VI. Conclusion

This paper investigated the impact of process variation on test quality of bridging faults. It has been shown that process variation influences two parameters (logic VT and gate drive strength) which affects the logic behavior of bridging faults leading to test escape. To quantify the impact of process variation on test quality, a metric called test robustness has been presented. The metric guides a novel PVAA method, which target the logic faults that have the largest impact on test quality. Experimental results show that for all considered benchmarks, the proposed method achieves better results (less test escapes) than tests generated without consideration of process variation.

References

[1] U. Ingelsson, B. M. Al-Hashimi, and P. Harrod, "Variation aware analysis of bridging fault testing," in *Proc. ATS*, Nov. 2008, pp. 206–211.

[2] S. Bhunia, S. Mukhopadhyay, and K. Roy, "Process variations and process-tolerant design," in *Proc. VLSID*, Jan. 2007, pp. 699–704.

[3] V. Iyengar, J. Xiong, S. Venkatesan, V. Zolotov, D. Lackey, P. Habitz, and C. Visweswariah, "Variation-aware performance verification using at-speed structural test and statistical timing," in *Proc. ICCAD*, Nov. 2007, pp. 405–412.

[4] V. R. Devanathan, C. P. Ravikumar, and V. Kamakoti, "Variation-tolerant, power-safe pattern generation," *IEEE Des. Test. Comput.*, vol. 24, no. 4, pp. 374–384, Jul. 2007.

[5] X. Lu, Z. Li, W. Qiu, D. M. H. Walker, and W. Shi, "PARADE: Parametric delay evaluation under process variation," in *Proc. ISQED*, Mar. 2004, pp. 276–280.

[6] X. Lu, Z. Li, W. Qiu, D. M. H. Walker, and W. Shi, "Longest-path selection for delay test under process variation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 12, pp. 1924–1929, Dec. 2005.

[7] G. Chen, S. Reddy, I. Pomeranz, J. Rajski, P. Engelke, and B. Becker, "An unified fault model and test generation procedure for interconnect open and bridges," in *Proc. ETS*, May 2005, pp. 22–27.

[8] M. Favalli and M. Dalpasso, "High quality test vectors for bridging faults in the presences of IC's parameters variations," in *Proc. DFT*, Sep. 2007, pp. 448–456.

[9] S. R. Nassif, "Modeling and analysis of manufacturing variations," in *Proc. CICC*, May 2001, pp. 223–228.

[10] M. Renovell, P. Huc, and Y. Bertrand, "The concept of resistance interval: A new parametric model for realistic resistive bridging fault," in *Proc. VTS*, Apr. 1995, pp. 184–189.

[11] P. Engelke, I. Polian, M. Renovell, and B. Becker, "Simulating resistive bridging and stuck-at faults," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 10, pp. 2181–2192, Oct. 2006.

[12] S. Khursheed, U. Ingelsson, P. Rosinger, B. M. Al-Hashimi, and P. Harrod, "Bridging fault test method with adaptive power management awareness," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 6, pp. 1117–1127, Jun. 2008.

[13] M. Favalli and M. Dalpasso, "Bridging fault modeling and simulation for deep submicron CMOS ICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 8, pp. 941–953, Aug. 2002.

[14] K. J. Kuhn, "Reducing variation in advanced logic technologies: Approaches to process and design for manufacturability of nanoscale CMOS," in *IEDM Tech. Dig.*, Dec. 2007, pp. 471–474.

[15] P. Oldiges, Q. Lin, K. Petrillo, M. Sanchez, M. Ieong, and M. Hargrove, "Modeling line edge roughness effects in sub 100 nanometer gate length devices," in *Proc. SISPAD*, Sep. 2002, pp. 131–134.

[16] A. Asenov, "Random dopant induced threshold voltage lowering and fluctuations in sub-0.1 μm MOSFETs," *IEEE Trans. Electron Devices*, vol. 45, no. 12, pp. 2505–2513, Dec. 1998.

[17] *Predictive Technology Model*, Arizona State Univ., Tempe, AZ, Apr. 2008. [Online]. Available: http://www.eas.asu.edu/~ptm

[18] L. H. A. Leunissen, W. G. Lawrence, and M. Ercken, "Line edge roughness: Experimental results related to a two-parameter model," *Microelectron. Eng.*, vol. 73/74, pp. 265–270, Jun. 2004.

