Fast Time-Parallel C-based Event-Driven RTL Simulation

Tariq Bashir Ahmad
ECE Department
University of Massachusetts Amherst
tbashir@ecs.umass.edu

Maciej Ciesielski
ECE Department
University of Massachusetts Amherst
ciesiel@ecs.umass.edu

***Abstract*—Simulation of the RTL model is one of the first and mandatory steps of the design verification flow. Such a simulation needs to be repeated often due to the changing nature of the design in its early development stages and after consecutive bug fixing. Despite its relatively high level of abstraction, RTL simulation is a very time consuming process, often requiring nightly or week-long regression runs. In this work, we propose an original approach to accelerating RTL simulation that leverages parallelism offered by multi-core machines. However, in contrast to traditional, parallel distributed RTL simulation, the proposed method accelerates RTL simulation in *temporal domain* by dividing the entire simulation run into independent simulation slices, each to be run on a separate core. It is combined with fast simulation model at ESL level that provides the required initial state for each independent simulation slice. The paper describes the basic idea of the method and provides some initial experimental results showing its effectiveness in improving RTL simulation performance in an automated way.**

Keywords—Simulation; Verfication; RTL; Verilog; ESL; C; SystemC; Testbench

I. Introduction

A. Current Verification Trends

According to Moore's law, density of hardware chips is doubling every 18 months. As the design size and complexity increase, the task of verifying a chip in 6-12 months becomes a challenge. Dramatic increase in simulation run time demanded by those designs, makes it harder to simulate and verify large designs. Today, simulation time has increased from minutes and hours to days and weeks. This has led to new verification trends. The use of hardware to assist simulation such as Field Programmable Gate Array (FPGA), Graphical Processing Unit (GPU) and custom hardware emulators has emerged. However, these hardware assisted simulators are limited in what they can simulate, are quite expensive, hard to set up, require partitioning of the design, have lengthy design compilation time and suffer from reduced signal controllability and observability making it harder to debug a design [1][5][13]. Companies designing Systems on a Chip (SoC) still use hardware assisted simulation because of time to market pressure but this approach is not viable for companies with limited budget [4].

B. Scope of Simulation

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 shows simulation speed at various levels of abstraction, using gate-level timing model as a base line. Note how simulation of higher level model is at least 10x faster than the level below it.

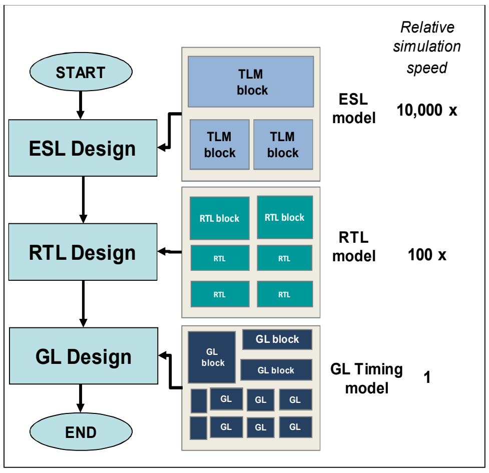

Figure 1. Simulation speed at various levels of abstraction

It is clear from above description that simulation has its own special place in the design flow and it is not going away in the near future. RTL simulation is necessary to validate the translation from design intent specification or a higher level model (SystemC/C/C++) to RTL. 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. As the design gets refined into lower levels of abstraction, and as more debugging features are added into the design, simulation time increases significantly. This is caused by a large number of events at the gate level or layout level, enabling timing checks, assertions and disk access to dump simulation data.

C. *Scope of this Work*

This work addresses performance improvement of event-driven RTL simulation. Section 2 surveys parallel simulation approaches to improve simulation performance. Section 3 introduces temporal RTL simulation. Experiments and results are presented in Section 4 and conclusions are stated in Section 5. Section 6 lists the references to the cited literature. Our contributions in this work span Sections 3 to 5.

