LEX - A Cell Switching Arcs Extractor: A Simple SPICE-Input Interface for Electrical Characterization

Rodrigo N. Wuerdig, Vitor H. Maciel, Ricardo Reis, and Sergio Bampi

Graduate Program in Microelectronics (PGMICRO) - Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre - Brazil
{rnwuerdig, vhsfmaciel, reis, bampi}@inf.ufrgs.br

2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) | 979-8-3503-2769-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISVLSI59464.2023.10238671

***Abstract*—The characterization of logic cells is a critical step in the design of digital circuits. Existing open-source cell characterization tools typically require significant extra information beyond the SPICE netlist. In this paper, we present a new open-source tool - LEX - that serves as a very useful interface for these characterization tools, enabling the extraction of essential input and output information, Boolean expressions, truth tables, and transition (switching) arcs directly from the SPICE netlist. Our LEX tool offers several advantages over existing open-source methods. First, it simplifies the cell electrical characterization process by eliminating the need for manual input of additional information. This saves time and reduces the incidence of errors. Second, our tool provides a more comprehensive set of information than existing tools, including Boolean expressions and truth tables. Third, LEX is highly flexible and can be integrated with a wide range of existing open-source cell characterization tools. We conducted experiments using a test set of netlists to demonstrate LEX effectiveness. By providing a more comprehensive set of information, eliminating the need for manual input of additional information, and improving efficiency, our tool offers a powerful new option to be integrated into already existing and future open-source characterization tools.**

***Index Terms*—VLSI, EDA, Cell-Based Design, Characterization, Microelectronics**

I. Introduction

The observation made by Gordon Moore in 1965 has correctly predicted the evolution of integrated circuits (ICs) for nearly six decades. Moore's law predicted that the number of transistors that can be packed onto a single chip doubled approximately every two years. This trend towards increasing transistor density on ICs has been largely made possible by advances in lithography and the ability to manufacture even smaller transistors. As a result of these advances and huge investments, modern ICs can integrate tens or hundreds (in flash EEPROMs) of billions of transistors onto a single chip [1].

Considering the high number of transistors in modern circuits, making a full-custom chip is challenging for today's technology. The bulk of the industry relies on cell-based digital designs, also known as semi-custom design (see Fig.1), to produce circuits with noticeably shorter time to market (TTM) and better yields. The cell-based design consists of a group of D-cells with different logical functions and driving strengths called standard D-cells. D-cells are extensively characterized, optimized, and made robust to PVT variations - i.e. fully validated - to secure the assured functionality of fabricated large digital blocks.

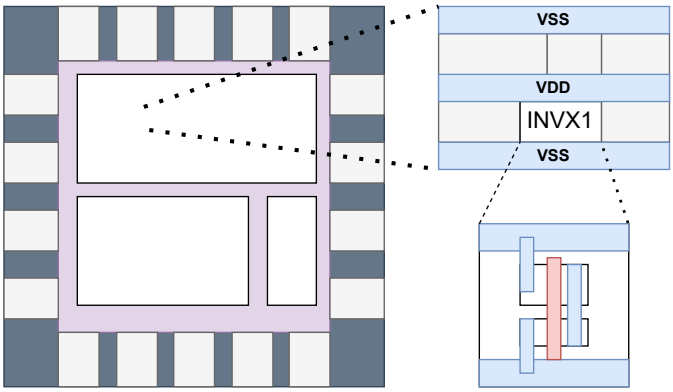

Fig. 1. Representation of a cell-based design.

There is a breed of electronic design automation (EDA) open tools that can be traced back to the early 1980s, like the MAGIC VLSI layout tool, or earlier, as SPICE electrical simulators. Recent years have witnessed an increasing trend in the adoption and creation of open EDA tools. This trend can be attributed to several factors, including the OpenROAD [2] project and the integration of different open-source tools by the OpenLANE [3] project. The open-source community has embraced developing integrated circuits without relying solely on commercial EDA tools. Additionally, open-PDKs are essential to provide a path to tape-out integrated circuits in selected fabs. Since the public availability of the Skywater 130nm PDK, more engineers have been enabled to experience IC design practice at a higher proficiency level, through contests aimed at being part of a Multi-Project Wafer (MPW) [4] [5].

