Problem C: GPU Accelerated Logic Re-simulation

Yanqing Zhang, Haoxing (Mark) Ren, Ben Keller, Brucek Khailany

NVIDIA

Revision History

August 10th – (Detailed revisions highlighted in document)

- Updated benchmarks to include only smaller netlists for easier time to test and debug
- Updated scoring requirements to only need to pass saifdiff. Contestants are no longer required to pass vcddiff to get a score – vcddiff is only used for debugging purposes from now on
- Updated benchmark debug VCD and input VCD files to maintain one consistent signal beginning and end format.
- (Minor) clarifications regarding not needing to simulate inputs and pseudo-inputs. Clarifications on required SAIF format.

Introduction

Timing-aware gate-level logic simulation is important to several EDA tasks, including simulation of test vectors for delay-dependent fault simulation, accurate power analysis, and functional verification signoff checks. Timing-aware gate-level simulation usually runs much slower than RTL simulation, from a few cycles per second on smaller unit-level designs to many seconds per cycle on today's largest full-chip SoC designs. Typically, when running timing-aware gate-level simulation, correct behavior is already known in advance from a previous RTL simulation result or from Automatic Test Pattern Generation (ATPG) vectors. As a result, we can define the input stimuli to be waveforms at every primary input and pseudo-primary inputs such as register/RAM outputs captured from an RTL simulation trace. We refer to such cases as logic "re"-simulation, since we can take a known trace on all primary and pseudo-primary inputs from a previous RTL simulation or ATPG vector, re-simulate the trace using propagation of signals through timing-aware gate-level combinational logic, and verify that results at the primary and pseudo-primary outputs match the reference RTL simulation results. In this logic re-simulation use case, a simulator can be parallelized both across cycles (stimuli) as well as the design (logic gates), and we expect to be able to exploit modern highly-parallel computing systems such as GPUs to achieve very large speedups.

Previous work in academic research and commercial tool development has proposed various techniques for accelerating gate-level logic simulation on GPUs. Early academic research demonstrated a 4-62x speedup when running gate-level simulation benchmarks on an NVIDIA 8000GT GPU compared to a 3.4 GHz Pentium 4 processor [1]. Commercial tools such as RocketSim have been reported to achieve up to a 23x speedup using GPUs on gate simulation [2] in 2013 on very large designs (hundreds of millions of gates). More recent research has exploited both gate-parallelism and stimuli-parallelism, similar to this proposed problem, to achieve up to 1000X speedup for timing-aware gate-level simulation [3], although only with 2-value simulation and on a restricted gate set. Many of the techniques described in these prior research publications may be useful references for accelerating this problem.

The GPU accelerated computing platform has also advanced significantly over the last 5-10 years, enabling advancements in deep learning by fueling its tremendous demand for computing power. We will make NVIDIA T4 GPUs [12] available to contestants via cloud GPU instances for development and benchmarking. These latest Turing-generation GPUs support 8.1 TFLOPS of peak single-precision floating-point performance with 16 GB of GDDR6 memory and 300 GB/s of memory bandwidth. T4 tensor cores support a peak of 130 TOPS of 8-bit integer performance and 260 TOPS of 4-bit integer performance, which contestants may be able to leverage for additional speedups. We also plan to support CUDA 10.0+,

including newer CUDA features [4] such as unified memory, cooperative groups, independent thread scheduling, CUDA graphs, and more to help enable more flexible and efficient parallelization of existing code base. Finally, recent progress in deep learning frameworks such as PyTorch can enable programming GPUs in Python and provide GPU-optimized library functions useful to EDA. For example, DREAMPLace [5] achieved a 40x speedup on VLSI global placement leveraging a number of optimization functions available in PyTorch.

In summary, the objective of this contest is to develop a GPU-accelerated, timing-aware, 4 value (0, 1, x, z) logic simulator for replaying RTL traces on gate-level netlists. In this contest, we will provide industrial-scale open-source benchmarks to evaluate contestants' code. We also will provide cloud GPU instances with NVIDIA T4 GPUs and a rich software stack. We expect contestants to leverage advanced GPU features to achieve significant speedup over existing CPU based simulators.

