Fast STA Prediction-based Gate-level Timing Simulation

Tariq B. Ahmad
ECE Department
University of Massachusetts
Amherst, MA, USA
tbashir@ecs.umass.edu

Maciej J. Ciesielski
ECE Department
University of Massachusetts
Amherst, MA, USA
ciesiel@ecs.umass.edu

***Abstract*— Traditional dynamic simulation with standard delay format (SDF) back-annotation cannot be reliably performed on large designs. The large size of SDF files makes the event-driven timing simulation extremely slow as it has to process an excessive number of events. In order to accelerate gate-level timing simulation we propose an automated fast prediction-based gate-level timing simulation that combines static timing analysis (STA) at the block level with dynamic timing simulation at the I/O interfaces. We demonstrate that the proposed timing simulation can be done earlier in the design cycle in parallel with synthesis.**

***Index Terms*— Gate-level timing, static timing analysis, dynamic timing simulation, ASIC, Opencores, RTL, Verilog**

I. Introduction

A. Literature Survey on Verification

As design size and complexity increase, so is the need to verify designs quickly and reliably. This, combined with the reduced design cycle of 3-6 months, makes verification an extremely challenging task. Today, verification consumes over 70% of the design cycle time and, on an average, the ratio of verification to design engineers is 3:1 [1][2].

Verification engineers use a wide variety of verification approaches, including constrained random simulation for datapath components, equivalence checking for pre- and post-synthesis netlists, and formal property verification for control and protocol checking. As the design gets refined into lower levels of abstraction, such as gate or layout level, in an application specific integrated circuit (ASIC) or field programmable gate array (FPGA) design flow, the performance of simulation drops significantly. This is due to the large size of gate-level and layout-level netlists, and gate and wire delays available only at these lower levels of abstraction. Formal property verification still cannot cope with the design complexity beyond register transfer level (RTL) of abstraction. Equivalence checking can only compare functionality (but not the timing) of two designs; it suffers from memory capacity limitations for large designs and may require defining structurally similar cut points as a basis of comparison. Other techniques, such as static timing analysis (STA), are prone to manually imposed constraints. A designer may inadvertently miss false or multi-cycle paths or add such paths that should not have been constrained [3]. Furthermore,

STA does not work for asynchronous interfaces [15] and there is no way to validate the results of STA, except by simulation. To accelerate simulation at lower levels of abstraction, hardware assisted simulation acceleration (based on FPGAs), emulation (such as Cadence Palladium platform or EVE/Synopsys Zebu platforms), and other techniques have been introduced. These techniques are expensive, quite complex to deal with, and may require redesigning testbench or the design under verification (DUV) [4]. Despite this, traditional hardware description language (HDL) simulation remains the most popular method of design verification, because of its ease of use, inexpensive computing platform, 100% signal controllability and observability [5]. Figure 1 illustrates the use of simulation in a typical ASIC design flow.

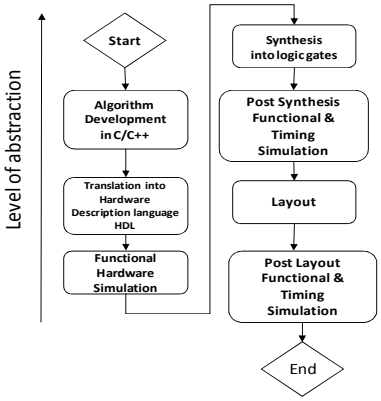

Figure 1. Simulation in ASIC/FPGA Design Flow

It is clear from the above description that simulation has its own special place in the design flow and it is not going away in the foreseeable future. As the design gets refined into lower levels of abstraction, such as gate-level and layout level, functional (zero-delay) and timing simulations can validate the results of synthesis, STA or equivalence checking. Moreover, neither STA nor equivalence checking can find bugs due to X (unknown) signal propagation. Even though RTL regression is run on a daily basis, industry insists on gate-level simulation before sign-off.

