Abstract

System-on-Chip (SoC) complexity growth has multiplied non-stop, and time-to-market pressure has driven demand for innovation in simulation performance. Logic simulation is the primary method to verify the correctness of such systems. Logic simulation is used heavily to verify the functional correctness of a design for a broad range of abstraction levels. In mainstream industry verification methodologies, typical setups coordinate the validation effort of a complex digital system by distributing logic simulation tasks among vast server farms for months at a time. Yet, the performance of logic simulation is not sufficient to satisfy the demand, leading to incomplete validation processes, escaped functional bugs, and continuous pressure on the EDA\textsuperscript{1} industry to develop faster simulation solutions. In this research, we will explore a solution that uses high-level parallel abstractions and parallel computing to boost the performance of logic simulation.

\textsuperscript{1}Electronic Design Automation
1 Project Description

1.1 Introduction and Background

SoC complexity is increasing rapidly, driven by demands in the mobile market, and increasingly by the fast-growth of assisted- and autonomous-driving applications. SoC teams utilize many verification technologies to address their complexity and time-to-market challenges; however, logic simulation continues to be the foundation for all verification flows, and continues to account for more than 90% [10] of all verification workloads. Logic simulation is the primary tool used to validate a wide range of design aspects, foremost among these being the correctness of the system’s functionality, both in its behavioral (functional verification) description, as well as in its structural (gate-level verification) one. A large body of research has been devoted to accelerate the logic simulation process [11]. Besides improving the efficiency of algorithms, parallel computing has long been widely considered as the essential solution to provide scalable simulation productivity. Unfortunately, logic simulation has been one of the most difficult problems for parallelization due to the irregularity of problems and the hard constraints of maintaining causal relations [8]. Today commercial logic simulation tools depend on multicore CPUs and clusters by mainly exploiting the task-level parallelism. In fact, large simulation farms could consist of hundreds of workstations, which would be expensive and power hungry. Meanwhile, the communication overhead might finally outweigh the performance improvement through integration of more machines.

Simulators can be grouped into two families based on their internal architecture: oblivious simulators compute all gates in the system during every simulation cycle and entail a simpler software design. Oblivious simulators have the advantage of low control overhead, but can spend significant computation time unnecessarily evaluating gates over and over whose output values do not change from cycle to cycle. Event-driven simulators limit the amount of computation by selectively simulating in each cycle only those gates whose inputs have changed since the previous cycle, and whose output may thus change in response to the switching stimulus. While the sequencing of gate evaluation in oblivious simulation can be statically determined at compile-time, event-driven simulators require a dynamic runtime scheduler, hence entail a more complex software structure. However, this latter approach is vastly more common in commercial tools because the scheduler performance overhead is largely offset by the fact that for many designs only 1 to 10% of the gates switch at each cycle, thus requiring significantly less computation. In investigating the potential for large performance improvements in logic simulators, it is noted that logic netlists present a high degree of structural parallelism that could be exploited by simulating individual gates concurrently. An ideal platform leveraging such concurrency is the modern graphics processing unit (GPU), as it includes many simple and identical computational units capable of operating concurrently by executing same instruction sequence on different data. GPU computing is a recent extension of traditional graphics processing, providing a general purpose programming interface for GPU devices, and making the vast parallel computational resources available for appli-
cations beyond the processing of graphic primitives. Platforms for GPU computing include AMD’s FireStream [12] and NVIDIA’s CUDA [13]. In addition, vendor-independent parallel computing standards such as OpenCL [3] have also been developed.

In recent years, accelerations of logic simulation have been proposed [9, 15]. The novel simulation architectures maximize the utilization of concurrent hardware resources while minimizing expensive communication overhead. The experimental results show that GPU-based simulators are capable of handling the validation of industrial-size designs while delivering more than an order-of-magnitude performance improvements on average, over the fastest multi-threaded simulators commercially available. However the deployment of such GPU-based simulators is expensive in terms of hardware resources as well as their integration in commercial simulators. The above approach did not find any breakthrough in IC² verification and as such the use of multi-threaded commercial simulators is considered as best practice.

