Pattern Selection For Testing Of Deep Sub-Micron Timing Defects

Mango, C.-T. Chao, Li-C. Wang, Kwang-Ting Cheng
Department of ECE, UC-Santa Barbara
mango,licwang, timcheng@ece.ucsb.edu

Abstract

Due to process variations in deep sub-micron (DSM) technologies, the effects of timing defects are difficult to capture. This paper presents a novel coverage metric for estimating the test quality with respect to timing defects under process variations. Based on the proposed metric and a dynamic timing analyzer, we develop a pattern-selection algorithm for selecting the minimal number of patterns that can achieve the maximal test quality. To shorten the run time in dynamic timing analysis, we propose an algorithm to speed up the Monte-Carlo-based simulation. Our experimental results show that, selecting a small percentage of patterns from a multiple-detection transition fault pattern set is sufficient to maintain the test quality given by the entire pattern set. We present run-time and accuracy comparisons to demonstrate the efficiency and effectiveness of our pattern selection framework.

1. Introduction

With the continual scaling of manufacturing technologies, process variations on physical devices (such as gate length fluctuation, sub-wavelength lithography and noise) are exercising increasing influence over design timing characteristics [1, 2, 3]. The timing effect from those variations, therefore, can no longer be characterized by a deterministic model such as a worst-case or nominal delay model. To accurately model the timing characteristics of those variations, using statistical timing models for each single device on a chip becomes mandatory for timing analysis in DSM technologies.

However, the delay variance on each single device results in a huge number of possibilities for a circuit delay configuration. This high computation cost associated with statistical timing models heavily increases the difficulty of their effective usage in delay testing, including applications such as static timing analysis, dynamic timing analysis, defect simulation, and timed ATPG.

Figure 1 compares the time complexities of some problems in the statistical timing domain. The static timing analysis reports the structural worst-case delay of the circuit. The dynamic timing analysis reports the circuit delays for each pattern. Hence, the complexity of dynamic analysis can be an order-of-magnitude higher than that of static analysis. Furthermore, the timing-defect simulation considers the effects of potential timing defects for each pattern and hence is more complex than dynamic analysis. As for *timed* ATPG, it needs to assign the patterns to maximize the effects of timing defects. Authors in [8, 9] proposed *timed* ATPG algorithms. However, due to their high complexity, timed ATPGs have not been practical for large circuits.

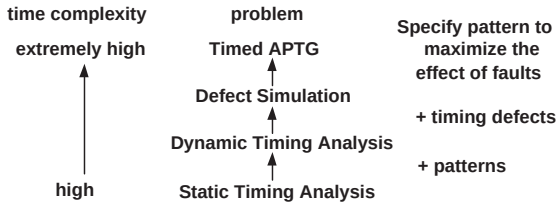

Figure 1: Comparison of time complexities in the statistical timing domain.

Because of the complexity issues, in the statistical timing domain past researches have focused more on the area of static timing analysis [4, 5, 6], which happens to be the problem with the lowest level of complexity in the hierarchy mentioned above. For dynamic timing analysis, the existing method [7] remains too slow for practical use. Also, there are no known efficient methods for timing-defect simulation.

Since we can never afford a timed ATPG to generate patterns specifically to target timing defects in the statistical domain, we may have to utilize an ATPG considering a less accurate timing model or one without timing (such as an ATPG for transition faults). Without an accurate ATPG, *pattern selection* becomes a mandatory step to obtain high-quality test patterns for testing of DSM timing defects. In this methodology, the job of ATPG is not to produce the final test but to produce a superset of test patterns such that they can be further processed in the pattern selection step.

The most intuitive way to do pattern selection for timing defects in the statistical timing domain is to apply a timing-defect simulator to evaluate the defect coverage for each pattern, just as we do in traditional fault simulation in the logic domain (i.e. transition fault simulation). After defect simulation, we may rank the patterns by their defect coverages and select the minimal pattern set that can achieve the highest coverage. Unfortunately, as described above, defect

1530-1591/04 $20.00 (c) 2004 IEEE

simulation in the statistical timing domain has a very high complexity, and hence cannot be used in practice.

