vee \neg lit(g_i)) \tag{1}$$

Moreover, let S_0 and S_1 be the sets of prime implicants of the offset and onset of f, respectively. The 0/1-specified TCF can be written as follows.

$$\chi^{f=1,\geq t} = f \wedge \bigwedge_{c \in S_1} \bigvee_{lit(g_i) \in c} (\chi^{g_i=pol(lit(g_i)),\geq t-d_{r_i}} \vee \neg lit(g_i)) \tag{2}$$

$$\chi^{f=0,\geq t} = \neg f \wedge \bigwedge_{c \in S_0} \bigvee_{lit(g_i) \in c} (\chi^{g_i=pol(lit(g_i)),\geq t-d_{f_i}} \vee \neg lit(g_i)) \tag{3}$$

where d_{r_i} and d_{f_i} is the rising and falling pin-to-pin delay from g_i to f, respectively. As noted in [5, 6], the above equalities can be replaced by logical implications $\rightarrow$ for direct CNF translation and fast SAT solving without loss of accuracy. Moreover, an equivalence relation can be defined on TCFs [9, 5, 6] based on arrival-time information. Two TCFs of a node f are equivalent if they have the same next larger-or-equal possible arrival time. Let $\{a_1, \ldots, a_m\}$ be all possible arrival times of f sorted in an ascending order. The TCF equivalence reduction can be performed as follows [5, 6]. $\chi^{f,\geq t} = 1$ for $0 \leq t \leq a_1$, $\chi^{f,\geq t} = \chi^{f,\geq a_i}$ for $a_{i-1} < t \leq a_i$, $\chi^{f,\geq t} = 0$ for $t > a_m$. Similar reduction applies to 0/1-specified TCFs as well.

3. FROM FUNCTIONAL TIMING ANALYSIS TO DELAY TESTING

In TCF-based functional timing analysis [14], a circuit is assumed to operate under the floating mode. Without explicit path enumeration, an input pattern is to be found such that, after the input pattern is applied, the signals of the circuit settle from the unknown initial value to their final values in time exceeding a specified delay upper bound. If such an input pattern exists, a true delay path can be further traced as a witness of timing violation. Otherwise, the circuit is guaranteed to operate correctly within the delay bound. A recent advancement showed that functional timing analysis based on the TCF formulation can be efficiently applied on multi-million gate designs [5, 6]. It motivates our investigation of applying TCF for delay testing.

There is a subtle difference between functional timing analysis and delay testing that need to be addressed. Essentially functional timing analysis is a single-pattern conservative analysis, it is particularly desirable in logic synthesis because the so-called *monotone speed-up property* has to be satisfied in the design phase [8]. In contrast, delay testing requires a two-pattern analysis to demonstrate timing violation. Since the unknown value is only a conceptual value used in logic simulation, it does not correspond to a real physical value needed in testing. In the following we extend the single-pattern analysis to two-pattern test generation.

Given a circuit C with its circuit formula φ_C and a fault site s in C of either an STR or STF fault with a fault size δ, suppose the circuit is to operate under a clock period T. We intend to search for a two-pattern test (p_1, p_2) to detect potential timing violation. We start with deriving the second pattern p_2 by solving

$$\left(\bigvee_{o \in \mathrm{PO}} \chi^{o,\geq T} \right) \wedge \varphi_C \wedge (s) \tag{4}$$

for the circuit under the presence of an STR fault, and

$$\left(\bigvee_{o \in \mathrm{PO}} \chi^{o,\geq T} \right) \wedge \varphi_C \wedge (\neg s) \tag{5}$$

for the circuit under the presence of an STF fault. Note that the third parts (s) and $(\neg s)$ of the above conjuncts are *unit clauses* (which contain only one literal) to assert fault activation for STR and STF, respectively.

If Formula (4) or (5) is unsatisfiable, the fault is guaranteed not testable for the given fault size δ because of the conservativeness of the floating mode analysis. On the other hand, if the formula is satisfiable, the circuit exhibits timing violation (under the floating mode assumption). The corresponding satisfying assignments to the PI variables are collected as the second pattern p_2 of the test vector (p_1, p_2) for the fault. With p_2, some true delay path(s) with delay longer than T can be identified by tracing the circuit using the exact sensitization criterion [5, 6].

