IEICE TRANSACTIONS

on Fundamentals of Electronics, Communications and Computer Sciences

DOI:10.1587/transfun.2024VLL0001

Publicized:2024/10/08

This advance publication article will be replaced by the finalized version after proofreading.

A PUBLICATION OF THE ENGINEERING SCIENCES SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

LETTER *Special Section on VLSI Design and CAD Algorithms*

Multithread Implementation of Open Source Library Characterizer

Shinichi NISHIZAWA[†a)], **Masahiro MATSUDA**[††], ***Members***, ***and*** **Shinji KIMURA**[†], ***Fellow***

SUMMARY This paper reports our open source cell library characterizer which can generate timing models and power models of standard cell library and its multitread implementation. Implementation results show 14.5 times speed-up from single-thread operation.

key words: *Open source VLSI design, Timing library, Characterizer*

1. Introduction

The progressive scaling of CMOS transistor fabrication technology and advanced technology in EDA tool-chain achieves faster operation speed, lower energy consumption and better area density of the state-of-the-art VLSI designs. However, only the limited company and academy can access this state-of-the-art environments to design the state-of-the-art VLSI chips. It is difficult to access modern VLSI design environment for designers in the other company, academy or personal designers.

On the other hands, several open-sourced VLSI design projects have been proposed [1], [2]. Many open EDAs are widely proposed and these EDAs enable the open-source VLSI design projects. However, to our knowledge, it lacks a leading library characterizer to extract timing and power of standard cells to enable precise timing and power estimation using Static Timing Analysis (STA). Several characterizers have been proposed include the previous work of this research [3][4][5][6][7], however they lacks the support for multithread simulation thus it uses huge computation time to the generate timing library.

This paper introduces the extension of our open and free timing characterizer, named libretto. libretto uses one of the major free spice simulator ngspice [8] for timing and power simulation. libretto automatically generates several spice files, launches spice simulation, accumulates simulation results and exports timing and power information as Synopsys Liberty format [9], which is the widely used industry standard open format of timing and power description. This version of libretto supports multithread simulation to speed-up the computation time utilizing the multi-core environment. Note that libretto does not have any advantage to the commercial characterizer, but it has basic but similar functionality of commercial characterizer for free, and accelerate open-source VLSI designs.

The rest of this paper is organized as follows. Section 2 describes related works on this field. Section 3 describes the overview of multithread implementation our characterizer. Section 4 describes the selection strategy and detail of multithread implementation. Section 5 describes the experimental results. Section 6 concludes this paper.

2. Related Works

Several works include technical papers and open-source repositories are available for non-commercial timing and power characterizing of standard cell library. Technical papers are already explained in our previous works [10][11]. Here we explain several open source-repositories of characterizer.

LibreCell(lctime) [6] is an open source characterizer which supports timing extraction for both combinational cell and sequential cells. It uses “template .lib” file to generate target .lib file. Template .lib file contains library common settings (unit for voltage, current, power, time, input/output voltage thresholds). LibreCell(lctime) supports ngspice and does not need a help of any commercial EDA tool. Disadvantage is the huge CPU time because of the lack of the support of multithread simulation.

CharLib [7] is an open source characterizer, which repository is generated after the release of libretto. CharLib uses YAML to configure the conditions for cell characterization. CharLib also does not support of multithread simulation.

This work, named libretto, is a free and open-sourced characterizer. libretto is implemented using Python3, and ngspice is selected for simulation engine, running on Linux system. It supports to characterize both combinational and sequential cells. The newest version of libretto supports multithread simulation so it can gain more computation speed than single-thread simulation. Also libretto supports document generation in Markdown format, and it can be converted to PDF by Pandox [13].

3. Overview of characterization

Figure 1 shows the overview of multithread characterization in libretto. Improved parts from [10][11] are colored in red. Designers need to prepare following files:

- netlist of target cells, and transistor model,

†Graduate School of Information, Production and Systems, Waseda University.
††Logic Research, Co., Ltd.
a) E-mail: nishizawa@aoni.waseda.jp

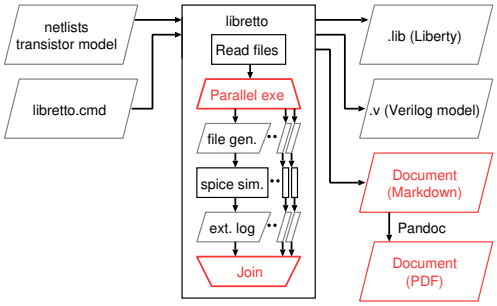

Fig. 1 Flow overview of libretto. Red parts has been added from previous.