[19] *zChaff Boolean Satisfiability Problem Solver*, Mar. 2007. [Online]. Available: http://www.princeton.edu/~chaff/zchaff.html

[20] *OSU FreePDK*, Oklahoma State Univ., Stillwater, OK, 2008. [Online]. Available: http://avatar.ecen.okstate.edu/projects/scells/OSUFreePDK.php

High-Speed Post-Layout Logic Simulation Using Quasi-Static Clock Event Evaluation

Myeong-Jin Kim, Eui-Young Chung, *Member, IEEE*, and Sungroh Yoon, *Member, IEEE*

***Abstract*—The post-layout gate-level simulation constitutes a critical design step for timing closure. The major drawback of traditional post-layout gate-level simulation is its long analysis time, which becomes exacerbated as design complexity increases. An alternative method is *static timing analysis* (STA), which can drastically reduce analysis time. However, STA sacrifices accuracy for speed and often produces unrealistic results such as false paths and overly pessimistic estimates. In this paper, we propose a hybrid analysis method that can significantly reduce analysis time, while preserving accuracy, with respect to the traditional gate-level simulation. Our key idea is that a large speedup would be possible by removing those events that are repetitious and unnecessary for simulation. In particular, we focus on reducing the number of clock-related events, which account for a major portion of all the events handled by a simulator. We tested the proposed method extensively with various benchmark circuits as well as industrial designs. Our experimental results exhibit that the proposed approach accelerates the total simulation speed by two times on average, yet maintaining the accuracy acquired by the traditional gate-level simulation.**

***Index Terms*—Clock tree analysis, CMOS integrated circuits, dynamic power analysis, gate-level logic simulator, static timing analysis (STA).**

I. Introduction

Since the term *System-on-Chip* (SoC) was first introduced in 1993, more and more functions have been integrated into a single chip thanks to rapid advances of process technology [1]. When designing such complex chips, one of the most time-consuming processes is often design verification. The aim of design verification is to sign off a given design in several aspects, particularly for functionality and timing. The major issue is the appropriate tradeoff between verification completeness and running time. To address this, several techniques have been proposed, which are typically classified into four categories: simulation methods, static methods, formal methods, and physical methods [2].

Simulation methods have been the most popular way for functional and timing verification. However, other methods are gradually taking its role for verification time reduction or completeness. For instance, formal methods are gradually receiving a large attention in the functional verification community. In addition, static methods have already become the *de facto* standard in the design flow of many commercial application-specific integrated circuit vendors for timing verification.

Manuscript received October 6, 2008; revised January 8, 2009. Current version published July 17, 2009. This work was supported in part by "System IC 2010" project of Korea Ministry of Knowledge Economy, by IDEC (IC Design Education Center), by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2009-0068969), and by KOSEF Grants funded by the Korean Government (MEST) (2009-007988 and 2009-0060369). This paper was recommended by Associate Editor C. V. Kashyap.

M.-J. Kim is with the School of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea, and also with the System IC Business Team, LG Electronics, Seoul 135-985, Korea (e-mail: myeongjinkim@yonsei.ac.kr).

E.-Y. Chung is with the School of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea (e-mail: eychung@yonsei.ac.kr).

S. Yoon is with the School of Electrical Engineering, Korea University, Seoul 136-701, Korea (e-mail: sryoon@korea.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2020716

However, these methods also have their own limitations. The limitation of a formal method is that it typically requires resources exponentially proportional to the design size, meaning that only small designs can be verified. Moreover, a static method is often too pessimistic by considering all possible cases which cannot be encountered in reality. The problem of "false paths" is one of the most well-known limitations of static methods for timing verification [3]. In addition, crosstalk effects can be precisely analyzed in a dynamic manner which is the generic feature of simulation methods. For these reasons, simulation is still an important verification method for large-scale SoCs.

