PAPER Special Section on Solid-State Circuit Design—Architecture, Circuit, Device and Design Methodology

Parallel Acceleration Scheme for Monte Carlo Based SSTA Using Generalized STA Processing Element

Hiroshi YUASA[†a)], ***Nonmember***, **Hiroshi TSUTSUI**[†], **Hiroyuki OCHI**[†], ***and*** **Takashi SATO**[†], ***Members***

SUMMARY We propose a novel acceleration scheme for Monte Carlo based statistical static timing analysis (MC-SSTA). MC-SSTA, which repeatedly executes ordinary STA using a set of randomly generated gate delay samples, is widely accepted as an accuracy reference. A large number of random samples, however, should be processed to obtain accurate delay distributions, and software implementation of MC-SSTA, therefore, takes an impractically long processing time. In our approach, a generalized hardware module, the STA processing element (STA-PE), is used for the delay evaluation of a logic gate, and netlist-specific information is delivered in the form of instructions from an SRAM. Multiple STA-PEs can be implemented for parallel processing, while a larger netlist can be handled if only a larger SRAM area is available. The proposed scheme is successfully implemented on Altera's Arria II GX EP2AGX125EF35C4 device in which 26 STA-PEs and a 624-port Mersenne Twister-based random number generator run in parallel at a 116 MHz clock rate. A speedup of far more than ×10 is achieved compared to conventional methods including GPU implementation.

key words: *statistical static timing analysis, delay distribution, slew rate, field-programmable gate array, Mersenne Twister*

1. Introduction

Timing analysis in modern VLSI design has never been as important as it is today. As the clock period becomes shorter, accuracy of the timing analysis becomes critically important. The timing analysis that takes process variability into account is also a pressing requirement. Static timing analysis (STA) is widely used since its computation time is linear to the number of gates in the design. The major drawback is its pessimism due to worst-case analysis employed to consider process variations [1].

To remedy this pessimism, statistical STA (SSTA) is becoming an extremely important tool. Circuit optimization and yield analysis, which consider device parameter variability, depend crucially on the timing predictions obtained from SSTA [2]–[4] since it is considered to be *statistically more correct* than the conventional STA. There are two major algorithms for SSTAs, block based [3], [5]–[7] and path based [8]–[10], both of which basically assume normal delay distributions. However, the normality assumption may not be applicable, especially for the shortest path analysis, in which the law of large numbers is inapplicable. Strongly nonlinear minimum operations may also significantly distort the timing distribution. Thus, more flexible timing model and framework that can handle non-normal timing distributions are desired.

One promising approach is to apply a Monte Carlo (MC) method for the timing analysis [11], [12]. In this approach, a conventional STA is repeatedly executed with randomly generated logic gate delays to calculate the timing distributions. Since MC methods use the delay samples directly, arbitrary timing distributions can be handled.

A drawback of the MC based SSTA (MC-SSTA) is its computational intensiveness, since large number of runs are required to obtain statistically meaningful results. However, since each MC run can be executed independently, a dramatic acceleration is possible using a pipeline processing. Motivated by this observation, we proposed a pipeline scheme with an STA engine called the *delay-sample generator and LAT calculator* (DGLC) [13]. In this scheme, the target netlist is first translated into a register-transfer level (RTL) description of dedicated pipelined STA engine. This RTL is then mapped into a field-programmable gate array (FPGA), and MC-SSTA is executed on the FPGA.

This scheme successfully achieves a high throughput. However, it requires long time to map the RTL description into a target FPGA device. Other FPGA-based acceleration methods [14] may also suffer from this problem. The amount of hardware resources also imposes a limitation on the applicability of MC-SSTA, since the required hardware resources is proportional to the size of the input netlist.

In this paper, a novel scheme for accelerating MC-SSTA is proposed. In this approach, we utilize a generalized STA engine called an *STA processing element* (STA-PE), which can calculate the latest or earliest arrival time of one logic gate within a clock period. Unlike the previous methods, the STA-PE is circuit-topology independent, and thus it eliminates the time-consuming FPGA remapping. Once we implement the proposed architecture on an FPGA, only the SRAM data that stores circuit-topology and delay distribution of each gate needs to be replaced when a new target netlist is given. In addition, the topology independence also eliminates the circuit size limitation.

This paper is organized as follows. In Sect. 2, we briefly review related works. We then present the proposed acceleration scheme and architectures for two types of MC-SSTA in Sect. 3. In Sect. 4, the proposed approach is evaluated in terms of throughput, hardware resources, and total processing time. Finally, in Sect. 5, we conclude this paper.

Manuscript received August 3, 2012.
Manuscript revised November 3, 2012.

[†]The authors are with the Department of Communications and Computer Engineering, the Graduate School of Informatics, Kyoto University, Kyoto-shi, 606-8501 Japan.
a) E-mail: paper@easter.kuee.kyoto-u.ac.jp

DOI: 10.1587/transele.E96.C.473

2. Preliminaries

In this section, we briefly review some existing timing analysis procedures and introduce some keywords used in this paper.

2.1 Static Timing Analysis (STA)

Static timing analysis (STA) is one of the most widely used timing analysis methods for designing synchronous digital circuits. In STA, critical paths in a given chip design are identified by propagating the latest arrival time (LAT) over all logic gates in the circuit. One of the biggest advantages of STA is its speed; computational complexity is $O(N)$ when the number of gates in the design is N since the STA process consists of simple graph tracing. Conservative timing is calculated without any input vectors.