Gate-level simulation is necessary to validate the results of RTL and logic synthesis. At this stage, gate-level simulation

978-3-9815370-2-4/DATE14/©2014 EDAA

can be functional (zero-delay) or unit-delay, where all gate-level cells are assumed to have delay value of 1 timescale unit. Later, gate-level timing simulation can be performed in the pre-layout or post-layout stage using standard delay format (SDF) back-annotation. Gate-level simulations are considered a must for verifying timing critical paths of asynchronous design as such paths cannot be handled by STA tools. Furthermore, gate-level simulation is used to verify the constraints of static verification tools such as STA and equivalence checking. These constraints are added manually, and the quality of results obtained with static tools is only as good as the imposed constraints. Gate-level simulation is also used to verify the power-up, power-down and reset sequences of the full chip. It is also used to estimate dynamic power drawn by the chip. Finally, gate-level simulation is used after engineering change order (ECO) to verify the applied changes [15].

B. Issues with Simulation

The dominant technique used for functional and timing simulation is event-driven HDL simulation [5]. However, event-driven simulation suffers from very low performance because of its inherently sequential nature and heavy event activities in gate-level simulation. As the design gets refined into lower levels of abstraction, and as more debugging features are added into the design, simulation time increases significantly. Figure 2 shows the simulation performance of Opencores [14] AES128 design [23] at various levels of abstraction with debugging features enabled. As the level of abstraction goes down to gate or layout level and debugging features are enabled, simulation performance drops down significantly. This is due to a large number of events at the gate-level or layout level, timing checks and disk access to dump simulation data.

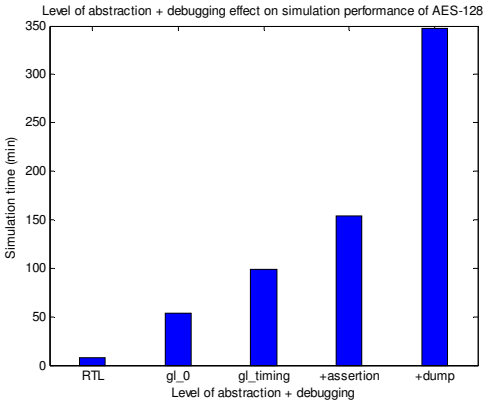

Figure 2. Drop down in simulation performance with level of abstraction + debugging

C. Scope of this Work

This work addresses the issue of improving performance of event-driven gate-level timing simulation using static timing analysis (STA) as "timing predictor" at the block level. We propose an automated partitioning scheme that partitions the gate-level netlist into blocks for SDF annotation and STA. We also propose a new design/verification flow where timing simulation can be done early in the design cycle using cycle-accurate RTL.

The next section briefly reviews literature on improving simulation performance using parallel simulation. Section 3 presents a new approach to accelerating gate-level timing simulation using STA. Section 4 describes the setup, experiments and results based on the new approach. Section 5 describes how to verify simulation results using the proposed flow. New simulation flow based on early simulation is discussed in Section 6. The paper is concluded in Section 7 and References are listed in Section 8. Our contributions in this work span Sections 3 through 7.

II. PARALLEL GATE-LEVEL SIMULATION

A. Parallel Discrete Event-Driven Simulation (PDES)

To address the performance of event-driven gate-level simulation (both functional and timing), distributed parallel simulation [6][7] has been proposed. Unfortunately, it has not been very successful for the following reasons: i) difficulty in design partitioning and load balancing; ii) communication overhead; iii) synchronization overhead between design blocks imposed by the distributed environment; and iv) lack of concurrency in the original design. The area of parallel simulation is rich in literature, and most of the known work concerns traditional parallel simulation, based on physical partitioning of the design into modules distributed to individual simulators. PDES is not practical for gate-level timing simulation as gate-level timing simulation involves processing huge number of events across partitions which severely degrades simulation performance. For this reason, recent parallel multi-core simulators provided by major EDA vendors [20][21] do not handle gate-level timing simulation in their multi-core simulators.

