Extending High-Level Synthesis with AI/ML Methods

Invited Paper

Nicolas Bohm Agostini*
Ankur Limaye, Claudio Barone, Marco Minutoli,
Vito Giovanni Castellana, Joseph Manzano,
Antonino Tumeo
Pacific Northwest National Laboratory
Richland, WA, USA

Giovanni Gozzi, Michele Fiorito, Serena Curzel,
Fabrizio Ferrandi
Politecnico di Milano
Milano, Italy

ABSTRACT

Artificial Intelligence (AI) and Machine Learning (ML) methods offer significant opportunities to improve the quality of results in high-level synthesis (HLS). For instance, they can be used to model and predict metrics of the final design (e.g., area, considering aspects such as interconnect overhead for different device technologies), thereby facilitating exploration when searching for the best design trade-offs. Additionally, they can help identify hidden correlations across various phases of synthesis and the optimizations performed, enabling the identification of the most effective pipelines. Furthermore, these methods can greatly facilitate and enhance the design space exploration for the synthesis process in terms of both time and quality of results. This paper discusses the opportunities and challenges of augmenting HLS with AI/ML, using as an example the SODA Synthesizer, an open-source hardware generation toolchain that includes SODA-OPT, a hardware/software partitioning and pre-optimization tool developed with the MLIR framework, and PandA-Bambu, a state-of-the-art HLS tool. SODA interfaces with OpenROAD to provide a complete end-to-end toolchain.

KEYWORDS

High-level synthesis, hardware/software co-design

ACM Reference Format:
Nicolas Bohm Agostini, Ankur Limaye, Claudio Barone, Marco Minutoli, Vito Giovanni Castellana, Joseph Manzano, Antonino Tumeo, and Giovanni Gozzi, Michele Fiorito, Serena Curzel, Fabrizio Ferrandi. 2024. Extending High-Level Synthesis with AI/ML Methods: Invited Paper. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), October 27–31, 2024, New York, NY, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3676536.3689923

1 INTRODUCTION

The introduction of machine learning (ML) and artificial intelligence (AI) methods is profoundly changing electronic design automation (EDA). These data driven methods provide opportunities to greatly improve quality of results, reducing iteration times for EDA tasks and significantly speeding up design space exploration.

All major EDA vendors have integrated some level of AI/ML into their workflows. However, while these advancements represent a step toward bridging the hardware design gap, significant technical and commercial challenges still limit the impact and adoption of these new optimization techniques.

First and foremost, since AI/ML approaches are primarily data-driven, data generation and management are crucial. Proprietary solutions, while effective, may not always provide access to the data needed to train models. This limitation hinders the development of more detailed models and the ability to fine-tune them for specific cases. The integration of open-source end-to-end pipelines for hardware design tools [5], spanning from high-level specification to silicon implementation, offers access to large datasets at each stage of synthesis and optimization. Even enabling proprietary tools to connect with and exchange data with open-source solutions can significantly enhance the potential to identify novel and improved methodologies [13].

Secondly, most current approaches focus on optimizing the back-end stages of the EDA flow, such as logic synthesis, physical design, and layout. However, there is significant untapped potential in combining higher-level abstractions, like those used in high-level synthesis (HLS), with AI/ML methods. This integration could further enhance and accelerate design space exploration.

HLS[11], which involves generating hardware designs in hardware description languages (HDLs) from specifications in higher-level programming languages, is undergoing a renaissance as a method to greatly accelerate the hardware design process. This is particularly true when developing custom accelerators for novel applications, where hardware designers must rapidly explore multiple architectural solutions across various metrics. Several new tools[4, 24] have emerged, further raising the abstraction level from general-purpose languages like C and C++ to high-level programming frameworks in Python, driven notably by the need to explore domain-specific accelerators for AI/ML. By leveraging highly flexible and extensible compiler infrastructures, these tools can incorporate novel abstractions that facilitate design space exploration through the inclusion of annotations and insights derived from predictive supervised or unsupervised ML models. Additionally, while much of the focus on AI/ML in HLS has been on using large language models (LLMs) to generate HDLs from natural language descriptions, we believe there is even greater potential for Generative AI in hardware generation, particularly in enabling "co-pilot" agents to assist with various complex tasks.