II. Survey of Parallel Simulation

A. *Spatial Parallel Simulation (PDES)*

A common approach to distributed parallel simulation, termed as Parallel Discrete Event-driven Simulation (PDES) [6], partitions the design into two or more partitions and assigns each partition to a different processing unit, called Logical Processor (LP). Simulation correctness is ensured by preserving event ordering among LPs, which makes synchronization among LPs important. Factors that affect the performance of such distributed parallel simulation are: interconnect between LPs, load-balancing among LPs, and synchronization overhead. PDES has not been very successful as it failed to achieve meaningful performance improvement of HDL simulation, caused by issues such as design partitioning, synchronization, inter-module communication, and load balancing.

B. *Temporal Parallel Simulation (TPS)*

TPS partitions the simulation time in temporal domain as opposed to partitioning the design in spatial domain as is done by PDES. It partitions the simulation run time into shorter intervals in time domain, called *slices*. In principle, if simulation of each slice can be run independently of each other, this technique should provide speedup as it does not suffer from communication and synchronization overhead. However, it has its own problems, namely finding initial state of each slice. A recently proposed parallel simulation method for gate-level simulations (functional and timing), MULTES (Multi-Level Temporal-Parallel Event Driven Simulation) [7], solves the initial state problem by finding state matching between the RTL (used for reference) and gate-level design (used for target simulation). However, state matching is a difficult problem for designs that are optimized, re-synthesized and retimed. For this reason the method is mostly applicable to designs that did not undergo retiming during synthesis.

C. *Issues with Multi-core Simulators*

Recently commercial EDA (Electronic Design Automation) tool vendors [2][3] have introduced multi-core simulators that run on multi-core machines. Unfortunately, they have limited success because of high cost, communication and synchronization overhead mentioned earlier, inability to support Verilog PLI (Programming Language Interface), gate-level timing and new SystemVerilog testbench features [7].

III. Temporal Parallel RTL Simulation

A. *Preliminaries*

RTL simulation performance can be improved if dependencies in RTL simulation are removed somehow. We discuss two types of dependencies

1. <u>Time dependency</u>: Before simulating the entire RTL design at a particular time *t*, the design must be simulated at all times from 0 to *t-1*.
2. <u>Spatial dependency</u>: At a particular time *t*, one component of RTL design depends on the value from another component of the RTL design.

In this paper, we concentrate on removing the time dependency in simulation of a design. Temporal parallel simulation (TPS) exploits time dependency while PDES exploits spatial dependency in the design. In TPS, simulation time intervals are made independent by pre-computing the initial state of each time interval. This allows TPS to achieve full parallelism by avoiding communication and synchronization overhead, which is inherent to PDES. To provide a correct initial state of each time interval (*slice*) for parallel RTL simulation, we follow the following two-step approach [7].

1. *Reference Simulation*: Simulation that provides initial state of each time slice in TPS. Normally, this simulation is fast as it uses a higher level model.
2. *Target Simulation*: Simulation of a time slice that uses initial state provided by reference simulation. Normally, this simulation is slower compared to the reference simulation, as it is done on a lower level of abstraction than the reference simulation.

MULTES [7] applied TPS idea to speed up functional gate-level simulation by using fast RTL simulation as reference. The initial states were obtained from checkpoints saved during reference simulation and then restored for gate-level target simulation. We extend this idea to RTL simulation and use a higher-level design model (often referred to as Electronic System Level, ESL) as reference for RTL simulation. The basic idea of TPS at RTL is illustrated in Figure 2, which shows fast reference simulation to provide the initial state of each slice for the target RTL simulation. In this work we compute the initial states for the RTL simulation slices using SystemC or C. The computation of the initial state is done "on the fly", when needed by the RTL simulation, rather than dumping and restoring state as proposed in [7]. This approach has additional advantage that it avoids saving and restoring the initial states from the disk.