Typically, the post-layout simulation of digital circuits means the gate-level simulation with back-annotated timing data in *standard delay format* (SDF) [4] which is calculated from the parasitic information extracted from physical layouts. Thus, a post-layout simulation method just inherits the properties of conventional gate-level simulation methods. Conventional gate-level simulation techniques can largely be classified into two categories: interpreted event-driven method and levelized compiled-code method. In addition, hybrid methods have been studied for many years [5], [6]. It is known that the interpreted method is more advantageous for simulating low activity circuits. In [7], the authors claim that the interpreted event-driven method outperforms the levelized compiled-code method when the circuit activity rate is lower than 50%–60%. Many commercial logic simulators also adopt the interpreted event-driven approach for gate-level simulations, since an interpreted event-driven method is effective for large-scale circuits particularly when the circuit activity is low [8].

In this paper, we propose a speedup technique for post-layout gate-level simulation, while keeping the analysis accuracy of estimates equivalent to that of traditional simulation. The key idea of our method is to use both event-driven and static methods in a hybrid fashion. We focus on reducing the number of clock-related events simulated, since clock signals repeat toggling periodically, and the portion of clock events out of the total events is not marginal. This observation motivates us to devise techniques for accelerating clock-event analysis, which can eventually yield overall simulation performance improvements. This is true for both pre- and post-layout stages, but the amount of savings would be higher for the post-layout stage, since post-layout designs have more buffers in signal lines to meet timing constraints. The proposed method is extensively tested with benchmark and industrial circuits, and the result is presented.

The rest of this paper is organized as follows. Section II presents related work, and in Section III, we explain the details of our method. We present our experimental results in Section IV, followed by a conclusion in Section V.

II. Related Work

The idea of speeding up simulation by exploiting clock networks has not been explored much in the event-driven simulation community. Only a few studies tried to exploit clock networks for parallel simulation [9]. Their major concern was how to reduce the synchronization overhead. That is, their major focus was not on reducing the evaluation time for clock events but on reducing communication overheads among processes. The work most closely related to the proposed technique is the *cycle-based simulation* method in which every event is evaluated at cycle-accurate level. Although this approach showed impressive speedups, it failed to survive in the commercial market, since it could not efficiently incorporate realistic delay models other than the simplistic zero-delay and unit-delay models. On the contrary, *static timing analysis* (STA) has been popularly used in the timing estimation of clock networks, particularly for timing-driven layouts. Its popularity is mainly due to its simplicity, short running time, and reasonable accuracy. The use of STA has further been expanded to consider *process, voltage, temperature, and crosstalk* variations as process technology scales down. To cope with such variations, STA has been in evolution to handle the variation effects in a statistical manner. The history of event-driven simulation and STA indicates that STA is appropriate for timing analysis of predictable (periodic) signals such as clocks in terms of speed, while unpredictable signal can effectively be managed by the event-driven simulation in terms of accuracy. No other techniques have exploited the merits of both techniques, to the best of the authors' knowledge. The method we propose is to exploit the merits of both event-driven method and STA in such a way that the analysis speed is in between the conventional event-driven simulation and STA, while its accuracy is identical to that of the event-driven simulation.

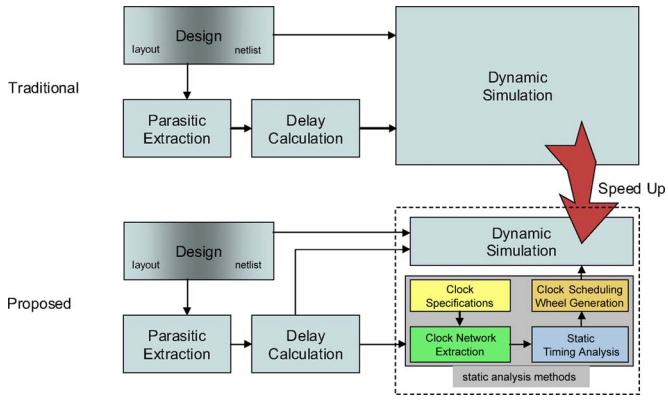


Fig. 1. Comparison of traditional and proposed flows.