- command file which contain characterization settings for library and individual logic cells.

libretto reads command file, generates several threads to perform spice simulation, reads spice outputs, and writes out as Liberty format. libretto also outputs Verilog-HDL model for gate level simulation, and specification of characterized library in Markdown.

Figure 2 shows the internal flow of libretto. First, second and third stages are the same as the previous. First stage defines the library common settings like unit of voltage, current and powers. Second stage defines the PVT (process voltage temperature) condition and logic threshold for delay definitions. Both settings are common for whole library. Third stage branches depend on the target cell function (combinational or sequential). In the third stage, individual settings for each logic cell are defined. Each cell require to set its spice subcircuit name, logic function, input/output pin names, and input slope/output loading conditions. Additionally, sequential cell requires clock, set and reset pin information.

In forth stage, spice files of target cells are generated and spice simulation is invoked. Red parts have been introduced to support multithread simulation. libretto launch several threads with different input slew and output loading conditions. For the combinational cells, each thread performs delay simulation (extraction of propagation delay, transient delay, start point of input waveform and end point of output waveform) then perform power simulation (extraction of leakage and dynamic power).

Sequential cell needs two stage simulation. First stage is the setup-hold simulation. We have two definition of setup; (1) Data-to-Clock (D2C) delay at minimum plus 5 to 10% delay of Clock-to-Q (C2Q) delay, and (2) D2C delay at minimum DATA-to-Q (D2Q) delay [12]. We select later definition explained in textbook [12], that searches minimum D2Q delay then decompose it to D2C delay as setup, and C2Q delay. Based on the obtained setup time, hold time is calculated. This setup-hold search needs to find a minimum value thus it need iterations. Dynamic and leakage power consumption are also extracted at this stage. Second stage is the recovery-removal simulation if the target sequential cell has asynchronous set and reset terminals, which is already implemented in [10]. Recovery-removal can be implemented with a similar function of setup-hold search. Recovery is the

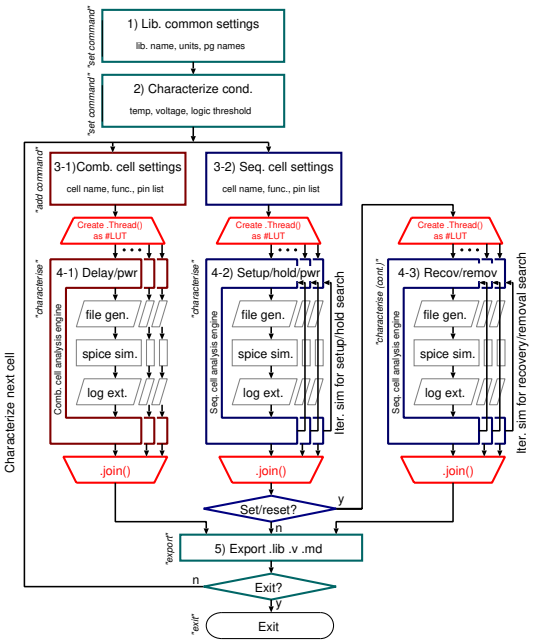

Fig. 2 Internal flow of libretto. Red parts has been added for multithread simulation.

minimum allowable time between the edge of set/reset to inactive state of clock signal. Removal is the minimum allowable time between the clock signal at the active set/reset to the edge of inactive set/reset. libretto search recovery constraint sweeping the signal edge of set/reset pin to find the minimal recovery time which succeed the recovery operation after set/reset. After recovery is found, libretto search removal constraint.

4. Detail of multithread implementation

We use a threading module of Python3 to implement multithread operation of characterization. There are several possible ways to apply multithread in characterization.

Cell-level parallelism simulates different cells at the same time.

Input-level parallelism simulates same cell but different input conditions at the same time.

Condition-level parallelism simulates the same input conditions but different input slopes and output loadings at the same time.

Loop-unrolling unrolls a loop for the setup-hold search and recovery-removal search.

We decided to use condition-level parallelism for achieving better potential parallelism, since modern library has large number of indexes for input slew and output loadings (e.g. load index 7 $\times$ slew index 7 = 49) . It might be exceeds the available CPU cores, however ngspice does not need any license thus we accept this inefficiency. Cell-level parallelism is not so efficient for libraries with small number of cells. Input-level parallelism is not efficient for cells with a few number of inputs. Loop-unrolling can be used only

Table 1 Characterization setting.