The number of target simulations that can be performed in parallel is determined by the number of CPU cores available. The theoretical performance of TPS, measured in total simulation time T_{tps} can be expressed by Equation 1.

$$T_{tps} = \max_{1 \le i \le n}(T_{ref}(i) + T_{tf}(i) + T_{target}(i))$$

Equation (1)

where $T_{ref}(i)$ denotes the time to run reference simulation to provide the initial state for target simulation of the i^{th} time slice; $T_{tf}(i)$ is the *testbench forwarding time,* simulation time needed to bring the testbench to the point where it can simulate the RTL circuit with the correct input vectors. For each slice, this simulation has to be done from the beginning. Therefore, this time increases with the number and the length of slices. $T_{tf}(i)$ should not be confused with T_{ref} which is the time to provide RTL with the design state. $T_{target}(i)$ denotes the target simulation time for the i^{th} time slice. As simulation runs on different CPU cores in parallel, the CPU core that takes the most amount of time to simulate determines the performance of time parallel RTL simulation as described in Equation (1).

For the simulation to be correct, the state (input, output and memory elements) of RTL simulation at the end of each slice must match the state at the beginning of the next time slice for all time slices. This is known as *horizontal state matching* [7]. Figure 2 shows horizontal state matching accomplished via comparators. We explain horizontal state matching for one slice i.e., *slice 2* but it applies to all the slices except *slice 0*. For example, the state information needed to run *slice 2* at RTL is provided by fast high level simulation as *state 2*. To confirm this *state 2* was indeed the correct state for simulating *slice 2*, it is compared against the state produced at the end of RTL simulation of *slice 1*. If the comparison passes, it is known that fast high level simulation correctly predicted the initial state for *slice 2*. If the comparison fails, it means the state provided by fast high level simulation was incorrect and the simulation for *slice 2* needs to be repeated by using state from *slice 1* instead.

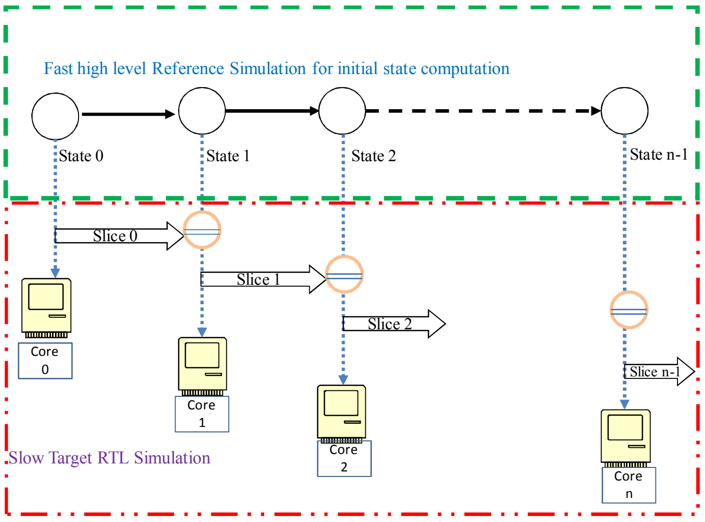
Fast high level Reference Simulation for initial state computation
State 0
State 1
State 2
State n-1
Slice 0
Slice 1
Slice 2
Slice n-1
Core 0
Core 1
Core 2
Core n
Slow Target RTL Simulation

Figure 2. Temporal parallel RTL simulation concept

B. *Integration with the existing ASIC design flow*

The concept of reference simulation is compatible with the standard ASIC and FPGA design flow where design is successively refined from a higher level of abstraction to a lower level of abstraction (refer to Figure 1). Thus, any simulation at a lower level of abstraction (target simulation) can be performed in parallel using a higher level of abstraction (reference simulation), as proposed in Figure 2. In this work, we use SystemC or C as reference simulation to enable parallel RTL target simulation. We assume that SystemC/C or any higher level model of the design is already available, as many designs are first simulated in SystemC/C or other higher level models. Furthermore, there are opensource tools, such as Verilator [8] that can convert RTL description into equivalent SystemC/C description.

It should also be noted that SystemC/C model is assumed to be at least compatible with RTL at the input/output (I/O) boundary. This means that such a model may not be detailed enough to provide internal state (memory and flip flops) of the RTL design but it must be able to predict output/s of RTL at the end of any clock cycle.

C. *Multi-core architecture of temporal RTL simulation*

We propose an architecture of Temporal RTL simulation that exploits multi-core architecture of the underlying hardware as shown in Figure 3.