One critical step in enabling cell-based design is to accurately characterize the behavior and electrical characteristics of each digital cell under defined process, voltage, and temperature (PVT) conditions. This characterization process results in a Liberty file ($< *.lib >$), which contains information about the electrical and timing properties of each cell, as well as their operating conditions. These known characteristics, such as voltage levels, current re-

979-8-3503-2769-4/23/$31.00 ©2023 IEEE

quirements, and timing parameters, are used as inputs in subsequent design steps during the elaboration of digital circuits. By using a library of well-characterized standard cells, designers can achieve predictable and reliable circuit performance, allowing for large-scale integration of complex digital circuits. The accuracy and completeness of the characterization process directly impact the quality of the Liberty file and ultimately affect the performance and functionality of the final digital circuit. Therefore, a thorough and rigorous cell characterization process is crucial for the success of cell-based digital design.

A. Motivation

In the literature, a variety of cell characterization tools are present. Libretto is an open-source characterization tool designed by Shinichi Nishizawa and Toru Nakura [6], which uses *ngspice* and supports the characterization of both combinational and sequential cells. Libretto still requires the user to specify the cell logic function, the input and output pins, the input values, and the expected outputs.

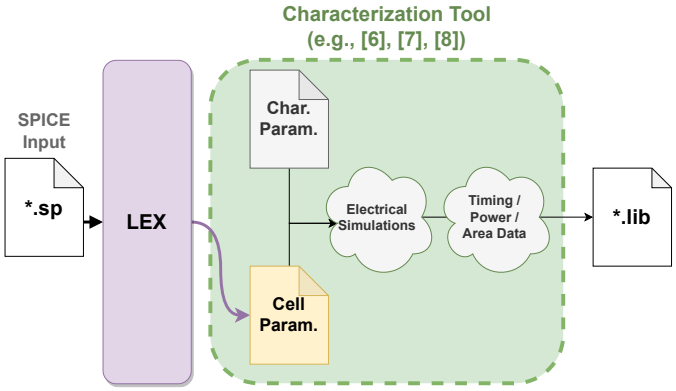

Fig. 2. LEX Application Usage

The ASCLIC tool [7], developed by bin Hussin et. al, and tested with Silterra 180nm and 130nm standard cell libraries, can characterize multi-inputs and multi-outputs combinational cells as well as partial sequential cells. Furthermore, it improves runtime performance by taking advantage of multi-core processing. However, it needs the cell function as an input inside its library configuration file.

Some cells commonly used in asynchronous circuits, such as the Muller C-element, exhibit unique characteristics such as feedback loops. In a Muller C-element, the output is directly fed back into the input expression, causing a circular dependency that can lead to issues in conventional characterization tools. Such tools may become trapped in a loop while trying to determine the input-output behavior of the cell, resulting in inaccurate or incomplete characterizations. LiChEn's advanced algorithms and modeling techniques are specifically designed to handle such complex cells, allowing for accurate and efficient characterization of cells with feedback loops [8].

LiChEn, like other characterization tools, needs prior information on the cell before starting the characterization process. This information is defined in a reserved line in the SPICE description and describes if each pin is either an input or an output.

Some extensions were also presented in the literature to aid the characterization process. Beg et. al presented a web-based collaborative platform [9] to ease the characterization of standard cells on teams that are in different geographical locations.

Those open-source characterization tools and extensions usually require lots of user inputs. Among these inputs, it is mostly required the input the Boolean expression and the assignment of input and output pins. This paper proposes an interface for characterization tools with minimal user input, as shown in Fig. 2. Each characterization tool has its requirements in terms of inputs. Usually, these requirements can be split into two main categories: I) characterization/technology parameters; II) Cell parameters. The characterization parameters can be either the output capacitance for each drive strength, input slew rates, supply V_{DD}, etc. This is usually set only once for a whole technology library. The cell parameters, however, change according to each cell, or logical function. These parameters can be the logical expression, inputs, outputs, etc.