Problem Formulation

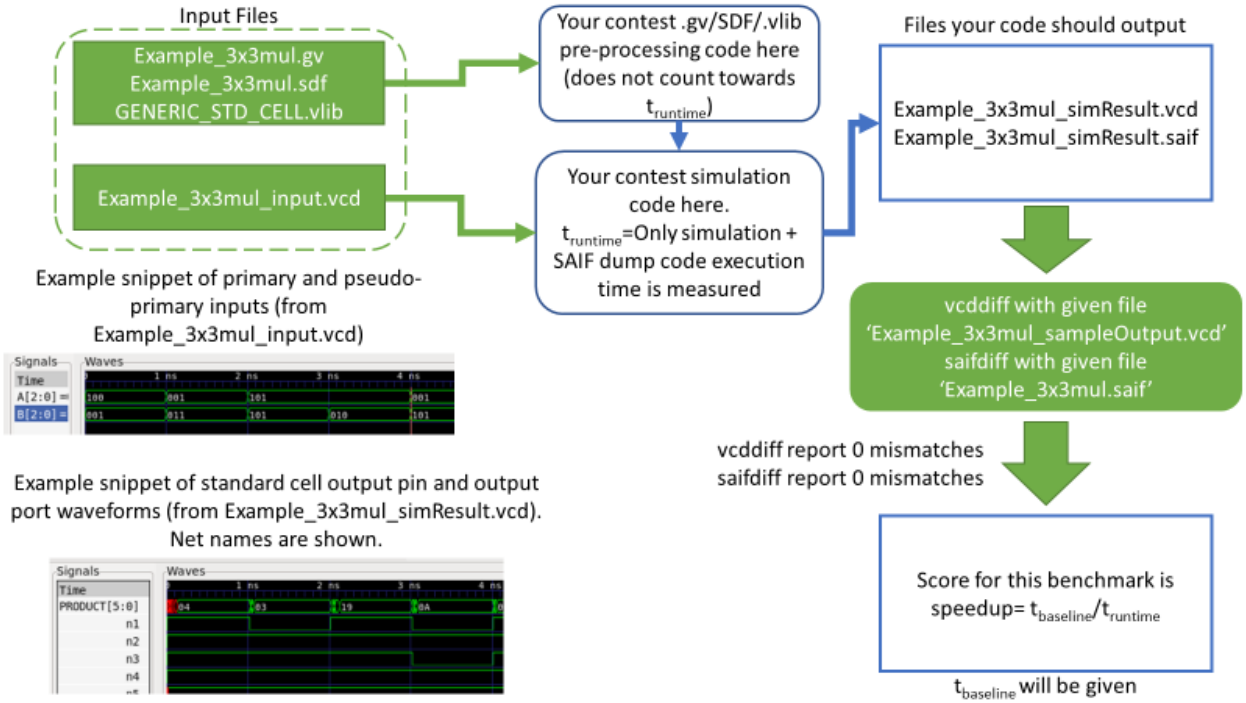

Figure 1: Flow chart of program requirements and contest participation.

Figure 1 shows a flow chart of program requirements and the contest problem definition. Each *benchmark* consists of a gate level Verilog *design* and one or more corresponding *testbenches* for that design. Contestants will have as input files:

1. A gate level netlist description of the *design*.
2. An SDF file [14] that describes the delays of each gate in the *design*.
3. The *testbench(es)* to be simulated, which is stored in an input VCD [6] file. The VCD file contains waveforms of the primary and pseudo-primary inputs of the *design* for the duration of the *testbench*.
4. A standard cell library .vlib file, which describes the behavior of each standard cell gate in the *design*.

From these input files, contestants are asked to write code to participate in the contest that will:

1. Read the gate level netlist .gv, SDF, and .vlib file and translate them into an intermediate representation format that can be easily acted upon by the ensuing simulator code. This portion of the code will not be timed for purposes of scoring in the contest.
2. Read, parse and interpret the simulation input stimuli VCD file.
3. Perform a delay-annotated, 4-value (0, 1, x, z) simulation on the *design* so that the output pin waveforms of the standard cell gates are calculated (as a hint, sequentials such as registers and RAMs will not need to be simulated, their waveforms are already given), and
4. Outputs a SAIF file [13] that contains the time nets were of value 0, 1, x, or z (T0, T1, TX, TZ) for all nets in the *design* for the duration of the specified timestamps (for example, the *benchmark* specification document that will be given may require you to record signal values from timestamp 5-5000ps) given for the *testbench*. For simplicity's sake, we only require the sum of TX and TZ to be correct.
5. (For your own debugging purposes only). It is highly recommended that your code has an option to output a VCD file that contains waveforms for all nets in the *design* for the duration of the specified timestamps (similarly, the *benchmark* specification document will give the timestamps) for the golden given VCD file for the *testbench*.

