PAPER

Test Pattern Ordering and Selection for High Quality Test Set under Constraints

Michiko INOUE[†a)], ***Member***, **Akira TAKETANI**[†*], ***Nonmember***, **Tomokazu YONEDA**[†b)], ***Member***, ***and*** **Hideo FUJIWARA**[††], ***Fellow***

SUMMARY Nano-scale VLSI design is facing the problems of increased test data volume. Small delay defects are becoming possible sources of test escapes, and high delay test quality and therefore a greater volume of test data are required. The increased test data volume requires more tester memory and test application time, and both result in test cost inflation. Test pattern ordering gives a practical solution to reduce test cost, where test patterns are ordered so that more defects can be detected as early as possible. In this paper, we propose a test pattern ordering method based on SDQL (Statistical Delay Quality Level), which is a measure of delay test quality considering small delay defects. Our proposed method orders test patterns so that SDQL shrinks fast, which means more delay defects can be detected as early as possible. The proposed method efficiently orders test patterns with minimal usage of time-consuming timing-aware fault simulation. Experimental results demonstrate that our method can obtain test pattern ordering within a reasonable time, and also suggest how to prepare test sets suitable as inputs of test pattern ordering.

key words: *small delay defects, SDQL, ATPG*

1. Introduction

Nano-scale VLSI design is facing the problems on increased test data volume. Small delay defects such as resistive-opens and resistive-shorts are becoming possible sources of test escapes [1]. Therefore, high quality test for small delay defects is required, and several approaches have been proposed [2]. High delay test quality and, in addition, increased circuit size require more test data volume, and therefore more tester memory and test application time, and these result in test cost inflation.

Test pattern ordering gives a practical solution to reduce test cost, where test patterns are ordered so that more defects can be detected as early as possible [3]–[9]. The ordered test patterns can be used 1) to reduce test data volume to satisfy a given constraint with minimal impact on defect detection by removing the later part of test patterns, or 2) to reduce test application time for defective chips in stop-on-first-failure testing strategy. The previously proposed methods address how to detect defective chips with small number of patterns, and some of works address test pattern ordering considering small delay defects.

Chao et al. [7] proposed a test pattern selection method for timing defects. Their method considers process variation using a probability density function for each pin-to-pin segment (between output or input pins of gates), and evaluates test patterns based on *critical probability*. A segment is critical if an arrival time of some primary output exceeds the system clock period by a given fixed size delay defect of the segment. Though their method considers the process variation, test patterns are evaluated under a fixed defect size. To handle various defect sizes, the method using time-consuming Monte-Carlo simulation have to be applied repeatedly.

Yilmaz et al. proposed test set pattern grading and selection method for small delay defects [9]. They leverage the method of *output deviations* [10] for screening small-delay defects. Their method calculates gate delay defect probabilities for a predetermined fixed size delay based on a probability density function of a delay distribution, and it then obtains signal-transition probabilities of observation points (outputs) for each test pattern by propagating the probabilities from the test application points. They select test patterns from the largest deviations for each output, where the deviation means a probability that the output does not have the expected signal-transition. In the method, they select test patterns in the order of the deviation, and do not consider any cumulative effect of a selected test set. The authors also proposed a time-efficient evaluation metric to avoid time-consuming timing-aware simulation. The selected test sets are evaluated by the number of sensitized long paths, where a long path is a path with at least 70% of the clock period. This metric evaluates test patterns with whether they activate long paths or not.

SDQM (Statistical Delay Quality Model) is proposed to evaluate test quality for small delay defects [11], [12]. It evaluates not only test pattern quality but also quality of fabrication, design and test timing, and it is adopted as a measure for timing-aware test generation in several EDA tools [13]–[15]. SDQL (Statistical Delay Quality Level) is a delay test measure based on SDQM. For a given circuit, SDQL of a test set represents a total amount of delay defects that have to be detected but cannot be detected by the test set. SDQL evaluates the minimum delay defect size detected by a test set, and obtains an amount of delay test escape by considering statistical delay defect distribution. Though SDQL

Manuscript received May 9, 2012.
Manuscript revised August 20, 2012.

[†]The authors are with the Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan. Also with Japan Science and Technology Agency, CREST, Tokyo, 102–8666 Japan.

[††]The author is with the Faculty of Informatics, Osaka Gakuin University, Suita-shi, 564–8511 Japan.

[*]Presently, with the Takenaka Corporation.

a) E-mail: kounoe@is.nasit.jp

b) E-mail: yoneda@is.naist.jp

DOI: 10.1587/transinf.E95.D.3001

is promising as a delay test quality measure, timing-aware ATPG and fault simulation based on SDQL tend to take long CPU time. In addition, a test set based on SDQL becomes large compared with test sets targeting other fault models.

In this paper, we address the problem of ordering test patterns based on SDQL. Using SDQL, we address the problem to order test patterns so that, when appending test patterns in the obtained order to an initially empty test set, each additional test pattern achieves maximal reduction of SDQL. The problem considers a cumulative effect of SDQL, that is, SDQL shrinks fast when test patterns are applied in that order. Since SDQL directly represents an amount of test escape, and hence an amount of detectable delay defect, it is a suitable metric for test pattern ordering. This is different from the previous works since a selected test set is optimized only for a fixed defect size in [7] and [9].

In our test generation flow, we first generate a test set called *a base test set* using existing ATPG and order test patterns in the set. We propose a test pattern ordering method for a given base test set. The proposed method orders test patterns based on the lengths of sensitized paths, those are efficiently evaluated and correlated with SDQL. The proposed method avoids to apply time-consuming SDQL evaluation repeatedly, and therefore, orders test patterns within a reasonable time. In the experiments, we demonstrate the efficiency of the proposed test pattern ordering method, and also evaluate ATPG methods as base test set generators.

