There and Back Again: A Netlist's Tale with Much Egraphin'

Gus Henry Smith[†], Zachary D. Sisco[‡], Thanawat Techaumnuaiwit[‡], Jingtao Xia[‡], Vishal Canumalla[†], Andrew Cheung[†], Zachary Tatlock[†], Chandrakana Nandi[§], Jonathan Balkind[‡]

[†] University of Washington [‡] University of California, Santa Barbara [§] Certora, Inc.

{gussmith, vishalc, acheung8, ztatlock}@cs.washington.edu {zsisco, thanawat, jingtaoxia, jbalkind}@ucsb.edu

chandra@certora.com

ABSTRACT

EDA toolchains are notoriously unpredictable, incomplete, and error-prone; the generally-accepted remedy has been to re-imagine EDA tasks as compilation problems. However, any compiler framework we apply must be prepared to handle the wide range of EDA tasks, including not only compilation tasks like technology mapping and optimization (the *"there"* in our title), but also decompilation tasks like loop rerolling (the *"back again"*). In this paper, we advocate for *equality saturation*—a term rewriting framework—as the framework of choice when building hardware toolchains. Through a series of case studies, we show how the needs of EDA tasks line up conspicuously well with the features equality saturation provides.

1 INTRODUCTION

Hardware development toolchains are notorious for their unpredictability [12], incompleteness [17], and incorrectness [4]. These issues stem from the fact that most common toolchains do not treat EDA tasks as compilation problems, and instead often use ad hoc, unprincipled approaches to solving each problem. Existing projects such as MLIR CIRCT [1], LLHD [15], and Calyx [13] have made great strides towards reframing and restructuring hardware design tools using consistent compiler frameworks.

Finding an appropriate compiler framework is difficult, as the EDA tasks that must be supported are diverse. For example, any framework should certainly be able to capture all standard optimization tasks, such as register retiming, pipelining, and common subexpression elimination. However, another essential task beyond standard optimization is *technology mapping*—the process of implementing a high-level design specification using the actual hardware primitives available on the target FPGA or ASIC process. To make things even more complicated, EDA tasks are not always "moving forward": recent work has established *hardware decompilation* as a valuable tool for design tasks such as speeding up netlist simulations [16]. Thus, an ideal compiler framework must also be able to easily break and lower between levels of abstraction.

Equality saturation [19] is a compiler framework which has already proven its prowess in all of these tasks. Equality saturation is a non-destructive term rewriting technique that uses the **e-graph data structure** [10, 11] to compactly store potentially infinitely many equivalent terms. Recent work [27, 29] has developed fast and extensible libraries for efficient equality saturation. Previous work has shown equality saturation's ability to implement decompilation [9], procedural abstraction [2], optimization [7, 8, 20, 24, 25, 30], and mapping [6, 18, 23].

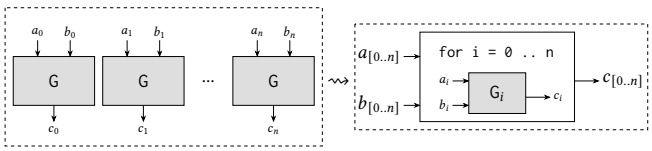
In this position paper, we advocate for the extensive application of equality saturation to EDA tasks. In fact, equality saturation has already shown early promise in being applied to various EDA tasks, including RTL optimization [5, 14], HLS optimization [3], multiplier optimization [21, 26], and repurposing CGRAs [28]. Ustun et al. also argue for equality saturation in datapath synthesis and optimization [22]. We make a larger claim in this paper that equality saturation has value beyond optimization in EDA tasks up and down the stack.

We now present four case studies highlighting different properties of equality saturation that makes it attractive at different stages in the hardware design workflow, from technology mapping to circuit-level analyses such as retiming and decompilation. These case studies demonstrate that the operational semantics of the various stages of the hardware design workflow are intuitively represented as rewrite rules. Each case study will explore the different properties of equality saturation that makes it attractive for implementing the different hardware passes.

2 CASE STUDY: HARDWARE LOOP REROLLING

Recent work considers the problem of hardware loop rerolling, that is, identifying repeated sequences of logic in a netlist and rerolling them into loops in higher-level HDL code [16]. This research fits into the larger problem of *hardware decompilation*, which lifts netlists to HDL code to help with design and analysis tasks. Loop rerolling for hardware decompilation uses a sketch-guided program synthesis technique to synthesize rerolled loops [16]. However, this technique scales poorly due to its reliance on SMT solvers to fill in the loop sketches.