The code may be written in any language, for example including, but not limited to, C/C++, CUDA, or Python. However, keep in mind that the code will be run on the contest-supplied GPU platform for evaluation. The runtime of the code for each *benchmark* will be measured as $t_{runtime}$, which includes the time from step 2 to step 4 described above, but does not include the time for preprocessing in step 1, nor includes the time to dump the VCD file in step 5. The output SAIF file should include signal values for all nets in the designs and will be compared using contest supplied saifdiff script to a given golden reference SAIF file. If saifdiffs do occur, one way to debug may be to output a VCD file which includes waveforms for all nets in the design and compare using vcddiff [7] to a supplied golden reference VCD file. Saifdiff must report 0 mismatches (unless waived by the contest organizers for odd corner cases) between the generated output file and the golden reference file, which signifies a successful logic re-simulation. Specific timestamps will be given specifying which time durations each output file should record. The contest output file formats are set up this way to limit the potential burden on file IO time while retaining proper checkers for correct re-simulation. Contestants are encouraged to debug potential waveform mismatches using open source waveform viewers such as gtkwave [8]. Finally, the score for each *benchmark* is recorded. The contest will supply a $t_{baseline}$ runtime, which signifies the baseline runtime needed to run the *benchmark* using conventional simulator tools. The score is simply the speedup $t_{baseline}/t_{runtime}$.

The delays for each gate are stored in the input SDF file for the *design*. The delays from 'specify' statements in the GENERIC_STD_CELL.vlib are only to describe the behavior of the standard cells, and should not be considered literally if the delay arc occurs in the SDF file. While IEEE Standard Delay Format (SDF) provides a rich syntax to describe the delays of each gate in a *design,* the delays we consider and display in the SDF files for this contest are described using a subset of the full SDF syntax. Specifically, contest-supplied SDF files will only include the ABSOLUTE and IOPATH keywords for consideration. The ABSOLUTE keyword simply means the numerical delay values recorded should be interpreted literally, while the unit for delay is specified in the header section of the SDF file (TIMESCALE keyword). IOPATH keyword signifies the related timing arc of the standard cell being described. In addition, we simplify the SDF file by not differentiating between minimum:typical:maximum timing values in the SDF triples. An

example on the process of interpreting SDF files is given in Figure 2. We will not disqualify contestants for mismatches in simulation arising from ambiguities in interpreting the SDF standard.

File Format Clarifications

Figure 3. provides additional clarification on interpreting the SDF files and how contestants' code should process them. In the case where separate input pins change **at the same time** causing an output pin transition, the smallest of the valid (all contributing transitions, regardless if the input pin transition results in a controlling value or not) IOPATH delays is taken during the simulation. In the case where multiple input pin value changes will result in conflicting output pin values, an ordering of the in-flight output pin value changes must be done before determining the order of value changes that occur on the output pin. In some extreme cases, glitches may be 'eaten' and not seen on the output waveform at all, as depicted in Figure 3. Otherwise, no glitch filtering is done. In some cases, some cells will have a delay of 0 in the SDF file. While this will result in no delay from input to output pin, keep in mind a delta delay exists between input and output pin, which may affect the ordering/scheduling of transitions.

Figure 4. provides clarification on the required SAIF output file format. Our contest will only require the recording of T0, T1, TX, and TZ times. For simplicity's sake, the contest provided saifdiff program will only require the sum of TX and TZ times to be equal between golden reference and simulated output SAIF to be considered correct. Though the golden reference SAIF files will also have TC and IG values recorded, our contest supplied saifdiff does not compare these values, and contestants are encouraged not to dump them in their generated SAIF file. It is suggested to follow the header format for the SAIF file, especially recording the TIMESCALE and DURATION keyword value.

Benchmark Suite

Benchmarks are derived from *designs* of different configurations of the NVDLA open source project [9] and different configurations of the RISC-V [10] open source project, and open source IWLS2005 designs[11] (possible hidden benchmark candidates). The purpose of sourcing from several different projects is to provide a range of small to large designs that also constitute a wide range of different activity factors and inclusion/exclusion of x/z values during simulation of their *testbenches*. The contest suggests contestants use smaller *designs* and shorter *testbenches* for code exploration and validation, while larger *designs* and longer *testbenches* should be used for code evaluation. Some *benchmarks* will be hidden from the contestants but will be used during the contest's final evaluation and scoring to encourage contestants to design a 'universal' type simulator capable of handling a fully diverse set of *benchmarks*, instead of designing their code to perform well only on the *benchmarks* given.