The rest of the paper is organized as follows. We introduce SDQM and SDQL in Sect. 2, and introduce the problem on test pattern ordering in Sect. 3. We then propose a test pattern ordering method in Sect. 4. Experimental results are given to evaluate the proposed test pattern ordering method and ATPG methods for base test set generation in Sect. 5. Finally, Sect. 6 concludes this paper.

2. Statistical Delay Quality Model (SDQM)

In this section, we introduce SDQM (Statistical Delay Quality Model) and SDQL (Statistical Delay Quality Level) proposed by Sato et al. [11], [12]. SDQM is proposed to evaluate test quality based on a delay defect distribution function which is derived from fabrication process. SDQL is a delay test quality measure that shows an amount of delay defects that should be detected but cannot be detected by a given test set and test timing.

SDQM considers rising and falling delay faults on each of input and output pins of each gate. Though the number of faults is the same as transition faults, a delay defect size is associated with each fault. SDQM considers delay defect sizes that should be detected and can be detected by a given test set and test timing. Let us consider an example shown in Fig. 1. Assume each gate has a delay of 1 ns and test timing is 5 ns. Test timing is an elapsed time from when test pattern is applied to primary inputs to when the responses are observed at primary outputs. In Fig. 1 (a) and (b), a rising fault f of the output of a gate G is detected. The path from

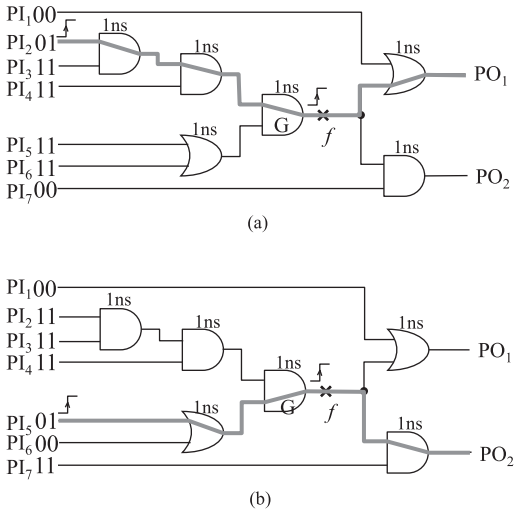

Fig. 1 Lengths of sensitized paths and detectable delay defect size.

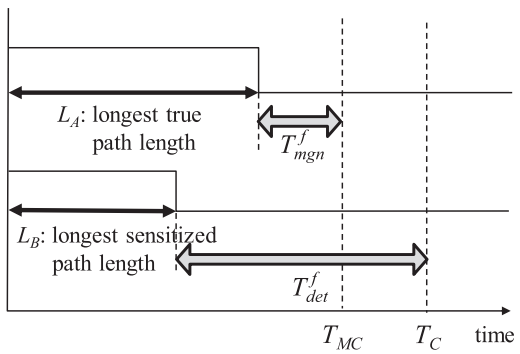

Fig. 2 Timing relations for the two types of paths.

a primary input PI_2 to a primary output PO_1 is sensitized in Fig. 1 (a) (a transition 0 to 1 applied at PI_2 is propagated to PO_1 through the path), while the path from a primary input PI_5 to a primary output PO_2 is sensitized in Fig. 1 (b). The lengths of the paths are 4 ns and 3 ns, respectively[†]. Since the test timing is 5 ns, a test pattern in Fig. 1 (a) detects delay defect of f if the size exceeds 1 ns, while Fig. 1 (b) detects delay defect of f if the size exceeds 2 ns. That is the test pattern in Fig. 1 (b) could not detect a small delay defect less than 2 ns though it may affect system behavior.

Figure 2 shows a concept of delay defect sizes that should be detected and can be detected by a given test set. Let f be a fault, and let L_A and L_B be the lengths of the longest true path passing through f and the longest path passing through f that is actually sensitized by the given test set, respectively. The true path is defined as a path that is designed to keep timing constraints. Let T_{MC} and T_C be system clock timing and test timing, respectively. The difference $T^f_{mgn} = T_{MC} - L_A$ is the minimum delay defect size that can affect system behavior and therefore should be detected. The difference $T^f_{det} = T_C - L_B$ is the minimum delay

[†]In this paper, a length of a path means an accumulated delay of the path.

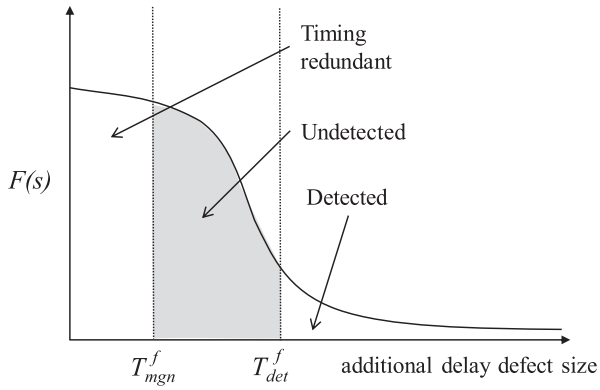

Fig. 3 SDQL for one fault.

defect size that can be actually detected by the given test set.

The SDQL for a given test set is defined as follow, where N is a set of faults and $F(s)$ is a delay defect distribution function of delay defect of size s.

$$SDQL = \sum_{f \in N} \int_{T^f_{mgn}}^{T^f_{det}} F(s)ds \quad (1)$$

It shows the total delay test quality of the chip based on the delay defect test escapes. A shadow area in Fig. 3 shows an amount of delay defect for one fault escaped during test. The SDQL is the total amount of such test escapes for the total faults. Therefore, smaller SDQL means better delay test quality.

