Democratizing Hardware Verification By Formalizing High-Level Abstractions

Luisa Cicolini

Verification is a fundamental step of the hardware design process, preventing expensive and potentially critical mistakes. As the complexity of hardware designs increases, so does the need for more efficient verification strategies to streamline the overall process. The CIRCT Electronic Design Automation (EDA) stack has recently shown that explicit representations of higher-level, domain-specific hardware enable the effective generation of optimized Verilog code. Such higher-level hardware representations can be a game changer in verifying the designs progressively as they are lowered and exploring novel, domain-specific verification approaches. However, taking advantage of these representations for verification purposes remains challenging due to the lack of formalized semantics for high-level abstractions, which often rely on domain-specific structures such as Finite State Machines (FSMs). In my PhD I will exploit modern multi-level EDA approaches to formally verify the semantics of high-level, domain-specific hardware abstractions, making the manipulation and verification of their semantics scalable and white-box. In particular, I aim to (1) formalize higher-level abstractions to guide lower-level verification efforts, (2) verify transformations from several high-level level abstractions to low-level ones (e.g. in the CIRCT ecosystem), and (3) explore domain-specific decision procedures and reasoning frameworks for verifying higher-level abstractions, such that the verification evolves as the design is lowered. This work will make hardware verification accessible in the early stages of the design process in a framework that can take advantage of different domain-specific abstractions and design paradigms.

1 INTRODUCTION

Designing modern hardware is incredibly complex, even more so as the demand for domain-specific architectures increases to compensate for the decline of general-purpose ones [5, 18]. Verification plays a crucial role in the process, as it can uncover expensive and potentially critical mistakes before the tape-out: yet, it also represents a bottleneck for the design procedure [15, 16, 30]. To address these challenges, the CIRCT project [2, 14] introduces a novel concept of hardware compiler that supports different frontends and backends and comprises various domain-specific abstractions (*dialects*), making the compilation of a design progressive, through different lowerings and dialects. Thanks to tailored domain-specific abstractions, CIRCT allows designers to use local reasoning to analyze and optimize the numerous domain-specific concepts it models. For example, in CIRCT [2, 14], high-level abstractions such as FSMs or latency-insensitive Intermediate Representations (IRs) are optimized and progressively lowered to Register Transfer Level (RTL) and Verilog, generating production-quality, optimized code [3]. Besides making hardware design more flexible, the high-level abstractions CIRCT exposes also have great potential for verification. Exploiting higher-level abstractions to bring verification tasks to a higher abstraction level and complement lower-level verification tooling already proved a successful approach [19, 20, 23]. CIRCT represents an excellent infrastructure to expand and make the most of this approach, given the domain-specific abstractions it includes.

Overall, **bridging domain-specific abstractions and verification tooling** is a fundamental step to make sure that verification is up-to-speed with the recent domain-specific hardware design trends, and CIRCT is the perfect environment to validate this approach, thanks to the numerous abstractions it provides. However, taking advantage of high-level CIRCT IRs remains a complex task due to the lack of precise semantics and documentation. In particular, the most critical challenges involved in the formal verification of the semantics of such diverse dialects and domain-specific abstractions are (1) precisely defining the semantics of the dialects relying on little-to-no documentation, (2) identifying and using effective automatic reasoning tools for the formalization, (3) managing the semantics evolution during the lowering through different abstractions.