III. Proposed Method

Fig. 1 compares the proposed method with the traditional post-layout simulation flow. Our key idea is that by combining the event-driven and static analysis methods, we can achieve a drastic speedup with no impact on accuracy. In particular, we focus on reducing the number of events occurring in clock trees. In a typical event-based simulation, most events are related to clocks since they keep toggling and generating events. It is thus expected that efficient handling of clock-related events would reduce the simulation time substantially.

A. Overview

The proposed method first partitions the input design into two parts, namely, clock network and nonclock network parts. The nonclock network part is then simulated by the conventional event-driven method. In contrast, the clock network part is first analyzed by STA to precalculate clock delays from a clock source to its downstream flip-flops. After the layout process is completed, annotations on cell and interconnect delays are available for each clock tree in the design. By STA, we can thus calculate the clock delay of each flip-flop in a clock tree by propagating rising and falling events from the clock source of the tree to the flip-flop [10]. As demonstrated in Section IV, overheads for the static analysis of clock trees are typically negligible.

During the simulation, we need not generate those events that occur at intermediate gates between the clock source and peripheral flip-flops. We refer to this notion as *clock event shadowing* (Fig. 2). Consequently, the proposed method can run faster than conventional simulators that should simulate events at intermediate gates repeatedly in every cycle. Another key idea is to eliminate unnecessary clock events at the clock pin of each flip-flop by using the fact that a flip-flop is only sensitive to either the positive or negative edge of a given clock. If a flip-flop changes its output at the positive edge of a given clock, we call the positive edge *active* and the negative edge *inactive*,

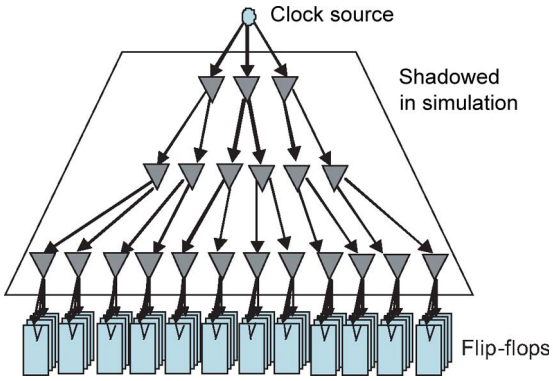


Fig. 2. Concept of the clock event shadowing.

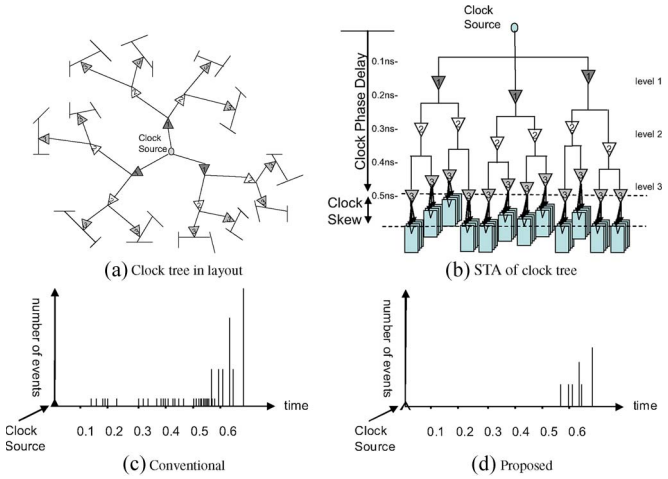


Fig. 3. Reducing the number of clock-related events. (a) Clock tree in layout. (b) Clock tree analysis by STA. (c) Number of events in clock network without event shadowing and inactive edge filtering. (d) Number of events in clock network with event shadowing and inactive edge filtering.

respectively. Hence, we can reduce the number of clock events at the pin of each flip-flop by half by eliminating inactive edge events. We call this technique *inactive-edge filtering*.

Example 1: Fig. 3(a) shows a clock network in the layout of a design. Using the RC information extracted from the layout, we perform STA on this clock network and compute the clock delays of the flip-flops in the clock network, as shown in Fig. 3(b). Fig. 3(c) and (d) compares the numbers of events without and with the proposed method employed, respectively. The events occurring between 0.1 and 0.5 ns are eliminated by clock-event shadowing, and the number of events from 0.5 ns to the end is reduced by inactive-edge filtering. □

B. Proposed Simulator

A traditional event-driven simulator uses a conceptual wheel called *the event wheel*, which rotates and supply events periodically to the simulator [Fig. 4(a)]. In the proposed method, we consider clock-related events separately from the rest of events. Thus, we create another wheel called *the clock scheduling wheel*, as shown in Fig. 4(b). The proposed simulator works based on the interplay between these two wheels. As simulation time proceeds, the two wheels rotate and generate events at the time points specified by either wheel.