3. Test Pattern Ordering

In this paper, we propose a method to order test patterns so that SDQL shrinks fast. We first formalize the problem with the desired input/output relationship.

Test Pattern Ordering Problem
Input: a set P of n test patterns and a circuit
Output: a sequence of test patterns such that, for any i ($1 \leq i \leq n$), i-th pattern achieves maximal reduction of SDQL among the last $n - i + 1$ test patterns when it is added to a test set of the first $i - 1$ test patterns

When we apply the test patterns in the obtained order, more defects can be detected as early as possible. We can use the result for test pattern selection problems under some constraints as follows.

Minimizing SDQL under Pattern Count Constraint
Input: a circuit, a set P of test patterns and an integer m
Output: a subset of P with the minimum SDQL among any subsets with m or less test patterns

Minimizing Pattern Count under SDQL Constraint
Input: a circuit, a set P of test patterns and a real number Q
Output: a subset of P with the minimum number of test patterns among any subsets with SDQL of Q or less

For both problems, we first generate a test set according to some criteria, and then order the generated test patterns. For the former problem, the first m test patterns in the obtained order give a good solution. For the latter problem, we can obtain a good solution by selecting test patterns in the obtained order until a selected test set satisfies a constraint on SDQL.

4. The Proposed Test Pattern Ordering Methods

In this section, we propose a method to solve the Test Pattern Ordering Problem. To solve the problems of the Minimizing SDQL under Pattern Count Constraint or the Minimizing Pattern Count under SDQL Constraint, we first generate a test set called a base test set. We will consider how to generate a base test set as an input of test pattern ordering later.

4.1 Ordering by Simulation

We first consider a straightforward method to solve the problem using timing-aware fault simulation that can evaluate SDQL. We call the method OrderBySimulation.

OrderBySimulation
P: a given test set
S: a sequence of test patterns, initially S is an empty sequence
1. Repeat Steps 2 and 3 until $P = \emptyset$
2. For each $p \in P$, calculate SDQL by simulation for a test set consisting of patterns in S and p
3. For p with the minimum SDQL, delete p from P, and append p to S

Let P_{base} be a base test set. The above OrderBySimulation requires fault simulation $|P_{base}|-(i-1)$ times to obtain SDQL to select the i-th test pattern. Therefore, fault simulation is applied $\sum_{i=1}^{|P_{base}|}(|P_{base}| - (i-1)) = \frac{1}{2}|P_{base}|(|P_{base}|+1)$ times. In general, timing-aware fault simulation to obtain SDQL is time-consuming. In timing-aware fault simulation, it cannot be accelerated by fault dropping like fault simulation to evaluate fault coverage. In case of fault simulation for fault coverage, once some fault is detected by some test pattern, the fault does not need to be cared by remaining test patterns. However, in fault simulation for SDQL, we have to consider not only detection but also the length of a sensitized path, and a fault can be dropped when it is detected by some test pattern with the longest true path passing through the fault. Therefore, fault simulation for SDQL treats many faults for every test pattern.

We examined CPU time required for fault simulation for ITC benchmark circuits. We used DesignComplier (Synopsys) for logic synthesis, PrimeTime (Synopsys) for static timing analysis, TetraMAX (Synopsys) for test pattern generation and fault simulation and SunFireX4100 with AMD Opteron256 3.0 GHz and 16 GB memory (Oracle). Table 1 shows circuit characteristics after logic synthesis and test generation results. In the table, the columns "CP",

Table 1 Circuit characteristics and timing-aware test generation results.

circuit	#gates	#FFs	T_{MC} (ns)	CP (ns)	#faults	#patterns	TGT (s)	FC (%)	FE (%)	SDQL	B in $F(s)$
b04	1,025	66	0.73	0.63	2,620	124	2.20	67.37	85.57	126.66	3.14
b12	1,730	121	0.55	0.46	4,794	698	6.71	88.40	91.20	51.27	4.22
b13	643	51	0.56	0.44	1,262	97	0.45	72.98	84.79	16.42	4.08
b14	8,460	215	3.27	2.94	22,904	1,325	2,429.06	84.27	86.74	8,497.64	0.70
b15	8,983	417	1.93	1.74	26,428	1,793	439.21	79.00	84.21	1,993.73	1.19
b17	27,766	1,317	1.93	1.74	80,612	5,745	2,193.66	85.96	88.17	5,654.23	1.19
b18	79,401	3,020	3.24	2.91	223,312	13,030	32,898.94	80.82	83.22	32,453.11	0.71
b19	152,599	6,042	3.24	2.91	433,410	24,058	103,917.61	81.24	83.04	63,921.51	0.71
b20	17,546	430	3.26	2.93	46,538	3,894	14,542.42	94.13	95.39	15,001.30	0.71

Table 2 CPU time of fault simulation(s).

circuit	transition	SDQL
b04	0.03	0.27
b12	0.14	3.29
b13	0.01	0.08
b15	1.98	86.30
b17	18.86	390.70
b18	185.07	5,501.43
b19	719.72	19,992.62
b20	12.87	1,504.55

"TGT", "FC", and "FE" are critical path length, test generation time, fault coverage, and fault efficiency, respectively.

Test patterns were generated under launch-on-capture (LoC) clocking scheme. TetraMAX Small Delay Defect Test mode are used for timing-aware test generation and fault simulation. We provided a delay defect distribution function $F(s)$ to TetraMAX in the form described as Eq. (2).

$$F(s) = A \cdot e^{-Bs} + C \tag{2}$$

In this experiment, we set $A = 1$, $C = 0$, and set B so that $F(T_{MC}) = 0.1$ holds. The values of B are shown in the column "B in $F(s)$" in Table 1.

