SODA Synthesizer: an Open-source, Multi-level, Modular, Extensible Compiler from High-level Frameworks to Silicon

Invited Paper

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

Serena Curzel†
Fabrizio Ferrandi
Politecnico di Milano
Milano, Italy

ABSTRACT

The SODA Synthesizer is an open-source, modular, end-to-end hardware compiler framework. The SODA frontend, developed in MLIR, performs system-level design, code partitioning, and high-level optimizations to prepare the specifications for the hardware synthesis. The backend is based on a state-of-the-art high-level synthesis tool and generates the final hardware design. The backend can interface with logic synthesis tools for field programmable gate arrays or with commercial and open-source logic synthesis tools for application-specific integrated circuits. We discuss the opportunities and challenges in integrating with commercial and open-source tools both at the frontend and backend, and highlight the role that an end-to-end compiler framework like SODA can play in an open-source hardware design ecosystem.

KEYWORDS

High-level synthesis, hardware/software co-design

ACM Reference Format:
Nicolas Bohm Agostini, Ankur Limaye, Marco Minutoli, Vito Giovanni Castellana, Joseph 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: Invited Paper. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22), October 30-November 3, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3508352.3561101

1 INTRODUCTION

Machine learning (ML) and artificial intelligence (AI) have become ubiquitous, with applications ranging from the smallest edge devices to large high-performance computing systems. For example, in sensor networks, ML algorithms can be used to filter and save only the relevant data or to reconstruct the data from missing links [12]. Autonomous systems, including vehicles or scientific experimental instruments, must quickly process newly acquired data to make real-time decisions and react to variations in the environment [18]. In areas such as finance or healthcare, they are used to identify anomalies and provide predictions [16]. Scientific simulations may employ surrogate models to speed up computation [1]. The diversity of these application domains and of the algorithms used in each area is leading to a *Cambrian* explosion [8] of specialized systems that try to accelerate computations while fitting specific domain requirements. These requirements include the usual metrics such as power, performance, and area, but may also impose new constraints on the system's overall size, security, cooling needs, and real-time awareness.

Domain scientists typically implement and validate their algorithms in high-level programming frameworks. On the other hand, providing hardware-accelerated solutions targeted to specific domains requires large teams of hardware experts to identify common algorithm patterns and design custom accelerators by directly implementing the register transfer level (RTL) code. The quick evolution of high-level programming frameworks and the intense research on novel algorithmic methods make the conventional hardware design approach impractical and time-consuming, leading to a significant hardware productivity gap.

Approaches based on High-Level Synthesis (HLS) allow semi-automatically translating algorithms described in higher-level languages to RTL. However, they typically require large restructuring of the existing C/C++ codes with the tool- and hardware-specific annotations, necessitating long porting from functional languages and significant optimization efforts by hand. The conventional HLS approaches still require a mix of manual and automated flows, which can fail to propagate relevant information downstream, leading to loss of information and integration issues. Once the custom RTL designs are finalized, they must deal with commercial tools and process development kits (PDKs) for fabrication. These tools and PDKs are often significantly diverse across multiple vendors and typically include proprietary technologies, thus increasing the efforts required to adapt designs to the specific process technology target. As the fabrication complexities increase with novel technology nodes, only the large hardware design companies can access state-of-the-art design capabilities and dedicate the necessary resources to address the mismatches and gaps at different levels of the design stack. However, these companies typically focus on fabricating solutions with broad commercial applicability. These challenges severely limit the capability to bring disruptive new

*Also with Northeastern University.
†Also with Pacific Northwest National Laboratory.