*Also with Northeastern University.

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10…$15.00
https://doi.org/10.1145/3676536.3689923

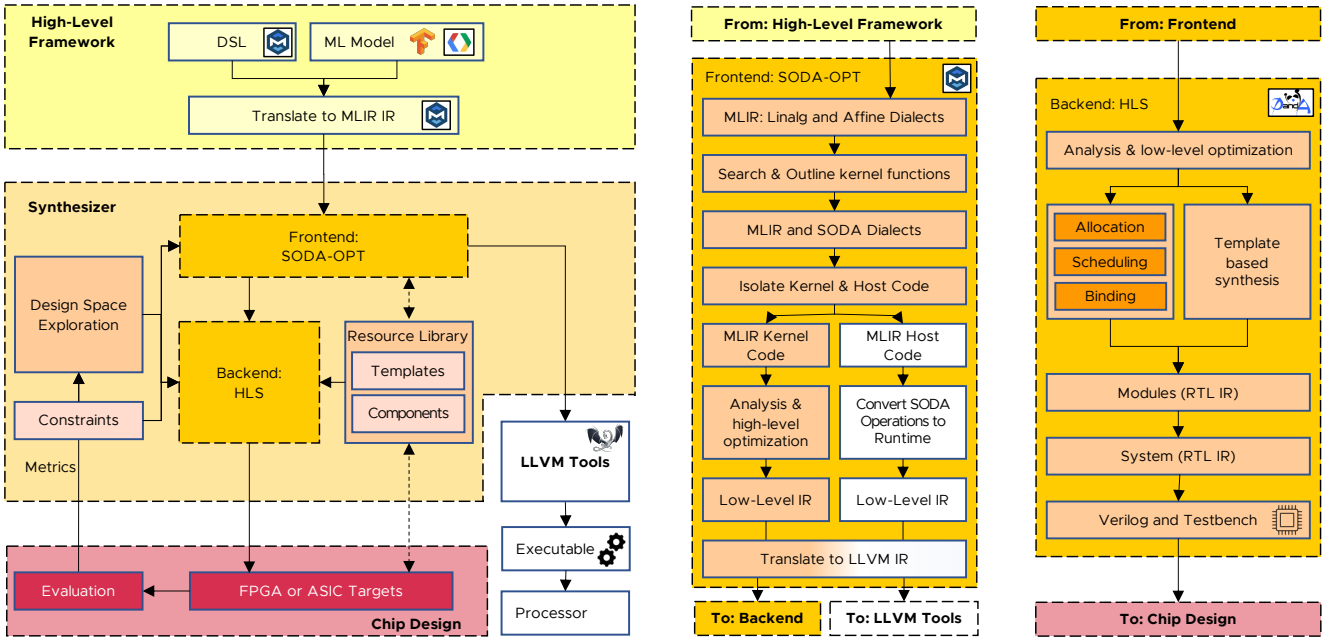

Figure 1: The SODA framework is an open-source, multi-level, modular, extensible, hardware generator composed of a high-level compiler and a lower-level HLS backend

In this paper, we first provide an overview of the SODA Synthesizer framework, an end-to-end open-source compiler that translates specifications written in high-level productive programming frameworks (e.g., Python) into silicon. We also discuss its integration with OpenROAD, an open-source suite of tools for lower-level logic synthesis and physical layout. Finally, we analyze the opportunities and challenges associated with enhancing this end-to-end toolchain using AI/ML methods.

2 THE SODA SYNTHESIZER

Figure 1a presents a high-level overview of the SODA Synthesizer framework. SODA consists of two distinct but interoperable components: SODA-OPT [4], a frontend compiler for system-level design and high-level optimizations, and Bambu [11], a state-of-the-art HLS tool from the PandA framework.

