Democratizing hardware verification by exploiting semantics for high-level abstractions

LUISA CICOLINI

Hardware verification is a fundamental step of the hardware design process, preventing expensive and potentially critical mistakes. As the complexity of hardware designs, so does the need for more efficient verification strategies that can make the overall design procedure smoother. In this context, exploiting higher-level hardware representations can be a game changer in verifying the designs progressively during its lowering and explore different verification approaches that do not work with with lower-level hardware representations. However, efficiently extracting such high-level information remains a challenge due to the lack of formalized semantics for high-level abstractions, which often rely on very specific structures, for example, Finite State Machines (FSMs). With this work, I aim to build the necessary infrastructure that can exploit such information, by formalizing the semantics of different hardware abstractions within the CIRCT ecosystem, to (1) extract relevant higher-level information that can guide lower-level verification efforts, (2) verify (parts of) the CIRCT compiler itself, (3) explore different approaches to the verification of higher-level representation, thus distributing verification efforts throughout the overall design procedure.

1 INTRODUCTION

Hardware design is inherently complex, requiring a thorough understanding of numerous components and tools. Verification is crucial in the design process as it can uncover expensive and potentially critical mistakes before the tape-out. The CIRCT[1, 10] compiler framework introduces several dialects representing domain-specific Intermediate Representations (IRs), allowing the progressive compilation of high-level abstractions, such as Finite State Machines, into lower-level representations, such as Verilog, via different lowerings. This approach proved successful in fostering the reusability of the code and the introduction of specific optimizations at different abstraction levels, relying on the progressive lowering of the code. Being able to optimize and reason on a design at various abstraction levels is especially appealing from a hardware designer's perspective. Moreover, CIRCT's higher-level abstractions provide valuable information about the design's structure, which is helpful to perform preliminary verification at a higher level, verify the compilation itself, and extract useful information that can serve lower-level verification tools[9, 12, 14].

Taking advantage of this aspect is fundamental to building a verification toolchain that fully exploits the unique characteristics of the CIRCT ecosystem, allowing designers to leverage the expressive power of different representations, for example, dataflow and FSM. However, extracting useful high-level information remains a complex task due to the scarce formalization and the lack of precise semantics for CIRCT dialects. Overall, accurately formalizing the semantics of such diverse dialects and abstractions remains a significant challenge and is critical for verifying the correctness of the designs, optimizations and lowerings involved in the compilation.

2 RELATED WORK

Due to the complexity of its environment, correctness guarantees in CIRCT require careful reflection on the semantics of the various IRs involved and the lowerings between them. To tackle this problem, CIRCT features different lowerings to standard hardware verification formats, such as `btor2`[17, 18], `aig`[6] and `SMT-LIB`, that contribute to integrating verification within the hardware design process. In particular, the SMT dialect represents one means towards formalizing dialects' semantics at different levels. However, the underlying complexity of SMT makes it hard to embed higher-level information into the actual SMT model[11]. `circt-lec` and `circt-bmc` are other tools that perform logical equivalence checking and bounded model checking, respectively, at the Register Transfer Level (RTL) level. A recent work[**?**] provided another formalization of CIRCT dialects, only focusing on lower-level IRs. While the current infrastructure features various efforts toward the verification of the CIRCT toolchain and its design, none of these take full advantage of CIRCT's high-level abstractions and eventually always fall back to lower-level representations.

Given the paramount importance of verification in hardware design, numerous works tackled the formalization and definition of semantics in hardware description languages, considering different levels of abstraction[15]. Koika[7] directly derives from Bluespec[19] and features novel and deterministic operational semantics, including a verified compiler to circuits. This work highlights 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. In particular, the authors highlight how traditional HDLs and HLS compilers lack precise semantics, making it especially hard to describe complex interactions between different modules. ReWire[20] 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[3] is another powerful structural language embedded in Scala, also used as a frontend for CIRCT. Finally, Bernstein et al. [4] suggest exploiting abstract interpretation to formalize and represent different hardware abstractions. This work also considers the differences in event-based semantics (e.g. the ones commonly used for dataflow abstractions) and semantics derived from sequential code description languages (e.g. in HLS workflow). Different abstractions require focusing on different aspects of the semantics. To deal with these differences, the authors use abstract interpretation to describe the semantics at different levels of abstraction, mainly focusing on the representations used in the early phase of the design when reasoning at the latency-insensitive level of the hardware representation. In particular, the semantics of latency-insensitive abstractions and their relationship to cycle-based representation are incredibly challenging. In this context, Carloni et al. [8] 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.

