A Position on Language Abstraction for Custom Circuit Design

Yongsheng Qin
The Chinese University of Hong Kong, Shenzhen
China

Zachary D. Sisco
The Chinese University of Hong Kong, Shenzhen
China

Abstract

Despite the growing importance of custom blocks and cells in modern chips, custom-circuit design still relies heavily on manual optimization and fragmented research and tooling for automation. In this paper, we argue that a language-driven representation and toolchain for transistor-level netlists can serve as infrastructure to enable transistor-level design automation for optimizations that scale to larger designs.

To argue this position, we present one step in this direction: We introduce TransiLog, a DSL for CMOS transistor networks. TransiLog combines behavioral and topological semantics to represent transistor-level structures. Using equality saturation, we optimize transistor netlists for PPA goals in a non-destructive way. TransiLog also emits SPICE netlists compatible with existing transistor-level physical design backends and offers a foundation for future hybrid flows that combine custom-circuit and cell-level optimization.

1 Introduction

As chips become increasingly specialized and cost-driven, standard-cell-library-based physical flows can no longer meet aggressive, design-specific PPA targets. Custom modules and custom standard cells have become a standard method to squeeze the last margin on the chip, and dedicated teams for custom cells and custom blocks have become a standard setup in industry.

While custom circuits can be extremely competitive, automation for custom-circuit designs remains limited and fragmented across both research and commercial tool flows. Despite a substantial amount of related work in areas such as cell layout generation[4, 5], CMOS network optimization[11, 18] and dedicated custom circuit generators[9, 19], most existing work either relies on oversimplified PPA metrics or works only on the logical or physical end, and fails to be applied to larger-scale or more general designs. Relatively little work treats general-purpose custom-circuit automation as a dedicated topic to explore. As a result, engineers still manually iterate on transistor topology and layout, moving through a test-and-run loop to converge on a PPA target.

Although custom-circuit development seemingly follows a familiar flow with standard-cell-level designs (netlist optimization, physical implementation (FP/PR), verification), the level of automation available for custom circuits is far less than what we take for granted in the cell flow. This gap stems from two fundamental differences:

(1) SPICE netlists do not explicitly encode Boolean logic semantics, which hinders logic optimization and PPA-driven automation.
(2) Unlike standard cells, custom circuits lack a compact, reusable abstraction that simultaneously encapsulates timing and physical constraints, making it difficult to scale to larger designs or to systematically address transistor-level physical design challenges.

Motivated by these observations, we propose TransiLog, a DSL that simultaneously captures the structure *and* logical behavior of CMOS transistor netlists. TransiLog is intended as a representation for custom-circuit netlists, enabling infrastructure of optimization flows and tooling for PPA-driven transistor-level netlist optimization. It can export results to transistor-level physical design flows and integrate seamlessly into cell-level flows.

In this paper, we first explain the key features of TransiLog with some examples (Section 2), show some preliminary results (Section 3), then show future work by explaining how this DSL points the way to more infrastructure to support a custom circuit automation flow, and how novel language abstractions can improve post-synthesis flows (Section 4).

2 TransiLog for Transistor Netlists

Boolean algebra cannot faithfully represent CMOS transistor networks: in CMOS circuits, the pull-up network (PUN) and pull-down network (PDN) realize different logical forms of the same Boolean function, and pure logic expressions cannot distinguish PUN/PDN from networks with both pull-up and pull-down capability, nor express certain circuit topologies such as non-series-parallel (NSP) networks. Therefore, we extend the syntax of boolean logic with a set of structural operators that describes circuit structure and logical function together.

Table 1 describes the semantics for TransiLog. Each operator in the language has two interpretations: a logical interpretation (as defined by a denotation into the Boolean logic domain), and a structural interpretation (the resulting device-level network). For ease of presentation, we give a natural-language description of the structural semantics.

Under this syntax, **join** together with **!** forms a node with both pull-up and pull-down capability, while expressions built from $\cdot$, $+$, **bridge**, and **X** form networks that can serve as either a pull-up or a pull-down network.

We next describe a subset of the rewrite rules for both structural and logical transformations in the DSL. Aside from conventional rules like distributivity (D) that rewrite logical expressions (i.e., transformations operating within a PUN/PDN network), the full rule set also includes cross-stage structure-modifying rules such as bubble-pushing rule (B), which explicitly change circuit topology but preserve logical behavior. These rules enable operations including buffer insertion, bubble pushing/merging, and cross-stage logic

Table 1: TransiLog operators, logical denotation, and device-level structural semantics.