The new architecture shows that ESL simulation runs as an independent thread on a CPU core. This thread simulates the design at ESL level, checkpoints the state and spawns RTL simulation of a slice on a free CPU core. At the end of each time slice simulation, ESL thread checks for horizontal state matching (explained in the Preliminaries Section). If there is a state matching between slice *i* and slice *i+1*, for every time slice *i*, ESL is known to be accurately predicting the initial state of slice *i+1* for every time slice *i*. This mode of the simulation is called "*Prediction Mode*", where ESL simulation correctly predicts the initial state of each time slice. If on the other hand, horizontal state matching fails for a *slice i+1*, the simulation result of *slice i+1* is discarded and then *slice i+1* is re-simulated using the state from previous *slice i* rather than the ESL. This mode of the simulation is called "*Actual Mode*". The actual mode imposes re-simulation overhead but it affects simulation of only the slice(s) where there is a state mismatch without affecting the rest of the simulation. In traditional simulation, the whole simulation needs to be restarted if there is a simulation mismatch or discrepancy.

The average number of cores that are busy at any time can be controlled by the ESL thread. As soon as a core is free, it is selected by the ESL thread to simulate the next time slice by providing it with the initial state.

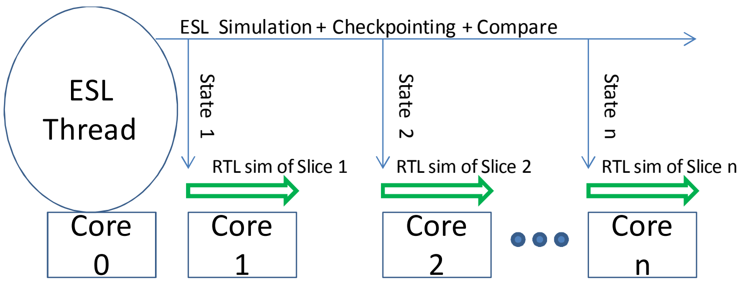

Figure 3. Multi-core architecture of temporal RTL simulation

D. *Experiment with a Sequential Design*

Figures 4 and 5 illustrate the simulation of a simple circuit under various unrolled configurations. Figure 4 shows a simple design [5] to be simulated. Figure 5 shows the same design unrolled twice. It shows that in order to get the output from the second vector (*f2*), we need output (f_1) of the circuit from the first vector. The sooner the first output is going to be available, the sooner the circuit can be simulated with the second vector. We simulated this circuit for unroll factor F = 1,2,4,6,8,10,12. Tables 1-3 show the speedup for F=1, 2 and 4. We used Cadence Incisive simulator 13.1 for RTL simulation on a quad-core Intel CPU with 8GB RAM. Figure 6 shows the setup using two cores. Figure 7 shows the plot for all the unroll factors.

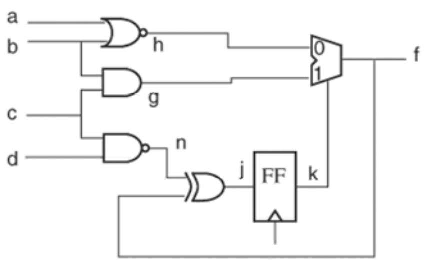

Figure 4. Simple circuit for RTL simulation

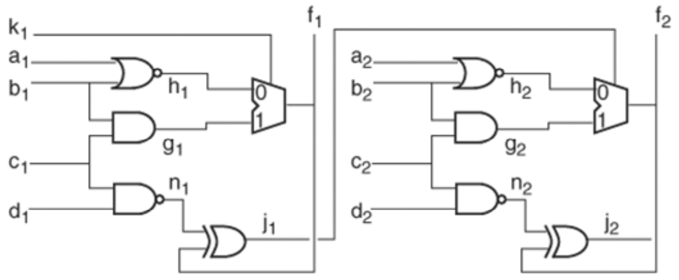

Figure 5. Simple circuit in Figure 4 unrolled twice.