Figure 3 depicts the tool steps. This tool parses the SPICE description file, creating two vectors of transistors, one responsible for the PMOS network, while the other is responsible for the NMOS network. The environment provides a transistor class with several attributes, like drain, source, gate, and bulk connections. The following steps performed by the tool are covered in the next sections. Sec. II, covers the automatic reconnaissance of the pin types. Sec. III shows the boolean expression extractor. Sec. IV the functionality of the boolean solver. Sec. V how the tool iterates through the input possibilities and outputs each of them. Sec. VI depicts the algorithm that finds each switching arc. Sec. VII displays the results, and Sec. VIII draws the main conclusions of this work and talks about possible future works.

II. Finding Inputs, Outputs, and Common Nets

During the parsing phase, a vector of pins is created using the information of the ".subckt" line, as shown in the SPICE description example on Listing 1. There is no information about the direction of each pin on the spice description. To find the direction, three other vectors were created: one for the inputs, one for the outputs, and the last for the common nets between the PDN and PUN.

```
1 .SUBCKT NAND3D0BWP A B C OUT VDD VSS
2 MP0 VDD A OUT VDD PCH W=2.5E-06 L=0.04E-06
3 MP1 VDD B OUT VDD PCH W=2.5E-06 L=0.04E-06
4 MP2 VDD C OUT VDD PCH W=2.5E-06 L=0.04E-06
5 MN0 OUT A rand0 VSS NCH W=2.5E-06 L=0.04E-06
6 MN1 rand0 B rand1 VSS NCH W=2.5E-06 L=0.04E-06
7 MN2 rand1 C VSS VSS NCH W=2.5E-06 L=0.04E-06
8 .ENDS
```

Listing 1. SPICE Netlist Example of a 3-input NAND

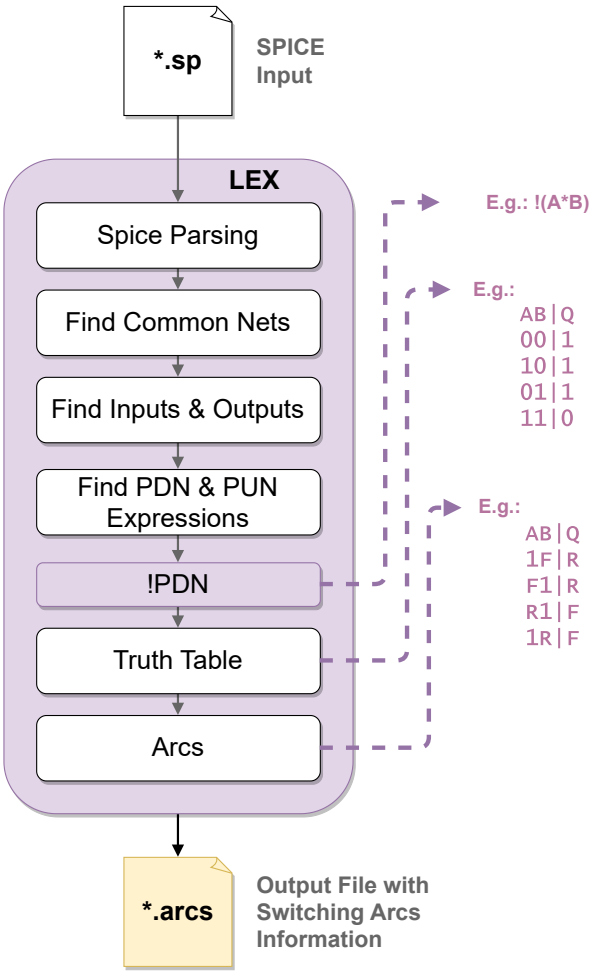

Fig. 3. LEX Flowchart

The subcircuit definition under the SPICE description, specifies the subcircuit name, power nets, inputs, and outputs. However, the definition of an input, output, or power net is not clear just by itself. To distinguish the pin category (if it is either input, output, or power pin), a code similar to the pseudo-code described in Alg.2 was created. Before the execution of the *DistrPins(&CNs, &$Pins_{IN}$, &$Pins_{OUT}$)* function, every pin (that could be either in or out) in the subcircuit line is included in the input pins list, or $Pins_{IN}$ in the pseudo-code. Then, with the common nets information given by the Alg. 1 presented earlier, the code can match if a common net is equal to one of these pins in the input. Every pin that is also a common net is likely an output, so, pins that match those criteria can be removed from the input pins vector and included in the output vector. Pins with a name similar to V_{DD}, V_{SS}, GND, etc., can be treated as power nets and outsourced to the power pin vector. The remaining pins in the input pins vector are likely inputs.