Given the so-obtained pattern p_2, the next task is to derive the first pattern p_1 of the test. For (p_1, p_2) to be a valid test, p_1 has to satisfy two conditions: First, p_1 must provide proper initial value for the target fault. That is, at the fault site s in the circuit C, the logic value of s should reset to 0 for an STR fault and set to 1 for an STF fault. Second, a transition induced by (p_1, p_2) at some PI must propagate through the fault site s to some PO along some long true delay path induced by p_2. Exact derivation of p_1 can be difficult, however, due to the fact that sensitization may arise from dynamic hazards. Therefore we approximate the above constraints by the formula

$$\varphi_C \wedge (\neg s) \wedge \bigwedge_{g \in \pi} (g \oplus c_g) \tag{6}$$

for an STR fault, and by the formula

$$\varphi_C \wedge (s) \wedge \bigwedge_{g \in \pi} (g \oplus c_g) \tag{7}$$

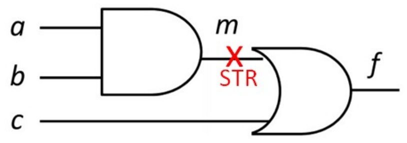

Figure 1: Circuit for timing-aware ATPG

for an STF fault, where π is a true delay path sensitized under p_2 and $c_g \in \{0,1\}$ is the stable logic value of signal g when p_2 is applied. Note that there can be multiple true delay paths sensitized under p_2. Empirical experience suggests that in most cases only very few paths need to be examined before a satisfying solution to Formula (6) or (7) is found.

Notice that Formulas (6) and (7) are only approximative since the transition propagation constraint is a static condition, which does not take into account dynamic timing information. Both false negatives and false positives may possibly arise. That is, if no p_1 can be found, it does not prevent the possibility that dynamic hazards propagate through π and cause timing violation from happening. On the other hand, if some p_1 is found, it does not guarantee that transitions will propagate all the way through π since a side-input may trigger a transition at a signal on π and block the intended propagation arrived later from an on-input. In general a fault simulator is needed to verify whether a generated test vector (p_1, p_2) indeed detects the target delay fault.

Our practical experience suggests that Formulas (6) and (7) may sometimes overly constrain p_1 pattern generation while dynamic transition events can still make a fault testable. Hence, in the implementation of p_1 generation, we relax the static path transition constraints and simply apply the reset constraint $\varphi_C \wedge (\neg s)$ and the set constraint $\varphi_C \wedge (s)$ for STR and STF faults, respectively. We then resort to a fault simulation to justify the validity of the generated pattern as a by-product of fault dropping.

EXAMPLE 1. *Consider the circuit of Figure 1 with a target operation clock period $T = 3$ unit delays. For simplicity, assume every pin-to-pin delay is of 1 unit delay, and the signal arrival times of PIs are 0. Suppose an STR fault is present at signal m with fault size 1. The two-pattern test vector (p_1, p_2) can be generated by SAT solving the following formulas.*

$$p_1 : \varphi_C \wedge (\neg m), \text{ and}$$

$$p_2 : \chi^{f,\geq 3} \wedge \varphi_C \wedge (m),$$

with

$$\begin{aligned}
\varphi_C &= (\neg a \vee \neg b \vee m)(a \vee \neg m)(b \vee \neg m)(m \vee c \vee \neg f) \\
&\quad (\neg m \vee f)(\neg c \vee f), \\
\chi^{f,\geq 3} &= (x^{f,\geq 3})(\neg x^{f,\geq 3} \vee x^{c,\geq 2} \vee x^{m,\geq 1}) \\
&\quad (\neg x^{f,\geq 3} \vee x^{c,\geq 2} \vee \neg c)(\neg x^{f,\geq 3} \vee x^{m,\geq 1} \vee \neg m) \\
&\quad (\neg x^{m,\geq 1} \vee x^{a,\geq 0} \vee \chi^{b,\geq 0})(\neg x^{m,\geq 1} \vee x^{a,\geq 0} \vee a) \\
&\quad (\neg x^{m,\geq 1} \vee x^{b,\geq 0} \vee b),
\end{aligned}$$

where variable $x^{f,\geq t}$ is used to represent the output variable of it corresponding characteristic function $\chi^{f,\geq t}$ to avoid confusion.