In this paper, we define a novel coverage metric to evaluate the test quality with respect to timing defects. The objective of this coverage metric is to guide pattern selection for finding a minimal set of patterns that can capture the maximal number of timing defects. More importantly, this coverage metric allows us to avoid using defect simulation in pattern selection. Only a dynamic timing analysis is required. To further shorten the runtime in the dynamic timing analysis, we also propose a speedup method for the Monte-Carlo-based dynamic timing analyzer to efficiently collect the timing information for computing the coverage metric from a large number of patterns. After the dynamic timing analysis, we propose a deterministic algorithm to select patterns based on the coverage metric.

2. Problem definition

2.1. Statistical timing models

In our statistical timing analysis, a cell-based approach is used to model the timing [7]. It requires a cell library of the output arrival time and output transition time for each pin-to-pin segment of each type of cell. In this library, the output arrival time is modeled as a *pdf* (*probability density function*). The input transition time and the output loading will be the indices to load the corresponding pdfs.

2.2. Timing defect model

The timing defect model in our pattern-selection framework is similar to a gate delay fault model. We call our model a *pin-to-pin timing defect* model. A pin-to-pin timing defect is notated by (d, $s(g_1,g_2)$), where d represents the delay size of the timing defect, g_i represents the gate output, and segment (g_1,g_2) represents the defect location. As shown in Figure 2, a timing defect between g and e could be a slow transition on the cell pin-to-pin, (g, g_1), or a slow transition on the interconnect, (g_1, e).

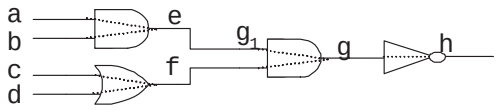

Figure 2: An example of a pin-to-pin timing defect.

2.3. Pattern selection problem

The primary idea of this paper is to select an applicable number of patterns from a given pattern set and, at the same time, to achieve the requirement for detecting timing defects. The best measurement of the capability for detecting timing defects is the fail rate (# of faulty instances detected over total # of faulty instances). Since the pattern selection is based on a given pattern set, the relative value of fail rates is more important than the absolute value. We define the term, *EPC*, (*effective pattern coverage*), to represent the percentage of capability of detecting timing defects for the selected patterns compared with the total given patterns.

$$EPC = \frac{\text{fail rate detected by the selected patterns}}{\text{fail rate detected by the total given patterns}} \quad (1)$$

In this paper, we first focus on the pattern selection for fixed-size timing defects. Then we will explain how to repeatedly apply our pattern selection for fixed-size timing defects to consider all timing defects.

The pattern selection problem based on the statistical timing model has the following as inputs:

- a netlist, *Net*,
- a statistical cell delay library, *Lib*,
- a set of patterns, P_{given},
- a clock period, *clock*,
- an *artificial* defect size *d_size* given by the user, and
- a desired effective pattern coverage, *EPC*.

The objectives of the pattern selection is the following:

- achieve the effective pattern coverage, and
- minimize the number of patterns.

3. The coverage metric

In this section, we first define $CP_i(s)$, the critical probability of a pin-to-pin segment s under pattern i.

- $CP_i(s)$: the probability that the pattern i can generate a circuit delay exceeding the clock period if there is a timing defect on the pin-to-pin segment s.

To consider the topological overlap of detecting timing defects for a set of patterns, we further define $CP_P(s)$, the critical probability of a pin-to-pin segment s under a set of patterns P.

- $CP_P(s)$: the probability that at least one pattern among the set of patterns P can generate a circuit delay exceeding the clock period if there is a timing defect on the pin-to-pin segment s.

The $CP_P(s)$ can be obtained by the following equation.

$$CP_P(s) = 1 - \prod_{i \in P}(1 - CP_i(s)) \quad (2)$$

We assume that timing defects may locate uniformly over all pin-to-pin segments. Therefore, the critical probability of each pin-to-pin segment should be treated with equal importance. Hence, the coverage metric of a set of patterns P, $Cov(P)$, can be formulated as the summation of each $CP_P(s)$.

$$Cov(P) = \sum_s CP_P(s) \quad (3)$$

In our pattern-selection algorithm, we use this coverage metric, $Cov(P)$, to guide the pattern selection process. To maximize the coverage metric, a new selected pattern needs not only to achieve maximum $CP_i(s)$ over all pin-to-pin segments, but also to avoid the topological overlap of pin-to-pin segments on which the existing patterns have already generated a high $CP_P(s)$.