III. Finding the Boolean Expression

This section presents a pseudo-code algorithm for retrieving the Boolean expression from a Spice netlist. Various algorithms can be used to find the Boolean expression, and in our initial LEX implementation, we employ the transistor flattening method. This method is described in Alg. 3.

Our greedy algorithm iteratively combines parallel and series transistors, in that specific order, in an attempt to minimize the number of transistors in each network (PDN and PUN). When parallel or series transistors are found (e.g., T1 and T2), a new pseudo-transistor is included in the network (e.g., PT1), and both transistors T1 and T2 are removed from the network. The algorithm stops when the amount of transistors in each network is equal to the amount of common nets (common nets are defined in the previous section), or $SizeCN$ in Alg. 3.

Algorithm 1: Algorithm for Finding Common Nets

Input: $PDN_{transistors}$ $PUN_{transistors}$
Output: CommonNets

```
for TranN in PDN_transistors do
    for TranP in PUN_transistors do
        if (TranN.Source == TranP.Source) then
            CommonNets.append(TranN.Source)
        else if (TranN.Source == TranP.Drain) then
            CommonNets.append(TranN.Source)
        else if (TranN.Drain == TranP.Source) then
            CommonNets.append(TranN.Drain)
        else if (TranN.Drain == TranP.Drain) then
            CommonNets.append(TranN.Drain)
        end if
    end for
end for
return CommonNets
```

Algorithm 2: Algorithm for Classifying Pins

Input: .
CNs: Common Net Expressions we are looking for
$Pins_{IN}$: Input Pins Vector
$Pins_{OUT}$: Output Pins Vector

DistrPins(&CNs, &$Pins_{IN}$, &$Pins_{OUT}$)

```
for CN in CNs do
    for Pin in Pins_IN do
        if (CN == Pin) then
            Pins_IN = remove_pin(Pins_IN, Pin)
            Pins_OUT.pushback(Pin)
            if Pins_IN.size() == 1 then
                break
            else
                DistrPins(CNs, Pins_IN, Pins_OUT)
            end if
        end if
    end for
end for
```

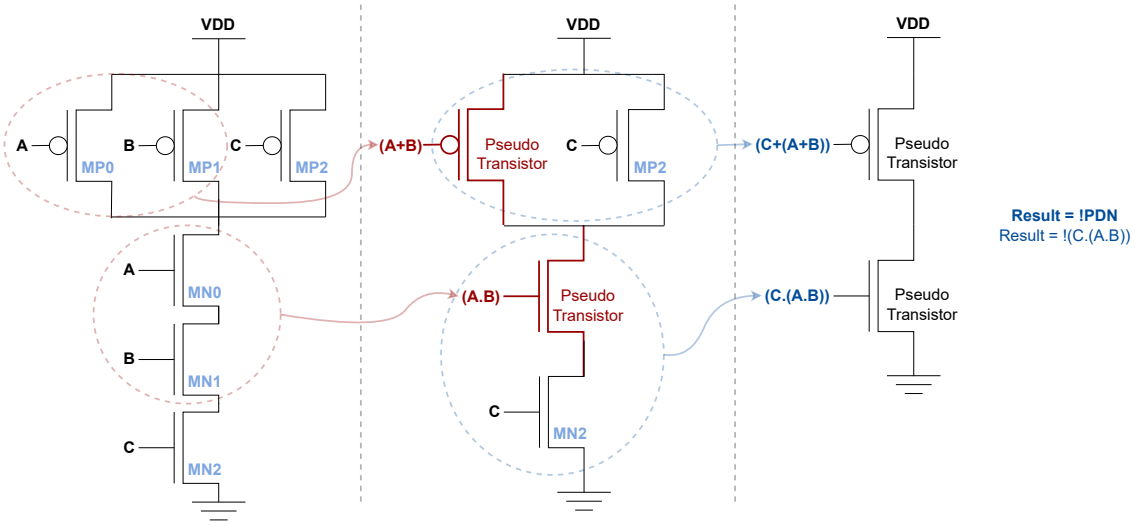

Fig. 4. Transistor Network Flattening Example

Algorithm 3: Algorithm for Finding the Boolean Expression for Every Common Net Between PDN and PUN