Originally, the longest delay of fault-free circuit is 2, which is smaller than T. After injecting the delay fault, there exists a delay path with delay ≥ 3. By solving the above two SAT-instances, we have satisfying assignments $p_1 = (a = 0, b = 1, c = 0)$ and $p_2 = (a = 1, b = 1, c = 0)$. By fault simulation, it can be verified that the STR fault on m is activated and propagated to f with 3 unit delays.

```
TCFTestGen
  input: circuit $C$, clock period $T$, fault list $L$
  output: test set $P$, detected faults $F_{\mathrm{D}}$,
          undetected faults $F_{\mathrm{UD}}$, untestable faults $F_{\mathrm{UT}}$
  begin
  01  $P, F_{\mathrm{D}}, F_{\mathrm{UD}}, F_{\mathrm{UT}} := \emptyset$;
  02  $L^* :=$ copy $L$;
  03  while $L^* \neq \emptyset$
  04    $fault := GetFault(L^*)$;
  05    $(p_1, p_2) := GeneratePattern(C, T, fault)$;
  06    if $p_1 =$ NULL or $p_2 =$ NULL
  07      move $fault$ from $L^*$ to $F_{\mathrm{UT}}$;
  08    else
  09      $(L', F'_{\mathrm{UD}}) := DropFault(C, (L^*, F_{\mathrm{UD}}), (p_1, p_2))$;
  10      if $L' \neq L^*$ or $F'_{\mathrm{UD}} \neq F_{\mathrm{UD}}$
  11        $P := P \cup \{(p_1, p_2)\}$;
  12        $F_{\mathrm{D}} := F_{\mathrm{D}} \cup \{L^* \backslash L'\} \cup \{F_{\mathrm{UD}} \backslash F'_{\mathrm{UD}}\}$;
  13      if $L^* = L'$
  14        move $fault$ from $L'$ to $F'_{\mathrm{UD}}$;
  15      $L^* := L'$, $F_{\mathrm{UD}} := F'_{\mathrm{UD}}$;
  16  return $(P, F_{\mathrm{D}}, F_{\mathrm{UD}}, F_{\mathrm{UT}})$;
  end
```

Figure 2: Algorithm: TCF Test Generation

4. ALGORITHMIC FLOW OF DELAY TESTING

In this section, we first show an ATPG flow with respect to pre-specified fault sizes, and then extend it to a comprehensive flow with adapted fault sizes to target various delay faults. Note that ATPG for delay defects with small fault sizes tends to sensitize long paths, whereas ATPG for defects with large fault sizes may sensitize short paths and is similar to timing-unaware ATPG.

4.1 ATPG with Specified Fault Sizes

Given a circuit C, its operation clock period T, and a fault list L, the test generation procedure of Figure 2 computes a test set P, along with the set F_{D} of detected faults, F_{UD} of undetected faults, and F_{UT} of untestable faults. Assume the fault list L provides the fault type (i.e., STR or STF) and fault size for each fault site. After the initialization steps (lines 1 and 2), a fault in the fault list L^* is examined iteratively for test generation (lines 3-15). The TCF based two-pattern test generation is performed (line 5). If p_1 or p_2 has no solution (line 6), then the fault is declared as untestable (with respect to the specified fault size) and placed in F_{UT} (line 7). Then a new iteration (line 3) begins with a new fault if it exists. Otherwise, fault simulation is performed under two-pattern input vector (p_1, p_2). By the simulation, faults in L^* and F_{UD} testable under the test vector are removed from L^* and F_{UD} yielding L' and F'_{UD}, respectively (line 9). If the test vector can effectively remove some fault (line 10), then it is added to the test set P (line 11) and the newly tested faults are added to F_{D} (line 12). On the other hand, if L^* cannot be reduced by fault dropping (line 13), then the current fault is moved from L' to temporarily untestable set F'_{UD} (line 14). Before a new iteration begins (line 3), L^* and F_{UD} are updated (line 15). When L^* is empty (line 3), the procedure returns the collected test vectors P, detected faults F_{D}, undetected faults

```
SDD_ATPG
  input: circuit $C$, clock period $T$, fault list $L$
  output: test set $P$, detected faults $F_D$,
          undetected faults $F_{UD}$, untestable fault $F_{UT}$
  begin
  01  $P, F_D, F_{UD}, F_{UT} := \emptyset$;
  02  $L^* :=$ copy $L$;
  03  repeat
  04      $L^* := AdaptFaultSize(C, T, L^*)$;
  05      $(P', F'_D, F_{UD}, F_{UT}) := TCFTestGen(C, T, L^*)$;
  06      $L^* := TrimFaultList(L^*, F'_D)$;
  07      $P := P \cup P'$;
  08      $F_D := F_D \cup F'_D$;
  09  until $L^* = \emptyset$ or fault size $> T$;
  10  return $(P, F_D, F_{UD}, F_{UT})$;
  end
```

Figure 3: Algorithm: SDD ATPG

F_{UD}, and untestable faults F_{UT}.

4.2 ATPG with Adapted Fault Sizes

The procedure of Figure 2 generates test vectors with respect to a given fault list. Since a fault with its longest path delay smaller than the clock period minus its fault size is not testable, different specifications of fault sizes may have strong impacts on the fault coverage. It is often beneficial to adapt the fault size according to the degree of circuit sensitivity in the delay fault and to the level of computation efforts needed for the test generation. Figure 3 presents a generic framework for SDD ATPG, which adjusts fault sizes to improve test quality.

In Figure 3, after initialization steps (lines 1 and 2), the procedure repeatedly generates test vectors (lines 3-9) for every fault list with adapted fault sizes (line 4). Given a fault list L^*, the procedure *TCFTestGen* of Figure 2 is performed for test generation (line 5). The detected faults F'_D are removed from the fault list L^* (line 6); the test set P is expanded accordingly (line 7); the set F_D of detected faults is updated as well (line 8). The process repeats until all faults are covered or the fault size has been increased exceeding the clock period (line 9). The final results are then returned (line 10).