Note that SDQL calculated by TetraMAX may be larger than the definition of SDQL. In the definition of SDQL, T^f_{mgn} is a difference of a system clock timing T_{MC} and the length of the longest true path passing through f. However, it is practically intractable to identify the longest *true* paths for all the faults. TetraMAX uses the longest path passing through each fault obtained by a *static timing analysis* instead. TetraMAX may use smaller values of T^f_{mgn} for some faults, and therefore, SDQL values in Table 2 do not reach zero.

Table 2 shows CPU time required for fault simulation to evaluate fault coverage of transition fault model and SDQL. From the table, we can find that fault simulation for SDQL takes too long CPU time, and therefore, OrderBySimulation is impractical for large circuits. Actually, the experiments in Sect. 5, we give up to apply OrderBySimulation.

4.2 Proposed Method

OrderBySimulation uses SDQL values to select a test pattern in each iteration, and therefore needs to apply time-consuming fault simulation repeatedly. In contrast, the proposed test pattern ordering uses the length of the longest sensitized path for selection. The lengths of the longest sensitized paths for all the faults for a test set can be easily found without fault simulation, once we obtain the length of the longest sensitized path for each pair of fault and test pattern. The proposed method first obtains, for each test pattern, an SDQL value and the lengths of the longest paths sensitized by the test pattern for all the faults. Though this first step needs timing-aware fault simulation to obtain SDQL, we do not need further fault simulation to order test patterns.

First, we explain how to find the lengths of the longest paths for a test set without fault simulation. Assume that we already order the first $i-1$ test patterns in S and the lengths of the longest paths sensitized by the patterns for all the faults are known. We also know the lengths of the longest paths sensitized by each test pattern p for all the faults. Let l^S_f and l^p_f be the length of the longest path sensitized by S and p for a fault f, respectively. It is obvious that the length of the longest path sensitized by S or p is $\max(l^S_f, l^p_f)$, and hence, we can easily obtain the the longest sensitized path length for a test set consisting of test patterns in S and p.

In the proposed method, we use the sum of the longest sensitized path lengths for all the faults as a metric to select patterns instead of an SDQL value. Though we do not directly evaluate SDQL values in each iteration like OrderBySimulation, the increase of the longest sensitized path length for some fault f implies the decrease of T^f_{det}. From Eq. (1), it implies the decrease of SDQL. Let L_S denote the sum of the longest sensitized path lengths for a test set consisting of patterns in S, and let $L_{S,p}$ denote the sum of the longest sensitized path lengths for a test set consisting of patterns in S and p. $L_{S,p}$ is obtained as follows.

$$L_{S,p} = L_S + \sum_{f \in N} \max(l^p_f - l^S_f, 0) \tag{3}$$

Let us define $Gain_{S,p}$ as follows.

$$\begin{aligned} Gain_{S,p} &= L_{S,p} - L_S \\ &= \sum_{f \in N} \max(l^p_f - l^S_f, 0) \end{aligned} \tag{4}$$

The proposed method orders test patterns based on Gain. The test pattern p with the largest $Gain_{S,p}$ is selected as the next pattern to S.

Let us explain the above idea using an example. In

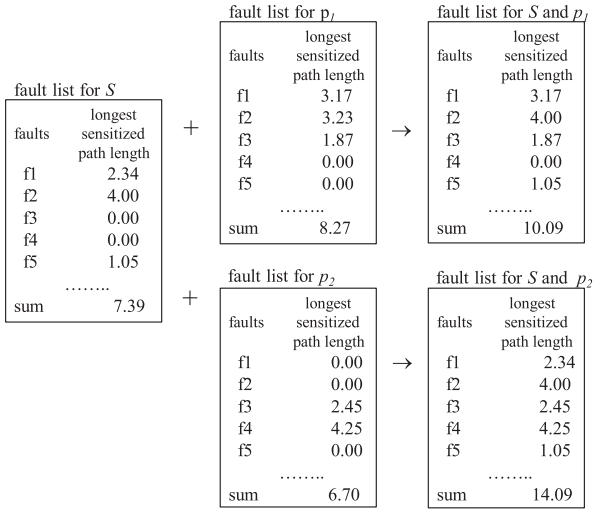

Fig. 4 Metric for test pattern selection.

Fig. 4, fault lists with the longest sensitized path lengths for a sequence S, test patterns p_1 and p_2 are shown. The longest sensitized path lengths when test patterns p_1 and p_2 are appended to S are also shown, respectively. These longest sensitized path lengths are obtained by fault lists for S and p_1, and fault lists for S and p_2, respectively. In this example, $Gain(S, p_1) = 10.09 - 7.39 = 2.70$ and $Gain(S, p_2) = 14.09 - 7.39 = 6.70$.

The outline of the proposed method is as follows. Note that the method applies timing-aware fault simulation for each test pattern in Step 1, and it reports SDQL for each test pattern. Therefore, we choose the test pattern with the minimum SDQL as the first test pattern.

Proposed
P: a given test set
S: a sequence of test patterns, initially S is an empty sequence
1. For each test pattern $p \in P$,
   apply timing-aware fault simulation
   and obtain SDQL and l_f^p for each f.
2. Select p with the minimum SDQL, and
   delete p from P, and append p to S
3. Repeat Steps 4 and 5 until $P = \emptyset$
4. For each $p \in P$, calculate $Gain_{S,p}$
5. For p with the maximum $Gain_{S,p}$
   delete p from P, and append p to S

In the proposed method, we apply timing-aware fault simulation to obtain SDQL for a test set with one test pattern only $|P_{base}|$ times, and therefore, it can order test patterns much faster than OrderBySimulation.