B. Time Parallel Simulation (TPSIM)

In contrast to the parallel discrete event HDL simulation described above, which partitions the design in spatial domain, there has been some interesting work on time-parallel discrete event HDL simulation [17]. This approach, called multi-level temporal parallel event-driven simulation (MULTES) [18], parallelizes simulation in time domain by dividing it into independent time intervals (simulation slices). Each slice is then simulated on a different processor. The key requirement of this technique is finding the initial state of each slice, which is a challenging problem, especially for a design obtained by re-timing and re-synthesis [18]. For functional gate-level simulation, RTL is used to find the initial state of each slice and for gate-level timing simulation, functional gate-level is used to find the initial state of each time slice. Limitations to this include space complexity (each simulation slice simulates the whole design) and limited applicability to multi-core architecture. Multi-core architecture is more suitable to design partitions rather than running entire design on every core. In general, the method does not scale well with the multi-core

architecture, cannot be fully automated and requires manual interaction. However, if the designer requires gate-level timing simulation with full SDF back-annotation, TPSIM can be used. This will need manual interaction and state matching.

III. HYBRID GATE-LEVEL TIMING SIMULATION

A. Basic Idea

We present a new approach to improve performance of gate-level timing simulation. The basic idea is to use static timing analysis (STA) as timing predictor at the block level. It uses worst case (critical path) delay, captured by STA, instead of the actual cell delays for annotating block-level timing during simulation. This idea is illustrated in Figures 3 and 4. Figure 3 shows gate-level timing simulation of a design consisting of two blocks, with timing simulation accomplished with SDF back-annotation applied to the entire design. However, for large designs, such SDF back-annotation will negatively impact the performance of gate-level timing simulation.

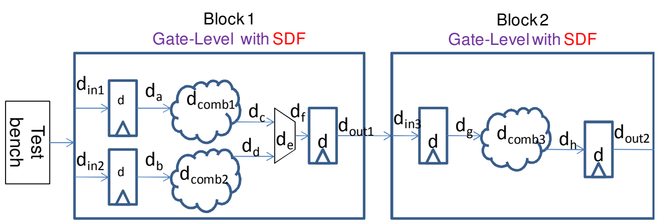

Figure 3. Gate-level timing simulation with full SDF annotation

To improve the performance of gate-level timing simulation, we propose a hybrid approach, shown in Figure 4, where only gate-level block 2 is SDF back-annotated. Gate-level block 1 is analyzed by STA tool which reports the maximum delay inside the block. Only this value is back-annotated during simulation as d_{sta} at the output of block1. This type of timing annotation is termed as selective SDF annotation. Note that STA can be performed on gate-level block 1 as part of the whole design or separately if I/O delays are modeled appropriately.

Essentially, block 1 is simulated in functional (zero-delay) mode i.e., without SDF back- annotation, while block 2 is simulated with SDF back-annotation. In case of multiple blocks, the proposed STA based timing prediction approach can be used for majority of the blocks to speed up gate-level timing simulation. Designers typically know the timing critical blocks in a design where selective SDF back-annotation can be used to quickly verify timing.

B. Partitioning

Partitioning of gate-level netlist into blocks for SDF annotation and STA is a challenging problem as verification engineer may not have sufficient insight in identifying timing-critical blocks. Furthermore, partitioning schemes are often manually driven. This may cause a problem when dealing with large gate-level netlists. Often gate-level netlist is flattened and hierarchy is not preserved. We propose a partitioning scheme based on STA that is fully automated and works for flat or hierarchical gate-level netlist. This is one of the most important contributions of this paper. Moreover, the partitioning does not have to be at the register boundary. For multi-clock designs, clock domain crossings (CDC) are always SDF back-annotated. Formal tools like Synopsys Formality can detect CDC paths in a design.