1) Clock Scheduling Wheel: This wheel is to trigger clock-related events and contains all possible clock event triggering scenarios. This can eliminate repetitive clock event propagation during simulation. If the design has a clock of period P, so is the circumference of the clock-event scheduling wheel. The use of clock scheduling wheel

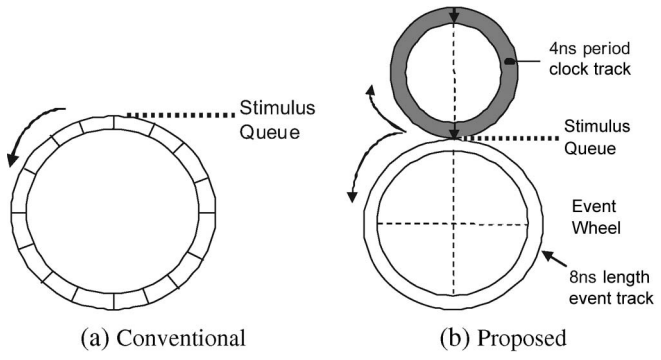
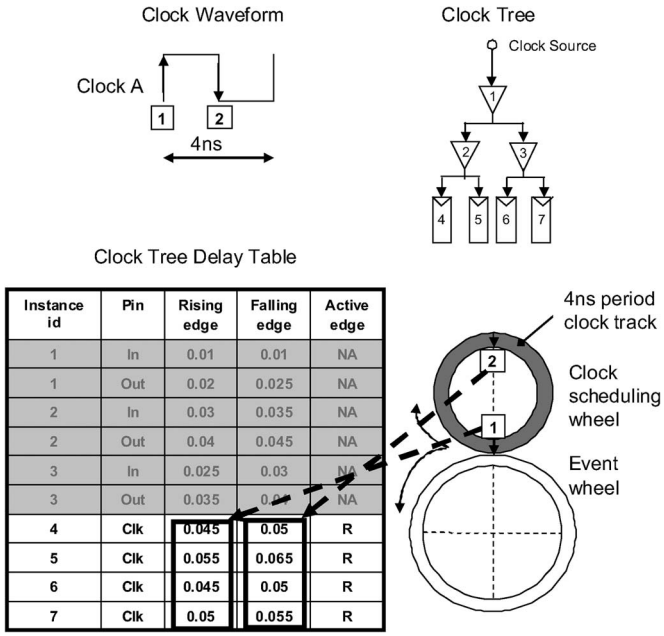


Fig. 4. Comparison. (a) Conventional event wheel. (b) Clock scheduling wheel and event wheel.



Instance id	Pin	Rising edge	Falling edge	Active edge
1	In	0.01	0.01	NA
1	Out	0.02	0.025	NA
2	In	0.03	0.035	NA
2	Out	0.04	0.045	NA
3	In	0.025	0.03	NA
3	Out	0.035	0.04	NA
4	Clk	0.045	0.05	R
5	Clk	0.055	0.065	R
6	Clk	0.045	0.05	R
7	Clk	0.05	0.055	R

Fig. 5. Example: clock event scheduling.

helps reduce the number of events scheduled for simulation. At active edges, the simulator schedules events related to the clock pins in flip-flops and updates the status of the other pins. At inactive edges, the simulator only updates a status table in the simulation database by considering pin phase delays and clock source triggering time without queuing additional events. That is, the events occurring only at active edges are scheduled for simulation.

When a simulation starts, the proposed simulator generates events for flip-flops directly using the clock-event scheduling wheel without going through intermediate gates, resulting in a steep decrease in simulation time. According to input stimuli, the events occurring at combinational gates are propagated with delay effects considered. Whereas, the event occurring at a storage element is propagated only when its clock or enable input is activated. Otherwise, the simulator updates only the status of input signals without propagating events.