5. Experiments

We made experiments to evaluate the proposed test pattern ordering method and also to analyze test generation methods suitable for base test sets. The experiment environment is the same as described in Sect. 3.

5.1 Evaluation of Test Pattern Ordering Method

To evaluate the test quality of the proposed method, we compared the method with other selection methods. For comparison, we prepared three different test pattern ordering methods: (1) OrderByATPG: select test patterns in the order that they are generated by ATPG, (2) OrderByCoverage: select test patterns based on a gain of transition fault coverage, and (3) OrderBySDQL: select test patterns based on a gain of SDQL.

For the second method OrderByCoverage, we slightly modified Steps 2–5 in the proposed method so that it evaluates transition fault coverage of a test set consisting of test patterns in S and p, instead of $Gain_{S,p}$. This evaluation is also possible without fault simulation, once we apply fault simulation for each test pattern at the beginning.

The third method OrderBySDQL was used to evaluate accuracy of the proposed method. In the proposed method, we use the length of the longest sensitized path for each pair of test pattern and fault. From this information and system clock timing T_{MC}, we can calculate SDQL corresponding to each pair of test pattern and fault. This evaluation is also possible without fault simulation, once we apply fault simulation for each test pattern at the beginning. In OrderBySDQL, we evaluate how much SDQL decreases when we select a test pattern p as the next pattern. This value is used instead of $Gain_{S,p}$ in Proposed. This method can get the same order as OrderBySimulation while reducing the number of applications of timing-aware fault simulation.

The efficiency of this method depends on the complexity to evaluate SDQL using a delay defect distribution function. Actually, in these experiments, we used a delay defect distributed function in the form described as Eq. (2) and, in this case, we can calculate SDQL with a little computational effort.

We applied the three test pattern ordering methods for base test sets generated by timing-aware ATPG in Table 1. Table 3 shows CPU time for the proposed method Proposed and OrderByCoverage. The columns "fsim", "other" and "total" show CPU times for fault simulation, the other computation, and total computation, respectively. The proposed method takes longer CPU time than OrderByCoverage, since the proposed method applies timing-aware fault simulation to obtain SDQL at the beginning. However, the total CPU time is less than double of fault simulation time for the whole circuit shown in Table 2 except for a few small circuits. That is, the proposed method can order test patterns in a reasonable time. In addition, proposed method obtains test pattern ordering by selecting test patterns one by one in the order of the final test pattern order. Therefore, when we use the test pattern ordering for test set selection under some constraints, we don't need to order all the test patterns. It is enough to select test patterns until it satisfies a given constraint. In this case, we can reduce CPU time more.

Figures 5 and 6 show SDQL transition curves for b18 and b19. From Figs. 5 and 6, we can find that both Proposed

Table 3 CPU time of test pattern ordering.

circuit	proposed			coverage based		
	fsim	other	total	fsim	other	total
b04	0.85	0.13	0.98	0.40	0.09	0.49
b12	5.20	0.94	6.14	1.50	0.65	2.15
b13	0.49	0.09	0.58	0.29	0.05	0.34
b14	190.34	5.55	195.89	25.23	2.92	28.15
b15	73.23	7.11	80.34	17.46	3.26	20.72
b17	537.21	65.72	602.93	175.17	23.57	198.74
b18	5,168.63	390.59	5,559.22	1,191.56	120.45	1,312.01
b19	24,727.96	1,828.55	26,556.51	4,807.91	534.31	5,342.22
b20	1,071.01	23.55	1,094.56	162.51	10.72	173.23

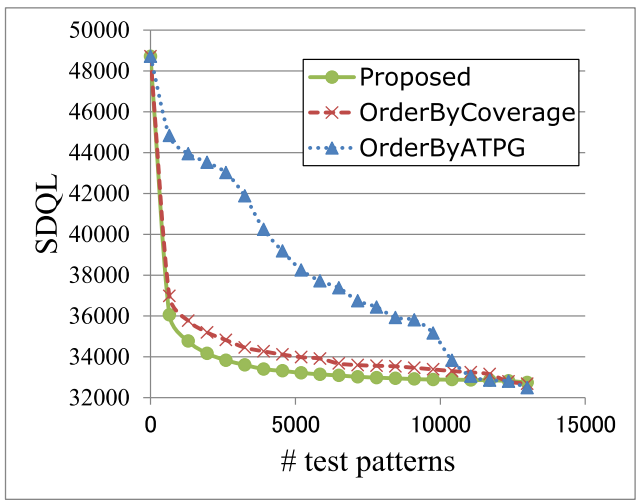

Fig. 5 SDQL and test pattern ordering methods for b18.

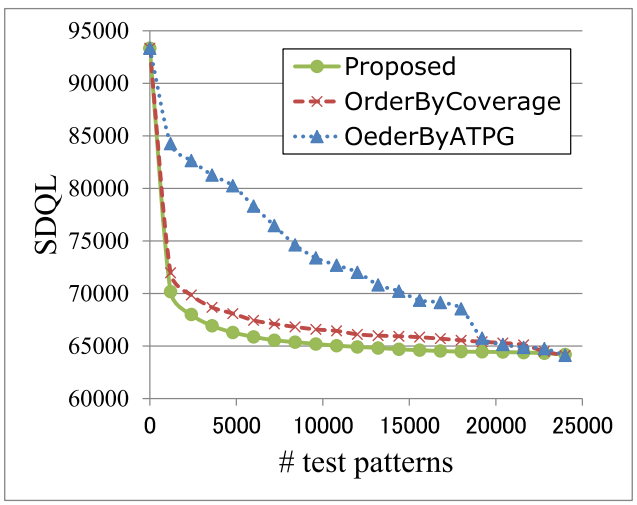

Fig. 6 SDQL and test pattern ordering methods for b19.

and OrderByCoverage shrink SDQL faster than OrderByATPG, and Proposed shrinks SDQL the fastest. When we use the proposed method for test pattern selection under a constraint, Proposed selects smaller (almost half size of) test set compared with OrderByCoverge under the same SDQL constraint, or Proposed achieves smaller SDQL under the same pattern count constraint.