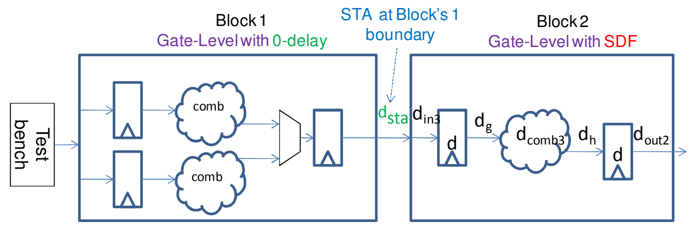

Figure 4.Gate-level timing simulation with hybrid approach

STA determines slowest (critical path) in a design. One can also choose to report not only the most timing critical path but the next most timing critical path and so on. STA report then reports these timing critical paths and the associated module instances. Since these paths are time critical, one would always want to do SDF back-annotated timing simulation on these module instances to make sure that their timing conforms to STA results. In brief, one can include all the module instances that are in the timing critical paths for SDF back-annotation. This group of instances is shown as Block2 in Figure 4. All the other module instances can be considered not timing critical. These module instances shall be simulated in functional (zero-delay) mode. This group of instances are in Block1. However, one needs to run STA on Block1 to find out their worst case delay d_{sta} as shown in Figure 4. All of this can be automated in a flow as shown in Figure 5.

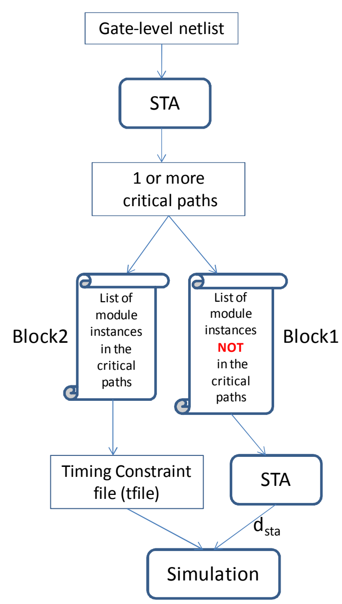

Figure 5. Automated Partitioning and simulation flow for hybrid gate-level timing simulation

C. Integration with the existing Design Flow

The flow for this approach is shown in Figure 6. The key idea is to capture peripheral timing of each block via static timing analysis or various estimates derived from time budgeting. As some (non-critical) of the design blocks are simulated in functional (zero-delay) mode, except at the block periphery, this should result in a significant speedup compared to the simulation with full SDF back-annotation.

To further improve the performance of gate-level timing simulation, the majority of gate-level blocks can be replaced by their cycle-accurate RTL blocks with peripheral timing captured via STA, time budgeting or other estimates to be explained next.

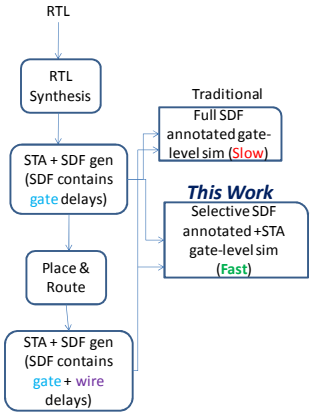

Figure 6. Proposed flow for hybrid gate-level timing simulation

D. Early Gate-level Timing Simulation

The idea of early simulation is shown in Figure 7, where gate-level Block1 is replaced by equivalent RTL. Now RTL is simulated instead of gate-level for Block1. The key idea is to perform timing simulation using estimated timing d_{est} early in the design cycle when the blocks have not been synthesized. The estimated timing can come from time budgeting or a tool like Synopsys DC Explorer [22]. This is in contrast to the conventional approach, where gate-level simulation is performed later in the design flow, after synthesis or place & route step, with all the detailed delay data already available. Major simulator vendors have already embraced the idea of early timing simulation based on the estimated delays realizing that performing gate-level timing simulations late in the design cycle is prohibitively slow. Verification engineers get around this problem by performing gate-level timing simulation of only time critical blocks with few test vectors. However, they are not able to perform full chip timing simulation with large number of test vectors, which often leaves certain timing bugs undetected. Synopsys [21] has recently announced a new product called DC Explorer [22] that is based on the same idea of early design exploration. It can do early synthesis, timing and other estimates with sufficient accuracy for designs to start the simulation process early in the design flow. For this reason, Synopsys DC Explorer is rapidly getting adoption in the industry.