SODA-OPT is a tool developed using the Multi-Level Intermediate Representation (MLIR)[15] compiler framework, which enables SODA to process code written in high-level programming frameworks. While our current focus is on machine learning and data science[6] due to the rapid advancements in these fields, SODA-OPT can also interface with more general scientific computing frameworks and support the development of ad-hoc domain-specific languages as needed. SODA-OPT serves multiple functions. First, it performs hardware/software partitioning of the input program. Second, it applies architecture-independent optimizations by leveraging features of the MLIR framework, which are well-suited for compute- and data-intensive applications developed with functional languages and represented through computational graphs. Third, it carries out system-level and HLS-related optimizations that facilitate detailed design space exploration of hardware accelerators. SODA-OPT produces two types of LLVM intermediate representation (IR) as output. The first is an HLS-optimized LLVM IR for kernels intended to be synthesized into hardware. The second is LLVM IR representing the host program, including runtime calls to the accelerators, which can be executed on a general-purpose processor within a system-on-chip (SoC).

LLVM IRs representing kernels to be synthesized into custom accelerators are passed as input to an underlying HLS tool. As previously mentioned, our open-source toolchain utilizes Bambu for this purpose. Bambu generates register transfer level (RTL) code for the accelerator designs based on the LLVM IR. The optimizations performed at the MLIR level include typical software compiler optimizations, restructuring or modifying the IR, and inserting information to trigger specific synthesis methodologies and additional optimizations. The power, performance, and area (PPA) metrics of the generated hardware design are significantly influenced by the sequence of transformations and their parameters, providing a large design space to explore in order to identify suitable Pareto points. Bambu can generate RTL code optimized for field-programmable gate arrays (FPGAs) from various vendors or application-specific integrated circuits (ASICs), and interfaces with lower EDA tools to gather relevant metrics that can drive the synthesis algorithms.

2.1 SODA-OPT

Figure 1b) provides a detailed view of the SODA-OPT flow. By extending the MLIR framework, SODA-OPT can leverage many of the readily available abstractions and transformations. MLIR uses the concept of *dialects*—specialized, self-contained IRs that adhere to

MLIR's meta-IR syntax—to enable the development of reusable compiler infrastructure. These dialects and their associated transformations are designed to address specific code optimization challenges.

onventional methods used in HLS design flows often require substantial code modifications or depend on compiler directives provided through `pragma` annotations. These directives guide optimizations, such as adjusting the degree of parallelism by controlling loop unrolling factors, during the HLS process. However, these methods typically rely on the specifics of the HLS tool being used. In contrast, SODA-OPT employs a different approach by leveraging the semantic information embedded in context-specific MLIR dialects, which allows for the automatic application of high-level transformations during the preparation of the input program for hardware synthesis. Additionally, SODA-OPT can automatically substitute and map calls to HLS libraries, rather than depending on mapping to HLS "code" templates written in C.

SODA-OPT offers a suite of compilation passes designed to *S*earch, *O*utline, *Optimize*, *D*ispatch, and *A*ccelerate computational kernels starting from specifications in high-level frameworks. Central to SODA-OPT is the definition of the `soda` dialect, a custom MLIR dialect that enables the automatic partitioning of the input application into a host program, which orchestrates runtime execution, and custom hardware accelerators [4]. Given a custom accelerator setup, SODA-OPT automatically generates the host driver code (for data movement and execution) based on the source application, identifying optimal data layouts and trade-offs to overlap computation with communication [1].

The SODA-OPT workflow begins by analyzing the MLIR input to identify code regions suitable for acceleration (search). These regions are then extracted into separate MLIR modules (outline), which are subsequently processed through SODA-OPT 's optimization passes. By leveraging the modularity of the MLIR framework, SODA-OPT can incorporate optimizations from existing MLIR dialects, such as `linalg` and `affine`, which are part of the LLVM compiler infrastructure, as well as from externally provided dialects and optimizations. For example, SODA-OPT utilizes the `linalg` and `affine` dialects to identify operators and perform loop optimizations, thereby improving the performance and efficiency of the synthesized hardware accelerators.

Since MLIR simplifies the support and integration of domain-specific languages (DSLs) into the compilation flow, various machine learning frameworks (e.g., TensorFlow, ONNX-MLIR, and TORCH-MLIR), scientific computing frameworks (e.g., NPCOMP), and even general-purpose programming languages (e.g., the FLANG Fortran compiler) are developing MLIR dialects, transformations, and lowering passes. This enables compiler-related optimizations for both existing and new dialects. Consequently, SODA-OPT can directly interface with all frameworks that lower to dialects included in the MLIR distribution, facilitating support for new high-level frontend languages.