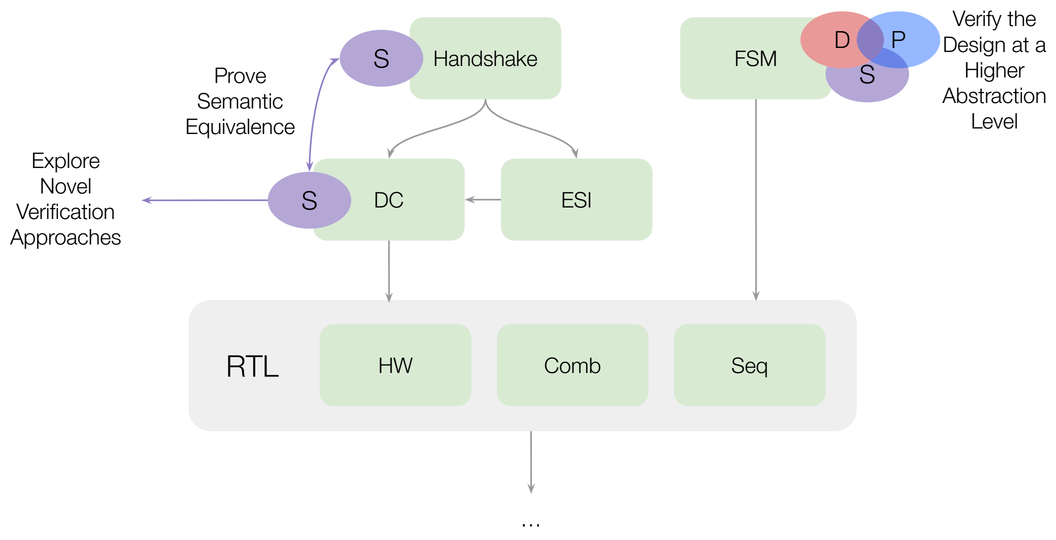
S
Handshake
FSM
D
P
S
Verify the Design at a Higher Abstraction Level
Prove Semantic Equivalence
Explore Novel Verification Approaches
S
DC
ESI
RTL
HW
Comb
Seq
...

Figure 1: Formalizing the semantics (S) of CIRCT dialects in Lean4 will allow us to (1) verify the compilation and equivalence between different abstractions, (2) explore novel verification approaches in a verified environment, (3) verify whether a property (P) holds for a design (D) at a certain level.

With this work, I intend to tackle these challenges, combining my experience with CIRCT and with the Lean4 Interactive Theorem Prover (ITP) to expose the semantics of high-level abstractions and improve the scalability and accessibility of verification, targeting the objectives in Figure 1.

2 RELATED WORK

Previous works tackled the formalization and definition of semantics in hardware description languages, considering different levels of abstraction [24]. Managing the different design paradigms on which domain-specific abstractions rely is incredibly challenging in this context. Promising efforts also exist bridging hardware compilers, and CIRCT specifically, with formal methods. However, these are currently limited in exploiting high-level abstractions due to the complexity and low flexibility of the verification tools they rely on, for example, SAT and SMT solvers. Lean4 is currently the best option for reasoning about the semantics of CIRCT-like infrastructures, as proven by the Lean-MLIR [9] framework, as it provides a flexible and verified environment to encode and manipulate domain-specific dialects.

Koika [11] directly derives from Bluespec [28] and features novel and deterministic operational semantics, including a verified compiler to circuits. In particular, Koika focuses on the importance of cycle-accurate description (and semantics) when dealing with the performance of circuits without compromising on the straightforward description of their functional properties. The authors also highlight how traditional HDLs and HLS compilers lack precise semantics, making describing complex interactions between different modules difficult.

ReWire [29] introduces a functional programming language with a compiler that translates the high-level description of a design into working hardware relying on the language's semantics to describe its behaviour.

Chisel [6] is another powerful structural language embedded in Scala, also used as a frontend for CIRCT.

Bernstein et al. [8] propose a description of hardware semantics at different abstraction levels within an abstract interpretation framework, mainly focusing on the representations used in the early phase of the design when reasoning at the latency-insensitive level of the hardware representation.

Another work exploits ACL2 [20] to encode the semantics of FSMs for the verification of hardware designs, extracting said FSMs from lower-level representations. This work is deeply inspiring for my PhD since CIRCT naturally exposes the FSM representation of its designs.

Managing the evolution of domain-specific semantics where different abstractions meet, e.g. latency-insensitive and cycle-accurate ones, is a core challenge of my work. In particular, the semantics of latency-insensitive abstractions and their relationship to cycle-based representations are incredibly challenging. In this context, Carloni et al. [12] introduce a theory to describe and represent latency-insensitive circuits and systems. The idea is to represent complex systems considering their single functional components and communication according to a specific protocol. The theory introduced allows for the composition of such single components into a complex system to satisfy synchronization and communication properties.

Due to the numerous abstractions CIRCT exposes, its correctness guarantees require careful reflection on the semantics of the various IRs involved and the lowerings between them. A first recent work [32] provided a preliminary formalization of CIRCT dialects, only focusing on lower-level IRs.