Conservatism, however, is one major drawback of STA. With modern process technologies, the delay variation of each logic gate due to process, voltage, and temperature variations has to be properly considered. A worst-case analysis has traditionally been employed to cope with this problem [1], but the timing report from the worst-case analysis tends to be extremely pessimistic since it assumes full correlations between the logic gate delays.

2.2 Monte Carlo Based SSTA (MC-SSTA)

MC-SSTA is realized by repetitively running STA on a *target circuit netlist* with randomly generated delay samples. A target circuit is a circuit whose timing is of interest. The MC-SSTA procedure can be summarized as follows.

1) Generate a *delay sample* and its related variables, such as slew rate, for a *delay arc* in each logic gate. We call a set of delay samples for all delay arcs an *MC sample* or just a *sample*.
2) Run an STA using the MC sample.
3) Store the latest or earliest arrival times of the endpoints and go back to 1). In this paper, we call the latest or earliest arrival time simply the *arrival time* (AT).

For each STA in step 2), the AT of each node is propagated down the circuit graph. The output of a single STA run is the ATs of all endpoints for the MC sample. The aim of the MC-SSTA is to obtain AT distributions at the endpoints by repeatedly running STA.

Since each MC run in MC-SSTA can be executed independently, it is possible to accelerate the algorithm dramatically using pipelined hardware engine. In [13], a pipeline scheme with an STA engine called the *delay-sample generator and LAT calculator* (DGLC) has been proposed. Its analysis flow is shown in Fig. 1. The target netlist (e.g., Fig. 3) is first translated into a synthesizable RTL description (e.g., Fig. 4). This RTL is then mapped into an FPGA, and MC-SSTA is executed on the FPGA.

This scheme successfully achieves a high throughput of 757,000 samples/s for any input netlist when the maximum operating frequency is 100 MHz. However, it requires long time to map the RTL description into a target FPGA device (the shaded portion of Fig. 1). For example, in the case of a 6-bit multiplier, the mapping requires 27 minutes, although runtime on the FPGA after mapping requires only 1.32 seconds for one million samples.

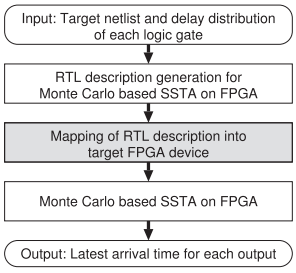

Fig. 1 MC-SSTA flow of the prior work [13].

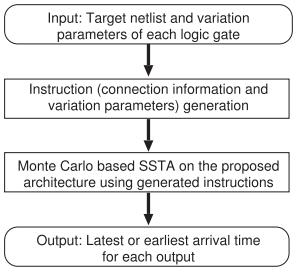

Fig. 2 MC-SSTA flow proposed in this paper.

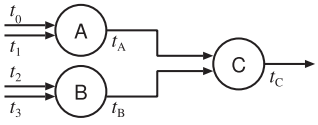

Fig. 3 Example of a target circuit netlist.

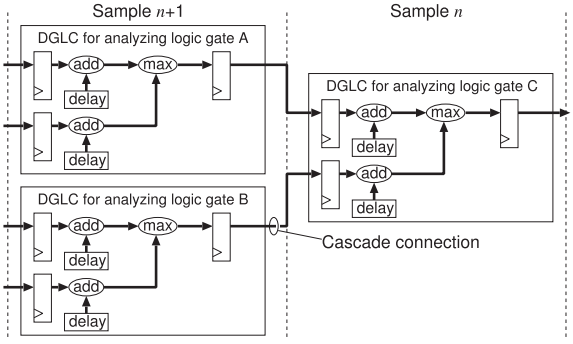

Fig. 4 Transformation of the target netlist into MC-SSTA implementation on an FPGA in the prior work [13].

3. Proposed Acceleration Scheme

In this section, we propose a versatile acceleration scheme to realize MC-SSTA. Unlike the previous methods, the proposed scheme uses a netlist-independent hardware accel-

erator, which eliminates the time-consuming FPGA mapping process. Instead, the proposed hardware accelerator refers to SRAMs in which netlist-dependent information is preloaded.

3.1 Overview

Figure 2 shows the overall flow of the proposed MC-SSTA. The inputs include the target circuit netlist to be analyzed and the variation parameters of each logic gate. For example, when delay samples are represented by normal distributions, the variation parameters include the mean and standard deviation of the delay distribution of each target logic gate. From the given input data, instructions are generated for the proposed MC-SSTA architecture, and the proposed architecture executes the MC-SSTA using the instruction.

3.1.1 Instruction Generation

Assuming that the given gate-level netlist of a combinational circuit (or combinational portion of a sequential circuit) is acyclic, an instruction sequence can be generated using following steps.