2.2 PandA-Bambu

Figure 1c illustrates the key conceptual tasks performed by Bambu, the state-of-the-art open-source HLS tool from the PandA framework. Bambu, used by the SODA framework, generates RTL code for accelerators based on the LLVM IR produced by SODA-OPT.

Bambu can accept input from several general-purpose programming languages by leveraging the two de facto standard open-source compiler frontends: GCC and Clang. It can also process LLVM IR through a Clang plug-in, which translates LLVM IR into Bambu's own intermediate representation (IR). By using a Clang plug-in, Bambu avoids issues related to LLVM IR versioning; instead of requiring an IR down-grader, the plug-in only needs to be updated to support new LLVM IR features. This capability allows Bambu to seamlessly interface with the MLIR framework, which can output LLVM IR, and consequently with SODA-OPT. Bambu ingests LLVM IR generated after SODA-OPT's high-level optimizations for HLS, leading to more efficient accelerators compared to those synthesized from general-purpose programming languages or the default MLIR optimization flow. Bambu performs the necessary HLS steps—such as bitwidth analysis, loop optimizations, resource allocation, scheduling, and binding algorithms—on its specialized IR and can generate RTL designs in both Verilog and VHDL. Additionally, it can automatically produce testbenches for verification alongside the synthesizable RTL code.

As previously highlighted, Bambu generates RTL that can target both FPGAs and ASICs. Although the generated RTL is primarily behavioral, Bambu's flexibility allows for the specialization of components within its resource library and of synthesis steps based on a specific target device. This includes integrating device-specific macros and performing resource library characterization. The addition of technology-specific interconnect models can significantly enhance the quality of the generated designs. The characterization process provides valuable information such as area, delay, and power for each element in the resource library. This information is then used during the HLS steps to meet constraints (such as area and frequency) while optimizing metrics like overall execution latency. Bambu's resource characterization tool is named *Eucalyptus* [2]. The release version of Bambu includes out-of-the-box the characterization for Xilinx, Altera, Lattice, and NanoXplore FPGAs, as well as for the FreePDK 45 nm and ASAP 7 nm [9] predictive Process Development Kits (PDKs) through the OpenROAD[14] EDA flow. Additionally, Bambu includes functionality to generate scripts that invoke the related logic synthesis and place-and-route flows.

In addition to ASIC support, we have significantly expanded Bambu's synthesis methodologies over the years. While Bambu defaults to generating RTL designs based on the finite state machine with datapath (FSMD) model, we have developed new methodologies to support parallel accelerator designs, which can also be triggered through SODA-OPT. These methodologies include integrating FSMD accelerators as processing elements in coarse-grained dataflow designs [7, 10] or in high-throughput, dynamically scheduled, multithreaded parallel templates [12, 19]. A key feature of Bambu is its support for modular HLS [20]: unlike other tools, it can generate modules representing functions that can be reused or replicated throughout an entire design and composed into complex multi-accelerator systems. Additionally, we have recently focused on improving the generation of memory subsystems and the orchestration of data movement across accelerators and memory to better target data-intensive applications and chiplet-based designs [17].

3 OPPORTUNITIES FOR AI/ML

Several studies have begun to explore ML-based approaches, particularly graph neural networks (GNNs), to identify correlations between parts of the IR and potential code optimizations. These approaches typically involve code restructuring according to known patterns and/or the automatic addition of pragma directives [22]. While these solutions show promise in simplifying the creation of input code for HLS tools, they primarily offer improvements in code translation, though these improvements can have a significant impact on the quality of results.

A modular open-source hardware generation tool based on compiler technology and high-level synthesis is crucial for providing the infrastructure needed to integrate AI/ML methods across the entire hardware generation pipeline. By exposing all potential optimization options and parameters for frontends, HLS tools, and physical design tools, it becomes possible to apply black-box optimization to identify relationships and order among these optimizations. Additionally, as highlighted in the introduction, the open-source nature of these tools enables the collection of data necessary for training models and evaluating results at each stage of the hardware design toolchain. For instance, CircuitOps [16] offers a data framework to organize and store data generated by OpenROAD for AI/ML optimizations. The open-source nature of the tools also facilitates the implementation of appropriate data-retrieval application programming interfaces (APIs). Moreover, intermediate information from each tool in the flow is readily accessible and not restricted by proprietary constraints [13].