Benchmark design gate level netlists are created from open source RTL, and synthesized with a commercial gate synthesis tool using a generic standard cell library that has no attachment to any real world technology. Timing targets have been gratuitously relaxed so timing checks do not need to be a part of this contest problem. *Testbench* input VCD, as well as golden reference VCD and SAIF files are created by simulating the provided open source *testbenches* related to the *benchmarks* using a commercial Verilog simulator and dumping the corresponding results into the VCD and SAIF files.

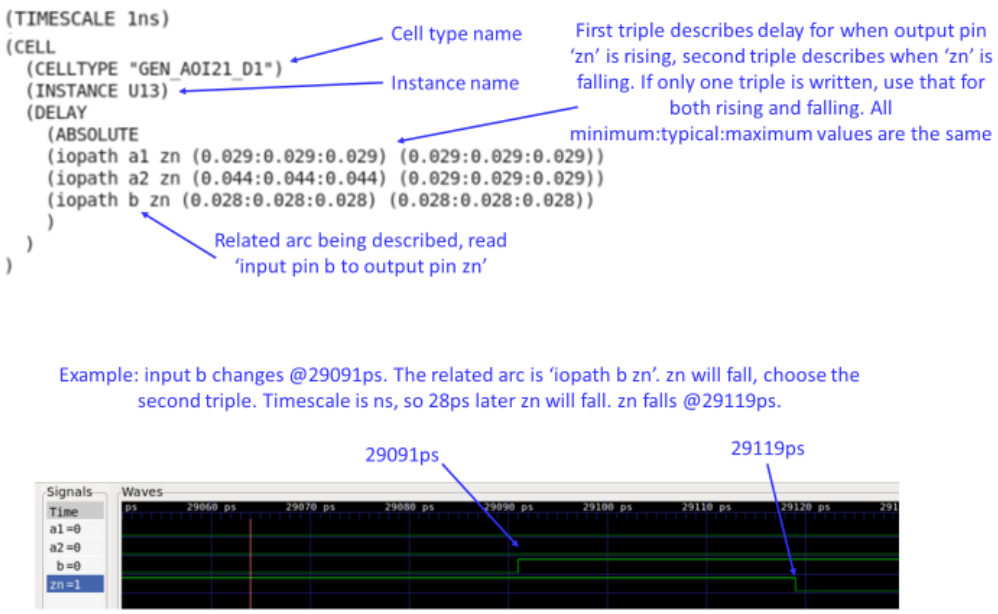

Figure 2. Example interpretation of SDF file and glitch enabled simulation results.