Operator	Logical Denotation	Structural Semantics
join(PUN, PDN)	$[\![\text{join}(\text{PUN},\text{PDN})]\!] = \neg[\![\text{PUN}]\!] = \neg[\![\text{PDN}]\!]$	Used to form dual networks using a pull-up network (PUN) and a pull-down network (PDN).
$!(x)$	$[\![!(x)]\!] = \neg[\![x]\!]$	Connect the operand to an inverter.
$+(a,b)$	$[\![a+b]\!] = [\![a]\!] + [\![b]\!]$	Form a series (PUN) / parallel (PDN) connection between operands, with *a* on the top/left.
$\cdot(a,b)$	$[\![a\cdot b]\!] = [\![a]\!] \cdot [\![b]\!]$	Form a series (PDN) / parallel (PUN) connection between operands, with *a* on the top/left.
bridge(a,b,c,d,e)	$[\![\text{bridge}(a,b,c,d,e)]\!] = [\![ab+cd+aed+bec]\!]$	Used to denote NSP structures. Form a bridge connection between operands (left-top, left-bottom (PDN) / right-top (PUN), right-top (PDN) / left-bottom (PUN), right-bottom, middle).
$\&(a,b)$	$[\![a]\!], [\![b]\!]$	Virtual output concatenation only.
X(Var, size)	$[\![X(\text{Var},\text{size})]\!] = [\![\text{Var}]\!]$	Size operator: sizes transistors.
Bool	True or False	Constant literal (VDD/GND).
Var	Var	Input literal.

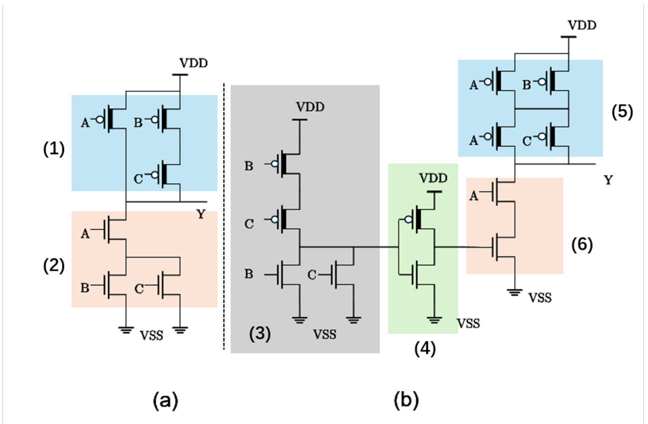

Figure 1: Two logically equivalent CMOS implementations of $\overline{A(B+C)}$ with different circuit structures.

restructure. In addition, there are transistor sizing rules to enable transistor sizing.

Figure 1 shows two logically equivalent CMOS implementations of $\overline{A(B+C)}$ with different circuit structures. In TransiLog, the circuits in Figure 1(a) and (b) are represented as join$\big(A(B+C),\ A(B+C)\big)$ and join$\big(AB+AC,\ A\cdot !(\text{join}(B+C,\ B+C))\big)$, respectively. We obtain (b) from (a) by applying the distributivity rule (D) to the pull-up network (PUN) sub-expression (1) produces (5), and the bubble-pushing rule (B) to the $B+C$ portion of the pull-down network (PDN), which pushes B+C out as an individual stage (3), and inserts inverter structure (4) to perform AND operation with A (6). The numbered annotations (1)–(6) in the figure correspond to the underbraced sub-expressions in the following transformation:

$$\text{join}\Big(\underbrace{A(B+C)}_{(1)},\ \underbrace{A(B+C)}_{(2)}\Big) \Rightarrow$$

$$\text{join}\Big(\underbrace{AB+AC}_{(5)},\ \underbrace{A\cdot}_{(6)}\ \underbrace{!}_{(4)}\ \underbrace{(\text{join}(B+C,\ B+C))}_{(3)}\Big).$$

$$\text{(D) Distributivity:}\quad x(y+z) \longrightarrow xy+xz$$

$$\text{(B) Bubble-pushing:}\quad x \longrightarrow !\big(\text{join}(x,\ x)\big)$$

By combining TransiLog with these rewrite rules, we can use e-graphs to compactly represent the transistor-level design space at the syntax layer, then use equality saturation [13, 16, 17] to explore the design space in a non-destructive way. The overall workflow is as follows in Figure 2.