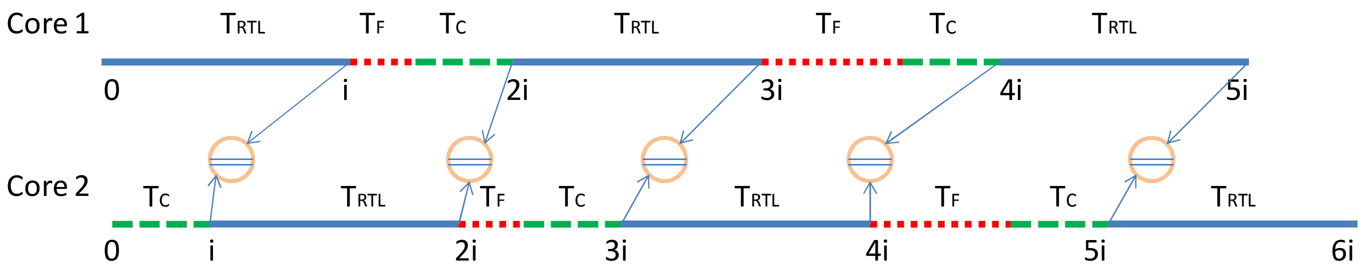

Figure 6. Accelerated RTL simulation setup on two cores.

Table 1. RTL simulation speedup for single-frame (*F*=1) circuit

# of clock Cycles	Traditional RTL sim time	*Fast RTL*				Speedup
(Billions)	(To) sec	# of slices	Forwarding time (Tf) sec	C sim time (Tc) sec	RTL sim time (TRTL) sec	To/Tf+Tc+TRTL
		2	0	104	465	1.64
1	764	4	0,47	49	600	1.27
		10	0,19,35,53,75	19	694	1.10
		2	0	197	933	1.60
2	1492	4	0,92	99	1200	1.24
		8	0,46,93,140	50	1242	1.20

Table 2. RTL simulation speedup for circuit unrolled 2x (*F*=2)

# of clock Cycles	Traditional RTL sim time	*Fast RTL*				Speedup
(Billions)	(To) sec	# of slices	Forwarding time (Tf) sec	C sim time (Tc) sec	RTL sim time (TRTL) sec	To/Tf+Tc+TRTL
		2	0	367	328	1.09
1	764	4	0,170	47	409	1.22
		10	0,18,38,58,110	30	446	1.09
		2	0	720	644	1.09
2	1492	4	0,268	152	836	1.18
		8	0,47,95,227	73	919	1.09

Table 3. RTL simulation speedup for circuit unrolled 4x (*F*=4)

# of clock Cycles	Traditional RTL sim time	*Fast RTL*				Speedup
(Billions)	(To) sec	# of slices	Forwarding time (Tf) sec	C sim time (Tc) sec	RTL sim time (TRTL) sec	To/Tf+Tc+TRTL
		2	0	345	302	1.18
1	764	4	0,150	72	362	1.30
		10	0,16,37,56,107	30	301	1.14
		2	0	650	603	1.19
2	1492	4	0,245	145	742	1.31
		8	0,48,98,228	72	827	1.17

Figure 7 shows the improvement in simulation speedup as the number of frames per simulation cycle (unroll factor *F*) increases. It approaches a factor of 2 for the case when *F*=12 and when the number of slices is 4. Note that the Tables 1-3 show the worst case time reported from the two cores.

IV. EXPERIMENTS AND RESULTS

In this experiment, we applied our parallel RTL simulation methodology to a real world design, AES-128 [9] design [12] configured in cipher block chaining mode (CBC) [10], taken from opencores.org [11]. We used Cadence Incisive simulator 13.1 for RTL simulation on a quad-core Intel CPU with 8GB RAM.

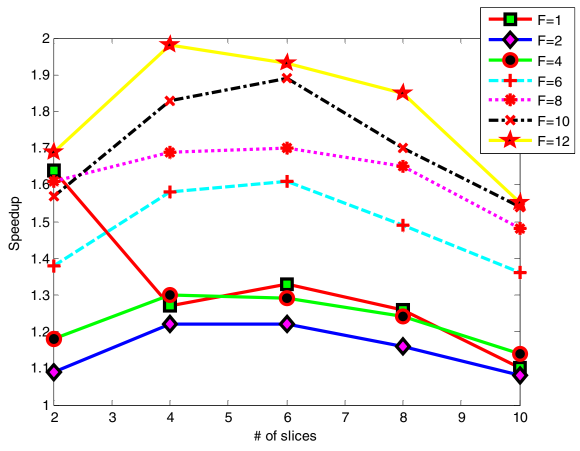

Figure 7. RTL simulation speedup as a function of the number of slices for different unroll factors.