1. Replacement of large fanin gates into two-input gates:
   Since the proposed STA-PE (explained later) handles only one- or two-input gates at a time, each logic gate in the given netlist that has more than two inputs should be replaced by two-input gates. For example, a three-input gate is replaced by two-input gates connected in series as shown in Fig. 5. In this case, the delay of an arc in the second two-input gate is set to zero, while the delays of the other three arcs (d_1, d_2, d_3) are associated with the three arcs of the original gate.
2. Deriving ordering of gate evaluation:
   We then sort gates topologically to determine the evaluation order of gate delay so that all arrival time (AT) at the gate inputs has been evaluated. For this purpose, several algorithms such as level sorting [15] and data flow sorting [16] are applicable. Figure 6 illustrates the level sorting algorithm. At first, all gates in the netlist are levelized so that fanins of all gates in a level depends only on the lower levels or primary inputs. After levelization, gates are numbered from lower levels. In the case of Fig. 6, A, B and C in the level 1 comes first, followed by D and E in the level 2, followed by F and G in the level 3.
3. Generation of instructions:
   Before generating instructions, address of scratch pad SRAM (explained later) should be allocated. For example, if there are k primary inputs and m gates, each of the first k words of the scratch pad SRAM are allocated to store the AT of every primary input, and each of other m words are allocated to store the AT of the output of every gate. Then, every gate in the netlist is mapped to an instruction, and the instructions are sorted according to the evaluation order explained above. As illustrated in Fig. 7, each instruction consists of (a) two read addresses (Src0 Addr. and Src1 Addr.) for the scratch pad SRAM(s), which correspond to the two inputs of the gate, (b) two sets of delay variation parameters, each of which corresponds to the variation parameters for each input arc, and (c) one SRAM write address (Dst Addr.) for the scratch pad SRAM(s), which corresponds to the AT of the output of the gate. Note that the number of delay variation parameters may differ depending on the delay model.

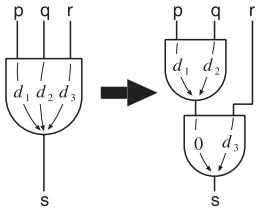

Fig. 5 Replacement of a 3-input gate into two 2-input gates.

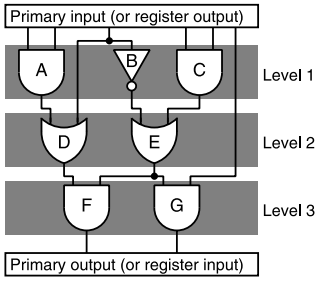

Fig. 6 Ordering of gate evaluation based on level sorting.

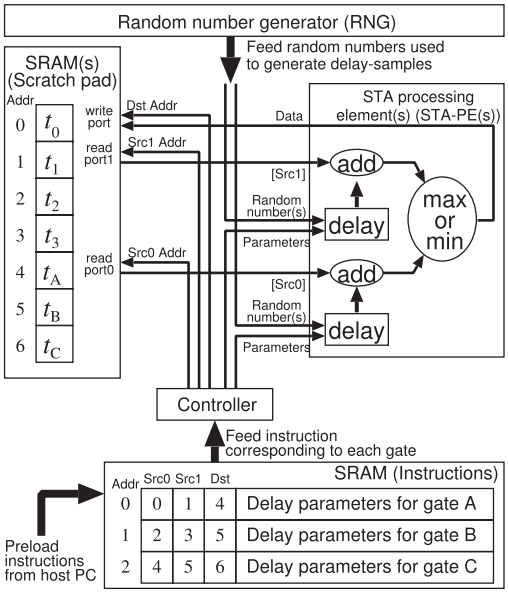

Fig. 7 Overview of the proposed architecture for MC-SSTA using generalized STA engine (STA-PE).

3.1.2 Proposed Hardware Architecture

After the instructions are generated and preloaded to the instruction SRAM, the proposed MC-SSTA architecture executes the MC-SSTA. The basic idea is illustrated in Fig. 7. It consists of a random number generator (RNG), STA-PE(s), scratch pad SRAM(s) to store AT for every wire, an SRAM to store instructions, and a controller.

The STA-PE executes the fundamental STA operations for a two-input logic gate, delay generation, accumulation, and comparison. The AT at the output of the two-input logic gate is calculated by comparing the ATs of the two inputs, each of which is added by a random delay sample of the gate. For each of two inputs, the arc delay between the input to the output is generated by the RNG as random sample(s) that follows delay distribution parameters specified in the instruction SRAM. For example, if we assume normal distribution with mean μ and standard deviation σ, the RNG draw a random sample x from standard normal distribution, and the DELAY module in Fig. 7 generates a delay sample as $d = \sigma x + \mu$. By adding the delay sample d to the AT of the input, the AT of the output of the gate through the input is obtained. Then the two ATs are compared and either maximum or minimum of the two is selected depending on the required analysis, i.e., maximum and minimum operations for the critical path analysis and the shortest path analysis, respectively.

Figure 7 also illustrates how the STA is executed for the sample target circuit shown in Fig. 3. Before starting the analysis using the proposed architecture, instructions that represent the given circuit are preloaded to the instruction SRAM. The region of scratch pad SRAM(s) for the primary inputs are initialized to a fixed value, such as 0[†]. In Fig. 7, t_0, t_1, t_2, and t_3 are preloaded to the addresses 0 to 3 of the scratch pad SRAM(s). When the controller is triggered to start the analysis, it fetches the instruction one by one from the instruction SRAM. According to the instruction at address 0, two ATs t_0 and t_1, determined by the two source addresses in the scratch pad SRAM, are provided to the STA-PE. Then the output AT of gate A, t_A, is stored to the address 4 of the scratch pad SRAM. Similarly, ATs of the output of the gate B and C are computed. Above analysis is repeated predefined times (e.g., 10^6 times) to obtain a statistical delay distribution.