We then compared Proposed with OrderBySDQL. We implemented OrderBySDQL using TetraMAX Small Delay Defect Test mode as follows. We first applied timing-aware fault simulation for each test pattern and obtained the longest path and the longest sensitized path for each fault. Then we calculated SDQL corresponding to each pair test pattern and fault. Since TetraMAX internally calculates SDQL corresponding to each fault to obtain total SDQL for all the faults, our implementation calculated SDQL for each fault twice. We do think it is fair to compare CPU time for both methods. Therefore, we use the result of OrderBySDQL only to evaluate the test quality.

We compared two methods for ITC99 benchmark circuits. Table 4 shows the results. We provided a delay defect distribution function $F(s)$ to TetraMAX in the form described as Eq. (4). We set $A = 1$, $C = 0$, and set B so that $F(T_{MC}) = 0.1$ holds for all the 7 circuits in Table 4. In addition, we used different delay defect distribution functions for circuits b17, b18 and b19, where we also set B so that $F(T_{MC}) = 0.05$ and $F(T_{MC}) = 0.025$ hold. The values of B are shown in the column "B in $F(s)$" in Table 4.

In this comparison, we found that differences of SDQL transition curves between Proposed and OederBySDQL are sufficiently small for all the cases. Therefore, instead of showing SDQL transition curves, we show the difference of SDQL transitions between two methods. Let $SDQL_{Proposed}(i)$ and $SDQL_{SDQL}(i)$ be SDQL for the first i patterns selected by Proposed and OrderBySDQL, respectively. We evaluated differences between $SDQL_{Proposed}(i)$ and $SDQL_{SDQL}(i)$ for all i from 0 to the number of test patterns as follows.

$$difference(i) = \frac{|SDQL_{Path}(i) - SDQL_{SDQL}(i)|}{SDQL_{SDQL}(i)} \times 100(\%) \quad (5)$$

In Table 4, "#no diff.", "max diff." and "ave. diff." show the number of cases with "$difference(i) = 0$" (cases where selected i patterns are the same for both methods), the maximum difference and the average difference among all i.

The maximum differences and the average differences between two methods are at most 1.05% and at most 0.16%, respectively. That is, for all the cases, differences between Proposed and OrderBySDQL are sufficiently small. For larger circuits b17, b18 and b19, we evaluated the two meth-

Table 4 Comparison between Proposed and OrderBySDQL.

circuit	#patterns	B in $F(s)$	#no diff.	max diff.(%)	ave. diff.(%)
b04	124	3.14	78	0.33	0.02
b12	698	4.22	263	0.37	0.08
b13	97	4.08	53	0.45	0.13
b15	1,793	1.19	503	0.81	0.09
b17	5,745	1.19	1,641	0.87	0.10
		1.55	1,752	0.55	0.05
		1.91	1,664	0.74	0.06
b18	13,030	0.71	3,533	1.05	0.16
		0.92	3,536	0.62	0.07
		1.14	3,623	0.35	0.04
b19	24,058	0.71	5,943	0.84	0.15
		0.92	5,965	0.56	0.07
		1.14	6,266	0.35	0.03

Table 5 Test pattern ordering for various base test sets.

circuit	b18							b19						
ATPG method	base test set				selection			base test set				selection		
	TGT (m)	#tp	SDQL	FC (%)	CPU time (m) fsim	other	total	TGT (m)	#tp	SDQL	FC (%)	CPU time (m) fsim	other	total
timing-aware	548.3	13,030	32,742	80.82	86.1	6.5	92.7	1,732.0	24,058	64,190	81.24	412.1	30.5	442.6
1-detect	8.3	7,494	36,816	74.83	51.0	3.8	54.8	27.9	13,403	70,944	75.26	210.3	14.6	225.0
2-detect	14.1	13,755	36,045	75.51	103.4	7.0	110.4	50.4	24,789	69,528	76.06	328.1	26.7	354.9
4-detect	25.1	25,932	35,429	76.27	196.3	13.7	210.0	88.7	46,878	68,440	76.73	680.2	55.8	736.0

ods with three different delay defect distribution functions. Though the maximum differences and the average differences are slightly different, they are sufficiently small. That is, Proposed can achieve almost the same test quality as OrderBySDQL. In addition, Proposed can achieve almost the same test quality as OrderBySimulation since OrderBySDQL obtains the same order of test patterns as OrderBySimulation.

5.2 ATPG for Base Test Set

Since the effectiveness of the proposed method depends on base test sets, in order to find a suitable base test set, we compared several base test sets generated by different test generation methods. In the experiment, we applied timing-aware ATPG and n-detect ATPGs for transition faults for $n = 1, 2, 4$ under LoC clocking scheme.