These solutions have been successfully demonstrated, for example, in optimizing IR drop and reducing iterations by estimating the feasibility of local routing from global routing information [8], thereby minimizing the need to revert to netlist generation. We see potential in extending this approach to higher layers of the synthesis toolchain. By enabling data collection at higher levels of abstraction, such as frontend optimizations, HLS resource allocation, operation scheduling, and resource binding, it becomes possible to develop unsupervised machine learning approaches that identify patterns across various HLS steps and the physical design and layout phases. This could, in turn, drive design space exploration towards promising design points with fewer iterations. Additionally, this integration allows for the accurate gathering of PPA metrics for specific ASIC technology nodes. This, in turn, enables the development of precise analytical models to predict area using resource library characterization information and more detailed interconnect data for ASIC technologies. Furthermore, it permits the implementation of more refined (semi-)supervised machine learning approaches that can predict the effects of typical code-level HLS optimizations on physical layouts. With flexible compiler abstractions such as those provided by MLIR, it is also possible to integrate such metrics as annotations to dialects or even define entirely new dialects that could embed layout information for resources and interconnect (wiring and interfaces across components).

As previously highlighted, generative AI and LLMs are of intense interest to the EDA community. Several works (e.g., ChipChat) [3] are exploring how LLMs might be used to generate RTL code from textual prompts, aiming to partially replace the HLS process. While these approaches seem effective for relatively small combinatorial circuits, significant challenges remain in managing large and complex designs. However, there are numerous other areas where generative AI could enhance HLS and hardware generation. One specific focus is developing co-pilots that empower domain scientists to optimize designs across various metrics and constraints. Currently, co-pilots based on foundational EDA models, such as ChipNemo [18], primarily assist with physical design and layout tools (from Verilog to GDS-II) and are mostly targeted at hardware designers. Often, a domain scientist might not fully understand the chip design constraints they can target, such as frequency or area, and may only have a high-level performance goal (e.g., latency for executing inference on a single input or a batch of inputs in ML) for a pre-defined target device. Instead, they might possess deep knowledge of domain-specific ML approaches required to accurately model the project and the mathematics needed for computations. Therefore, envisioning co-pilots trained across the entire end-to-end hardware generation toolchain could greatly simplify iterative design refinements and provide natural language feedback. Another clear opportunity for generative AI in HLS is verification and testbench generation [23]. Creating tests for hardware designs is complex due to the need for relevant testbenches and test vectors. Testability approaches often require verifying designs both in isolation and as part of larger communicating modules, using appropriate hierarchical methods. Generative AI could offer efficient methods for generating inputs and testbenches for various configurations, while also accounting for hierarchical structures.

4 POTENTIALS OF THE APPROACH

present comparisons between hand-designed ML accelerators and matrix multiplication units generated with the SODA framework [5, 10, 25], for variable precision and 32-bit floating point (FP32) precision, respectively. One advantage of the synthesis flow is its capability to explore different precisions through quantization and re-synthesis of designs with narrower datapaths and specialized functional units. For SODA, we report results using both open-source EDA tools with predictive open-source PDKs, as well as commercial tools with commercial technology nodes. The tables estimate the benefits that could be achieved by enhancing the design space exploration through AI/ML-based optimizations. In a previous paper [2], we demonstrated the results of Bambu's ASIC characterization using the ASAP7 PDK through OpenROAD. We observed that for PolyBench [21], the requested frequency constraint was not always met when designs were automatically optimized for performance (latency in terms of execution time). We identified congestion, and the lack of detailed interconnect modeling for ASICs, as the primary causes of the issue. Bambu's synthesis and optimization approach involves increasing overall execution latency (in number of clock cycles) through additional pipelining to meet the frequency constraints, but failed in those cases due to inadequate modeling of the target technology. We anticipate that improved and faster modeling will address these issues. Additionally, better utilization of layout information at higher levels of abstraction should reduce congestion before actual synthesis, thereby further improving quality of results at its end. Effective