3.1. Computing the coverage metric

3.1.1 Notations and Terminologies

First, we define two terms, *controlling-transition input* and *non-controlling-transition input*, on a gate under a 2-vector pattern (i_1, i_2). A gate input is called a *controlling-transition input* (or conversely, a *non-controlling-transition input*) if there is a transition on the input and the logic value given by i_2 is the controlling (or non-controlling) value of the gate. For example, in Figure 4, *a* is a controlling-transition input of the NAND gate, and *e* is a non-controlling-transition input of the AND gate.

Then, we define the notations for a gate with an output pin, *out*, and *n* input pins, in_1, $in_2, \ldots, in_n$.

- $MEDR(out)$: The minimum extra delay required for output pin *out* to result in a delay exceeding the clock (the *slack*).
- $MID(out, in_j)$: The maximum possible increase of delay on the segment $s(out, in_j)$.
- $arrival(p)$: The arrival time of pin *p*.
- $delay(out, in_j)$: The pin-to-pin delay on segment $s(out, in_j)$.

The other notations in this section follows the notations defined in Section 2.3.

3.1.2 General trace-back method

The most crucial part of computing the coverage metric is to obtain the $CP_i(s)$ of each pin-to-pin segment *s* and each pattern *i*. For each pattern *i*, a Monte-Carlo-based timing analyzer is applied to generate timing configuration samples based on the statistical timing model shown in section 2.1. The timing analyzer will stop sampling once the sampled mean and variance of the circuit delay converge. For each timing configuration sample, a trace-back method is applied to decide which pin-to-pin segment is *critical* (which means that lumping a timing defect on the segment will generate a circuit delay exceeding the clock period). Then a variable, called *critical counter*, is used to record the number of timing configuration samples where the particular pin-to-pin segment is considered to be critical. $CP_i(s)$ is just the critical count of *s* over the number of total timing configuration samples in the Monte-Carlo simulation for pattern *i*. Figure 3 shows the flow of this Monte-Carlo-based timing analysis for a pattern *i*.

Procedure: Monte-Carlo-Based Timing Analysis for Pattern *i*
Input: *Net*, *Lib*, *d_size*, *clock*, Pattern *i*;
Output: Circuit Delay Distribution, $CP_i(s)\ \forall s$;
\\ $Mean_{new}$, Var_{new}–New sampled mean and variance of circuit delay
\\ $Mean_{old}$, Var_{old}–Old sampled mean and variance of circuit delay
\\ *t*–Precision threshold
\\ $arrival(p)$–Output arrival time of pin *p*
\\ $c\_counter(s)$–Critical counter of segment *s*
1 **while** ($|Mean_{new} - Mean_{old}| > t$ or $|Var_{new} - Var_{old}| > t$){
2 generate a timing configuration sample based on *lib*;
3 **for**(all PO){
4 **if** $arrival(PO) > clock - d\_size$ {
5 trace back and increase $c\_counter(s)$ for each critical *s*
6 }
7 }
8 update $Mean_{new}$, Var_{new}, $Mean_{old}$, and Var_{old}
9 }

Figure 3: Procedure of Monte-Carlo-based timing analysis for pattern *i*.

On each timing configuration sample in the Monte-Carlo simulation (shown in Figure 3), we check to see whether the arrival time on each PO is larger than the clock period minus the given artificial defect size. If yes, then we trace back the segments from the PO to locate which pin-to-pin segments contribute to this delay, mark those segments as *critical*, and then increase the critical counts of those critical segments by 1. Figure 4 shows a simple example of locating critical segments. The number on each input transition represents the input arrival time, and the number on each pin-to-pin dash line represents the sampled pin-to-pin delay. Segments (h,g), (g,e), and (e,a) contribute to the delay on the primary output *h*. Since $arrival(h) + d\_size > clock$, (h,g), (g,e), and (e,a) are critical on this timing configuration sample.

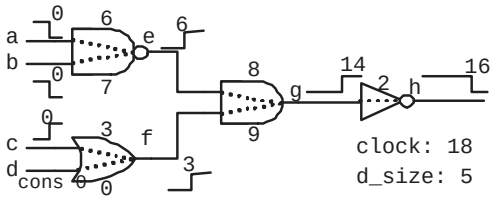

Figure 4: A example of our trace-back method. (h,g), (g,e), and (e,a) are critical segments.