Table 5 shows CPU times to order test patterns in base test sets, where "# tp" and "FC" show the number of test patterns and transition fault coverage of the base sets. Though timing-aware APTG takes much longer CPU time compared with n-detect ATPGs, it achieved less SDQL and higher fault coverage. That is timing-aware ATPG can generate test sets with higher test quality. In addition, the numbers of test patterns of timing-aware ATPG are comparative or less than 2-detect and 4-detect ATPGs. That means each test patterns generated by timing-aware ATPG detect more amount of defects than n-detect ATPGs.

We ordered test patterns using the proposed selection method for four types of base test sets. Figures 7 and 8 show the SDQL transition curves for b18 and b19. In the figures, "(Proposed)" and "(ATPG)" mean the results on the proposed test pattern ordering and ordering by ATPG, re-

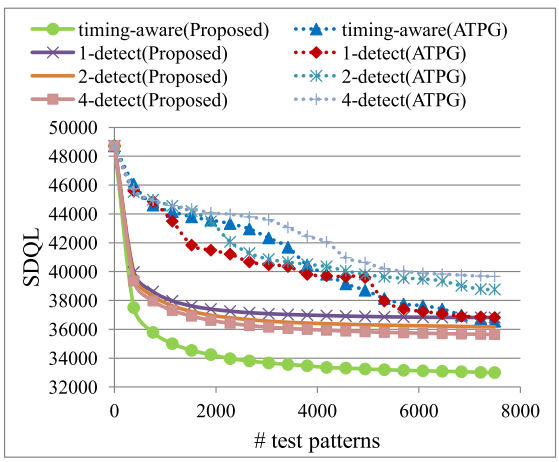

Fig. 7 SDQL and base test sets for b18.

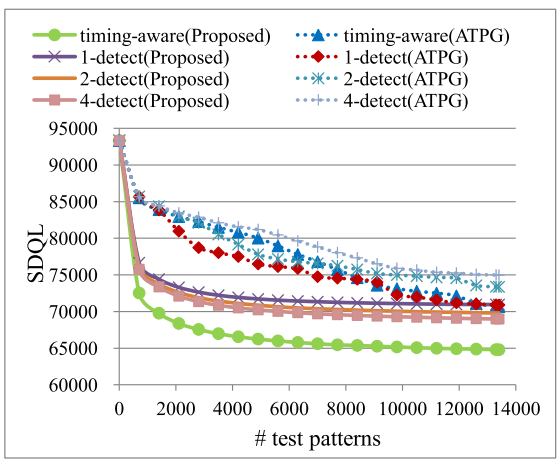

Fig. 8 SDQL and base test sets for b19.

spectively. We compared the curves up to the numbers of test patterns generated by 1-detect ATPG. From these figures, it can be found that base test sets generated by timing-aware ATPG can effectively shrink SDQL. That is, timing-aware ATPG is suitable to generate base test sets for test pattern ordering.

6. Comparison with Existing Methods

As we mentioned in Sect. 1, there are existing pattern selection methods. In this section, we compare the proposed method with existing methods.

Chao et al. [7] proposed a test pattern selection method for delay defects. The motivation of their method is to provide a high quality test sets for small delay defects without using timing-aware test generation or fault simulation, since they consider such timing-aware tools are time-consuming and impractical to be used. They used a large number of test patterns for n-detection of transition faults, and select small number of test patterns sufficient to detect small delay defects. That is different from our proposed method where we adopt timing-aware ATPG to generate a base test set, and our experiments revealed timing-aware ATPG obtained higher test quality than n-detection ATPG. In addition, Chao et al. proposed a dynamic timing analysis method to analyze capability of detecting a fixed size delay defect for each test pattern. The role of this method is like timing-aware fault simulation. This method selects, for every fault site, a test pattern with a criteria whether the pattern can detect a given fixed size of delay defect or not. This is different from our proposed method where SDQL is considered to evaluate test quality. That is, our selection method tries to select, for every fault site, a test pattern that can detect smaller defect. Therefore, we consider our proposed method provides more precise analysis to test quality for small delay defects.

Yilmaz et al. proposed test set pattern grading and selection method for small delay defects [9]. They used output deviations for screening small-delay defects, where the output deviation means a probability that the output does not have the expected signal-transition for a given test pattern. Their method achieved high test quality in terms of *long path coverage* that evaluates the number of sensitized long paths (a long path is a path with at least 70% of the system clock timing). This metric evaluates test patterns with whether they activate long paths or not. That is different from our method where we consider SDQL and hence consider total amount of test escape. We think comparison between different metrics of test quality is out of the range of this paper.

7. Conclusions and Remarks

In this paper, we proposed a test pattern ordering method which is a practical solution to reduce test data volume and hence test cost. Test pattern ordering orders test patterns so that more defective chips can be detected as early as possible to reduce test cost. We addressed test pattern ordering problem base on SDQL since it represents an amount of delay defects escaped to be detected by a test set, and therefore, it is suitable metric for test pattern ordering.

We proposed time efficient test pattern ordering method based on SDQL. The proposed method obtains test pattern ordering, which shrink SDQL fast, in a reasonable time. Furthermore, we can reduce CPU time more when we use the proposed test pattern ordering for test pattern selection under some constraints. We also examined suitable test generation for base test sets, and found the timing-aware test generation can lead to high delay test quality.

The proposed method can be used to reduce test data volume, and it can be combined with other test data volume reduction methods. For example, seeds for LFSR-based test compression method can be ordered, if we apply the proposed method for test patterns decompressed from seeds. Indeed, there have been proposed such seed ordering methods [16], [17].

Acknowledgment

The authors would like to thank Prof. Seiji Kajihara and Prof. Yasuo Sato with Kyusyu Institute of Technology and Prof. Satoshi Ohtake with Oita University for their variable comments to this work.