Example 2: In Fig. 5, assume a clock signal whose period and duty cycle are 4 ns and 50%, respectively. Suppose that all flip-flops connected to the clock tree are positive-edge sensitive. Then, the clock events at the clock source will be scheduled on the clock scheduling wheel. Each event at the clock source is linked to the delayed events occurring at the downstream flip-flop clock pins. These delayed events are listed in the clock tree delay table, which is constructed by performing STA of the clock tree. Hence, whenever a clock source event

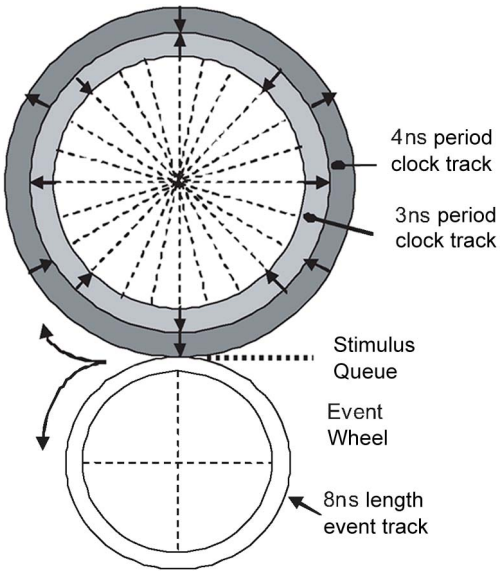


Fig. 6. Multi-clock scheduling wheel and event wheel.

recorded on the clock scheduling wheel is evaluated, the events linked to this clock source event are scheduled and recorded on the event wheel with the specified delays, as long as their polarity corresponds to the active edge. The shaded part of the table indicates the events removed by clock event shadowing. □

2) Event Wheel: The event wheel is to generate and manage non-clock events. The circumference of the event scheduling wheel is twice[1] as long as the longest clock period and is divided into subintervals.[2] Each time point is specified in the event scheduling wheel (this time point serves as a coarse-grained timing guideline), and a table called *pin instance status table* is referred to for retrieving more fine-grained timing information for the rising and falling events associated with pin instances. These events are inserted into the queue for simulation, and the corresponding entry in the pin instance status table is updated with most recent status information.

3) Estimating the Amount of Savings: Let R denote the ratio of the total events simulated by the proposed method to that by the conventional one. We can compute an estimate of R as follows:

$$R = \frac{\sum \text{event} - 4\sum \text{clock buffer} - \sum \text{flipflop}}{\sum \text{event}} \qquad (1)$$

where $(\sum \text{event})$, $(\sum \text{clock buffer})$, and $(\sum \text{flipflop})$ represent the number of total events in the traditional simulator, the number of clock buffers in the clock tree, and the number of flip-flops in the clock tree, respectively. In the numerator, we subtract $(4\sum \text{clock buffer})$ from $(\sum \text{event})$ because four events happen (two at the input and two at the output) for each gate per period (two events at rising edges and two at falling edges). This term originates from the clock event shadowing. The last term in the numerator is due to the inactive-edge filtering, and we can reduce the number of events by the number of flip-flops in the design.

C. Extensions

We can apply the proposed technique to multi-clock designs by adding additional tracks to the clock-event scheduling wheel, one track per clock in the design, and adjusting the parameters of the wheel accordingly.

Example 3: Assume a design has two clocks with periods of 3 and 4 ns. The least common multiple of the periods is 12 ns, and a clock scheduling wheel having the circumference of 12 ns is created, as shown in Fig. 6. The time interval corresponding to the circumference is further divided into equal-sized subintervals. Each interval has a length of 0.5 ns, which corresponds to the greatest common divisor of all falling and rising edge periods of the two clocks used (0 and 1.5 ns for the 2-ns clock; 0 and 2 ns for the 3-ns clock). For simulation, the event wheel is formed and then interacts with the clock scheduling wheel. The circumference of the event wheel is 8 ns (twice as large as the longer clock period). □

The proposed method can also handle clock-gated circuits (for low-power design) by shadowing events in the clock tree rooted at the output pin of each clock gating cell. It is thus possible to apply our

[1]Here, we assume that the phase delay of a clock network can be at most twice the longest clock period. This is a conservative estimate and can further be optimized for additional speedups.