However, the general trace-back method is not sufficient. Two more rules have to be listed in Sec 3.1.3 and Sec 3.1.4

3.1.3 Mark more non-controlling-transition inputs

First, we show how to compute $MEDR(out)$ for a gate output, *out*. From a primary output *PO* where $arrival(PO) + d\_size > clock$, $MEDR(PO)$ is the *clock* minus $arrival(PO)$. Then we propagate the $MEDR(PO)$ along the the segments contributing the delay. For any other gate *other* , we first set $MEDR(other)$ as 0. For example, in Figure 4, $MEDR(h)$=$MEDR(g)$=$MEDR(e)$=2. The other *MEDR*s are all set as 0 initially.

For a gate with multiple non-controlling-transition inputs, the output arrival time is determined by the segment (out, in_j) having the longest arrival time. To mark the segment (out, in_j) as critical is correct. However, any other segment (out, in_k) with a non-controlling transition may also be critical in the case when a timing defect occurs. In this case, $MEDR(in_k)$ is not 0 and can be obtained by the following equation.

$$\begin{aligned} MEDR(in_k) &= arrival(out) - arrival(in_k) \\ &\quad - delay(out, in_k) + MEDR(out) \end{aligned} \quad (4)$$

If the size of a timing defect, *d_size*, is larger than $MEDR(in_k)$, another trace-back will be performed from in_k with $MEDR(in_k)$ in equation 4. For example, in Figure 5, segment (out, in_1) contributes to the output arrival time of 14. According to equation 4, $MEDR(in_1)$=2, $MEDR(in_2)$=8, $MEDR(in_3)$=5. $MEDR(in_1)$ and $MEDR(in_3)$ are both less than *d_size*, 6. Segment (out, in_1) and (out, in_3) are hence critical, so a trace-back will be performed from each of in_1 and in_3. $MEDR(in_2)$ is larger than *d_size*. Segment (out, in_2) is hence not critical, so no further trace-back will be performed from in_2.

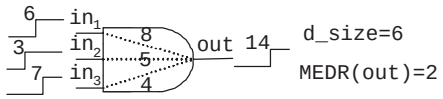

Figure 5: A example of marking another non-controlling-transition input.

3.1.4 Block some controlling-transition inputs

If a gate has more than one controlling input transition, the shortest arrival time among the input controlling transitions will dominate the output arrival time. Let this input fall on the segment $s(out, in_j)$ where *out* is the gate output and in_j is the input. If a defect falls on $s(out, in_j)$ to make the new arrival time on $s(out, in_j)$ exceed the second shortest arrival time among the controlling input transitions, then the segment with the second shortest arrival time, $s(out, in_k)$, will dominate the gate arrival time. It implies that the maximum possible increase of delay on $s(out, in_j)$ is limited by $s(out, in_k)$. Hence, we have

$$\begin{aligned} MID(out, in_j) &= arrival(out) - arrival(in_k) \\ &\quad - delay(out, in_k) \end{aligned} \quad (5)$$

During the trace-back, we check to see whether $MID(out, in_i)$ is larger than $MEDR(out)$. If yes, we keep on tracing back from in_i; if no, we stop tracing. For example, in Figure 6, segment $s(out, in_3)$ contributes to the output arrival time of 9. However, $MID(out, in_3)$=(6+4)-(4+5)=1 is less than $MEDR(out)$. So the segment $s(out, in_3)$ is not critical, and no further trace-back is performed from in_3.

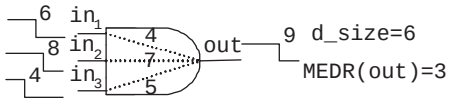

Figure 6: A example of blocking a controlling-transition input. $s(out, in_3)$ contributes to the output arrival time but is not critical

With all the trace-back rules in this section, the critical segments in Figure 4 should be (h, g), (g, e), (g, f), and (f, c).

4. Pattern-selection algorithm

In this section, we propose a pattern-selection algorithm, named *PSFTD*, for fixed-size timing defects. In PSFTD, the proposed coverage metric is used to guide each selection of a pattern by measuring the capability of detecting timing defects. Basically, the PSFTD uses a greedy method to select patterns. For each selection, the non-selected pattern which can generate the largest increase in the coverage is selected. The PSFTD stops when the ratio of the computed coverage of selected patterns over the computed coverage of the total patterns reaches the given effective pattern coverage *EPS*.