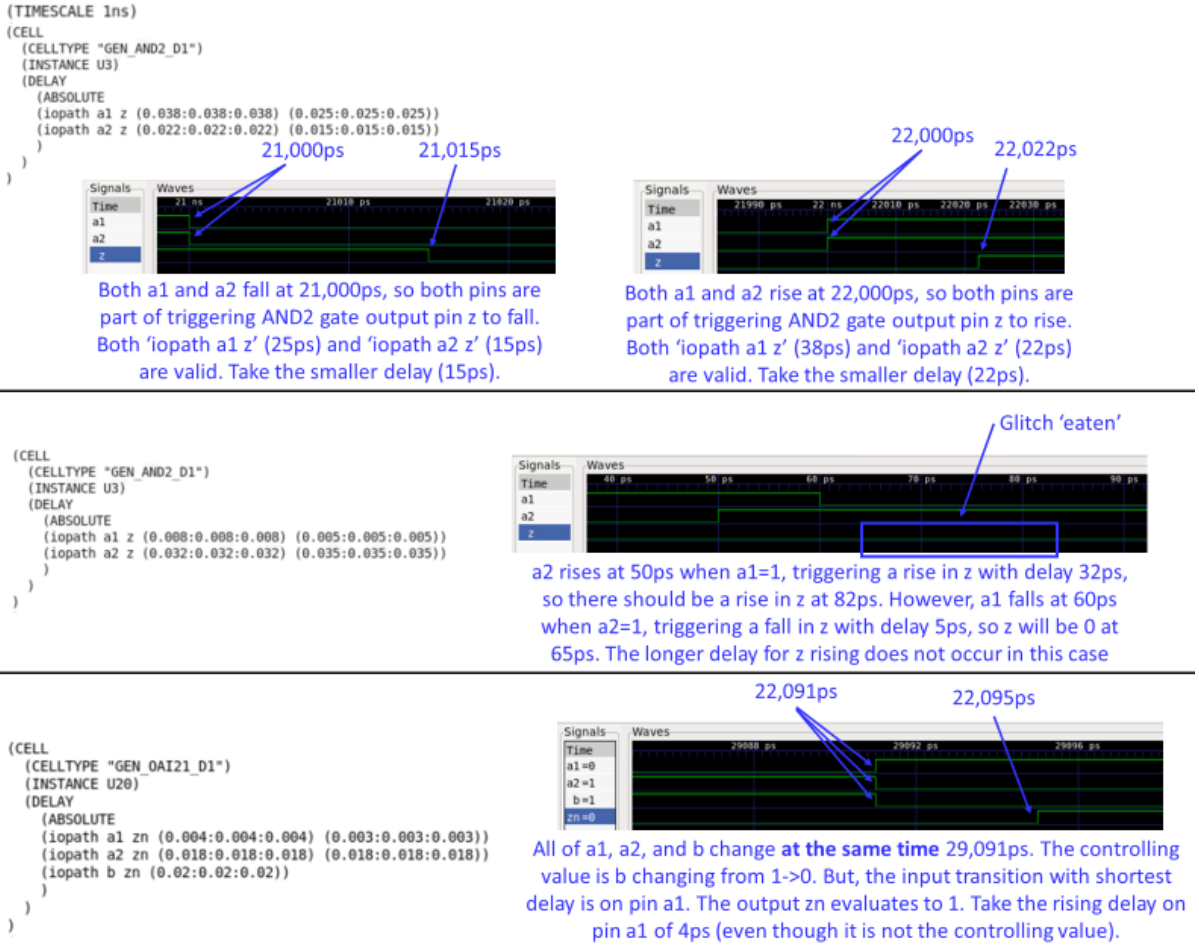

Figure 3. Additional clarifications regarding SDF file interpretation.

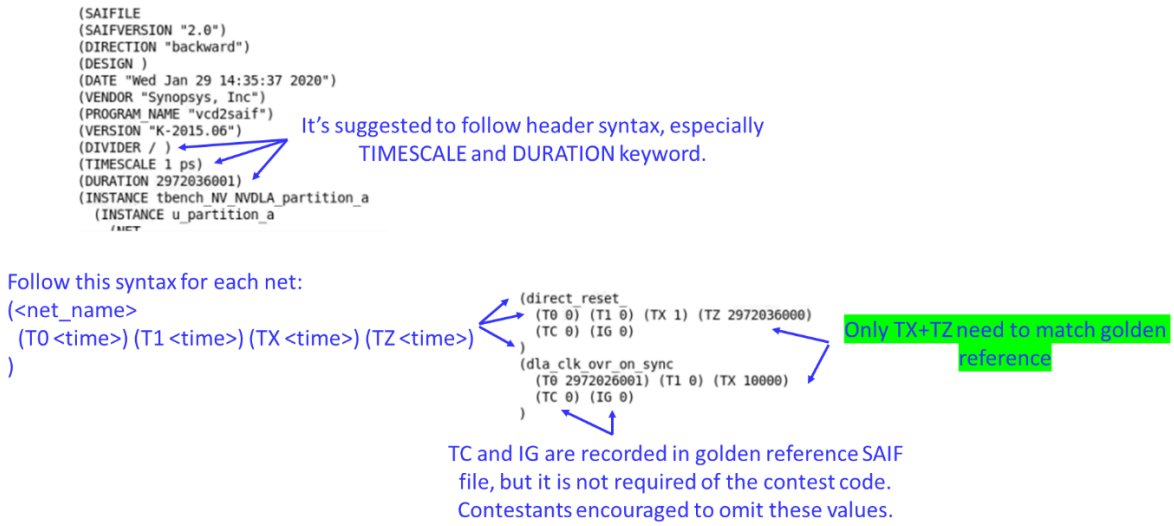

Figure 4. Clarification on required SAIF output file format.