[2]The length of an interval is specified in the SDF file used.

TABLE I
ISCAS 99 BENCHMARK CIRCUITS

Design	Gate Count	Speed (MHz)	Net Count
B19	198,352	80	106,415
B18	95,422	90	49,591
B17	26,573	100	12,199
B22	34,068	100	23,112
B14	11,194	100	7,593
B12	1,517	250	596
B10	225	330	109

TABLE II
COMPARING THE NUMBER OF SIMULATED EVENTS

Design	Traditional (T)	Proposed (P)	P/T (%)
B19	12,958,540	5,902,540	45.6
B18	8,294,167	4,756,167	57.3
B17	4,180,004	2,057,004	49.2
B22	1,358,035	599,035	44.1
B14	649,019	308,019	47.5
B12	394,005	255,005	64.7
B10	90,823	59,823	65.9

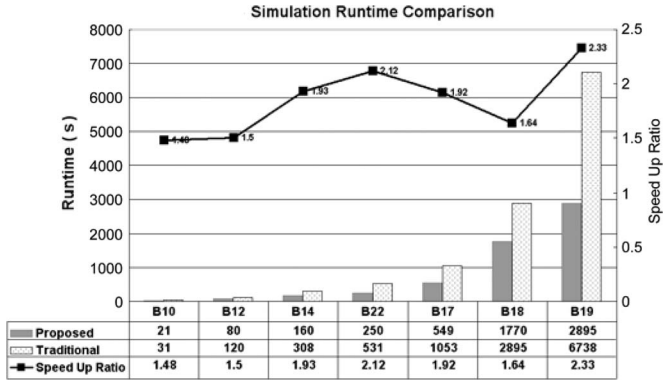


Fig. 7. Runtime comparison.

TABLE III
INFORMATION ON THE MULTICLOCK DESIGN TESTED

Clock ID	Period (ns)	# FFs	# CLK buffers
A	6	1,355	114
B	8	214	17
C	12	232	18
D	14	16	0
E	14	27	0
F	14	96	3
G	14	32	0

TABLE IV
PERFORMANCE COMPARISON FOR MULTICLOCK DESIGN

Statistic	Traditional (T)	Proposed (P)	P/T (%)	Event-shadowed	Inactive-edge-filtered
Runtime (s)	564	278	49.3	-	-
# events simulated	799,374	416,622	52.1	-	-
# active edge events	312,186	288,574	92.4	23,612	0
# inactive edge events	310,653	0	0	23,612	287,041

approach to vectored dynamic power analysis for enhancing the analysis speed. According to our experiments, we could obtain about two times speedup over the traditional simulation. Due to space limit, more details are not presented.

IV. EXPERIMENTAL RESULTS

We tested the proposed simulator with the ISCAS 99 benchmarks implemented using 0.25-μm process technology. The designs used and their characteristics are listed in Table I. The designs were synthesized, placed, routed, and optimized by Synopsys Design-Compiler and Astro. The target design speed was achieved by conducting postrouting optimization. We used Synopsys Star-RCXT [11] to generate *standard parasitic format* (SPF) files and exploited Synopsys PrimeTime [12] for calculating the SDF delay information from the SPF files.

The proposed simulator, simulation models, and data structures were written in the Scheme program language [13], and all the information for experiments were maintained in the Synopsys Milkyway environment [14]. For comparison, we also implemented in Scheme the traditional simulation environment that lacks the clock event shadowing and inactive-edge filtering capabilities proposed in this paper.

To verify the correctness of the proposed method, we first ran it over the test circuits to generate a set of *value change dump* (VCD) files [15]. Using a traditional gate-level simulator, we then produced another set of VCD files. The VCD files generated by the two simulators matched, which confirmed the accuracy of the proposed simulator. This is as expected, since our method does not approximate any events, although clock tree events are handled in a static manner.