The strategy of adapting fault sizes plays a key role determining both test quality and test efficiency. A reasonable strategy is to first prune easy-to-detect delay faults, e.g., by timing-unaware ATPG. The remaining hard-to-detect faults can then be covered by gradually increasing the fault sizes whenever they are needed. This fault-size relaxation strategy is complete in the sense that every detectable fault can be tested under some fault size possibly close to its smallest possible. Ultimately when the fault size is increased to exceed the clock period, every fault that can be activated and sensitized through some path should be covered by the ATPG.

5. IMPLEMENTATION ISSUES

We propose the following enhancement techniques to improve ATPG efficiency for SDDs.

5.1 Fault Simulation

Fault simulation is essential in delay testing for two reasons. First, it validates the generated test vectors are indeed legitimate. Second, it facilitates fault dropping to remove additional undetected faults covered by the newly generated test vector from the fault list. Since fault simulation needs to be frequently applied, its efficiency strongly affects the runtime of ATPG. Deploying a proper fault simulation by striking a balance between accuracy and efficiency is vital to achieve efficient and effective ATPG.

Event-driven simulation is a common technique for accurate waveform simulation to faithfully capture transition events at the switch level. Due to its effectiveness, it has been widely used in logic simulation. Nevertheless, in application to delay testing, it has to be executed for many times and imposes great computation overhead. Essentially, there can be enormous transition events present in simulating a single delay fault. Moreover, there are numerous faults to be tested for a given circuit. For practicality, the high cost of event-driven simulation should be avoided whenever possible.

In our implementation, we adopt the exact sensitization criterion [3] for delay simulation. That is, the simulation implicitly assumes the floating mode operation, which is consistent with the underlying assumption of TCF formulation. Although floating-mode based simulation is not as accurate as event-driven simulation, it is desirably conservative and computational more affordable than event-driven simulation. Under the two-pattern test generation scenario, we assume the first test pattern reaches propagation stablization before the second test pattern is applied. As a result, the accuracy of floating-mode analysis can be refined to take into account the early stablization of the first test pattern. Precisely, a signal x has arrival time 0 if both the first pattern and the second pattern of a test vector impose the same stable values at every signal in the transitive fanin cone of x. Otherwise, x is assumed to have an unknown initial value as usual.

5.2 Incremental Arrival-Time List Computation

As mentioned in Section 2.5, an arrival-time list is useful for TCF equivalence reduction. However the list of every gate cannot be computed once and for all because injecting a delay fault into a circuit changes the possible arrival times in the circuit. Since the induced arrival-time lists of every fault can be different, the arrival-time lists for the gates of a circuit have to be updated for every fault. We only incrementally update the arrival-time lists of the gates in the transitive fanout cone of the fault site.

5.3 TCF Time-Order Strengthening

The TCFs of any gate f obey the monotone property that the logic implication $\chi^{f,\geq t_1} \rightarrow \chi^{f,\geq t_2}$ holds for $t_1 > t_2$. This property can be explicitly added to ϕ_{TCF} to strengthen the search space of SAT solving. Precisely, suppose, in some test pattern generation instance, several TCFs $\chi^{f,\geq t_1}, \ldots, \chi^{f,\geq t_n}$, with $t_1 > \cdots > t_n$, are invoked for gate f. Then we may add the constraints $\chi^{f,\geq t_1} \rightarrow \chi^{f,\geq t_2}$, $\chi^{f,\geq t_2} \rightarrow \chi^{f,\geq t_3}$, $\ldots$, $\chi^{f,\geq t_{n-1}} \rightarrow \chi^{f,\geq t_n}$ into ϕ_{TCF}. Empirical experience suggests that such time-order constraints often improve solving performance.

5.4 Delay Search Strategy

To determine whether a delay fault can be tested, there can be different strategies in choosing target circuit delays for verification. One obvious choice is to set the target delay slightly greater than the clock period and determine whether there exists a two-pattern test vector sensitizing the fault and causing delay violation. Following this strategy, however, the correspondingly constructed TCF can be too complex to be solved efficiently, especially when the longest true delay is much larger than the target delay. As was proposed in [5, 6], a linear search strategy of pruning spurious arrival times from the largest one is particularly desirable. Although the linear search strategy may potentially require several runs of SAT checking, it allows constant propagation during TCF construction and may substantially simplify the formulas to be solved. In addition, as a by-product the so-generated test vector tends to sensitize a longer delay path and is particularly desirable.