The PSFTD can be repeatedly applied to select patterns based on a sequence of artificial defect sizes in order to achieve the balance of detecting both small-size and large-size defects. In essence, a small artificial defect size favors the selection of patterns that focus on the coverage of long paths. A large artificial defect size favors the selection of patterns that can cover a wide region of the circuit (increase topological coverage). Whether to cover more long paths or to achieve higher topological coverage depends on the user. In any case, the control of the artificial defect size in our pattern selection scheme controls the tradeoff between these two.

4.1. Experimental results for pattern selection

In our experiment, the statistical timing models were obtained through pre-characterization of cell libraries using a Monte-Carlo-based SPICE simulator (ELDO) [10] based on a 0.25μm, 2.5V CMOS technology. The patterns to be selected are the test patterns for 15-detection of transition faults generated by a commercial ATPG tool. A defect simulator is applied for each pattern set to simulate 1000 circuit instances, each with a random-location fixed-size timing defect. With the defect simulation results, the fail rate of each pattern set can be calculated and hence, the actual *EPC* for each pattern set can be obtained by Equation 1. The dynamic timing analyzer used for computing the coverage metric in pattern selection is a speedup version of a Monte-Carlo-based timing analyzer, which will be discussed in Section 5.The unit for clock period and defect size in this section is 10ps.

Table 1 shows the number of patterns and the actual *EPC* for each pattern set on benchmark circuit s1488 with a clock

Table 1: **Actual** *EPC* **of each selected pattern set. Circuit s1488, clock 220.**

	desired *EPC*	0.2	0.4	0.6	0.8	0.9	0.99	0.999	0.9999	1	eff. pttn	P_{given}
d_size 20	# of pttn	1	2	3	4	9	40	61	69	89	133	6229
	actual *EPC* of PSFTD	0.318	0.545	0.795	0.886	0.932	1	1	1	1	1	1
	actual *EPC* of RPS	0.045	0.500	0.250	0.455	0.727	0.932	0.977	0.955	1	1	1
d_size 40	# of pttn	1	2	4	7	13	54	103	139	251	593	6299
	actual *EPC* of PSFTD	0.317	0.485	0.634	0.842	0.950	0.950	0.970	0.990	1	1	1
	actual *EPC* of RPS	0.158	0.099	0.347	0.554	0.743	0.743	0.931	0.970	0.970	1	1
d_size 60	# of pttn	1	3	5	14	25	88	163	206	485	1802	6299
	actual *EPC* of PSFTD	0.271	0.517	0.662	0.860	0.952	0.981	0.981	0.981	0.995	1	1
	actual *EPC* of RPS	0.130	0.251	0.420	0.609	0.725	0.865	0.923	0.918	0.952	1	1

of 220 and three different artificial defect sizes. The nine columns with labels from 0.2 to 1 show the results by setting the desired EPC equal to the label. The last two columns correspond to the results based on the effective patterns (eff. pttn) and the original patterns (P_{given}). The effective patterns are selected using our coverage metric. An *effective* pattern here means a pattern i which might contribute an increase to our coverage metric, i.e., generating at least one non-zero $CP_i(s)$ for any segment s.

From column "0.2" to column "1," the "# of pttn" rows show the numbers of patterns selected by PSFTD based on the effective patterns. To evaluate PSFTD, we also select patterns randomly from the effective patterns. The results are labeled with the term *RPS*. In RPS, we randomly select the same number of patterns as that by PSFTD.

For the selected defect size of 20, 133 effective patterns can be first selected out of the 6228 patterns and achieve an actual *EPC* of 1, i.e, exactly the same capability of defecting timing defects as the 6229 patterns. With a desire *EPC* of 1, we can select 89 out of the 133 effective patterns, and still achieve an actual *EPC* of 1. Then, with a desired *EPC* of 0.9, we can select 9 out of 89 patterns, and achieve an actual *EPC* of 0.932. Furthermore, the pattern sets selected by PSFTD always achieve higher actual *EPC*s than those achieved by RPS. This indicates that the PSFTD indeed has used a good metric for pattern selection.

Table 2 reports the number of patterns and actual *EPC* of each pattern set selected by PSFTD on some other benchmark circuits. We arbitrarily choose the defect size for each circuit, and the similar trends among the actual *EPC*s of pattern sets can be observed on each circuit as well.