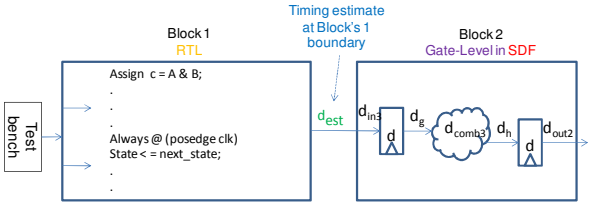

Figure 7. Early timing simulation using RTL with estimate of peripheral timing

IV. Experiments

A. Setup

We tested the proposed approach by measuring the performance of gate-level timing simulation of several Opencores Verilog designs [14], namely AES-128 [23], 3-DES [24], VGA controller [25] and JPEG encoder [26] designs. We used Cadence [20] Incisive Unified Simulator 13.1 on quad-core Intel CPU with 8GB RAM. The designs were synthesized with Synopsys Design Compiler using TSMC 65nm standard cell library. All these designs except VGA controller are single clock designs. The following Table 1 shows essential statistics for these designs.

Table 1. Gate-level design statistics

Implementations	Synthesized Area in NAND2 equivalents
AES-128 Iterative	18400
3-DES	96650
VGA	144189
JPEG	968788

B. Results

First, we show simulation results with the AES-128 design. We started with SDF annotation of majority of blocks (to accommodate many timing critical paths) and then gradually decreased the number of blocks in SDF annotation to one (to accommodate the worst case timing path). Table 2 shows that significant speedup (~5x) over full SDF annotated timing simulation can be obtained.

The waveforms in Figure 8 illustrate the difference between full SDF annotation and selective SDF annotation. It shows that signal from selective SDF annotation is delayed more than the SDF-annotated signal due to STA delay, but contains no glitches. This means fewer events to process during simulation and hence faster simulation. Both signals match at the clock cycle boundary (positive edge of the clock).

In the next set of experiments, all designs were divided into two gate-level blocks, Block1 and Block2 as in Figure 4. Block 2 contains module instances from the most timing

critical path. Here, the number of timing critical paths considered is one.

The proposed approach has an additional advantage that it validates the result of STA which is depends on manual constraints entry. If the simulation exhibits timing failure, it will help debug STA constraints. Once the constraints are corrected, STA is run again to provide new $\#d_{sta}$ value. This STA-to-simulation cycle is repeated until all timing failures are debugged and removed from the simulation.

Table 2. Gate-level timing simulation speedup of AES-128 for variable number of blocks in SDF annotation

# of modules instances in SDF Annotation /17	Module Instances in 0-delay	Full SDF annotated timing sim (*T1*) Min	Selective SDF annotated Timing sim (*T2*) Min	Speedup (*T1*/*T2*)
16	test.u0.us00 (one Sbox)	172	115	1.49
16	test.u0.u0 (key_expand)	172	84	2.04
15	test.u0.us00 to test.u0.us01 (two Sboxes)	172	110	1.56
13	test.u0.us00 to test.u0.us03 (Four sboxes)	172	100	1.72
9	test.u0.us00 to test.u0.us13 (8 sboxes)	172	77	2.23
7	test.u0.us00 to test.u0.us23 (12 sboxes)	172	56	3.07
1	test.u0.us00 to test.u0.us33 (16 sboxes)	172	37	4.64

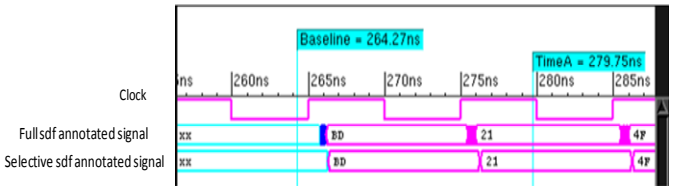

Figure 8. Full SDF annotation vs selective SDF annotation in waveform

Table 3. Speedup with hybrid gate-level timing simulation

Implementations	Full SDF annotated timing sim *T1* (min)	Hybrid timing simulation *T2* (min)	Speedup (*T1*/*T2*)
AES-128	172	37	4.64
3-DES	196	51	3.92
VGA Controller	812	232	3.50
JPEG Controller	273	79	3.45

V. Verification of Simulation Results

In order to verify the timing correctness of the approach, we propose the following dumping-based flow, shown in Figure 9. Note that this is an optional step, used only to verify the proposed simulation approach. In practice, verification engineer can skip this step to reduce the verification time.