Within CIRCT, `circt-lec` and `circt-bmc` are other tools that perform logical equivalence checking and bounded model checking, respectively, at the RTL level. CIRCT also features different lowerings to standard hardware verification formats, such as `btor2` [26, 27], `aig` [10] and `SMT-LIB` [7], that contribute to integrating verification within the hardware design process.

Moreover, the SMT dialect represents one means of formalizing dialects' semantics at different levels. This dialect comprises most SMT-LIB [7] operators. It is the endpoint of various lowerings from different dialects, meaning that their semantics are encoded in SMT-LIB for verification purposes. The dialect is part of the compiler verification and optimization efforts currently under development in Prof. Grosser's research group at the University of Cambridge, with whom I have worked for the past year. However, the underlying complexity of SMT solvers such as z3 [13] and their low flexibility make it hard to embed such domain-specific information into the actual SMT model. Lean4 can tame this complexity: it is an open-source programming language and theorem prover providing a flexible environment to write verified code.

Works such as Lean-MLIR [9] have already proved the benefits of embedding MLIR [21] dialects' semantics in Lean4. This framework can also be game-changing in expanding CIRCT verification tooling. Lean4 enables writing theorems that encode certain domain-specific behaviours and successfully guide the verification with significantly higher trust than standard verification tooling. We typically need to trust SMT and SAT solvers when verifying a design. For instance, consider z3 [13]: it consists of a vast C++ codebase, it is not verified, and these characteristics make extending its tactics and logic incredibly hard, as shown, for example, by previous efforts to expand it for Constrained Horn Clauses (CHCs) reasoning [17]. As a consequence, optimizing the job of the SMT solver, for example, by introducing tactics that exploit domain-specific knowledge available at higher abstraction levels, is not a practically viable solution. On the contrary, Lean4 has already proved very effective in building white-box automation techniques for bitvector manipulation (to which I contributed) and verification. In particular, the `bv_decide` tactic [1] represents a first effort in this space, implementing a verified bit-blaster and LRAT checker. This fact demonstrates the potential of Lean4 in the verification field and the community's interest in working towards this direction. Moreover, thanks to its flexibility, Lean4 represents the perfect environment to experiment with further verified verification methods such as abstract interpretation.

3 RESEARCH DIRECTION AND CHALLENGES

During my PhD, I aim to enhance verification efforts for the CIRCT infrastructure by formally verifying the semantics of its dialects to enable the verification of lowerings and optimizations, the progressive verification of designs' properties, and the adoption of novel verification techniques (Figure 1). My work combines CIRCT's high-level information at different abstraction levels with Lean4's flexibility and minimal trusted codebase. Until now, verifying high-level abstractions with standard lower-level hardware verification techniques (such as assertion-based verification or SMT solvers) has been very complex [4, 31], due to the scarce flexibility of the tools involved (e.g. z3 [13]) and the limited number of abstractions they support. With this work, I aim to formalize the high-level IRs CIRCT exposes to bring verification to a higher abstraction level and complement the effort of lower-level, classical verification approaches, by relying on the flexibility of Lean4.