Overall, these works focus on precise abstraction levels - either low or high. Instead, the CIRCT compilation toolchain exposes different abstractions: reasoning about their semantics allows **taking advantage of CIRCT's flexible approach to hardware design**, complementing it with solid correctness guarantees concerning the designs and, ultimately, the compilation itself.

3 RESEARCH DIRECTION AND CHALLENGES

During my PhD, I aim to formalize the semantics of CIRCT dialects to complement the existing verification infrastructure, enabling the verification of compilation lowerings, optimizations, and the progressive verification of designs' properties. To address this challenge, I propose a systematic formalization of CIRCT dialects' semantics at different abstraction levels, paving the way for verifying the framework's lowerings and optimizations. Works such as Lean-MLIR[5] have already proved the benefits of embedding dialects' semantics in Lean4, an open-source programming language and theorem prover providing a flexible environment to write verified code. Pursuing this direction, I aim to formalize more high-level abstractions and dialects, starting from the Finite State Machine (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. Besides introducing a powerful means to reason about dialects' semantics and the overall correctness of the compilation, this work will also lay the foundations for including higher-level verification techniques into CIRCT, for example, those leveraging automata-theoretic and abstract-interpretation-based approaches.

This work takes advantage of CIRCT's great potential, lying in the high-level information available at different abstraction levels it provides. Until now, exploiting such information has been very complex, since encoding it in standard lower-level hardware verification techniques (such as assertion-based verification or SMT solvers) is very complex [2, 21]. With this work, the high-level information we extract by accurately formalizing dialects' semantics can contribute to guiding the effort of lower-level, classical verification approaches. Moreover, formalizing the semantics of CIRCT dialects allows semantics manipulation up to a point where the application of different verification approaches is possible and beneficial. Exploring different approaches in the context of hardware verification is currently a challenge, requiring significant effort to bridge the hardware level with a suitable abstraction these verification methodologies can digest[13, 16]. Nevertheless, preliminary studies suggested the effectiveness of these methodologies, especially at specific abstraction levels[4]. Formalizing the semantics of different abstractions represents a game-changer in this context. Finally, the formalization of dialects within an Interactive Theorem Proving (ITP) also enables the verification of compiler passes, offering further guarantees for the correctness of CIRCT itself. In particular, Lean4 provides a nurturing open-source environment to develop alternative verification frameworks due to its flexibility and versatility. 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 strict peer-review processes - and creates impact, being available to numerous potential users.

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. 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.

Overall, 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:

(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 step is fundamental for the verification of the entire compilation. To achieve this goal, I will spend the first four months of the PhD implementing in Lean4 the semantics of two high-level dialects (FSM, DC) and the optimizations they involve. 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. Subsequently, I will spend the next three months implementing the semantics of the CIRCT dialects necessary for the RTL-level representation (Hardware, Comb, Seq). One more month will be necessary to tackle the engineering aspects involved in combining all these dialects in the Lean-MLIR framework and ensuring they work flexibly and their integration is functional. I will spend the last four months of the first year proving the correctness of the lowerings from the higher-level abstractions to RTL, including the optimizations involved.

(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.

(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 further take advantage of

the formalized semantics to improve the guarantees concerning the design's behaviour, potentially reducing the number of necessary iterations. I will also devote part of the last year to writing the PhD thesis.

REFERENCES

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

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

[3] 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.

[4] 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).

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

[6] 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.

[7] 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.

[8] 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.

[9] Chen, J., and He, F. Leveraging control flow knowledge in smt solving of program verification. *ACM Transactions on Software Engineering and Methodology (TOSEM) 30*, 4 (2021), 1–26.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] Niemetz, A., Preiner, M., Wolf, C., and Biere, A. Btor2 , btormc and boolector 3.0. In *International Conference on Computer Aided Verification* (2018).

[18] 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.

[19] 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.

[20] 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.

[21] 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.