While testbench can verify functional correctness of the two simulations, the proposed verification scheme helps in verifying timing correctness of the two simulations. In order for both simulations to be timing correct, the monitored signals from the two simulations should match at the clock cycle boundary. Unfortunately, dumping, as shown in Figure 2 can drastically reduce simulation performance. Further, the amount of dumping can cause the disk to quickly become full. Therefore, it is recommended that dumping should be done for a small time interval rather than for the entire simulation. We used small simulation intervals to verify timing correctness of the output signals of the designs. Cadence Comparescan tool was used to compare the dumped signals. The tool reported the signals to be matching at the clock cycle boundary. Table 3 shows comparison between full SDF gate-level timing simulation and proposed hybrid gate-level timing simulation for all the flip-flops/registers in VGA and AES-128 designs. The fact that the values of the registers match at the clock cycle boundary during the entire simulation confirms the accuracy of our approach.

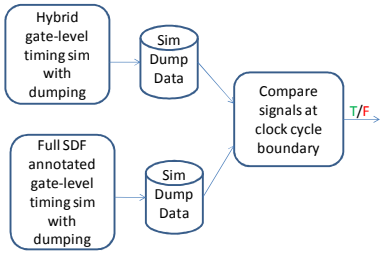

Figure 9. Verification flow for hybrid gate-level timing simulation

Table 4. Accuracy of hybrid gate-level timing simulation at the register boundary

Design name	A: Total # of registers	B: # of Full SDF timing vs selective SDF timing register match	Lower bound on hybrid prediction accuracy *(B/A)*100*
VGA controller	1611	1611	100 %
AES128	530	530	100 %

VI. New Gate-Level Timing Simulation Flow

We also propose the design/verification flow in which gate-level timing simulation is performed early in the design cycle, using estimates from time budgeting and/or STA. Tools like Synopsys DC Explorer [22] can provide timing estimates for running gate-level timing simulation. As already mentioned performing gate-level timing simulation late in the design cycle is prohibitively slow and may result in design changes back in the RTL or may require ECO. Further, the idea of performing long full chip timing simulation in a short amount of time is much welcomed by the industry. Figures 10 and 11 show the traditional and new flow for simulation,

respectively. The obvious advantage of the new flow is rapid gate-level timing simulation early in the design cycle so that timing checks are validated and bugs are caught early on.

Figure 10. Traditional simulation flow in ASIC design

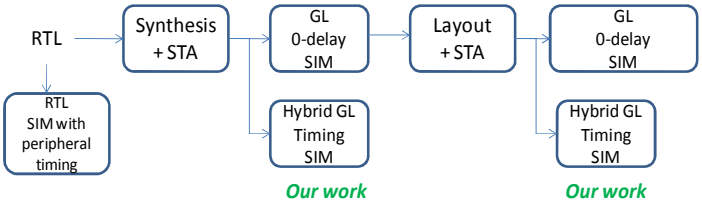

Figure 11. Proposed flow of early simulation in ASIC design

VII. Conclusion and Future Work

Today, system-on-chip (SoC) designs have become widespread. These designs integrate multiple hardware cores working at different frequencies. Timing simulation of such multi-clock domain designs is critical. Traditional dynamic simulation with SDF back-annotation cannot be done on such large designs. In addition, event-driven timing simulation is extremely slow, suffers from capacity issues because of large SDF files (exceeding 10GB for small SoC designs) and is generally done late in the ASIC design cycle after synthesis or layout.

This paper provides a proof of concept of hybrid gate-level timing simulation that makes use of STA and selective SDF back-annotation to accelerate gate-level timing simulation. STA acts as timing predictor for blocks which are run without SDF back-annotation. The approach also validates the result of STA which depends on manual constraints entry. The proposed approach is applicable to multi-clock domain designs with clock domain crossings (CDC). We are actively working on such larger designs. Further, we proposed a flow for early simulation in the ASIC/FPGA design flow that includes rapid hybrid gate-level timing simulation.