Platform	Technology	Precision	Power [W]	Clock [MHz]	GFLOPS/W	Notes
A100 GPU Tensor Core	7nm	TF32	400	1410	780	Theoretical peak (with sparsity)
A100 GPU Tensor Core	7nm	INT8	400	1410	3120	Theoretical peak (with sparsity)
TPU v3	16 nm	FP16	450	940	273.33	Theoretical peak
TPU v4	7 nm	BF16	175	N/A	1432.29	Theoretical peak
SIGMA Sparse	28 nm	FP16	22.3	500	480	Average across GEMMs
(KU115) DNN Builder	20 nm	Fixed16	22.9	235	90.2	Batched execution of VGG
SODA (current) MatMul	45 nm	Fixed16	0.05	500	162.25	Derived from Execution time, FreePDK
SODA (current) MatMul	12/14 nm	INT8	0.00727	800	> 17000	Synopsys Design Compiler, GF12
AI/ML SODA MatMul	7 nm and newer	Fixed16	< 0.005	>500	> 1500	Estimated, ASAP 7nm
AI/ML SODA MatMul	7 nm and newer	INT8	< 0.0005	>800	> 20000	Estimated, ASAP 7nm

Table 1: Comparison across deep neural network (DNN) accelerators adapted from [10] to include projected results for an AI/ML enabled SODA framework. Results with variable precision.

Platform	Technology	Power [W]	Clock [MHz]	GFLOPS/W	Notes
V100 GPU	12 nm	300	1246	52.33	Theoretical peak
A100 GPU	7 nm	400	1410	48.75	Theoretical peak
TPU v3	16 nm	450	940	8.89	Theoretical peak
SODA (current) MatMul	45 nm	0.42	500	17.46	Derived from Execution time, freePDK
AI/ML SODA MatMul	7 nm and newer	< 0.07	> 500	> 100	Estimated, ASAP 7nm

Table 2: Comparison across deep neural network accelerators (DNN) adapted from [10] to include projected results for an AI/ML enabled SODA framework. Results at 32 bit floating point precision.

predictions will also enable quicker evaluation of the results and help eliminate solutions that do not meet timing requirements.

5 CONCLUSIONS

This paper explores how AI/ML can enhance hardware synthesis, using the SODA framework as an example. SODA is an open-source hardware synthesizer that includes a high-level frontend for interfacing with productive programming frameworks, performing hardware/software partitioning, and optimizing kernels for HLS, as well as a state-of-the-art HLS tool for generating RTL code. By integrating with open-source EDA tools, SODA provides a complete Python-to-silicon pipeline. We highlight the potential of AI/ML to improve the HLS process and demonstrate how applying these techniques within a fully open-source toolchain can create new opportunities. These advancements can drastically reduce turnaround times, significantly enhance the quality of results, and overall simplify and overcome barriers to hardware design.

ACKNOWLEDGMENTS

This research was partially supported by the Software Defined Architectures for Ultra-low Latency Reasoning (SODA-ULTRA) in the Adaptive Tunability for Synthesis and Control via Autonomous Learning on Edge (AT SCALE) Laboratory Directed Research and Development (LDRD) Initiative, and the ASCR Project Compilers Frameworks and Hardware Generators to Support Innovative US Government Designs at Pacific Northwest National Laboratory (PNNL).

REFERENCES

[1] Nicolas Bohm Agostini, Jude Haris, Perry Gibson, Malith Jayaweera, Norm Rubin, Antonino Tumeo, José L. Abellán, José Cano, and David R. Kaeli. 2024. AXI4MLIR: User-Driven Automatic Host Code Generation for Custom AXI-Based Accelerators. In *IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2024, Edinburgh, United Kingdom, March 2-6, 2024*. 143–157.