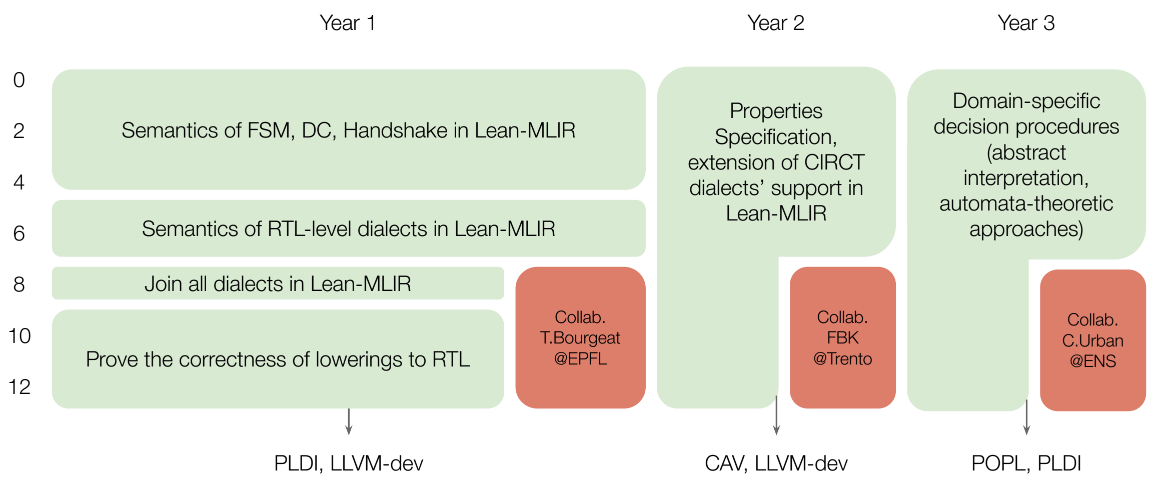

Figure 2: I will spend the first four months of the PhD implementing in Lean4 the semantics of three high-level dialects (FSM, Handshake, DC) and studying the state-of-the-art works concerning the semantics of these domain-specific representations. Then, I plan to spend three more months implementing the semantics of the CIRCT dialects necessary for the RTL-level representation (Hardware, Comb, Seq). During the following month, I will work on combining these dialects in the Lean-MLIR framework to ensure the correctness of the overall semantics and their flexibility and functionality. During the last four months of the first year, I will prove the correctness of the lowerings from the higher-level abstractions to RTL, collaborating with Thomas Bourgeat's research group at EPFL. The second year will focus on implementing the property specifications in Lean-MLIR. I also aim to visit Fondazione Bruno Kessler (FBK) and the University of Trento. I will investigate alternative decision procedures and verification frameworks for domain-specific representations during the last year, collaborating with Caterina Urban's research group at ENS. PLDI, POPL, and CAV are the leading conferences I intend to target, in addition to LLVM developer meetings. Besides these three years, I plan to spend 6 additional months working on my thesis.

Following the path indicated by Lean-MLIR [9], I intend to formalize more high-level abstractions and dialects, starting from the FSM dialect and also considering those relying on radically different design paradigms from traditional imperative or functional IRs, such as the Dynamic Control (DC) dialect. This aspect will require careful mathematical modelling of complex, dataflow-like behaviour to fill gaps in the informal model, e.g. where latency-insensitive and circuit-based semantics meet. Moreover, the formalization of dialects within Lean4 will also enable the verification of compiler passes, offering further guarantees for the correctness of CIRCT itself.

Overall, Lean4 is a fertile, open-source environment to work on white-box verification frameworks thanks to its flexibility, versatility, and trust level. Moreover, working in an open-source environment is an incredibly valuable aspect of this work, as it significantly increases the quality of the output - thanks to peer-review processes - and creates impact, being available to numerous users.

Besides introducing a powerful means to reason about dialects' semantics and the overall correctness of the compilation and the designs, this work will also lay the foundations for including higher-level verification techniques into CIRCT, for example, automata-theoretic and abstract-interpretation-based approaches. Formalizing the semantics of CIRCT dialects allows semantics manipulation up to a point where the application of domain-specific verification approaches and decision procedures is possible and beneficial. Exploring different approaches in the context of hardware verification is currently a challenge [22, 25], requiring significant effort to bridge the hardware level with a suitable abstraction that alternative verification methodologies, such as abstract interpretation frameworks, can digest. Nevertheless, preliminary studies suggested the effectiveness of these methodologies, especially at specific abstraction levels [8]. Formalizing the semantics of CIRCT dialects in an ITP is a game-changer in this context, as it proves a stable foundation to explore many verification strategies in a white-box way, by simply adding theorems and tactics.

Overall, formalizing the semantics of a large subset of CIRCT's dialects is a first step towards integrating the EDA toolchain with automatic verification, which is simultaneous and progressive with the design procedure and comprises different verification techniques. While this aspect can increase the compilation time for a single design iteration, its correctness guarantees can significantly reduce the number of iterations required to generate the desired design, eventually improving the overall design pipeline. Moreover, this work paves the way for filling and reasoning about subtle semantics gaps in hardware abstractions, by introducing formally verified semantics in the CIRCT ecosystem.