5.5 Formula Complexity Filtering

One effective way to control the runtime of ATPG is to avoid solving time-consuming TCF formulas. To quantify the complexity of a TCF formula, the formula size and the number of decision conflicts during SAT solving are good measures. When the formula size or the number of decision conflicts of an SAT instance exceeds a pre-specified threshold, the test derivation of the corresponding delay fault is skipped. Nevertheless, it may possibly be covered by the test patterns of other faults, whose test derivations are more efficient.

6. EXPERIMENTAL RESULTS

The proposed ATPG procedure for SDDs was implemented in the C++ language and used MiniSat version 2.2 [7] for SAT solving. All experiments were executed on a Linux machine with a Xeon 2 GHz CPU and 40 GB RAM.

Circuits from the ISCAS benchmark suits were selected for experiments. The circuits were technology mapped with respect to a library consisting of simple gate types, including INV, AND, NAND, OR, NOR, XOR, and XNOR gates, although our method is applicable to arbitrary complex gate types. Table 1 shows the gate counts and logic levels of the circuits. The timing model is based on TSMC 0.18μm library with combined rise/fall times and with delay precision set to the first digit after the decimal point. In this paper we focused on testing combinational circuits. So for sequential circuits we assumed the launch-on-capture (LOC) testing method for scanning in test patterns.

Table 1: Circuit Profile

Circuit	#gate	#level
C880	383	24
C1355	554	25
C1908	932	42
C2670	1202	33
C3540	1703	47
C5315	2330	49
C6288	2480	124
C7552	3568	43
S9234	6094	58
S13207	9365	59
S15850	11053	82
S35932	19588	29
S38417	25367	47

In practice, many faults can be handled by traditional timing-unaware ATPG methods. Hence there is no need to perform timing-aware ATPG for every fault in a circuit. Under this observation, for every circuit in the experiments, 1,000 faults are created under testing, the clock period is set to be about 1% larger than the maximal arrival time at the POs of a circuit, and the fault size is set to be some percentage of the clock period depending on different sets of experiments. Fault coverage (denoted FC, the percentage of detected faults among all 1,000 faults), test length (denoted TL, the total number of generated test patterns), and CPU runtime are under evaluation. In addition, SDQL (statistical delay quality level) [17], a well-known standard for test quality measure, is also adopted for evaluation.

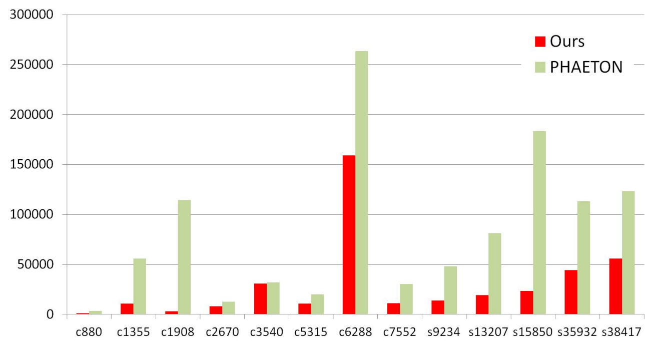

Figure 4: Comparison on the number of generated clauses

Three experiments were performed and a time limit of 3,600 seconds was set. We compared our method against the SAT-based timing-aware ATPG tool PHAETON [18] (based on our re-implementation, which might not directly reflect the actual performance of the original tool) and a commercial ATPG tool for TDFs. On the other hand, attempts were made to have a fair setting on the commercial tool for comparison.

Table 2 shows the results of the three methods under three different fault size settings, namely 10%, 20%, and 40% of the clock period. In the experiment, limits were set on the CNF formula size. For our method, we experimented with the limits of 10,000 and 1,000,000 clauses per SAT instance. For PHAETON, a limit of 1,000,000 clauses per SAT instance was set. (Reducing the clause limit to 100,000 or less results in many unavailable data of PHAETON due to formulas beyond the size limit.) Note that the fault coverage reported by our floating-mode fault simulator is fault size dependent since some faults going through short paths are ATPG untestable with small fault sizes. That is, the best fault coverage is not necessarily 100%, and the comparison is relative among the three methods. When our method and PHAETON are compared, the former is significantly faster and achieved better fault coverage than the latter in almost all benchmark circuits. The large computation overhead of PHAETON is due to its excessive number of T-variables [18] in a CNF formula. When the commercial tool is compared, our method has larger CPU runtime since the commercial tool determines timing violations by computing timing slacks whose computation overhead is much lighter than SAT-solving. For faults whose minimum slacks are greater than the maximum timing margin, the commercial tool applies timing-unaware ATPG and may lose fault coverage although computational efficiency is gained. Our method yielded better fault coverage and shorter test lengths, which means our method has better quality of the generated patterns for SDDs. Notice that comparison on SDQL was not performed in this experiment. It is because our method and PHAETON generated test patterns only for faults that are testable with respect to the specified fault size, whereas the commercial tool generated patterns for all faults regardless of their fault sizes. This difference makes SDQL comparison unfair.