Our in-progress work considers hardware loop rerolling through the lens of rewriting. Consider the following illustration of a rewrite rule which identifies a repeated logic block `G` and rewrites it into a `for`-loop with `G` parameterized over the loop variable `i`:

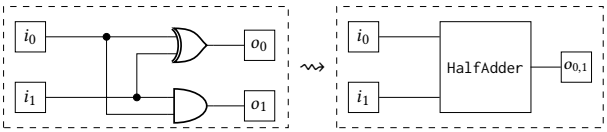


LATTE '24, April 28, 2024, San Diego, CA, USA

While in this example, the indices a and b appear in monotonically increasing order on the logic blocks, this may not always be the case, which would make it harder to infer the closed form for the `for`-loop. More generally, loop rerolling is particularly challenging when the initial, unrolled program does not expose any high-level structure, i.e., the repetitive patterns of the program are obfuscated. Prior work shows that equality saturation can be used to discover this latent structure by applying carefully designed rewrite rules [9]. We envision scaling hardware loop rerolling by leveraging similar techniques.

3 CASE STUDY: STANDARD LIBRARY COMPONENT IDENTIFICATION

This case study is about finding components from a hardware standard library within a compiled artifact such as a netlist. The compiler optimizes the component in ways that using a sub-graph-isomorphism algorithm for identification will fail, and, for large enough designs, will not scale. An e-graph solves these problems in two ways: (1) it allows us to explore semantically equivalent versions of the same design to find the one where we can extract the standard library component and (2) it allows sub-graphs to be extracted out more efficiently due to the internal union-find structure. For standard library component identification, we can directly take the standard library component we are looking for and turn it into a rewrite rule within the `egglog` equality saturation engine [29]. For example, here is an illustration of the rewrite rule for a half adder:



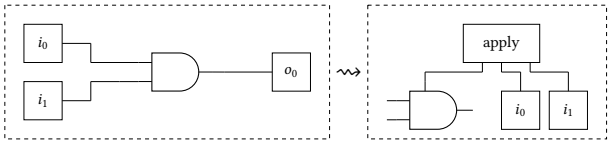
Within this rewrite rule, i_0 and i_1 are *any arbitrary circuits*. Equality saturation runs this rewrite rule (along with standard rules for Boolean algebra) on a larger design which pattern-matches parts of the design with the half-adder definition—rewriting that definition into an abstract half-adder component.

Challenges with this approach include matching on components where the compiler optimized away parts of the module or fused two modules together which share resources. Anti-unification techniques, as presented in babble [2], can help with the problem of partial matching. Further, a generalized problem of standard library identification is *procedural abstraction*, finding repeated instances of a procedure where there is no standard library as reference to match against.

4 CASE STUDY: SCALING TECHNOLOGY MAPPING VIA LIBRARY LEARNING

Our previous work Lakeroad [17] demonstrates how the process of FPGA *technology mapping*—converting a high-level hardware design description into an implementation using FPGA-specific primitives—can be vastly improved via program synthesis. However, program synthesis is known to face scaling issues. Meanwhile, the process of technology mapping must scale to potentially massive hardware designs.

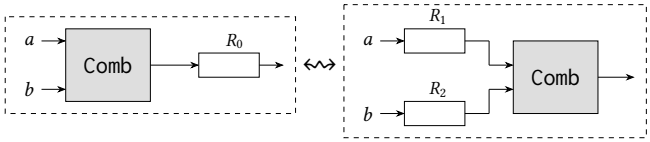
With equality saturation, we can scale these state-of-the-art technology mapping techniques via the application of *library learning* [2]. Library learning is the process of finding abstractions commonly used throughout a corpus of code—in our setting, finding hardware modules used repeatedly within a larger design. Within the equality saturation framework, library learning can be expressed simply as a rewrite which converts an expression into an abstracted module applied to a list of concrete inputs:



When applied repeatedly across a large design, this rewrite will find larger and larger abstracted submodules. By default, equality saturation deduplicates identical expressions, allowing us to discover submodules which are frequently reused across the design. These abstracted submodules are then perfect candidates for program synthesis. Furthermore, we can use information about frequency of appearance and other contextual information in the e-graph to filter and rank candidates. Thus, equality saturation gives us a path towards scaling currently limited state-of-the-art techniques using simple algebraic rewrites and its native deduplication ability.