In this research, we explore the use of high-level parallel abstractions and parallel computing for functional and gate-level simulation acceleration aiming to develop a fast simulation tool that can be used by medium-scale size company that cannot afford to spend $50K on existing commercial simulators. The future tool will also be used by universities to train hardware verification engineers.

1.2 Methodology

The state of IC verification practice is characterized by the following facts:

- The preeminence of three majors companies (Cadence, Mentor Graphics, and Synopsys), which control the complete IC verification market.

- The existence of advanced methodologies for functional verification:
  - OSVVM: Open Source VHDL Verification Methodology[4]
  - OVM: Open Verification Methodology[5]
  - UVM: Universal Verification Methodology[6]

- The issues related to time closing in gate-level simulation, leading to delay in product delivery.

- The nonexistence of viable open-source software for functional and gate-level simulation. Those available, i.e, Icarus Verilog[2], Verilator[7], and EDA Playground[1] cannot be used in a serious IC design project.

- The high demands in industry for well-trained verification engineers.
The above considerations and personal industrial experiences have led us to start this research project with the motivations to:

- Provide Open-Source/Open Hardware Engines for Hardware Verification.
- Address the ever-increasing computing demand for VLSI verification.
- Provide a framework for training of new verification engineers
- Provide low-cost verification solutions for industry.

The proposed architecture of the logic simulator to be developed is given below:

![Architecture of the Logic Simulator](image)

**Figure 1: Architecture of the Logic Simulator (EBC: Embedded Computing Module, STA: Static Timing Analyzer)**

### 1.3 Aim and Fundamental Questions

The central focus of this research is to develop a fast logic simulator that uses parallelism in abstraction and computation. Most existing simulators leverage parallelism at the gate level and deploy the simulation using multi-threading. The approach to be investigated is
based of RTL parallelism partitioning using machine learning techniques to build a fast and
effective logic simulator. The fundamental questions for this research include:

1. Given a synthesizable Verilog-based design (DUT\textsuperscript{3}), what is the adequate verification
flow that allows fast verification and complete functional coverage?

2. How to deploy hardware accelerators within the different modules of the logic simulator
such as the parallel Discrete-event simulator kernel and the OpenCL Client Driver?

3. How to build a cost effective and fast embedded computing cluster to be used to
leverage parallel computation?

Our goal is to provide effective solutions to each research question, which at the end will
allow us to develop a cost-effective fast logic simulator with its associated parallel computing
platform.

1.4 Student Role

The project is in its initial stage and the focus will be on the research question #1. The
initial work to be conducted by the researcher will be on the software architecture of the
Timing-based C/C++ Model generator. Machine learning techniques will be considered in
the development of an efficient RTL-based timing model. The proposed timing-based model
will have the following characteristics:

- The timing is considered at the module level (RTL) not at the gate-level.
- Machine learning techniques will be used to optimize the timing model.
- A module path delay is developed by considering Interface Logic Models (ILMs).
- An interface logic timing model (ILM) has partial timing of the block that includes the
timing at boundary logic only, but hides most of the internal register-to-register logic
(Figure 2).
- Advantages of using Interface Logic Models include:
  - Good performance improvements both in STA (Static Timing Analysis) and GLS
(Gate-level Simulation), since only a partial design is used.
  - High accuracy as interface logic and interface cells are preserved.

\textsuperscript{3}Design Under Test
1.5 Proposed Timeline with Project Goals

The researcher will meet several times a week with Dr. Ngalamou in addition to attending the SURE Wednesday seminars.

1.5.1 General Timeline

Weeks 1-4:

- Read selected papers on timing analysis of digital circuits along with the first four chapters of Farzad Nekoogar’s Timing Verification of ASICs.

- Get familiar with the software tool Verilator.

- Read the first four chapter of Aurelien Geron’s Hands on Machine Learning with Scikit-Learn and TensorFlow.
**Weeks 5-7:** Develop a detailed software architecture of the Timing-based C/C++ Model generator.

**Weeks 8-9:** Start the implementation and testing of the different modules of the Timing-based C/C++ Model generator.

**Week 10:** Prepare for SURE presentation.

**References**


2 Description of Funding for Materials and Supplies

We don’t expect a need for any extra materials.

3 Criteria for Student Applicants

This research requires a student to have a background in Digital Logic, Algorithms, and Programming.