We then compared the performance of the traditional post-layout simulation method and that of the proposed technique in terms of the number of total events simulated and running time. We let each simulator run for 1000 cycles with identical input stimuli. Table II compares the number of total events simulated by the two methods. The third column of this table represents the ratio of the number of events simulated by the traditional method to that by the proposed method. This ratio was 44.1%–65.9%, indicating a steep reduction in the event counts. Fig. 7 shows the runtime comparison result, in which the runtime of the proposed method was on average 57.4% of the traditional technique.

For testing with multi-clock designs, we compared the performance of the two simulators using a bus arbitration logic circuitry designed for a commercial HDTV chip designed using 0.13-μm process technology. As listed in Table III, the design used has approximately 100 000 gates and seven clocks. The design has 33 787 pins and 8729 nets. We ran each simulator for the simulation time of 1040 ns, achieving the result summarized in Table IV. The running time and the number of simulated events of the proposed technique were only 49.3% and 52.1% of the traditional method, respectively. Table IV also compares two simulators for the number of events simulated for each clock event type. We could confirm that the performance of the proposed simulator is largely improved by completely eliminating the evaluation of inactive clock edge events thanks to the inactive clock edge filtering. The impact of clock event shadowing technique was less than expected, as the number of active edges simulated shows in Table IV. The effectiveness of clock event shadowing primarily depends on the depth of clock tree, which was only at most five in this design. Using a design with a longer clock buffer chain would result in more reduction.

V. CONCLUSION AND FUTURE WORK

We have proposed a fast simulation method for post-layout gate-level simulation, where clock events take the dominant portion out of all the events to be simulated. The major techniques we used for running time reduction are clock event shadowing and inactive-edge filtering. The experimental results we obtained exhibit that the proposed method runs over two times faster than the traditional method on average. Future work includes extensions to various design components with regular or predictable event activities. If we also exploit approximations, we would be able to increase the portion of design analyzed in a static manner, which will result in more speedups while reasonably sacrificing accuracy.

REFERENCES

[1] K. Mori, H. Yamada, and S. Takizawa, "System on chip age," in *Proc. Tech. Papers Int. Symp. VLSI Technol., Syst., Appl.*, 1993, pp. K15–K20.
[2] P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-Chip Verification Methodology and Techniques*. Norwell, MA: Kluwer, 2001.
[3] C. Rim, S. Kim, J. Park, M. Jang, J. Lee, K. Choi, and J. Kong, "Fast and practical false-path elimination method for large SoC designs," in *SOC Conf.*, 2003, pp. 397–400.
[4] *IEEE Standard for Standard Delay Format (SDF) for the Electronic Design Process*, IEEE Std. 1497-2001, 2001.
[5] Y. Lee and P. Maurer, "Two new techniques for compiled multi-delay simulation," in *Southeastcon*, 1992, pp. 175–179.
[6] K. Taniguchi, H. Fujii, S. Kajihara, and X. Wen, "Hybrid fault simulation with compiled and event-driven methods," in *Design Test Integr. Syst. Nanoscale Technol.*, 2006, pp. 240–243.
[7] P. Maurer, "The inversion algorithm for digital simulation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 7, pp. 762–769, Jul. 1997.
[8] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing: Digital, Analog, and Mixed-Signal VLSI Circuits*. Boston, MA: Kluwer, 2000.
[9] K. Chang, W. Tu, H. Wang, Y. Yeh, and S. Kuo, "Techniques to reduce synchronization in distributed parallel logic simulation," in *Parallel Distrib. Comput. Syst.*, 2004, pp. 1–5.
[10] V. Wason, R. Murgai, and W. Walker, "An efficient uncertainty- and skew-aware methodology for clock tree synthesis and analysis," in *VLSI Design*, 2007, pp. 271–277.
[11] Star-RCXT. [Online]. Available: http://www.synopsys.com/Tools/Implementation/SignOff/Pages/Star-RCXT.aspx
[12] PrimeTime. [Online]. Available: http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx
[13] *IEEE Standard for the Scheme Programming Language*, IEEE Std. 1178-1990, 1990.
[14] Milkyway. [Online]. Available: http://www.synopsys.com/solutions/endsolutions/galaxyimplementation/pages/milkyway.aspx
[15] *IEEE Standard Verilog Hardware Description Language*, IEEE Std. 1364-2001, 2001.