Characterizer	libretto	PrimeLib
Simulator	ngspice	hspice
# of threads	1, 49 (# of index)	32, 64
Process tech.	Commercial 180-nm	
Waveform	ramp	
Process	Typical	
Voltage	1.8 V (Nominal)	
Temperature	25°C	
Input slope	0.1 ns, 0.2 ns, 0.4 ns, 0.8 ns, 1.6 ns, 3.2 ns, 6.4 ns	
Output load	0.01pF, 0.02pF, 0.04pF, 0.08pF, 0.16pF, 0.32pF, 0.64pF	
DUT	INVx1, NAND2x1, NAND3x1, NAND4x1, NOR2x1, NOR3x1, NOR4x1, Flip-Flop w/ pos. edge clock	

for the sequential cells. Combinations of these parallelisms are the best solution for characterization speed. Commercial characterizer PrimeLib utilize these parallelisms with Distributed Processing management system. This is very powerful, however hard to implement since it need to decompose all of loops and manage all of jobs by ourselves. To balance the parallelism and ease of implementation, condition-level parallelism is selected for libretto.

Code 1 shows the internal description of libretto to utilize multithread simulation based on condition-level parallelism. We use simple and straightforward method for our implementation. Dictionary type array are prepared to store the simulation result before the multithread operation. It starts from the two nested loops of input slope and output loading, and *Thread()* object [14] has been called inside the loops. After that, *start()* object has been called and it invoke a method *runSpiceCombDelaySingle()* to generate spice file, which runs spice simulation and accumlates the result. All of the thread will be joined at *join()* method. When all simulation finishes, simulation results are extracted from the dictionary type array and stored into another array type variable for .lib generation.

Example in code 1 shows the implementation for combinational cells, but a similar way can be used for sequential cells to achieve condition-level multithread simulation.

Since libretto uses *Thread()* method, all of the threads will be parallelized without any respect to the available CPU resources. This will be improved by utilizing *ThreadPoolExecutor()* class [15] to control the number of multithread operations.

Code 1 Multithread implementation

```
 1: import threading
 2: def runSpiceCombDelayMultiThread(targetLib, targetCell, tar-
    getHarness, spicef):
 3: # prepare dict
 4:     res_prop_io = dict()
 5:     res_trans_o = dict()
 6:     threadlist = list()
 7: # set target methods to multithread
 8:     for tmp_slope in targetCell.slope:
 9:         for for tmp_load in targetCell.load:
10:             thread = threading.Thread( [ target=runSpiceCombDelaySingle,
11:             args=(targetLib, targetCell, targetHarness, spicef, tmp_slope,
12:             tmp_load, res_prop_io, res_trans_o),
13:             name="%d" % thread_id ] )
14:             threadlist.append(thread)
15:             thread_id += 1
16: # start multithread operation
17:     for thread in threadlist:
18:         thread.start()
19: # join
20:     for thread in threadlist:
21:         thread.join()
22: # extract result
23:     thread_id = 0
24:     for tmp_slope in targetCell.slope:
25:         list_prop = [ ]
26:         list_tran = [ ]
27:         for tmp_load in targetCell.load:
28:             list_prop.append(res_prop_io [ str(thread_id) ] )
29:             list_tran.append(res_tran_io [ str(thread_id) ] )
30:             ....
```

5. Experimental Results

We implement multithread simulation on libretto using Python3 and threading objects, and ngspice is used as the simulation engine. We prepare several netlists of the cell from PDK which targets 180-nm process as an evaluation. We use Synopsys PrimeLib for the performance comparison of libretto. Table 1 shows the common settings for both characterizers. Inverter, NANDs and NORs are selected as a representative of combinational cells, and positive edge Flip-Flop is selected as a representative of sequential cell.

5.1 Effect of parallelism on characterization result

Here we explain how parallel implementation of characterization affect to the result. We compare the generated .lib from single-thread operation and multithread operation both by ngspice, but we cannot see any difference on the two .lib files. libretto with ngspice, hspice and PrimeLib with hspice show some difference that are already reported in our previous works [10][11].

5.2 Runtime comparison

We use Linux machine with AMD Ryzen 2900wx 32-core 64-thread CPU for characterization, which specification is summarized in table 2. Table 3 shows the CPU time for characterization. To characterize eight cells, libretto with ngspice in single thread simulation requires 57.9 hour in total. libretto with ngspice in multithread simulation consumes 4.00 hours which achieves 14.5× faster than single thread simulation. This gap of speed gain (14.5× speed up with 32-core CPU) comes from unbalance of the simulation conditions. Different slew and load condition need different simulation time. In condition-level parallelism, it need to wait for all simulations of the current cell to finish, before it

Table 2 Environment for characterization.