5 CASE STUDY: CIRCUIT RETIMING

With an algebraic representation of the netlist we form a bidirectional rewrite rule that captures forward and backward retiming:



where `Comb` is a combinational gate. With only these two rules, equality saturation explores all possible ways of arranging registers in the design through non-destructive rewrites. Then, we use ILP (Integer Linear Programming) to retime the circuit according to a cost function—following prior work that effectively uses ILP extraction from an e-graph [3, 24].

The other side of retiming is *undoing* the effects of a retimed cicuit by, for example, moving all registers as close as possible to their source. This pass is useful for decompilation by moving registers outside of a section of combinational logic to expose latent structure for other analyses such as standard library component identification and loop rerolling (Sections 2 and 3).

6 CONCLUSION

We present case studies demonstrating how equality saturation can be used to improve state of the art techniques for mitigating four concrete hardware challenges: decompilation through loop rerolling, library component identification and technology mapping through library learning, and optimum circuit retiming through efficient state space exploration and ILP extraction. We are already working on some of these topics and hope this paper encourages other researchers to consider equality saturation as a technique to mitigate EDA challenges in the future.

REFERENCES

[1] CIRCT Authors. 2024. CIRCT. https://circt.llvm.org/. Accessed 2024-03-06.

[2] David Cao, Rose Kunkel, Chandrakana Nandi, Max Willsey, Zachary Tatlock, and Nadia Polikarpova. 2023. babble: Learning Better Abstractions with E-Graphs and Anti-unification. *Proc. ACM Program. Lang.* 7, POPL, Article 14 (jan 2023), 29 pages. https://doi.org/10.1145/3571207

[3] Samuel Coward, George A. Constantinides, and Theo Drane. 2022. Automatic Datapath Optimization using E-Graphs. In *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*. 43–50. https://doi.org/10.1109/ARITH54963.2022.00016

[4] Yann Herklotz and John Wickerson. 2020. Finding and understanding bugs in FPGA synthesis tools. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 277–287.

[5] Kuo-Wei Ho, Shao-Ting Chung, Tian-Fu Chen, Yu-Wei Fan, Che Cheng, Cheng-Han Liu, and Jie-Hong R Jiang. 2023. WolFEx: Word-Level Function Extraction and Simplification from Gate-Level Arithmetic Circuits. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[6] Bo-Yuan Huang, Steven Lyubomirsky, Yi Li, Mike He, Gus Henry Smith, Thierry Tambe, Akash Gaonkar, Vishal Canumalla, Andrew Cheung, Gu-Yeon Wei, Aarti Gupta, Zachary Tatlock, and Sharad Malik. 2024. Application-level Validation of Accelerator Designs Using a Formal Software/Hardware Interface. *ACM Trans. Des. Autom. Electron. Syst.* 29, 2, Article 35 (feb 2024), 25 pages. https://doi.org/10.1145/3639051

[7] Shadaj Laddad, Conor Power, Tyler Hou, Alvin Cheung, and Joseph M. Hellerstein. 2023. Optimizing Stateful Dataflow with Local Rewrites. arXiv:2306.10585 [cs.PL]

[8] Kazuaki Matsumura, Simon Garcia De Gonzalo, and Antonio J Peña. 2023. ACC Saturator: Automatic Kernel Optimization for Directive-Based GPU Code. *arXiv preprint arXiv:2306.13002* (2023).

[9] Chandrakana Nandi, Max Willsey, Adam Anderson, James R. Wilcox, Eva Darulova, Dan Grossman, and Zachary Tatlock. 2020. Synthesizing structured CAD models with equality saturation and inverse transformations. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 31–44. https://doi.org/10.1145/3385412.3386012

[10] Charles Gregory Nelson. 1980. *Techniques for program verification.* Ph. D. Dissertation. Stanford, CA, USA. AAI8011683.

[11] Robert Nieuwenhuis and Albert Oliveras. 2005. Proof-producing congruence closure. In *Proceedings of the 16th International Conference on Term Rewriting and Applications* (Nara, Japan) *(RTA'05)*. Springer-Verlag, Berlin, Heidelberg, 453–468. https://doi.org/10.1007/978-3-540-32033-3_33

[12] Rachit Nigam, Sachille Atapattu, Samuel Thomas, Zhijing Li, Theodore Bauer, Yuwei Ye, Apurva Koti, Adrian Sampson, and Zhiru Zhang. 2020. Predictable accelerator design with time-sensitive affine types. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 393–407.