Acknowledgment

This work was supported in part by funding from the National Science Foundation, award No. CCF 1017530.

References

[1] T. Anderson, and R. Bhagat, “Tackling Functional Verification for Virtual Components,” ISD Magazine, pp. 26, November 2000.

[2] P. Rashinkar, and L. Singh, “New SoC Verification Techniques,” Abstract for tutorial, IP/SOC 2001 Conference, March 19, 2001.

[3] Symbolic Simulation speeds Timing Closure (http://www.techdesignforums.com/eda/eda-topics/verified-rtl-to- gates/symbolic-simulation-speeds-timing-closure)

[4] D. Kim, M. Ciesielski, and S. Yang, “A new Distributed Event-driven Gate-level HDL Simulation by Accurate Prediction,” Design and Test Europe (DATE 2011), pp. 547-550, March 2011.

[5] W.K. Lam, “Hardware Design Verification: Simulation and Formal Method-Based Approaches,” Prentice Hall, 2005.

[6] SimCluster datasheet, Avery Design Automation (http://www.averydesign.com)

[7] MP-Sim datasheet, Axiom Design Automation (http://www.axiomda.com)

[8] R.M. Fujimoto, “Parallel Discrete Event Simulation,” Communication of the ACM, Vol. 33, No. 10, pp. 30-53, Oct. 1990.

[9] A. Gafni. “Rollback Mechanisms for Optimistic Distributed Simulation Systems,” SCS Multiconference on Distributed Simulation, vol.3, pp 61-67, July 1988.

[10] R.M. Fujimoto, “Time Warp on a Shared Memory Multiprocessor,” Transactions of the Society for Computer Simulation, Vol, 6, No. 3, pp. 211-239, July 1989.

[11] D.M. Nicol, “Principles of Conservative Parallel Simulation,” Proceedings. of the 28th Winter Simulation Conference, pp. 128–135, 1996.

[12] D. Chatterjee, A. DeOrio, and V. Bertacco, “Event- driven Gate-level Simulation with General Purpose GPUs,” Proceedings. of Design Automation Conference (DAC09), pp. 557-562, 2009.

[13] Y. Zhu, B. Wang, and Y. Deng, “Massively Parallel Logic Simulation with GPUs,” article 29, ACM Trans. Design Automation of Electronic Systems, June 2011.

[14] Opencores designs (www.opencores.org)

[15] VerificationBlog (http://whatisverification.blogspot.com/2011/06/gate-level-simulations-necessary-evil.html)

[16] L. Li, and C. Tropper, “A design-driven Partitioning Algorithm for Distributed Verilog Simulation,” in Proc. 20th International Workshop on Principles of Advanced and Distributed Simulation (PADS), pp. 211–218, 2007.

[17] D. Kim, M. Ciesielski, and S. Yang, "MULTES: MUlti-Level Temporal-parallel Event-driven Simulation," IEEE Trans. on CAD of Integrated Circuits and Systems 32(6): pp. 845-857 (2013).

[18] D.Kim, “MULTES : Multi-level Temporal-parallel Event-driven Simulation, ” PhD Thesis, University of Massachusetts Amherst, 2011.

[19] F. Rodriguez-Henriques, N. Saqib, A. Perez, and C. Koc, “Cryptographic Algorithms on Reconfigurable Hardware,” Springer, 2006.

[20] Cadence (http://www.cadence.com)

[21] Synopsys (http://www.synopsys.com)

[22] Synopsys DC Explorer (http://www.synopsys.com/tools/implementation/rtlsynthesis/dcexplorer/Pages/default.aspx)

[23] AES-128 Opencores design (http://opencores.org/project,aes_core)

[24] DES-3 Opencores design (http://opencores.org/project,des)

[25] VGA Controller Opencores design (http://opencores.org/project,vga_lcd)

[26] JPEG Encoder Opencores design (http://opencores.org/project,jpegencode)