References

[1] P. Nigh and A. Gattiker, "Test method evaluation experiments & data," Proc. International Test Conf., pp.454–463, 2000.
[2] M. Tehranipoor, K. Peng, and K. Chakrabarty, Test and Diagnosis for Small-Delay Defects, Springrer, 2011.
[3] W. Maly, "Optimal order of the VLSI IC testing sequence," Proc. Design Automation Conf., pp.560–566, June 1986.
[4] W. Jiang and B. Vinnakota, "Defect-oriented test scheduling," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.9, no.3, pp.427–438, June 2001.
[5] X. Lin, J. Rajski, I. Pomeranz, and S.M. Reddy, "On static test compaction and test pattern ordering for scan designs," Proc. International Test Conf., pp.1088–1097, 2001.
[6] I. Pomeranz and S. Reddy, "On maximizing the fault coverage for a given test length limit in a synchronous sequential circuit," IEEE Trans. Comput., vol.53, no.9, pp.1121–1133, Sept. 2004.
[7] M.C.T. Chao, L.C. Wang, and K.T. Cheng, "Pattern selection for testing of deep sub-micron timing defects," Proc. Design, Automation and Test in Europe Conf. and Exhibition, pp.1060–1065, 2004.
[8] K.Y. Cho and E.J. McCluskey, "Test set reordering using the gate exhaustive test metric," Proc. VLSI Test Symposium, pp.199–204, 2007.
[9] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "Test-pattern grading and pattern selection for small-delay defects," Proc. VLSI Test Symposium, pp.233–239, 2008.
[10] Z. Wang and K. Chakrabarty, "Test-quality/cost optimization using output-deviation-based reordering of test patterns," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.27, no.2, pp.352–365, 2008.
[11] Y. Sato, S. Hamada, T. Maeda, A. Takatori, and S. Kajihara, "Evaluation of the statistical delay quality model," Proc. Asia and South Pacific Design Automation Conf., pp.305–310, 2005.
[12] Y. Sato, S. Hamada, T. Maeda, A. Takatori, Y. Nozuyama, and S. Kajihara, "Invisible delay quality? SDQM model lights up what could not be seen," Proc. International Test Conf., p.47.1, 2005.

[13] X. Lin, K.H. Tsai, C. Wang, M. Kassab, J. Rajski, T. Kobayashi, R. Klingenberg, Y. Sato, S. Hamada, and T. Aikyo, "Timing-aware ATPG for high quality at-speed testing of small delay defects," Proc. Asian Test Symp., pp.139–146, 2006.
[14] Synopsys, TetraMAX ATPG User Guide, Version C-2009.06-SP2, Sept. 2009.
[15] A. Uzzaman, M. Tegethoff, B. Li, K.M. Cauley, S. Hamada, and Y. Sato, "Not all delay tests are the same - SDQL model shows true-time," Proc. Asian Test Symp., pp.147–152, 2006.
[16] M. Yilmaz and K. Chakrabarty, "Seed selection in LFSR-reseeding-based test compression for the detection of small-delay defects," Proc. Design, Automation and Test in Europe, pp.1488–1493, 2009.
[17] T. Yoneda, M. Inoue, A. Taketani, and H. Fujiwara, "Seed ordering and selection for high quality delay test," Proc. Asian Test Symp., pp.313–318, 2010.

Michiko Inoue received her B.E., M.E, and Ph.D. degrees in computer science from Osaka University in 1987, 1989, and 1995 respectively. She worked at Fujitsu Laboratories Ltd. from 1989 to 1991. She is a Professor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST). Her research interests include distributed algorithms, parallel algorithms, graph theory and design and test of digital systems. She is a member of the IEEE, the IPSJ, and the Japanese Society for Artificial Intelligence.

Akira Taketani received the B.E. degree in science from Kochi University in 2008 and the M.E. degree in information science from Nara Institute of Science and Technology in 2010. Presently, he works at Takenaka Corporation.

Tomokazu Yoneda received the B.E. degree in information systems engineering from Osaka University, Osaka, Japan, in 1998, and the M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology, Nara, Japan, in 2001 and 2002, respectively. Presently he is an Assistant Professor in Graduate School of Information Science, Nara Institute of Science and Technology. His research interests include VLSI CAD, design for testability, low power testing and delay testing. He is a senior member of the IEEE.

Hideo Fujiwara received the B.E., M.E., and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1969, 1971, and 1974, respectively. He was with Osaka University from 1974 to 1985, Meiji University from 1985 to 1993, Nara Institute of Science and Technology (NAIST) from 1993 to 2011, and joined Osaka Gakuin University in 2011. Presently he is Professor Emeritus of NAIST and a Professor at the Faculty of Informatics, Osaka Gakuin University, Osaka, Japan. His research interests are logic design, digital systems design and test, VLSI CAD and fault tolerant computing, including high-level/logic synthesis for testability, test synthesis, design for testability, built-in self-test, test pattern generation, parallel processing, and computational complexity. He has published over 400 papers in refereed journals and conferences, and nine books including the book from the MIT Press (1985) entitled "Logic Testing and Design for Testability." He received the IECE Young Engineer Award in 1977, IEEE Computer Society Certificate of Appreciation Awards in 1991, 2000 and 2001, Okawa Prize for Publication in 1994, IEEE Computer Society Meritorious Service Awards in 1996 and 2005, IEEE Computer Society Continuing Service Award in 2005, and IEEE Computer Society Outstanding Contribution Award in 2001 and 2009. Dr. Fujiwara is a life fellow of the IEEE, a Golden Core member of the IEEE Computer Society, and a fellow of the IPSJ (the Information Processing Society of Japan).