Table 2: **Actual** *EPC* **of each PSFTD-selected pattern set for c880, c1355, s5378, and s35932 with clocks in Table 3.**

	desired EPC	0.2	0.6	0.9	0.999	1	eff. pttn	P_{given}
c880	# of pttn	1	4	16	41	61	239	3008
d_size 80	actual *EPC*	0.311	0.651	0.866	0.956	0.969	0.992	1
c1355	# of pttn	2	8	24	58	90	1715	4911
d_size 40	actual *EPC*	0.289	0.671	0.927	1	1	1	1
s5378	# of pttn	2	14	36	80	224	385	11035
d_size 50	actual *EPC*	0.324	0.69	0.915	0.958	0.986	1	1
s35932	# of pttn	1	2	9	424	865	1341	2769
d_size 50	actual *EPC*	0.396	0.680	0.971	0.991	0.997	1	1

Table 3 shows the runtime of the PSFTD and the runtime of defect simulation over the total given patterns P_{given}. Both the PSFTD and defect simulation deal with the same number of patterns, and the defect simulation only consider

Table 3: **Runtime of pattern selection and defect simulation.**

circuit	c880	c1355	s1488	s5378	s35932
clock (10ps)	240	240	220	275	210
d_size (10ps)	80	40	40	50	50
runtime of PSFTD (s)	77	796	278	2875	7084
runtime of defect sim (s)	5659	17632	19869	166383	447664

1000 instances. As the results, the runtime of the defect simulation is from 22X to 73X longer than the PSFTD. This big gap in runtime implies that applying our pattern selection is more efficient than using a defect simulator directly. Also, the inefficiency of defect simulation prevents us from simulating more faulty instances for large circuits. For example, the number of total pin-to-pin segments in s35932 is 33773. If the number of instances in defect simulation is the same as the number of pin-to-pin segments, the defect simulation will take 15118956 seconds (almost 175 days) to finish. This is exactly why we need the proposed pattern selection that depends on dynamic timing analysis rather than defect simulation.

5. Fast dynamic timing analysis

Most of computation in our pattern select framework comes from the statistical dynamic timing analysis. The original Monte-Carlo timing analysis in [7] requires simulating a large sample of instances in order to reach the stopping point where both the sampled mean and variance have converged. Monte-Carlo simulation is inherently expensive for selecting patterns from large pattern sets. In this section, we propose an efficient method to speed up the Monte-Carlo timing analysis to be used in our pattern-selection framework.

The purpose of the original dynamic timing analysis is to capture the circuit delay distribution for each pattern. Therefore, in Figure 3, line 1, the Monte-Carlo simulation for each pattern stops until the sample mean and variance converge. However, in our pattern-selection framework, the critical probability $CP_i(s)$ for each pattern i and each segment s is the real target. It means that we may stop sampling earlier, as long as the critical probability $CP_i(s)$ has converged.

In Figure 3, line 4, only a delay on a PO larger than $clock - d\_size$ could affect the critical probability $CP_i(s)$. Hence, we set the reference line as $clock - d\_size$. If the sampled circuit delays are always far below (or above) this reference line, then the sampling can stop much earlier. In

Table 4: Comparison of average runtime and average difference of critical probabilities.

circuit	# of total patterns	OLD	$k = 10$		$k = 20$		$k = 40$	
		avg runtime(s)	avg accuracy(%)	avg runtime(s)	avg accuracy(%)	avg runtime(s)	avg accuracy(%)	avg runtime(s)
c880	3008	468	0.107	43	0.100	57	0.093	84
c1355	4911	4922	0.202	389	0.158	462	0.141	563
s1488	6229	1639	0.147	171	0.123	219	0.119	292
s5378	11035	21325	0.186	2831	0.168	3408	0.162	4285

this case, a small number of samples is enough to determine that $CP_i(s)$ is 0 (or 1).

To quickly determine the range of a distribution so that we can decide if the Monte-Carlo sampling should stop earlier, a 3-sigma bound measurement is used. That is, after the first k samples, we test to see whether the 3rd standard deviation above the mean is smaller than the value of the reference line (likewise, we test to see whether the 3rd standard deviation below the mean is larger than the value of the reference line). If yes, we stop sampling. If no, we keep on sampling until the sampled mean and variance converge.

5.1. Experimental results

In this section, we compare the speedup version of the Monte-Carlo-based timing analyzer with the original version, denoted as *OLD*. We set the different default numbers of samples, k, to 10, 20, and 40. All versions in this experiment use the same precision threshold (shown in Figure 3), $t = 0.001$. We use the same delay library and pattern set (15-detection of transition faults) in Section 4.1.