[13] Rachit Nigam, Samuel Thomas, Zhijing Li, and Adrian Sampson. 2021. A compiler infrastructure for accelerator generators. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 804–817.

[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] Fabian Schuiki, Andreas Kurth, Tobias Grosser, and Luca Benini. 2020. LLHD: A multi-level intermediate representation for hardware description languages. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 258–271.

[16] Zachary D. Sisco, Jonathan Balkind, Timothy Sherwood, and Ben Hardekopf. 2023. Loop Rerolling for Hardware Decompilation. *Proc. ACM Program. Lang.* 7, PLDI, Article 123 (jun 2023), 23 pages. https://doi.org/10.1145/3591237

[17] Gus Henry Smith, Ben Kushigian, Vishal Canumalla, Andrew Cheung, Steven Lyubomirsky, Sorawee Porncharoenwase, René Just, Gilbert Louis Bernstein, and Zachary Tatlock. 2024. FPGA Technology Mapping Using Sketch-Guided Program Synthesis. *arXiv preprint arXiv:2401.16526* (2024).

[18] Gus Henry Smith, Andrew Liu, Steven Lyubomirsky, Scott Davidson, Joseph McMahan, Michael Taylor, Luis Ceze, and Zachary Tatlock. 2021. Pure tensor program rewriting via access patterns (representation pearl). In *Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming* (Virtual, Canada) *(MAPS 2021)*. Association for Computing Machinery, New York, NY, USA, 21–31. https://doi.org/10.1145/3460945.3464953

[19] Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. 2009. Equality saturation: a new approach to optimization. *SIGPLAN Not.* 44, 1 (jan 2009), 264–276. https://doi.org/10.1145/1594834.1480915

[20] Samuel Thomas and James Bornholt. 2024. Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors. (2024).

[21] Ecenur Ustun, Ismail San, Jiaqi Yin, Cunxi Yu, and Zhiru Zhang. 2022. Impress: Large integer multiplication expression rewriting for fpga hls. In *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 1–10.

[22] Ecenur Ustun, Cunxi Yu, and Zhiru Zhang. 2023. Equality Saturation for Datapath Synthesis: A Pathway to Pareto Optimality. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 1–2. https://doi.org/10.1109/DAC56929.2023.10247948

[23] Alexa VanHattum, Rachit Nigam, Vincent T. Lee, James Bornholt, and Adrian Sampson. 2021. Vectorization for digital signal processors via equality saturation. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 874–886. https://doi.org/10.1145/3445814.3446707

[24] Yisu Remy Wang, Shana Hutchison, Jonathan Leang, Bill Howe, and Dan Suciu. 2020. SPORES: sum-product optimization via relational equality saturation for large scale linear algebra. *Proc. VLDB Endow.* 13, 12 (jul 2020), 1919–1932. https://doi.org/10.14778/3407790.3407799

[25] Zhengrong Wang, Christopher Liu, Aman Arora, Lizy John, and Tony Nowatzki. 2023. Infinity stream: Portable and programmer-friendly in-/near-memory fusion. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 359–375.

[26] Andy Wanna, Samuel Coward, Theo Drane, George A Constantinides, and Miloš D Ercegovac. 2023. Multiplier Optimization via E-Graph Rewriting. *arXiv preprint arXiv:2312.06004* (2023).

[27] 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. https://doi.org/10.1145/3434304

[28] Jackson Woodruff, Thomas Koehler, Alexander Brauckmann, Chris Cummins, Sam Ainsworth, and Michael FP O'Boyle. 2023. Rewriting History: Repurposing Domain-Specific CGRAs. *arXiv preprint arXiv:2309.09112* (2023).

[29] Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock, and Max Willsey. 2023. Better Together: Unifying Datalog and Equality Saturation. *Proc. ACM Program. Lang.* 7, PLDI, Article 125 (jun 2023), 25 pages. https://doi.org/10.1145/3591239

[30] Xiaolei Zhao, Zhaoyun Chen, Yang Shi, Mei Wen, and Chunyun Zhang. 2023. Automatic End-to-End Joint Optimization for Kernel Compilation on DSPs. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.