For this contest problem, the benchmarks will be made available for download from a Google drive repository link in[15]. The repository will also provide documentation on the timestamps needed to correctly produce the output SAIF files, instructions on how to install utility programs such as vcddiff and gtkwave, $t_{baseline}$ and other *benchmark* statistics, the contest python saifdiff program, and instructions on how to decompress the *benchmark* files. Submitted code will be evaluated using Amazon EC2 G4 instances, powered by NVIDIA T4 GPUs. After finishing their contest registrations, competing teams who need access to GPUs will be provided with an allocation of credits for using Amazon EC2 G4 instances and AWS Cloud Storage Services to develop and test their code. Competing teams will be responsible for managing the spend of their credit allocation on compute instances and persistent storage during the contest. Additional documentation on setting up and configuring Amazon EC2 G4 instances will be provided to the registered teams before the start of the contest, and can also be found in the repository.

Evaluation

The submitted code must produce output SAIF files across all *benchmarks* that report 0 mismatches across all golden reference SAIF files compared. Submission code which contains mismatches when run during evaluation will unfortunately be disqualified and will not be given a score. If this pre-requisite is met, each contest submission's final score will be determined by taking the geometric mean of each individual *benchmark* score, as described below. The total benchmark suite will include both released *benchmarks* to the contestants and some hidden *benchmarks* for evaluation only. Submissions that do not utilize GPUs will not be evaluated.

$$Each\ benchmark\ score = \frac{t_{baseline}}{t_{runtime}}, final_{score} = \sqrt[n]{\prod_{i}^{n=number\ of\ benchmarks} score_i}$$

The required code submission format is detailed in Figure 5. The preprocessing script for .gv/SDF/.vlib takes in file names as command line arguments and outputs an intermediate representation to be taken as input to the simulator code. The simulator code also takes as input the file name for the input waveform VCD, and command line variables that signify the start and stop times (in picoseconds) for signal recording/dumping for SAIF files. Only the running of the simulator code ("GPUSimulator.exe" in Figure 5) will be timed.

```
GraphPreprocessing.exe <netlist.gv> <netlist.SDF> <std_cells.vlib>
[intermediate_representation.file]
#Preprocessing script of gate level netlist, SDF, and standard cell .vlib takes
the files for each as input, outputs some compiled intermediate representation.
As an example, code can be in CUDA or Python.
GPUSimulator.exe <intermediate_representation.file> <input.vcd> <dumpon_time>
<dumpoff_time> [SAIF.saif]
#Actual execution of simulator code. Execution of this code will be timed. The
code should dump an output SAIF file.
```

Figure 5. Desired code submission format

Contestants may be required to submit code with executables that have different names than the example naming given in Figure 5. Regardless of which coding language was used to build/compile the code, submissions of code should be in binary executable format which take the above described variables as command line variables.

Conclusion

Logic re-simulation is an important industrial task that has potential for speedup through parallelism exploitation through parallel processing hardware platforms such as GPUs. We hope our ICCAD-2020 contest problem spurs interest in re-thinking this important EDA task, and perhaps sheds light on accelerating standard RTL and gate level simulations with GPU as well. Finally, we look forward to the contestants' creative solutions and thank them for their participation.

Reference

[1] Debapriya Chatterjee, Andrew DeOrio, Valeria Bertacco, "GCS: High-Performance Gate-Level Simulation with GP-GPU", DATE, 2009

[2] https://www.deepchip.com/items/0523-04.html

[3] Stefan Holst, Michael E. Imhof, and Hans-Joachim Wunderlich, "High-Throughput Logic Timing Simulation on GPGPUs", TODAES, 2015

[4] Stephen Jones, "CUDA New Features and Beyond", GTC, 2019

[5] Yibo Lin et al., "DREAMPlace: Deep Learning Toolkit Enabled GPU Acceleration for Modern VLSI Placement", DAC, 2019

[6] https://en.wikipedia.org/wiki/Value_change_dump

[7] https://github.com/veripool/vcddiff

[8] http://gtkwave.sourceforge.net/

[9] https://github.com/nvdla

[10] https://github.com/riscv

[11] https://iwls.org/iwls2005/benchmarks.html
[12] https://www.nvidia.com/en-us/data-center/tesla-t4/
[13] IEEE/IEC International Standard - Design and Verification of Low-Power Integrated Circuits," in *IEC 61523-4 Edition 1.0 2015-03 (IEEE Std 1801-2013)*, vol., no., pp. 303-351, 24 March 2015
[14] IEEE Standard for Standard Delay Format (SDF) for the Electronic Design Process," in IEEE Std 1497-2001 , vol., no., pp.1-80, 14 Dec. 2001
[15] https://drive.google.com/drive/u/2/folders/1IrWXkHEED_gVsLPUGrIKNIOAE6BIp0WY