For each circuit and each speedup method, we compare the runtime and accuracy over 8 different reference lines ($clock - d\_size$). The accuracy is measured by the average difference in the calculated critical probabilities over all patterns using the answer from the OLD version as the true answer. Table 5 shows the runtime for benchmark circuit s5378. The runtime of each speedup method changes with a different reference line. For large or small reference lines, dramatic speedup can be achieved. For example, with $k = 10$, if the reference line is set as 260, the runtime is 639. If the reference line is set as 180, the runtime is 6682.

Table 5: Comparison of runtime(sec). Circuit s5378.

method	reference line (10ps)							
	260	240	220	200	180	160	140	120
OLD	21426	21305	21328	21271	21348	21358	21332	21230
$k = 10$	639	846	943	1870	6682	6437	3771	1457
$k = 20$	1014	1285	1318	2479	7410	7267	4459	2033
$k = 40$	1829	2063	2214	3365	8275	8259	5359	2916

Table 4 shows the average runtime and the average accuracy for each circuit, also based on 8 different reference lines. For example, with $k = 40$ on circuit c880, the average runtime is 84, and the average accuracy is 0.093%, i.e., the average difference of the calculated probabilities for each pattern between the speedup method and the OLD method. Compared with the runtime of OLD, method $k = 40$ can gain about a 5.6X speedup while sacrificing 0.093% in accuracy. A tradeoff between the accuracy and runtime by assigning a different k can be observed in Table 4. The smaller the k is, the faster the runtime will be but the less the accuracy will be.

As the results show, versions of dynamic timing analysis can attain at least 5X speedup on average with a limited loss in accuracy. Therefore, the speeded-up version of dynamic timing analysis can significantly shorten the pattern selection process and still maintain high accuracy. In PSFTD, we assign k to 40.

6. Conclusion

In this paper, we have proposed an effective coverage metric to guide pattern selection for detecting timing defects in the statistical timing domain. Instead of using a timing-defect simulator directly for pattern selection, our approach utilizes a dynamic timing analysis with less complexity than that of a timing-defect simulator. In addition, a speedup method is proposed to further shorten the runtime of dynamic timing analysis. We have demonstrated the effectiveness of the proposed approaches through various experiments based on benchmark circuits.

References

[1] A. Kahng, and Y. Pati, Subwavelength lithography and its potential impact on design and EDA. *ACM/IEEE Design Automation Conference*, pp. 799-804, June 1999

[2] M. Orshansky, L. Milor, P. Chen, K. Keutzer, and C. Hu, Impact of Spatial Intrachip Gate Length Variability on the Performance of High-Spead Digital Circuits. *IEEE Transactions on Computer-Aided Design*, pp. 544-553, May, 2002.

[3] K.-T. Cheng, S. Dey, M. Rodgers, and K. Roy, Test Challenges for Deep Sub-Micron Technologies, *ACM/IEEE Design Automation Conference* 2000.

[4] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic, Fast Statistical Timing Analysisby by Probabilistic Event Propagation, *ACM/IEEE Design Automation Conference*, pp. 661-666, June 2001.

[5] A. Agarwal, D. Blaauw, V. Zolotov, and S. Vrudhula, Statistical Timing Analysis using Bounds, *ACM/IEEE Design, Automation and Test in Europe Conference and Exhibition*, pp. 62-67, March 2003.

[6] M. Orshansky, J. Chen, and C. Hu, A Statistical Performance Simulation Methodology for VLSI Cirucits, *ACM/IEEE Design, Automation and Test in Europe Conference and Exhibition*, pp. 62-67, March 2003.

[7] J.-J. Liou, A. Krstic, Y.-M. Jiang, and K.-T. Cheng, Path Selection and Pattern Generation for Dynamic Timing Analysis considering power supply noise effects, *ACM/IEEE International Conference on Computer Aided Design*, pp. 493-496, Nov 2000.

[8] W.-Y. Chen, S. K. Gupta, and M. A. Breuer, Test Generation for Crosstalk-Induced Delay in Integrated Circuits. *ITC*, pp. 191-200, Oct. 1999.

[9] Y-M. Jiang, A. Krstic, K.-T. Cheng, Estimation for Maximum Instantaneous Current Through Supply Lines for CMOS Circuits. *IEEE Tran. on VLSI*, Vol. 8 No. 1, Feb, 2000. pp. 61-73

[10] Eldo v4.4.x User's Manual. 1996.