We first build an initial e-graph from an eqn file or an S-expression representation, then apply rewrite rules to (partially) saturate the design space. Finally, we run extractors for different optimization objective, e.g., minimizing critical-path delay, minimizing area, or minimizing transistor count. For the delay-optimization cost, we use SPICE simulations performed on SPICE netlist emitted from TransiLog, together with a test bench template.

3 Preliminary Results

We implemented multiple delay-minimization extractors, including a brute-force enumeration extractor, a uniform-sampling-based extractor, and a random-walk-based extractor. For simulation, we use Xyce[1] and conducted preliminary experiments with the ASAP 7nm PDK[7]. Using a set of naive implementation of CMOS circuits serving as baseline, nearly all of them extracted a better result. For larger designs, the extraction speed still needs fine-tuning.

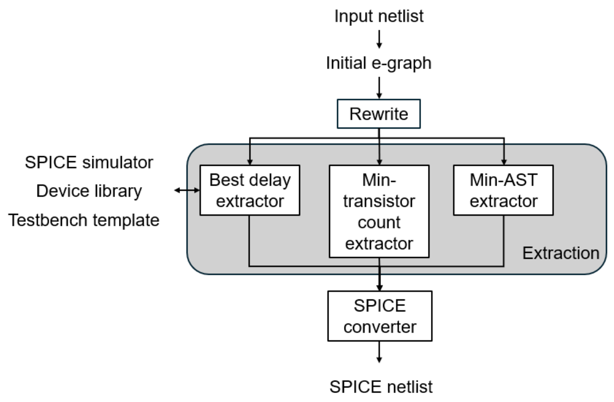

Figure 2: Equality saturation flow of transistor level netlist optimization.

Other simpler metrics such as minimal transistor count are also used for netlist optimization. We implemented a minimal-transistor-count extractor and evaluated it on the 53 Handcrafted Optimum Switch Networks testbench [12]; with our initial prototype, the total transistor count is 395 transistors with ~3 min runtime. Although not yet optimal (i.e., 356 transistors), the result surpasses several prior approaches as summarized in [18], leaving room for further optimization.

4 Conclusions

Clearly defined interfaces such as TransiLog will aid interoperability between EDA tools. Previous research also shows that e-graph-based circuit representations are effective across abstraction levels, from specification level to gate level [2, 3, 6, 8, 10, 14, 15]. In this work, we extend the scope to the transistor level. With these different layers represented as e-graphs, we believe there is potential in unifying these representations to close the semantic gap between design layers (e.g., the RTL and transistor level) and expose more optimizations in the resulting circuit.

TransiLog demonstrates our position that novel language abstractions can improve automation for physical design flows by unifying logical and structural operators. Beyond network topology, there are other aspects of physical design that can be modeled via language abstractions which we leave for future work and discussion: for example, routing, geometry, and timing. In our preliminary results, we show that within an e-graph, TransiLog's abstractions often offer richer expressiveness than Verilog and SPICE netlists, enabling multiple optimization goals in the same representation. Moreover, we believe this initial work points to the benefit of designing new DSLs/IRs for post-synthesis design flows. We argue that at the physical level, for problems not yet addressed by traditional language abstractions, there exists a broad class of optimization and satisfiability problems founded on congruence relations, which can, in principle, be modeled and optimized within the framework of equality saturation.

Acknowledgments

The authors would like to thank Jonathan Balkind, Minh Bui, and Sijie Kong for comments and discussion over the ideas presented in this manuscript.

References

[1] 2013. Xyce(™) Parallel Electronic Simulator. [Computer Software] https://doi.org/10.11578/dc.20171025.1421. doi:10.11578/dc.20171025.1421

[2] Chen Chen, Guangyu Hu, Cunxi Yu, Yuzhe Ma, and Hongce Zhang. 2025. E-morphic: Scalable Equality Saturation for Structural Exploration in Logic Synthesis. In *Proceedings of the 62nd Annual ACM/IEEE Design Automation Conference* (San Francisco, California, United States) *(DAC '25)*. IEEE Press, Article 319, 7 pages. doi:10.1109/DAC63849.2025.11133110