To justify the performance gap between our method and PHAETON, Figure 4 compares for every circuit the average

Table 2: Comparison of ATPG techniques for SDDs with specified fault size

Fault size = 10%

Circuit	Ours (limit=10k)			Ours (limit=1M)			PHAETON (limit=1M) [18]			Commercial tool		
	Time (s)	FC (%)	TL	Time (s)	FC (%)	TL	Time (s)	FC (%)	TL	Time (s)	FC (%)	TL
C880	0.5	13.5	23	0.5	13.5	23	8.1	4.4	2	0.2	0	119
C1355	3.9	85.0	121	6.1	90.8	171	18.6	88.6	82	0.4	83.4	170
C1908	7.7	16.4	39	7.7	16.4	39	95.3	16.5	27	0.3	14.3	131
C2670	7.7	86.7	144	13.2	90.2	151	5.2	90.0	11	0.2	79.5	124
C3540	17.8	31.1	33	188.6	35.1	75	383.4	34.4	43	0.3	31.8	187
C5315	3.3	9.1	35	3.9	9.5	39	20.6	4.9	13	0.2	7.4	130
C6288	17.1	13.9	4	933.9	18.2	10	717.5	20.2	24	0.4	0	54
C7552	5.5	11.0	56	5.5	11.0	56	32.8	7.1	20	0.3	9.0	132
S9234	2.2	74.2	16	2.9	74.2	16	20.8	58.2	3	0.4	65.9	214
S13207	6.5	2.6	23	8.2	2.6	23	138.8	2.0	14	0.6	2.1	176
S15850	37.8	96.7	198	48.2	96.7	198	1499.7	51.7	1	0.5	99.0	152
S35932	46.9	13.3	116	69.5	13.3	116	1093.5	7.2	64	0.9	12.4	53
S38417	27.4	17.5	120	45.7	17.5	121	876.4	18.4	72	1.2	13.3	176
average	14.2	36.2	71.4	102.6	37.6	79.8	377.7	29.5	29.3	0.5	32.2	139.8

Fault size = 20%

Circuit	Ours (limit=10k)			Ours (limit=1M)			PHAETON (limit=1M) [18]			Commercial tool		
	Time (s)	FC (%)	TL	Time (s)	FC (%)	TL	Time (s)	FC (%)	TL	Time (s)	FC (%)	TL
c880	0.6	22.3	50	0.7	22.3	50	8.2	9.1	9	0.2	4.2	139
c1355	2.9	87.3	128	7.8	92.3	175	50.3	92.4	83	0.4	85.4	170
c1908	5.8	20.9	60	8.3	20.9	60	140.6	19.0	29	0.3	17.1	145
c2670	6.1	91.7	154	13.0	91.3	163	4.4	90.3	19	0.2	59.1	125
c3540	15.8	35.5	39	274.4	40.2	90	1042.6	40.8	63	0.4	39.0	207
c5315	2.6	9.5	36	4.0	9.9	41	24.6	5.7	14	0.3	7.9	154
c6288	14.4	26.2	5	1342.3	30.4	13	3600	0	0	0.5	0	68
c7552	4.6	11.6	62	6.3	11.6	62	40.2	7.3	20	0.4	10.2	166
s9234	2.2	74.2	16	2.9	74.2	16	16.5	58.2	3	0.4	66.1	240
s13207	6.6	2.6	23	8.2	2.6	23	107.1	2.2	14	0.7	2.2	183
s15850	36.6	96.7	198	53.2	96.7	198	1266.6	51.7	1	0.7	99.2	172
s35932	47.7	13.3	116	68.1	13.3	116	447.4	7.2	64	0.9	12.9	60
s38417	28.2	17.5	120	46.1	17.5	121	389.3	18.5	68	1.3	13.2	209
average	13.4	39.2	77.5	141.2	40.2	86.7	549.0	33.5	32.2	0.5	32.0	156.8