4 RESEARCH ORGANIZATION

This research proposal comprises three milestones, which I plan to distribute in the three years of the PhD, as Figure 2 summarizes:

(1) The goal of the first year is to **formalize the semantics of a first subset of high-level CIRCT dialects** (namely FSM, Handshake and DC) and **verify their lowering to RTL level within Lean-MLIR**. This phase will require careful study of existing works, especially concerning the formalization of latency-insensitive behaviours in DC and their interface with cycle-accurate representation. Then, I will implement the semantics of the CIRCT dialects necessary for the RTL-level representation (Hardware, Comb, Seq) and spend some time combining all these dialects in Lean-MLIR. As a final task for the first year, I will prove the correctness of the lowerings from the higher-level abstractions to RTL, including some of the optimizations involved. I will target PLDI to publish the formally verified semantics resulting from this work and present it at LLVM-dev meetings. Thomas Bourgeat and his research group, with whom I am already working, represent an excellent collaboration opportunity at this stage of the PhD because of their expertise in hardware semantics definition and verification with ITPs.

(2) **Bringing designs' verification at a higher abstraction level, exploiting different dialects and their formalized semantics**. At this point, a crucial step is understanding how to correctly specify relevant properties, which will require thorough readings in the state of the art. Moreover, I plan to add further dialects to the Lean-MLIR framework during the second year. During the second year, I intend to collaborate with Stefano Tonetta's research group at Fondazione Bruno Kessler (FBK) due to their expertise in developing verification techniques and solvers. CAV is the ideal conference to publish the output of this second year.

(3) **Explore different verification strategies**, such as automata-theoretic approaches and abstract interpretation frameworks, to fully take advantage of the formalized semantics. During the last year, I plan to take advantage of the effort put into effectively formalizing CIRCT dialects. I will investigate how other verification techniques can take advantage of the formalized semantics to improve the guarantees concerning the design's behaviour, aiming to reduce the number of necessary iterations. To work on these topics, I intend to collaborate with Caterina Urban's research group at ENS, given the high-impact work they have carried out in abstract interpretation and semantics analysis. PLDI and POPL are the conference targets for publishing the output from this last year.

(4) I plan to spend six additional months writing the **PhD thesis**, which will focus on how CIRCT and Lean4 are a unique combination of tools to make hardware verification flexible and extensible, bending its tools to fit different designs and techniques.

5 CONCLUSION

This proposal targets the formalization of CIRCT dialects semantics to improve the scalability and flexibility of verification, by manipulating higher-level hardware abstractions in the Lean4 ITP, in order to (1) make white-box verification accessible at higher abstraction levels (2) verify the compiler, its lowerings and optimizations (3) explore domain-specific verification techniques and procedures, whose adoption for hardware has been incredibly complex until now. This work is a starting point to becoming an expert in hardware verification, building valuable connections in the community and laying the foundations for the research work to come as I continue my career in academia and pursue my dream of becoming a professor.

REFERENCES

[1] bv_decide. https://github.com/leanprover/lean4/, Online.

[2] Circt. https://circt.llvm.org/, Online.

[3] Circt for sifive. https://github.com/sifive/chisel-circt, Online.

[4] Symbiyosys (sby) – front-end for yosys-based formal verification flows. Accessed 24/11/21.

[5] Asanovic, K., Bodik, R., Catanzaro, B., Gebis, J., Husbands, P., Keutzer, K., Patterson, D., Plishker, W., Shalf, J., and Williams, S. W. The landscape of parallel computing research: A view from berkeley.

[6] Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., Wawrzynek, J., and Asanović, K. Chisel: constructing hardware in a scala embedded language. In *Proceedings of the 49th Annual Design Automation Conference* (New York, NY, USA, 2012), DAC '12, Association for Computing Machinery, p. 1216–1225.

[7] Barrett, C., Stump, A., Tinelli, C., et al. The smt-lib standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)* (2010), vol. 13, p. 14.

[8] Bernstein, G. L., and Ragan-Kelley, J. What are the semantics of hardware. In *Workshop on Languages, Tools, and Techniques for Accelerator Design (LATTE)* (2021).

[9] Bhat, S., Keizer, A., Hughes, C., Goens, A., and Grosser, T. Verifying peephole rewriting in ssa compiler irs. *arXiv preprint arXiv:2407.03685* (2024).

[10] Biere, A., Heljanko, K., and Wieringa, S. AIGER 1.9 and beyond. Tech. Rep. 11/2, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2011.

[11] Bourgeat, T., Pit-Claudel, C., Chlipala, A., and Arvind. The essence of bluespec: a core language for rule-based hardware design. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (2020), pp. 243–257.