[3] Chen Chen, Guangyu Hu, Dongsheng Zuo, Cunxi Yu, Yuzhe Ma, and Hongce Zhang. 2024. E-syn: E-graph rewriting with technology-aware cost functions for logic synthesis. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[4] Chung-Kuan Cheng, Andrew B. Kahng, Byeonggon Kang, Seokhyeong Kang, Jakang Lee, and Bill Lin. 2025. SO3-Cell: Standard Cell Layout Automation Framework for Simultaneous Optimization of Topology, Placement, and Routing. In *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD66269.2025.11240677

[5] Chung-Kuan Cheng, Byeonggon Kang, Bill Lin, and Yucheng Wang. 2025. *Standard Cell Layout Generation: Review, Challenges, and Future Works*. Association for Computing Machinery, New York, NY, USA, 372–378. https://doi.org/10.1145/3658617.3703146

[6] Jianyi Cheng, Samuel Coward, Lorenzo Chelini, Rafael Barbalho, and Theo Drane. 2024. Seer: Super-optimization explorer for high-level synthesis using e-graph rewriting. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 1029–1044.

[7] Lawrence T. Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. 2016. ASAP7: A 7-nm finFET predictive process design kit. *Microelectronics Journal* 53 (2016), 105–115. doi:10.1016/j.mejo.2016.04.006

[8] Samuel Coward, Theo Drane, and George A Constantinides. 2024. ROVER: RTL optimization via verified E-graph rewriting. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 12 (2024), 4687–4700.

[9] Matthew R. Guthaus, James E. Stine, Samira Ataei, Brian Chen, Bin Wu, and Mehedi Sarwar. 2016. OpenRAM: an open-source memory compiler. In *Proceedings of the 35th International Conference on Computer-Aided Design* (Austin, Texas) *(ICCAD '16)*. Association for Computing Machinery, New York, NY, USA, Article 93, 6 pages. doi:10.1145/2966986.2980098

[10] Matthew Hofmann, Berk Gokmen, and Zhiru Zhang. 2025. EqMap: FPGA LUT Remapping using E-Graphs. In *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD66269.2025.11240672

[11] Henrique Kessler, Marcello Muñoz, Plínio Finkenauer Junior, Leomar Jr, and Vinícius Camargo. 2021. Electrical Evaluation of Logic Network Generation Methods for On-the-Fly Supergate Design. *Journal of Integrated Circuits and Systems* 16 (12 2021), 1–7. doi:10.29292/jics.v16i3.527

[12] Logics Lab. 2012. Catalog of 53 Handmade Optimum Switch Networks. Federal University of Rio Grande do Sul (UFRGS), Institute of Informatics. https://www.inf.ufrgs.br/logics/docman/53_NSP_Catalog.pdf Accessed: 2026-01-15.

[13] Charles Gregory Nelson. 1980. *Techniques for program verification*. Stanford University.

[14] Yan Pi, Hongji Zou, Tun Li, Wanxia Qu, and Hai Wan. 2023. ESFO: Equality Saturation for FIRRTL Optimization. In *Proceedings of the Great Lakes Symposium on VLSI 2023*. 581–586.

[15] Gus Henry Smith, Colin Knizek, Daniel Petrisko, Zachary Tatlock, Jonathan Balkind, Gilbert Louis Bernstein, Haobin Ni, and Chandrakana Nandi. 2024. Scaling Program Synthesis Based Technology Mapping with Equality Saturation. *arXiv preprint arXiv:2411.11036* (2024).

[16] Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. 2009. Equality saturation: a new approach to optimization. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 264–276.

[17] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. egg: Fast and extensible equality saturation. *Proc. ACM Program. Lang.* 5, POPL, Article 23 (Jan. 2021), 29 pages. doi:10.1145/3434304

[18] Weihua Xiao, Shanshan Han, Yue Yang, Shaoze Yang, Cheng Zheng, Jingsong Chen, Tingyuan Liang, Lei Li, and Weikang Qian. 2023. MiniTNtk: An Exact Synthesis-based Method for Minimizing Transistor Network. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 01–09. doi:10.1109/ICCAD57390.2023.10323691

[19] Jinshan Zhang, Bo Jiao, Yunzhengmao Wang, Haozhe Zhu, Lihua Zhang, and Chixiao Chen. 2021. ALPINE: An Agile Processing-in-Memory Macro Compilation Framework. In *Proceedings of the 2021 Great Lakes Symposium on VLSI* (Virtual Event, USA) *(GLSVLSI '21)*. Association for Computing Machinery, New York, NY, USA, 333–338. doi:10.1145/3453688.3461532