Input: .
PXN: Either PDN or PUN
CommonNets: Common Net Expressions we are looking for
SizeCN: The amount of Common Nets in the Circuit
Output: Expression
FindExpression(PXN,*CommonNet*, *SizeCN*)

```
while (size(PXN) > SizeCN) do
    SolveParallelTransistors(PXN,CommonNet, SizeCN)
    if SIZE(PXN) = SizeCN then
        for CommonNet:CommonNets do
            Transistor t1
            t1 = find_pseudo_t (PXN, CommonNet)
            return t1.getAlias()
        end for
    else
        SolveSeriesTransistors(PXN,CommonNet, SizeCN)
    end if
end while
return Expression
```

IV. Boolean Solver

The Boolean solver is an integral part of the tool. Different parts of the tool rely on the Boolean solver. The implemented Boolean solver iterates through the given Boolean expression. The Boolean solver call for further improvements since the current implementation is very simple: it now finds patterns among literals and substitutes them accordingly. As an example, if it finds the pattern "0 + 0" it will replace it with logic "0". If it finds the pattern "0 + 1", it will replace it with logic "1".

Before	After
0 * 0	0
0 * 1	0
1 * 0	0
1 * 1	1
0 + 0	0
0 + 1	1
1 + 0	1
1 + 1	1
!1	0
!0	1
(1)	1
(0)	0

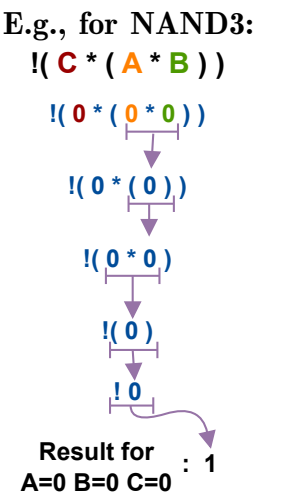

Fig. 5. Replacement table and an example showing the replacement interaction for the 3-input NAND boolean expression.

This process is kept running, following the replacement table as the one depicted in Fig. 5, until the expression size is down to just one character.

V. Truth Table

Boolean expressions are essential representations of functions that describe logical arguments and determine the function's output Boolean value. A truth table displays a Boolean expression's result for all possible combinations of inputs.

To generate the truth table, we used the previously introduced Boolean solver to solve all input combinations. The number of combinations for an n-input logic cell can reach up to 2^n possible combinations. We create a variable

with a length equal to the number of unique literals in the expression. This variable is shifted at each of the 2^n iterations, and a bit-wise operation is performed. The resulting output is then fed to the Boolean solver at each iteration.

VI. Transition Arcs Search

One important task in electrical circuit analysis is determining a netlist's transition arcs. A transition arc is a directed edge (rising- or falling edge) in the circuit graph that represents the transition of a digital signal from one logic level to another. Knowing the transition arcs for a given logic cell is essential for timing analysis, power estimation, and other circuit optimization tasks. In this section, we describe the part of LEX responsible for finding the transition arcs of a given netlist.

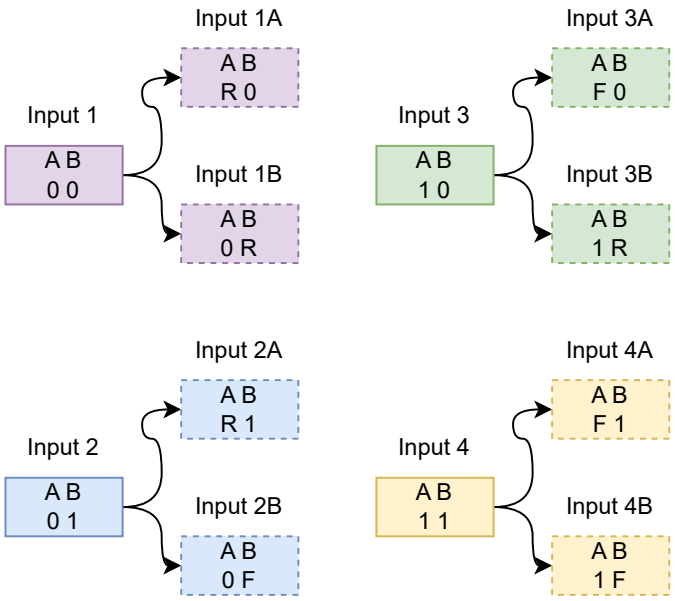

Fig. 6. Methodology for Finding the Transition Arcs

Figure 6 illustrates the various input arcs that may be present in a two-input (A and B) cell. To identify any transition arcs, we need to evaluate each input case labeled as "Input 1, Input 2, Input 3, and Input 4" by inputting them into a Boolean solver to obtain the corresponding output.