Fault size = 40%

Circuit	Ours (limit=10k)			Ours (limit=1M)			PHAETON (limit=1M) [18]			Commercial tool		
	Time (s)	FC (%)	TL	Time (s)	FC (%)	TL	Time (s)	FC (%)	TL	Time (s)	FC (%)	TL
c880	1.0	50.8	127	1.2	50.8	127	6.2	41.4	59	0.2	39.1	190
c1355	2.5	89.4	120	7.8	94.4	162	231.9	94.3	81	0.4	87.2	170
c1908	7.2	37.2	96	10.3	37.2	96	141.3	37.5	66	0.5	38.5	194
c2670	5.0	91.9	166	11.8	92.4	159	6.8	93.1	39	0.4	97.3	195
c3540	13.8	35.5	39	284.1	40.2	90	139.8	39.3	53	0.6	39.7	276
c5315	2.3	10.3	43	4.1	10.7	48	29.7	7.4	15	0.4	9.0	218
c6288	13.3	45.3	13	1451.9	53.5	31	3600	0	0	0.8	45.8	120
c7552	4.1	12.9	69	6.0	12.9	69	55.6	9.0	24	0.6	12.1	251
s9234	2.5	74.4	20	3.3	74.4	20	23.5	58.2	3	0.5	51.2	274
s13207	6.6	2.6	23	7.8	2.6	23	146.4	1.6	12	0.7	2.3	194
s15850	37.6	96.7	198	52.6	96.7	198	1705.0	51.7	1	0.9	98.8	200
s35932	67.1	17.5	157	84.8	17.5	157	472.3	14.3	87	0.9	18.4	75
s38417	42.3	32.5	246	69.1	32.6	247	657.3	33.7	135	1.4	34.0	263
average	15.8	45.9	101.3	153.4	47.3	109.7	555.0	40.1	47.9	0.6	44.1	201.5

number of clauses generated for SAT solving by our method and PHAETON (under the upper limit of 1,000,000 clauses). As can be seen, our method produces much fewer clauses than PHAETON in all circuits.

In the second experiment, we combined our ATPG technique with the commercial tool in the following setting to enhance SDQL. We intended to focus on hard-to-detect faults that the commercial tool cannot generate patterns to sensitize them through a long enough path. Firstly, the commercial tool was run with its SDD algorithm applied for all faults. A fault is classified as detected if it can be detected through a path whose delay is more than 95% of the maximum delay of all paths through the fault site. After fault simulation, the faults that the commercial tool claimed undetected through the longest path were collected. Our method then tried to generate patterns for these faults with adapted fault sizes. This hybrid method is compared against with the pure commercial tool ATPG targeting pattern generation for sensitizing the longest path for all faults in the given fault list. Table 3 shows the runtime, SDQL, and test length data for the two methods. The runtime and test length of the hybrid method were broken down into Timec (runtime by the commercial tool) and Timeo (runtime by ours), and TLc (test length of the commercial tool) and TLo (test length of ours), respectively. As can be seen, our ATPG can enhance SDQL by generating patterns for the faults which cannot be detected through long path sensitization. On average, the hybrid method achieved better (smaller) SDQL with small test lengths.

In the third experiment, we studied the ATPG performance of our method with three different formula size limits, namely, 1,000,000, 100,000, and 10,000. The results are shown in Table 4. It shows a tradeoff between the test length and runtime. For circuits C1355, C2670, C3540 and C6288, sacrificing some test patterns that are hard to generate would improve the runtime. The speed-up can be significant especially for large or structurally complex circuits such as C6288 and yet without much fault coverage degradation.

Table 3: Comparison of our hybrid ATPG technique with commercial approach

Circuit	Commercial tool			Hybrid method					
	Time (s)	SDQL	TL	Timec (s)	Timeo (s)	SDQL	TLc	TLo	TLtotal
C880	0.5	0.3	210	0.5	1.1	0.1	214	156	370
C1355	1.4	4.5	346	0.4	12.8	5.1	133	134	267
C1908	1.9	11.7	310	1.1	76.9	7.9	175	108	283
C2670	1.7	1.5	340	1.6	29.9	1.5	311	175	486
C3540	2.0	15.1	446	1.1	119.3	15.9	219	32	251
C5315	1.7	3.8	369	0.8	11.1	3.8	127	77	204
C6288	7.1	12.4	352	7.1	124.4	11.8	327	64	391
C7552	3.5	3.0	429	1.5	20.1	4.7	117	86	203
average	2.5	6.5	350.2	1.8	49.4	6.3	202.8	104.0	306.8

Table 4: Our ATPG with different clause limits (10% fault size)

Circuit	Clause limit = 1000000			Clause limit = 100000			Clause limit = 10000		
	Time (s)	FC	TL	Time (s)	FC	TL	Time (s)	FC	TL
C880	0.5	13.5	23	0.5	13.5	23	0.5	13.5	23
C1355	6.1	90.8	171	6.1	90.8	170	3.9	85.0	121
C1908	7.7	16.4	39	7.7	16.4	39	7.7	16.4	39
C2670	13.2	90.2	151	13.2	90.2	151	7.6	86.7	144
C3540	188.6	35.1	75	80.8	34.7	71	17.8	31.1	33
C5315	3.9	9.5	39	3.9	9.5	39	3.3	9.1	35
C6288	933.9	18.2	10	96.1	15.9	6	17.1	13.9	4
C7552	5.5	11.0	56	5.5	11.0	56	5.5	11.0	56
average	144.9	35.6	70.5	26.5	35.2	69.4	7.9	33.3	56.8