[2] Nicolas Bohm Agostini, Ankur Limaye, Marco Minutoli, Vito Giovanni Castellana, Joseph B. Manzano, Antonino Tumeo, Serena Curzel, and Fabrizio Ferrandi. 2022. SODA Synthesizer: An Open-Source, Multi-Level, Modular, Extensible Compiler from High-Level Frameworks to Silicon. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2022, San Diego, California, USA, 30 October 2022 - 3 November 2022*, Tulika Mitra, Evangeline F. Y. Young, and Jinjun Xiong (Eds.). ACM, 18:1–18:7. https://doi.org/10.1145/3508352.3561101

[3] Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond Pearce. 2023. Chip-Chat: Challenges and Opportunities in Conversational Hardware Design. In *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE. https://doi.org/10.1109/mlcad58807.2023.10299874

[4] Nicolas Bohm Agostini, Serena Curzel, Vinay Amatya, Cheng Tan, Marco Minutoli, Vito Giovanni Castellana, Joseph Manzano, David Kaeli, and Antonino Tumeo. 2022. An MLIR-based Compiler Flow for System-Level Design and Hardware Acceleration. In *41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD'22)*. To appear.

[5] Nicolas Bohm Agostini, Serena Curzel, Jeff Zhang, Ankur Limaye, Cheng Tan, Vinay Amatya, Marco Minutoli, Vito Giovanni Castellana, Joseph Manzano, David Brooks, Gu-Yeon Wei, and Antonino Tumeo. 2022. Bridging Python to Silicon: The SODA Toolchain. *IEEE Micro* (2022). https://doi.org/10.1109/MM.2022.3178580

[6] Vito Giovanni Castellana, Nicolas Bohm Agostini, Ankur Limaye, Vinay Amatya, Marco Minutoli, Joseph B. Manzano, Antonino Tumeo, Serena Curzel, Michele Fiorito, and Fabrizio Ferrandi. 2023. Towards On-Chip Learning for Low Latency Reasoning with End-to-End Synthesis. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference, ASPDAC 2023, Tokyo, Japan, January 16-19, 2023*. ACM, 632–638.

[7] Vito Giovanni Castellana, Antonino Tumeo, and Fabrizio Ferrandi. 2021. High-Level Synthesis of Parallel Specifications Coupling Static and Dynamic Controllers. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS'21)*. 192–202. https://doi.org/10.1109/IPDPS49936.2021.00028

[8] Vidya A. Chhabria, Wenjing Jiang, Andrew B. Kahng, and Sachin S. Sapatnekar. 2024. A Machine Learning Approach to Improving Timing Consistency between Global Route and Detailed Route. *ACM Trans. Design Autom. Electr. Syst.* 29, 1 (2024), 18:1–18:25.

[9] 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 (July 2016), 105–115. https://doi.org/10.1016/j.mejo.2016.04.006

[10] Serena Curzel, Nicolas Bohm Agostini, Vito Giovanni Castellana, Marco Minutoli, Ankur Limaye, Joseph B. Manzano, Jeff Zhang, David Brooks, Gu-Yeon Wei, Fabrizio Ferrandi, and Antonino Tumeo. 2022. End-to-End Synthesis of Dynamically Controlled Machine Learning Accelerators. *IEEE Trans. Computers* 71, 12 (2022), 3074–3087.

[11] Fabrizio Ferrandi, Vito Giovanni Castellana, Serena Curzel, Pietro Fezzardi, Michele Fiorito, Marco Lattuada, Marco Minutoli, Christian Pilato, and Antonino Tumeo. 2021. Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications. In *58th ACM/IEEE Design Automation Conference (DAC'21)*. 1327–1330. https://doi.org/10.1109/DAC18074.2021.9586110

[12] Giovanni Gozzi, Michele Fiorito, Serena Curzel, Claudio Barone, Vito Giovanni Castellana, Marco Minutoli, Antonino Tumeo, and Fabrizio Ferrandi. 2024. SPARTA: High-Level Synthesis of Parallel Multi-Threaded Accelerators. *ACM Trans. Reconfigurable Technol. Syst.* (jul 2024). https://doi.org/10.1145/3677035 Just Accepted.

[13] Andrew B. Kahng. 2022. A Mixed Open-Source and Proprietary EDA Commons for Education and Prototyping. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2022*. 17:1–17:6.

[14] Andrew B. Kahng and Tom Spyrou. 2021. The OpenROAD Project: Unleashing Hardware Innovation. In *Government Microcircuit Applications and Critical Technology Conference*. 1–6.