We then proceed to compare the input cases composed of the parent case "Input 1, Input 2, Input 3" and the child cases "Input 1A, Input 2A, Input 3A, and Input 4A" and "Input 1B, Input 2B, Input 3B, and Input 4B". Suppose the output of the parent case is different from the output of the child case. In that case, it indicates the presence of a switching arc (e.g., if solve(Input 1) = 0 and solve(Input 1B) = 1, then a transition arc $A \uparrow \ B : 0 \rightarrow Out \uparrow$ is identified).

We can systematically identify all transition arcs by repeating this process for all input cases. This approach effectively finds all possible switching arcs and can be used as a basis for further analysis of the D-cell behavior. However, this is not the most efficient method for finding all possible switching arcs since it compares each state of the truth table with its possible transitions. Other algorithms, such as those based on graph theory, may be more efficient for larger circuits.

TABLE I
Some of the types of cells that were tested (not including the driving strengths of each) and their respective number of arcs found by the tool.

Type	Funct.	#Arcs	Funct.	#Arcs
Comb.	a2111o	22	a2111oi	22
	a211o	16	a211oi	16
	a21boi	10	a21o	10
	a221o	32	a221oi	32
	a222o	64	a222oi	64
	a22o	24	a22oi	24
	a2bb2o	16	a2bb2oi	16
	a311oi	34	a32o	46
	a32oi	46	a41oi	38
	inv	2	buf	2
	clkinv	2	clkbuf	2
	einvn	4	maj3	12
	and2	4	nor2	4
	and3	6	nor3	6
	and4	8	nor4	8
	nand2	4	or2	4
	nand3	6	or3	6
	nand4	8	or4	8
	nand2b	4	xnor2	8
	nand3b	6	xnor3	8
	nand4b	8	nor2b	4
	nand4bb	16	nor3b	6
	einvp	4	nor4b	8
	o41ai	38	o2111a	22
	o2111ai	24	o211a	16
	o211ai	16	o21a	10
	o21ai	10	o221a	42
	o221ai	40	o22a	16
	o22ai	16	o2bb2ai	24
	o311a	32	o311ai	34
	o31a	20	o31ai	20
	o32a	32	o41a	38
	aoi22	24	...	...
	...	✗	...	✗
Seq.	...	✗	...	✗
Async.	CELEM2	✗	...	✗

VII. Results

The LEX tool herein proposed, with a simple Spice-input format, was tested for different technologies, driving strengths, and types of logic. The LEX tool for logic and switching arcs extractions was meant to be as technology agnostic as possible. The tool successfully retrieved the switching arcs for D-cells described in Spice syntax for technologies such as 16 nm FinFet, 28 nm Bulk CMOS, 180 nm Bulk CMOS, and Skywater 130 nm PDK. Being open, the latter is the main focus of the results in this work.

The tool was tested with all cells available at the Skywater 130 nm high-speed standard cell library [10]. Currently, the tool is capable of discovering what are the inputs and outputs of all cells. However, it can only extract the arcs of 166 out of 392 cells or 42.3% of the library without any prework. It is important to note that out of these 392, there are tap cells, filler cells, and more. The real execution time for the 392 cells took 4.683 seconds in a consumer-grade laptop equipped with an Intel I5-8400h processor and 16 gb of RAM. For the same set, it took 3.995 seconds in the mode that only outputs the arcs i.e., without any visual feedback on the Command-Line Interface (CLI).

The tool correctly extracted the Boolean expression for the 2-input Muller C-element; However, since the output pin is part of the output expression, the tool entered a loop while trying to solve the Boolean expression. The solving process was stopped by the emergency timeout scheme. Not being able to fetch the truth table and switching arcs for the c-element cell.