To accelerate cipher text computation, we used high level model of the design together with RTL to parallelize the computation across multiple CPU cores. In this experiment we used a 2-core configuration and the simulation run was partitioned into 5 time slices (three on the first core and two on the second) as this offered the best overall simulation performance. Figure 8 shows the configuration.

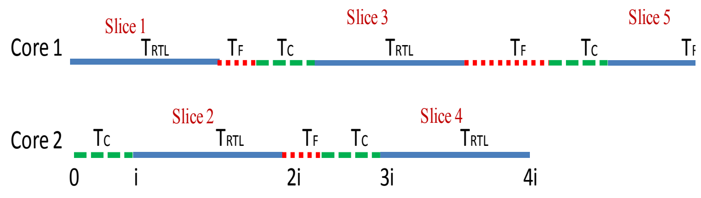

Figure 8. Simulation configuration for AES-128 on two cores

The results are shown in Table 4. Results indicate that the simulation performance was capped at about 1.5x speedup which is close to theoretical speedup of *n*=2, where *n* = number of cores used.

Table 4. Simulation speedup for AES-128 design using parallel RTL simulation

# of cores	# of time slices	# of Plain Texts	Traditional RTL sim time (T1) sec	Parallel RTL sim time (T2) sec	Speedup (T1/T2)
2	5	0.1M	5	5	1.00
2	5	1M	52	33	1.57
2	5	10M	517	340	1.52
2	5	100M	4200	2700	1.55

V. CONCLUSION

The paper presents a proof of concept of accelerating RTL simulation targeting multi-core CPUs. It proposes a new technique for accelerating RTL simulation based on temporal partitioning of the simulation and using higher level model (SystemC/C or any higher level model) to provide the initial states for the individual simulation slices. We showed that simulation can be accelerated by making intelligent choices in terms of the number of slices and number of CPU cores. To the best of our knowledge, this is the first paper that has considered RTL simulation acceleration using *temporal partitioning* with higher level model targeting multi-core machines. All of this has been automated. The methodology also allows early exploration of which configuration (number of slices and CPU cores) provides best RTL simulation speedup.

We are currently working on other designs from the Opencores [11] benchmark suite to demonstrate scalability of our approach and its applicability to industrial grade designs.

ACKNOWLEDGMENT

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

VI. REFERENCES

[1] K.H. Chang, and C. Browy, "Parallel Logic Simulation: Myth or Reality?." IEEE Computer Society, April 2012.
[2] Cadence (http://www.cadence.com)
[3] Synopsys (http://www.synopsys.com)
[4] R. Wiśniewski, A. Bukowiec, and M. Węgrzyn, "Benefits of Hardware Accelerated Simulation," DESDes' 2001.
[5] W.K. Lam, "Hardware Design Verification: Simulation and Formal Method-Based Approaches," Prentice Hall, 2005.
[6] R.M. Fujimoto, "Parallel Discrete Event Simulation," Communication of the ACM, Vol. 33, No. 10, pp. 30-53, Oct. 1990.
[7] 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).
[8] Wilson Snyder, P. Wasson, and D. Galbi. "Verilator." (2007) (http://www.veripool.org/wiki/verilator).
[9] J. Daemen, and V. Rijmen, "Rijndael." Proceedings from the First Advanced Encryption Standard Candidate Conference, National Institute of Standards and Technology (NIST). 1998.
[10] M. Dworkin," Recommendation for Block Cipher Modes of Operation. Methods and Techniquesm," No. NIST-SP-800-38A. National Institute of Standards and Technology, 2001.
[11] Opencores designs (www.opencores.org)
[12] AES-128 Opencores design (http://opencores.org/project,aes_core)
[13] W. Chen, X. Han, C. Chang, and R. Dömer. "Advances in Parallel Discrete Event Simulation for Electronic System-Level Design." (2011): 1-1.