[15] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *IEEE/ACM International Symposium on Code Generation and Optimization (CGO'21)*. 2–14. https://doi.org/10.1109/CGO51591.2021.9370308

[16] Rongjian Liang, Anthony Agnesina, Geraldo Pradipta, Vidya A. Chhabria, and Haoxing Ren. 2023. Invited Paper: CircuitOps: An ML Infrastructure Enabling Generative AI for VLSI Circuit Optimization. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–6. https://doi.org/10.1109/ICCAD57390.2023.10323611

[17] Ankur Limaye, Claudio Barone, Nicolas Bohm Agostini, Marco Minutoli, Joseph Manzano, Vito Giovanni Castellana, Giovanni Gozzi, Michele Fiorito, Serena Curzel, Fabrizio Ferrandi, and Antonino Tumeo. 2024. Towards Automated Generation of Chiplet-Based Systems. In *Proceedings of the 29th Asia and South Pacific Design Automation Conference*. 771–776.

[18] Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, Bonita Bhaskaran, Bryan Catanzaro, Arjun Chaudhuri, Sharon Clay, Bill Dally, Laura Dang, Parikshit Deshpande, Siddhanth Dhodhi, Sameer Halepete, Eric Hill, Jiashang Hu, Sumit Jain, Ankit Jindal, Brucek Khailany, George Kokai, Kishor Kunal, Xiaowei Li, Charley Lind, Hao Liu, Stuart Oberman, Sujeet Omar, Ghasem Pasandi, Sreedhar Pratty, Jonathan Raiman, Ambar Sarkar, Zhengjiang Shao, Hanfei Sun, Pratik P Suthar, Varun Tej, Walker Turner, Kaizhe Xu, and Haoxing Ren. 2024. ChipNeMo: Domain-Adapted LLMs for Chip Design. arXiv:2311.00176 [cs.CL] https://arxiv.org/abs/2311.00176

[19] Marco Minutoli, Vito Giovanni Castellana, Nicola Saporetti, Stefano Devecchi, Marco Lattuada, Pietro Fezzardi, Antonino Tumeo, and Fabrizio Ferrandi. 2022. Svelto: High-Level Synthesis of Multi-Threaded Accelerators for Graph Analytics. *IEEE Trans. Comput.* 71, 3 (March 2022), 520–533. https://doi.org/10.1109/TC.2021.3057860

[20] Marco Minutoli, Vito Giovanni Castellana, Antonino Tumeo, and Fabrizio Ferrandi. 2015. Inter-procedural resource sharing in High Level Synthesis through function proxies. In *25th International Conference on Field Programmable Logic and Applications (FPL'15)*. 1–8. https://doi.org/10.1109/FPL.2015.7293958

[21] Louis-Noël Pouchet and Tomofumi Yuki. 2021. Polybench/C 4.2.1. Retrieved August 07, 2022 from https://web.cse.ohio-state.edu/~pouchet.2/software/polybench

[22] Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. 2023. Robust GNN-Based Representation Learning for HLS. In *IEEE/ACM International Conference on Computer Aided Design, ICCAD 2023, San Francisco, CA, USA, October 28 - Nov. 2, 2023*. IEEE, 1–9.

[23] Lily Jiaxin Wan, Yingbing Huang, Yuhong Li, Hanchen Ye, Jinghua Wang, Xiaofan Zhang, and Deming Chen. 2024. Invited Paper: Software/Hardware Co-design for LLM and Its Application for Design Verification. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 435–441. https://doi.org/10.1109/ASP-DAC58780.2024.10473893

[24] H. Ye, C. Hao, J. Cheng, H. Jeong, J. Huang, S. Neuendorffer, and D. Chen. 2021. ScaleHLS: Scalable High-Level Synthesis through MLIR. *arXiv preprint arXiv:2107.11673* (2021), 1–13.

[25] Jeff Jun Zhang, Nicolas Bohm Agostini, Shihao Song, Cheng Tan, Ankur Limaye, Vinay Amatya, Joseph B. Manzano, Marco Minutoli, Vito Giovanni Castellana, Antonino Tumeo, Gu-Yeon Wei, and David Brooks. 2021. Towards Automatic and Agile AI/ML Accelerator Design with End-to-End Synthesis. In *32nd IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP*. 218–225.