VIII. Conclusion & Future Work

In conclusion, the method for analyzing digital circuits that we have herein described is simple and efficient. Our tool can parse a SPICE netlist and then generates a Boolean expression, a truth table, and switching arcs only by entering the netlist. This strategy has significant advantages over other approaches as it just needs one input and does not require labor-intensive human analysis, as in the case of large complex cells. The characterization of transistor networks in library-free methods [11] [12] can also take advantage of this tool, as the amount of transistor networks, or Static CMOS Complex Gates (SCCGs), to be characterized can be so large that it becomes unpractical for human analysis.

Our tool is highly accessible, as it does not require extensive knowledge or experience to use. Its user-friendly interface and efficient processing make it an ideal resource for engineers, students, and researchers alike. The C++ source code of the tool and some test-cases are openly available at our GitHub repository [13] and distributed under the MIT license.

Future work in LEX will be on the expansion of this tool as a complete minimal-input characterization tool-chain. Another future aim is the improvement of the limitations of this tool to correctly extract and find relevant switching arcs in sequential D-cells, e.g., type D Flip-flops, and also for the cells that are commonly used in the design of asynchronous systems.

Acknowledgment

This research was partially funded by Brazilian government funding agencies, CNPq, Capes Code 001, and Fapergs.

References

[1] T. Hoang, "Cerebras systems smashes the 2.5 trillion transistor mark with new second generation wafer scale engine," Jul 2022. [Online]. Available: https://www.cerebras.net/press-release/cerebras-systems-smashes-the-2-5-trillion-transistor-mark-with-new-second-generation-wafer-scale-engine/

[2] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem, G. Pradipta, S. Reda, M. Saligane, S. S. Sapatnekar, C. Sechen, M. Shalan, W. Swartz, L. Wang, Z. Wang, M. Woo, and B. Xu, "Toward an open-source digital flow: First learnings from the openroad project," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3316781.3326334

[3] A. Ghazy and M. Shalan, "Openlane: The open-source digital asic implementation flow," in *Workshop on Open-Source EDA Technology (WOSET)*, 2020, p. 21.

[4] E. Corporation, "Open shuttle program," https://efabless.com/open_shuttle_program, 2021, [Accessed: March 16, 2023].

[5] I. S.-S. C. Society, "Ieee solid-state circuits society pico program," https://sscs.ieee.org/about/solid-state-circuits-directions/sscs-pico-program, 2021, [Accessed: March 16, 2023].

[6] 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. advpub, p. 2022VLP0007, 2022.

[7] M. S. I. bin Hussin, Y. W. Lim, N. A. Kamsani, S. J. Hashim, and F. Z. Rokhani, "Development of automated standard cell library characterization (asclic) for nanometer system-on-chip design," in *2017 IEEE 15th Student Conference on Research and Development (SCOReD)*, 2017, pp. 93–97.

[8] M. T. Moreira, C. H. M. Oliveira, N. L. V. Calazans, and L. C. Ost, "Lichen: Automated electrical characterization of asynchronous standard cell libraries," in *2013 Euromicro Conference on Digital System Design*, 2013, pp. 933–940.

[9] A. Beg, A. Elchouemi, and R. Beg, "A collaborative platform for facilitating standard cell characterization," in *Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2013, pp. 202–206.

[10] G. , "Github - google/skywater-pdk-libs-sky130_fd_sc_hs: "High speed" digital standard cells for SKY130 provided by SkyWater." https://github.com/google/skywater-pdk-libs-sky130_fd_sc_hs, nov 16 2020.

[11] F. S. Marques, L. S. Rosa, R. P. Ribas, S. S. Sapatnekar, and A. I. Reis, "Dag based library-free technology mapping," in *Proceedings of the 17th ACM Great Lakes Symposium on VLSI*, ser. GLSVLSI '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 293–298. [Online]. Available: https://doi.org/10.1145/1228784.1228857

[12] A. I. Reis, R. Reis, D. Auvergne, and M. Robert, *Library Free Technology Mapping*. Boston, MA: Springer US, 1997, pp. 303–314. [Online]. Available: https://doi.org/10.1007/978-0-387-35311-1_25

[13] R. Wuerdig, "Github - rodrigowue/LEX: Lex - Spice Standard-cell Arcs Extractor," https://github.com/rodrigowue/LEX, mar 17 2023.