The instruction SRAM has one write port for transferring instructions and one read port for executing MC-SSTA, which enables instruction load into the instruction SRAM can be conducted simultaneously with executing delay calculations. The time required for transferring instructions can be concealed. Further, this feature also enables us to analyze larger circuits than to fit in the instruction memory at once.

The STA-PE is implemented to analyze one logic gate every clock cycle. To maintain throughput, we use multiple STA-PEs in parallel along with the RNG that can output multiple random numbers required by the STA-PEs.

3.1.3 Comparison with Conventional Methods

Compared with the previous architecture [13] that requires hardware area proportional to the number of gates in the circuit, the hardware area needed in the proposed architecture is compact and constant regardless of the size of the input netlist. The hardware area of the proposed architecture is determined only by the number of STA-PEs run in parallel. It is also noteworthy that the proposed architecture can be implemented on non-configurable platforms, such as on ASIC, since the proposed hardware engine is netlist independent, while the previous architectures [13], [14] require configurable platform due to their netlist-dependence.

Regarding the performance, the previous architecture [13] requires N_{NDRNG} cycles to generate one normally distributed random number. This means that other parts of the architecture, excluding RNG, operate once every $N_{\mathrm{NDRNG}} = 132$ cycles. Thus, the previous architecture is inefficient in terms of temporal hardware utilization. In contrast, the RNG and all the STA-PEs of the proposed architecture operate every cycle. In terms of effective hardware utilization, the proposed architecture becomes more efficient than the previous one, and the throughput of the proposed scheme is given by

$$F_{\mathrm{MAX}} \times N_{\mathrm{PE}} / N_{\mathrm{Instruction}} \ [\mathrm{sample/s}], \quad (1)$$

where F_{MAX}, N_{PE}, and $N_{\mathrm{Instruction}}$ are the maximum operating frequency, number of STA-PEs in the proposed architecture, and number of instructions for the target netlist, respectively. Since analysis of n-input gate (if $n > 2$) requires $(n-1)$ instructions as illustrated in Fig. 5, $N_{\mathrm{Instruction}}$ is given by

$$N_{1-\text{input gates}} + N_{2-\text{input gates}} + \Sigma_{n\geq 3}(n-1)N_{n-\text{input gates}}, \quad (2)$$

where $N_{n-\text{input gates}}$ is the number of n-input gates in the netlist. Note also that scratch pad SRAMs, in order to realize this throughput, should be three-port SRAMs that have two read ports and one write port as depicted in Fig. 7.

In the following subsections, two implementation examples of the proposed architecture are presented. The first one considers statistical gate delay variations only, and the second one additionally considers statistical variations of the output capacitance and input slew rates for more practical timing analysis.

3.2 Timing Analysis for Gate Delay

In the existing studies [13], [14], [17], it is assumed that the delay samples are represented by normal distributions. To compare the proposed scheme with these studies, we use normally distributed delay variations in applying MC-SSTA.

An example realization of our scheme is shown in

[†]If distributions of ATs of the primary inputs also need to be considered, we can put a dummy buffer with desired delay distribution to each primary input.

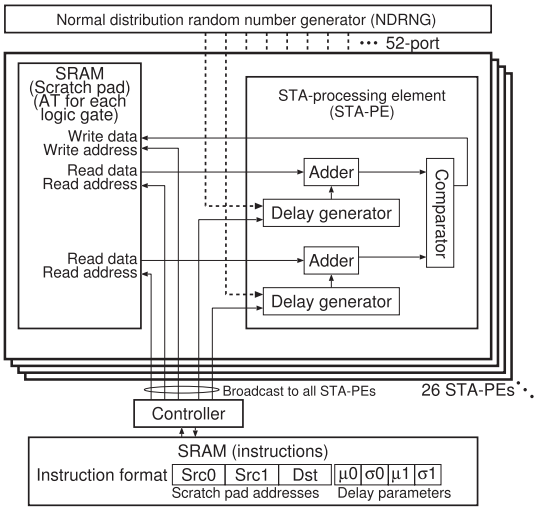

Fig. 8 Block diagram of the proposed architecture for MC-SSTA considering only delay variation.

Fig. 8 in which the normal distribution random number generator (NDRNG) has 52 output ports, and the number of STA-PEs is 26. In this figure, the instruction format is also depicted. The flow of MC-SSTA execution is same as described or exemplified in the previous section.

3.2.1 STA-PE

The STA-PE consists of (1) two delay generators, (2) two adders, and (3) one comparator. The delay generators are used to generate normally distributed random delay values.

Let μ and σ be the mean and standard deviation of the delay distribution of a target logic gate; a random delay sample d_{sample} is generated from $X_{\text{norm}} \sim \mathcal{N}(6, 1)$, an output of our NDRNG, as follows,

$$d_{\text{sample}} = (X_{\text{norm}} - 6) \times \sigma + \mu. \tag{3}$$

By using the adders, ATs propagated from the preceding logic gates and the delay samples are added to calculate the ATs. The latest or earliest AT is selected by the comparator and then stored on the SRAM for the ATs.

3.2.2 NDRNG

The NDRNG should be very efficient in terms of hardware cost and throughput since it limits the generation speed of the delay samples in MC-SSTA. In order to generate normal distribution random numbers, the central limit theorem (CLT) is utilized because it includes only simple arithmetic operations. Normally distributed random numbers from uniform random numbers $X_i \in (0, 1]$ are generated by

$$\mathcal{N}(\mu, \sigma) \sim \frac{2\sqrt{3}}{\sqrt{N}} \left(\sum_{i=1}^{N} X_i - \frac{N}{2} \right) \times \sigma + \mu, \tag{4}$$

where μ, σ, and N are the mean, standard deviation, and an integer constant, respectively.