[12] Carloni, L. P., McMillan, K. L., and Sangiovanni-Vincentelli, A. L. Theory of latency-insensitive design. *IEEE Transactions on computer-aided design of integrated circuits and systems 20*, 9 (2001), 1059–1076.

[13] De Moura, L., and Bjørner, N. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems* (2008), Springer, pp. 337–340.

[14] Eldridge, S., Barua, P., Chapyzhenka, A., Izraelevitz, A., Koenig, J., Lattner, C., Lenharth, A., Leontiev, G., Schuiki, F., Sunder, R., Young, A., and Xia, R. MLIR as Hardware Compiler Infrastructure. In *WOSET '21: Workshop on Open Source EDA Technology* (2021), SiFive.

[15] Foster, H. The 2020 Wilson Research Group Functional Verification Study. Tech. rep., Siemens, 2020.

[16] Foster, H. D. Why the design productivity gap never happened. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2013), IEEE, pp. 581–584.

[17] Gurfinkel, A. Program verification with constrained horn clauses. In *International Conference on Computer Aided Verification* (2022), Springer, pp. 19–29.

[18] Hennessy, J., and Patterson, D. A new golden age for computer architecture: domain-specific hardware/software co-design, enhanced. In *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)* (2018).

[19] Huang, B.-Y., Zhang, H., Subramanyan, P., Vizel, Y., Gupta, A., and Malik, S. Instruction-level abstraction (ila) a uniform specification for system-on-chip (soc) verification. *ACM Transactions on Design Automation of Electronic Systems (TODAES) 24*, 1 (2018), 1–24.

[20] Hunt Jr, W. A., and Reeber, E. A sat-based procedure for verifying finite state machines in acl2. In *Proceedings of the sixth international workshop on the ACL2 theorem prover and its applications* (2006), pp. 127–135.

[21] Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N., and Zinenko, O. Mlir: A compiler infrastructure for the end of moore's law, 2020.

[22] Malik, S. Hardware verification: Techniques, methodology and solutions. In *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings 14* (2008), Springer, pp. 1–1.

[23] Mattarei, C., Mann, M., Barrett, C., Daly, R. G., Huff, D., and Hanrahan, P. Cosa: Integrated verification for agile hardware design. In *2018 Formal Methods in Computer Aided Design (FMCAD)* (2018), IEEE, pp. 1–5.

[24] Melham, T. F. Abstraction mechanisms for hardware verification. In *VLSI Specification, Verification and Synthesis*. Springer, 1988, pp. 267–291.

[25] Mukherjee, R., Kroening, D., and Melham, T. Hardware verification using software analyzers. In *2015 IEEE Computer Society Annual Symposium on VLSI* (2015), IEEE, pp. 7–12.

[26] Niemetz, A., Preiner, M., Wolf, C., and Biere, A. Btor2 , btormc and boolector 3.0. In *International Conference on*

Computer Aided Verification (2018).

[27] Niemetz, A., Preiner, M., Wolf, C., and Biere, A. Btor2 , btormc and boolector 3.0. In *Computer Aided Verification* (Cham, 2018), H. Chockler and G. Weissenbacher, Eds., Springer International Publishing, pp. 587–595.

[28] Nikhil, R. Bluespec system verilog: efficient, correct rtl from high level specifications. In *Proceedings. Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. MEMOCODE '04.* (2004), pp. 69–70.

[29] Procter, A., Harrison, W. L., Graves, I., Becchi, M., and Allwein, G. Semantics driven hardware design, implementation, and verification with rewire. In *Proceedings of the 16th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems 2015 CD-ROM* (2015), pp. 1–10.

[30] Vasudevan, S., Jiang, W. J., Bieber, D., Singh, R., Ho, C. R., Sutton, C., et al. Learning semantic representations to verify hardware designs. *Advances in Neural Information Processing Systems 34* (2021), 23491–23504.

[31] Witharana, H., Lyu, Y., Charles, S., and Mishra, P. A survey on assertion-based hardware verification. *ACM Computing Surveys (CSUR) 54*, 11s (2022), 1–33.

[32] Zhao, J., Kang, J., and Zhao, Y. K-circt: A layered, composable, and executable formal semantics for circt hardware irs. *arXiv preprint arXiv:2404.18756* (2024).