OS	CentOS 7.8
CPU	AMD Ryzen 2990wx 3 GHz 32-core, 64-thread
MEM	DDR4-2400 96GB ECC
SSD	3TB

Table 3 CPU time for characterization.

	libretto		PrimeLib	
Simulator	ngspice		hspice	
# threads	1	49 (# of index)	32	64
CPU time	57.9 hours	4.00 hours	92.0 seconds	106 seconds
Speed up	1 (baseline)	14.5×	2265×	1966×

characterize next cell.

libretto is also compared with the commercial characterizer PrimeLib with hspice on the same computation environment. PrimeLib with hspice in 32-thread simulation requires only 1.53 minuets and 156× faster characterization than multithread operation on libretto with ngspice. As reported before [10][11], libretto needs two-stage simulations to obtain timing and power information since ngspice does not support nested ".measure" statement available in hspice. Current version of libretto has pessimistic guard bands to accept various types of simulation conditions. libretto uses gradient method to search setup and hold but PrimeLib with hspice uses internal bisection search. These difference affects speed difference on libretto and PrimeLib. Reduction and optimization of these guard bands are our future work.

5.3 Scalability test

We test the scalability of our multi-thread implementation by different number of simulation threads. To remove the simulation condition difference, we use same Inverter cell with same slew (6.4 ns) load (0.64 pF) conditions for all 11×11 LUT. Calculate speed improvement by changing the number of threads.

Figure 3 shows the characterization speed improvement. From one to 32-thread, characterization speed increases as thread increases. 32-thread simulation achieves 21.8× speed up, and this result indicates resource conflict (memory, disk) affects the performance. If number of threads exceeds its physical cores, speed improvement become smaller.

Result shows our implementation achieves 21.8× speed improvement at 32-thread simulation if all of the simulation conditions are the same. In real workload, LUT has different slew and load conditions and it reduce the speed improvement to wait all of the simulation of current cell to finish and synchronize.

6. Conclusion

In this paper, we describe the multithread implementation of our open-sourced characterizer. It supports the characterization of both combinational cells and sequential cells. Current version supports the multithread operation and achieves

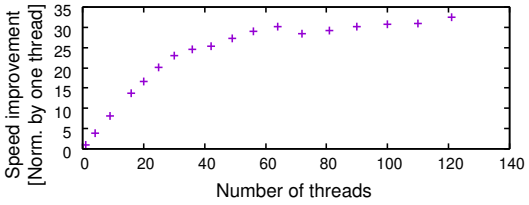

Fig. 3 Speed improvement by different number of threads.

faster characterization than single-thread operation. Experimental result show 32-thread operation of ngspice achieves 14.5 times faster than single-thread operation. 32-thread operation of PrimeLib with hspice is 159 times faster than libretto, however the characterization time of libretto is now becoming reasonable. Program code and environment is uploaded in author's GitHub page [16].

Our future work is performance improvement and accuracy improvement in characterization. Input waveform model should be improved from current simple ramp waveform to realistic driver model. Interdependency of setup time and hold time should be consider in the characterization to improve the accuracy of post-layout simulation and STA.

Acknowledgment This work is supported by Logic Research (190402, 210104), and partly supported by JSPS KAKENHI JP21K17723, VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc..

References

[1] "MakeLSI Project,"*https://scrapbox.io/makelsi/*
[2] R.T. Edwards, "Google/SkyWater and the Promise of the Open PDK," Workshop on Open-Source EDA Technology, 2020.
[3] G. Bronstein, et al., "Asic standard cell library design by graham petley.," 1991.
[4] M.T. Moreira, et al., "LiChEn: Automated electrical characterization of asynchronous standard cell libraries," Euromicro Conference on Digital System Design, pp.933–940, 2013.
[5] C.H. Oliveira, et al., "ASCEnD-FreePDK45: An open source standard cell library for asynchronous design," International Conference on Electronics, Circuits and Systems, pp.652–655, 2017.
[6] "lctime," *https://codeberg.org/librecell/lctime*
[7] "CharLib," *https://github.com/stineje/CharLib*
[8] "Ngspice," *https://ngspice.sourceforge.io*
[9] Synopsys, Liberty User Guide Volume 1.
[10] S. Nishizawa and T. Nakura, "Library characterizer for open-source VLSI design," The Workshop on Open-Source EDA Technology, 4–pages, 2022.
[11] S. Nishizawa and T. Nakura, "libretto: An Open Cell Timing Characterizer for Open Source VLSI Design," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.E106-A, no.No.3, pp.551–559, 2023.
[12] N.H.E. Weste and D.M. Harris, CMOS VLSI Design, 4 ed., Addison Wesley, 2010.
[13] "Pandoc," *https://pandoc.org*
[14] "docs.python.org: threading — Thread-based parallelism," *https://docs.python.org/3/library/threading.html*
[15] "docs.python.org: concurrent.futures — Launching parallel tasks," *https://docs.python.org/3/library/concurrent.futures.html*
[16] "libretto," *https://github.com/snishizawa/libretto*