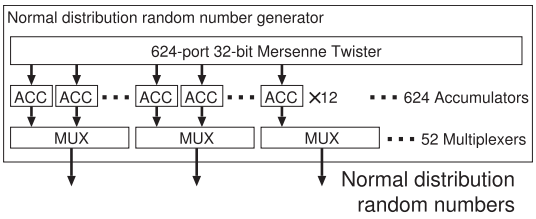

Fig. 9 MT-based normal distribution random number generator for MC-SSTA considering only delay variation.

A uniform RNG requires N_{RNG} cycles to generate a uniform random number, and the NDRNG requires N_{NDRNG} ($= N \times N_{\text{RNG}}$) cycles to generate a normal distribution random number. Following the result in [13], $N = 12$ is selected to balance efficiency, accuracy, and implementation simplicity. In this case, $\mathcal{N}(6, 1)$ can be generated by simply accumulating 12 uniform random numbers as follows,

$$\mathcal{N}(6, 1) \sim \sum_{i=1}^{12} X_i. \tag{5}$$

Therefore, we use $\mathcal{N}(6, 1)$ as the outputs of our NDRNG.

In the proposed architecture, we utilize the multi-port Mersenne Twister (MT) implementation [18] as the uniform RNG since it is highly efficient in terms of both hardware cost and throughput. We utilize a 624-port MT since this is the most efficient number of output ports. This MT can output 624 32-bit uniform random numbers every cycle.

The RNG for the proposed MC-SSTA is shown in Fig. 9: 624 accumulators for each output of the 624-port MT are divided into 52 groups, each of which has 12 accumulators. By interleaving the 12 accumulators through multiplexers (MUXs), our NDRNG generates 52 normal distribution random numbers every cycle.

3.3 Timing Analysis Considering Statistical Load Variations

The proposed MC-SSTA scheme can be extended to realize statistical delay changes due to the statistical load capacitance and input slew rate variations. In a similar manner to the previous realization example, the ATs themselves can be computed while propagating slew rates as well as ATs. As an example, let us consider the following basic sensitivity-based model:

$$d_{\text{o}} = a + bC_{\text{L}} + cS_{\text{i}}, \tag{6}$$

$$S_{\text{o}} = x + yC_{\text{L}} + zS_{\text{i}}, \tag{7}$$

where d_{o}, S_{o}, S_{i}, and C_{L} are the delay, output slew rate, input slew rate, and output capacitance of the target logic gate, respectively. The sensitivity parameters a, b, c, x, y, and z are specific to the target logic gate. Assuming that a, x, and C_{L} fluctuated because of process variations that follow normal distributions, d_{o} and S_{o} can be computed using a variation of the STA-PE described in Sect. 3.2 and shown in Fig. 10. In this figure, the instruction format and data format

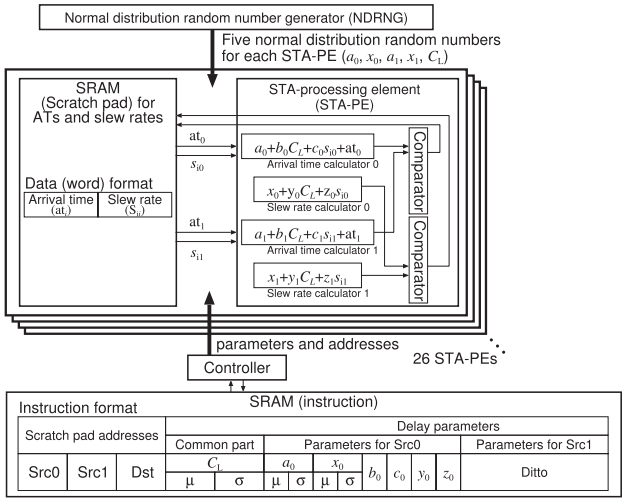

Fig. 10 Block diagram of the proposed architecture for MC-SSTA considering output capacitance and input slew rates.

are also depicted. Each instruction includes a set of delay parameters to calculate d_o and S_o for each arc. Note that the AT and slew rate for each logic gate can be packed into one word.

In this case, the NDRNG has 130 output ports, but 26 STA-PEs are still used. Each scratch pad SRAM stores the temporal ATs and slew rates to be propagated.

3.3.1 STA-PE

The modified STA-PE now consists of (1) two arrival time calculators, (2) two slew rate calculators, and (3) two comparators.

The STA-PE generates samples of a_0, a_1, x_0, x_1, and C_L following their respective distributions to analyze two logic arcs every cycle. These samples are generated in the same way as Eq. (3) by using five normal distribution random numbers following $\mathcal{N}(6, 1)$.

The output ATs and slew rates are calculated using those of the preceding logic gates and the five generated samples. The latest or earliest AT and the slew rates are selected by the two comparators and stored on the scratch pad SRAM.

3.3.2 NDRNG

In this application, we again utilize the 624-port MT. The NDRNG for this application is shown in Fig. 11 and generates 5×26 normal distribution random numbers every cycle to fully utilize the 26 STA-PEs, each of which requires five normal distribution random numbers every cycle.