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10.
https://doi.org/10.1145/3508352.3561101

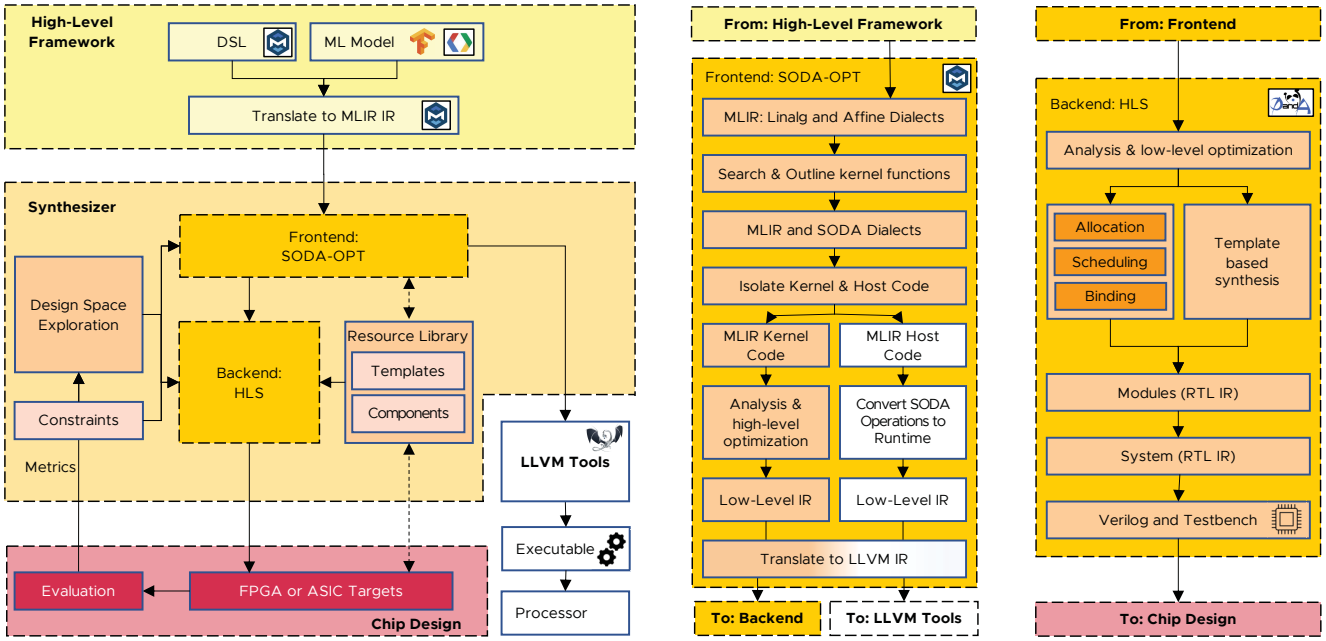

Figure 1: The SODA framework is an open-source, multi-level, modular, extensible, hardware generator composed of a high-level compiler and a lower-level HLS backend

advanced computing concepts to late-stage product development, thus not permitting their use in mission-critical applications with limited commercial interest.

We argue that a new ecosystem of open-source design automation tools spanning the whole stack is required to address all these challenges. This ecosystem should include high-level compilers, hardware generators, simulation and verification infrastructures, intellectual properties (IPs), logic synthesis and physical design tools, and PDKs. Existing tools must be strengthened and integrated to enable various end-to-end flows, thus enabling *agile* hardware designs. The open-source ecosystem also needs to effectively and efficiently incorporate not only the existing standalone open-source tools, but also inteface with the proprietary hardware design ecosystems to leverage advanced methodologies with well-established and proven industry practices. We furthermore argue that novel HLS methodologies, well integrated into such an open-source ecosystem, are vital to establishing a fast, automated, iterative, and economically sustainable design process.

In such a context, we introduce the SODA (Software Defined Accelerators) Synthesizer [3], an open-source, modular, and extensible end-to-end compiler-based framework for generating highly specialized hardware accelerators from algorithms designed in high-level programming frameworks. SODA is composed of a compiler-based frontend to interface with high-level programming frameworks and apply high-level optimizations, and a compiler-based backend to generate Verilog code and interface with external tools that compile the final design (either application-specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs). A key feature of SODA is interoperability with other tools and extensibility that enable the realization of an *end-to-end agile hardware design flow*. This paper presents an overview of the SODA Synthesizer framework and highlights two case studies demonstrating interoperability with lower-level open-source logic synthesis and physical layout tools. We leverage the SODA framework to synthesize designs with the OpenROAD [9] flow and two of the supported open-source physical design kit (FreePDK 45 nm and ASAP 7 nm). We conclude the paper by discussing future research directions for the SODA framework in the context of the open-source design automation ecosystem.