7. CONCLUSIONS AND FUTURE WORK

This paper has proposed a timing-aware ATPG technique for SDDs based on TCFs. Experimental results showed that our method outperforms state-of-the-art SAT-based timing-aware ATPG method PHAETON in terms of both runtime and test pattern quality. Moreover, in comparison with the commercial tool, our method improves test quality in terms of both fault coverage and test length, though with some performance loss. For future work, the application of timing-aware ATPG on large sequential designs awaits further development. Enhancing the test coverage without loss of false negatives is also an important issue. Furthermore, for ATPG of SDDs in an industrial setting for multi-million gate designs, our method can be used to cover the most critical SDDs that commercial timing-aware ATPG can not manage to detect.

REFERENCES

[1] M. E. Amyeen, S. Venkataraman, A. Ojha, and S. Lee. Evaluation of the Quality of N-Detect Scan ATPG Patterns on a Processor, *In Proc. Int'l Test Conference* (ITC), pp. 669-678, 2004.

[2] M. Bushnell and V. Agrawal. *Essentials of Electronics Testing.* Kluwer Publishers, 2000.

[3] H.-C. Chen and D. Du. Path sensitization in critical path problem. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 12(2): 196-207, Feb. 1993.

[4] K.-T. Cheng. Transition Fault Testing for Sequential Circuits. *IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems,* 12(12):1971-1983, Dec. 1993.

[5] Y.-T. Chung and J.-H. R. Jiang. Functional Timing Analysis Made Fast and General. In *Proc. Design Automation Conference* (DAC), pp. 1055-1060, 2012.

[6] Y.-T. Chung and J.-H. R. Jiang. Functional Timing Analysis Made Fast and General. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2013.

[7] N. Een and N. Sorensson. An Extensible SAT-Solver. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing* (SAT), pp. 502-518, 2003.

[8] J.-H. R. Jiang and S. Devadas. Logic Synthesis in a Netshell. In *Electronic Design Automation: Synthesis, Verification, and Test*, L.-T. Wang, Y.-W. Chang, and K.-T. Cheng (ed.), pp. 299-404, Morgan Kaufmann Publishers, 2009.

[9] Y.-M. Kuo, Y.-L. Chang, and S.-C. Chang. Efficient Boolean characteristic function for timed automatic test pattern generation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 28(3): 417-425, March 2009.

[10] H. Lee, S. Natarajan, S. Patil, and I. Pomeranz. Selecting High-Quality Delay Tests for Manufacturing Test and Debug. In *Proc. IEEE Int. Symp. Defect Fault Tolerance Very Large Scale Integr. Syst.*, pp. 59-70, 2006.

[11] X. Lin, K.H. Tsai, C. Wang, Y. Sato, and S. Hamada. Timing-Aware ATPG for High Quality At-speed Testing of Small Delay Defects. In *Proc. Asian Test Symposium* (ATS), pp. 139-146 2006.

[12] S.-Y. Lu, M.-T. Hsieh, and J.-J. Liou. An Efficient SAT-Based Path Delay Fault ATPG with an Unified Sensitization Model. In *Proc. Int'l Test Conference* (ITC), pp. 1-7, 2007.

[13] R. Mattiuzzo, D. Appello C. Allsup. Small Delay Defect Testing. *Test and Measurement*, 2009.

[14] P. C. McGeer and R. K. Brayton. *Integrating Functional and Temporal Domains in Logic Design.* Kluwer Academic Publishers, 1991.

[15] Mentor Graphics. Understanding How to Run Timing-Aware ATPG, Application Note, 2006.

[16] M. Sauer, A. Czutro, I. Polian and B. Becker. Small-Delay-Fault ATPG with Waveform Accuracy. In *Proc. Int'l Conf. on Computer-Aided Design* (ICCAD), pp. 30-36, 2012.

[17] Y. Sato, S. Hamada, T. Maeda, A. Takatori, and S. Kajihara. Evaluation of the Statistical Delay Quality Model. In *Proc. Asia and South Pacific Design Automation Conference* (ASP-DAC), pp. 305-310, 2005.

[18] M. Sauer, J. Jiang, A. Czutro, I. Polian, and B. Becker. Efficient SAT-Based Search for Longest Sensitisable Paths. in *Proc. Asian Test Symposium* (ATS), pp. 108-113, 2011.

[19] M. Sauer, S. Kupferschmid, A. Czutro, I. Polian, S. Reddy and B. Becker. Functional Test of Small-Delay Faults using SAT and Craig Interpolation. In *Proc. Int'l Test Conference* (ITC), pp. 1-8, 2012

[20] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, pp. 466-483, 1970.