To generate 5×26 normal distribution random numbers every cycle using the CLT ($N = 12$) of Eq. (5), $5 \times 26 \times 12 = 1{,}560$ uniform random numbers must be generated every cycle. Therefore, in this NDRNG, the 624-port 32-bit MT is used as a 1,560-port 11-bit MT; every two output ports of the 624-port 32-bit MT are bundled to construct a

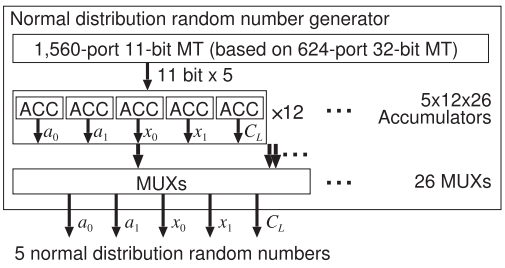

Fig. 11 MT-based normal distribution random number generator for MC-SSTA considering output capacitance and input slew rates.

312-port 64-bit MT, and each 64-bit output port is divided into five output ports. Theoretically, this MT should be a $624/2 \times 5 = 1{,}560$-port $\lfloor 64/5 \rfloor = 12$-bit MT. However, to successfully map the proposed architecture into the target FPGA device, we use this MT as a 1,560-port 11-bit MT. As shown in Fig. 11, we implement 1,560 accumulators for each output of the MT and divide them into 26 groups each of which has 5×12 accumulators. By interleaving the 12 accumulators through MUXs, our NDRNG generates 5×26 normal distribution random numbers every cycle.

3.4 Extension of the Proposed Scheme

The proposed scheme can be extended further to analyze the clock path delay by considering delay variations of the clock buffers in a similar manner to the data paths with which we can consider clock skew. Subtracting the AT of each data path from the result of the clock-delay analysis, slack distributions are also obtained. Furthermore, the extension of the NDRNG to support arbitrary delay distributions by utilizing an arbitrarily distributed RNG, such as the existing study [19], will also be possible.

4. Experimental Results and Comparisons

In Sect. 4.1, we show the experimental results of the proposed scheme, and in Sect. 4.2, we compare the proposed scheme with existing works.

4.1 Experimental Results

4.1.1 Implementation

In the following experiments, we use a mid-range FPGA, Altera's Arria II GX EP2AGX125EF35C4 for implementing the architectures presented in Sect. 3.2 and Sect. 3.3. The number of STA-PEs (N_{PE}) for parallel MC operations, resource usage of the target FPGA device, and maximum operating frequencies (F_{MAX}) are shown in Table 1.

Although the latter architecture executes more complicated operations than the former one (and indeed the latter needs more FPGA resources), the latter achieved higher F_{MAX}. This is due to the following reasons:

- In the latter architecture, the number of bits to represent

Table 1 Implementation of the proposed architectures on Altera's Arria II GX EP2AGX125EF35C4.

Statistical parameters	Number of STA-PEs (N_{PE})	Resource utilization	Maximum Frequency (F_{MAX})
Gate delay only (Sect. 3.2)	26	78%	116 MHz
Load capacitance and slew rates as well (Sect. 3.3)	26	97%	126 MHz

Table 2 Throughput of the architecture presented in Sect. 3.2.

Target circuit	N_{Gates}	$N_{Instructions}$	Throughput [$\times 10^3$ samples/s]
C432	145	213	141,596
C499	207	231	130,563
C880	228	327	92,232
C1355	207	231	130,563
C1908	239	289	104,360
C2670	396	594	50,774
C3540	599	1,001	30,130
C5315	794	1,330	22,677
C6288	1,897	1,957	15,411
C7552	997	1,366	22,079

Table 3 Required time breakdown of the flows of the prior work and the proposed scheme (6-bit multiplier).

Process	Prior work [13]	Proposed
Netlist file read	1.28 ms	
Delay parameter file read	8.45 ms	
RTL description generation	92.2 ms	-
Mapping of RTL description into target FPGA device	27 min	-
Generation of instructions for STA-PEs	-	60.4 ms
MC-SSTA with one million samples on FPGA	1,320 ms	78.9 ms
Total	27 min	159 ms

the ATs is smaller in order to consider both the ATs and input slew rates under a hardware resource limitation.

- STA-PEs in the latter architecture are implemented with more pipeline stages than those of the former architecture since the analysis operation is more complicated.

Note that there is a resource margin of about 20% for the former architecture on the target FPGA device. This means that a higher throughput can be expected by utilizing this resource margin — if more number of STA-PEs with a higher throughput NDRNG were implemented.

4.1.2 Performance

Table 2 shows the throughput of the architecture presented in Sect. 3.2 for ISCAS'85 benchmark circuits. We used mapped netlists for these benchmark circuits that are taken from [20].

Table 3 shows the overall processing time and its breakdown of our scheme (Fig. 2) using the architecture presented in Sect. 3.2. For comparison, the result of the prior work [13] (Fig. 1) is also shown in this table. As can be seen from this table, the proposed flow eliminates the mapping process, which requires 27 minutes in the prior work. The proposed flow instead requires an instruction generation process, which requires only 70.0 ms including file I/O. In addition, a process for transferring the generated instructions into the instruction SRAM is required for the proposed flow. However, as mentioned in Sect. 3, since this process and MC-SSTA on FPGA can be performed concurrently, the time required for this process can be concealed. Therefore, the proposed scheme can achieve significant acceleration in the entire MC-SSTA flow.

Table 4 Comparison of the preprocessing time.

Approach	Preprocessing	Time
Proposed (fixed hardware engine)	Netlist scan	Very fast (≈10 ms)
Compiled software (CPU or GPU [17])	Netlist scan	Very fast (≈10 ms)
Dedicated hardware engine on FPGA [13], [14]	FPGA mapping	Very slow (≈10min)

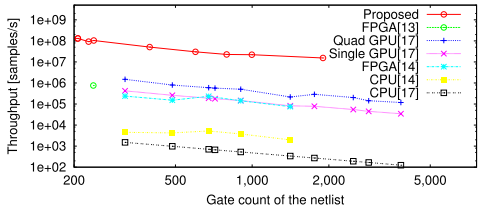

Fig. 12 Comparison of performance.

4.2 Comparisons with Existing Works

4.2.1 Overall Processing Time

Although our goal is to derive the analysis result as fast as possible from the given circuit, most of the previous works only presents the throughput of the core part of the analysis, and little descriptions can be seen on the preprocessing phase. Necessary preprocessing and qualitative time required can be summarized as Table 4. Those methods which use dedicated hardware engine mapped on an FPGA device should suffer from long mapping time. On the other hand, the proposed method and possibly GPU implementation require very short time for preprocessing.

We then focus on the throughput of the core part of analysis in more detail.

4.2.2 Throughput

Figure 12 shows the comparison of performance. FPGA [13] is the result on 6-bit multiplier circuit, and all the other results are on the ISCAS'85 benchmark circuits. The horizontal axis is the number of gates of the given netlist in log scale. Proposed method uses the mapped netlist for ISCAS'85 benchmark circuits that are taken from [20], while

GPU [17] and CPU [17] use mapped netlist using SIS[21], and thus the gate counts are not equal. FPGA [14] and CPU [14] also use mapped netlist using SIS, but the gate counts are not clarified in [14], hence we assume that the gate count of [17] and [14] are equal. The vertical axis is the throughput in samples/s in log scale. As seen from the figure, throughput of the analysis by each method is approximately inversely proportional to the gate count. We can also observe that the proposed method is far more than $\times 10$ faster than the conventional methods. This can be explained by the throughput of the random number generator. While our implementation generates $624 \times 116\,\text{MHz} = 72 \times 10^9$/s uniform random numbers utilizing the multi-port MT implementation [18], GPU implementation [17], for example, generates only 2.23×10^9/s uniform random numbers.

4.2.3 Accuracy

When software algorithms are implemented on hardware, there can be accuracy deterioration due to (1) implementation of floating-point numbers using fixed-point representation, and (2) finite number of digits of the fixed-point representation. According to [13], MC-SSTA hardware implementation using linear feedback shift register (LFSR) based RNG, CLT based NDRNG, and fixed-point representation achieves only 0.005% error in yield analysis compared with an MC-SSTA software implementation using Mersenne Twister based RNG, Box Muller (BM) based NDRNG, and floating-point representation, when the digits of uniform random numbers are at least 11 bits. Hence, the hardware engine proposed in Sect. 3.2 and Sect. 3.3 uses Mersenne Twister instead of LFSR, and the digits of uniform random numbers are given 11 bits, the analysis error of the proposed method is guessed to be sufficiently small, e.g., 0.005%.

5. Conclusions

In this paper, we have proposed an acceleration scheme for MC-SSTA by utilizing a generalized STA engine called an STA-PE, which efficiently calculates the latest or earliest ATs of one logic gate for one sample. In order to execute MC-SSTA, a target netlist is converted into instructions to execute the timing calculation using the STA-PEs. Unlike other hardware implementations on FPGAs, the time-consuming mapping process is not required in the proposed scheme. Once we implement the proposed architecture on FPGAs, only the contents of the instruction SRAM is replaced when the target netlist changes. In the case of a 6-bit MUL, generating instructions requires only 70.0 ms, which is considerably faster than the 27 minutes of the existing scheme. The proposed architecture was successfully implemented on a mid-range FPGA device with a 116 MHz clock in which 26 STA-PEs and a 52-port NDRNG utilizing a 624-port MT-based RNG run in parallel. The scheme achieves far more than $\times 10$ speed-up compared with conventional methods including GPU implementation [17].

Acknowledgment

This work has been partly supported by Semiconductor Technology Academic Research Center (STARC), and by KAKENHI Grant-in-Aid for Scientific Research (B) 22360143 from JSPS. This work has been also partly supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Mentor Graphics, Inc. and Synopsys, Inc.

References

[1] S.R. Nassif, A.J. Strojwas, and S.W. Director, "A methodology for worst-case analysis of integrated circuits," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.5, no.1, pp.104–113, Jan. 1986.
[2] Y. Zhan, A.J. Strojwas, M. Sharma, and D. Newmark, "Statistical critical path analysis considering correlations," Proc. IEEE/ACM International Conference on Computer-Aided Design, pp.699–704, Nov. 2005.
[3] X. Li, J. Le, M. Celik, and L.T. Pileggi, "Defining statistical sensitivity for timing optimization of logic circuits with large-scale process and environmental variations," Proc. IEEE/ACM International Conference on Computer-Aided Design, pp.844–851, Nov. 2005.
[4] M. Pan, C.C.N. Chu, and H. Zhou, "Timing yield estimation using statistical static timing analysis," Proc. IEEE International Symposium on Circuits and Systems, pp.2461–2464, May 2005.
[5] H. Chang and S.S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single PERT-like traversal," Proc. IEEE/ACM International Conference on Computer-Aided Design, pp.621–625, Nov. 2003.
[6] J. Le, X. Li, and L.T. Pileggi, "STAC: Statistical timing analysis with correlation," Proc. IEEE/ACM Design Automation Conference, pp.343–348, June 2004.
[7] C. Visweswariah, K. Ravindran, K. Kalafala, S.G. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," Proc. IEEE/ACM Design Automation Conference, pp.331–336, June 2004.
[8] A. Agarwal, V. Zolotov, and D.T. Blaauw, "Statistical timing analysis using bounds and selective enumeration," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.22, no.9, pp.1243–1260, Sept. 2003.
[9] L. Lee, L.C. Wang, T.M. Mak, and K.T. Cheng, "A path-based methodology for post-silicon timing validation," Proc. IEEE/ACM International Conference on Computer-Aided Design, pp.713–720, Nov. 2004.
[10] M. Orshansky and A. Bandyopadhyay, "Fast statistical timing analysis handling arbitrary delay correlations," Proc. IEEE/ACM Design Automation Conference, pp.337–342, June 2004.
[11] M. Imai, T. Sato, N. Nakayama, and K. Masu, "Non-parametric statistical static timing analysis: An SSTA framework for arbitrary distribution," Proc. IEEE/ACM Design Automation Conference, pp.698–701, June 2008.
[12] V. Veetil, D. Sylvester, and D. Blaauw, "Efficient Monte Carlo based incremental statistical static timing analysis," Proc. IEEE/ACM Design Automation Conference, pp.676–681, June 2008.
[13] H. Yuasa, H. Tsutsui, H. Ochi, and T. Sato, "A fully pipelined implementation of Monte Carlo based SSTA on FPGAs," Proc. IEEE International Symposium on Quality Electronic Design, pp.785–790, March 2011.
[14] J. Cong, K. Gururaj, W. Jiang, B. Liu, K. Minkovich, B. Yuan, and Y. Zou, "Accelerating Monte Carlo based SSTA using FPGA," Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp.111–114, Feb. 2010.
[15] M.A. Breuer and A.D. Friedman, Diagnosis & reliable design of

digital systems, Computer Science Press, 1976.
[16] N. Ishiura, H. Yasuura, and S. Yajima, "High-speed logic simulation on vector processors," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.6, no.3, pp.305–321, May 1987.
[17] K. Gulati and S.P. Khatri, "Accelerating statistical static timing analysis using graphics processing units," Proc. IEEE/ACM Asia and South Pacific Design Automation Conference, pp.260–265, Jan. 2009.
[18] V. Sriram and D. Kearney, "An FPGA implementation of a parallelized MT19937 uniform random number generator," EURASIP J. Embedded Systems, pp.7:1–7:6, Jan. 2009.
[19] D.B. Thomas and W. Luk, "Non-uniform random number generation through piecewise linear approximations," IET Computers & Digital Techniques, vol.1, no.4, pp.312–321, 2007.
[20] "PATMOS'2011 timing analysis contest." http://patmos-tac.inesc-id.pt/ (accessed on Nov. 1, 2012).
[21] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Tech. Rep. UCB/ERL M92/41, EECS Department, University of California, Berkeley, 1992.

Hiroshi Yuasa received his B.E. degree in Electrical and Electronic Engineering and his M.E. degrees in Communications and Computer Engineering from Kyoto University in 2009 and 2011, respectively. Presently, he is with Sony Corporation.

Hiroshi Tsutsui received his B.E. degree in Electrical and Electronic Engineering and his master and Ph.D. degrees in Communications and Computer Engineering from Kyoto University in 2000, 2002, and 2005, respectively. He is currently an assistant professor in the Department of Communications and Computer Engineering, Kyoto University. His research interests include circuits and systems for image processing and VLSI design methodology. He is a member of IEEE, ACM, IPSJ, IEEJ, and IIEEJ.

Hiroyuki Ochi received the B.E., M.E., and Ph.D. degrees in Engineering from Kyoto University in 1989, 1991, and 1994, respectively. In 1994, he joined Department of Computer Engineering, Hiroshima City University as an associate professor. Since 2004, he has been an associate professor of Department of Communications and Computer Engineering, Kyoto University. His research interests include low-power/reliability-aware VLSI design and reconfigurable architectures. He is a member of IPSJ, IEEE, and ACM.

Takashi Sato received B.E. and M.E. degrees from Waseda University, Tokyo, Japan, and a Ph.D. degree from Kyoto University, Kyoto, Japan. He was with Hitachi, Ltd., Tokyo, Japan, from 1991 to 2003, with Renesas Technology Corp., Tokyo, Japan, from 2003 to 2006, and with the Tokyo Institute of Technology, Yokohama, Japan. In 2009, he joined the Graduate School of Informatics, Kyoto University, Kyoto, Japan, where he is currently a professor. He was a visiting industrial fellow at the University of California, Berkeley, from 1998 to 1999. His research interests include CAD for nanometer-scale LSI design, fabrication-aware design methodology, and performance optimization for variation tolerance. Dr. Sato is a member of the IEEE. He received the Beatrice Winner Award at ISSCC 2000 and the Best Paper Award at ISQED 2003.