2 THE SODA SYNTHESIZER

The SODA Synthesizer (Figure 1a) consists of two interoperating parts: i) SODA-OPT [2], the frontend compiler for system-level partitioning and high-level optimizations, and ii) PandA-Bambu [7], a state-of-the-art HLS compiler. The input to the SODA framework is code written in high-level programming frameworks, typically using Python, which is transformed into valid and optimized Verilog RTL designs that can be implemented on different FPGA or ASIC targets.

The input provided to the compilation pipeline is translated into a high-level intermediate representation (IR) in the early stages of the high-level optimizer. This high-level IR is defined with many dialects in the Multi-Level Intermediate Representation (MLIR) [10].

SODA-OPT performs hardware/software partitioning of the input program and architecture-independent optimizations by leveraging features of the MLIR framework. SODA-OPT produces two different types of LLVM IR as outputs. The first output is an optimized LLVM IR file without external dependencies that represents the kernels identified for acceleration and is passed as an input to the PandA-Bambu [7] HLS compiler. The other output is an LLVM IR

file representing the host program that orchestrates calls to the accelerators. All the optimizations performed by the SODA framework are implemented as different compiler passes. The generated hardware design's performance, area, and power are highly influenced by the sequence of transformations and their parameters, which are presented in detail in Section 3. SODA-OPT and PandA-Bambu offer great control over the optimization process, which is integrated with the design space exploration engine that selects a suitable combination of compiler passes and parameters, optimizing the design for a chosen target metric (performance, area, power, etc.).

2.1 SODA-OPT Frontend

SODA-OPT (Figure 1b) is the SODA Synthesizer's compiler frontend. It is developed by extending MLIR- a framework recently contributed to the LLVM project - that allows building modular and reusable compiler infrastructure by defining *dialects*, i.e., specialized, and self-contained IRs that respect MLIR's meta-IR syntax.

The conventional approaches followed by HLS design flows require significant code changes or use compiler hints provided via `pragma` annotations. These hints guide the optimizations (e.g., exposing more or less parallelism by controlling the unrolling factor of loops) during the HLS process. SODA-OPT takes an orthogonal approach by leveraging the semantic information carried by context-specific MLIR dialects and automatically applies the high-level transformations while preparing the input program for hardware synthesis.

SODA-OPT provides compilation passes to *S*earch, *O*utline, *Optimize*, *D*ispatch, and *A*ccelerate parts of the initial specification coming from high-level frameworks. SODA-OPT defines the `soda` dialect. This custom MLIR dialect allows automatic partitioning of the input application into a host program responsible for orchestrating the runtime execution, and the custom hardware accelerators [2].

SODA-OPT analyzes the MLIR input and identifies code regions (search) amenable for acceleration. The code regions identified are then extracted into separate MLIR modules (outline). The outlined modules undergo the SODA-OPT optimization passes. Taking advantage of the modular design of the MLIR framework, SODA-OPT can leverage MLIR dialects and the associated optimizations directly provided with the MLIR distribution in the LLVM compiler framework or provided externally. For example, SODA-OPT leverages MLIR's `linalg` and `affine` dialects to identify operators and perform loop optimizations.

Machine Learning frameworks (e.g., TensorFlow, ONNX-MLIR, and TORCH-MLIR), software for scientific computing (e.g., NPCOMP), and general-purpose programming languages (e.g., the FLANG Fortran compiler) are designing MLIR dialects, optimizations, and lowering passes to optimize their input programs. SODA-OPT can directly interface with all the frameworks that lower to dialects provided in the MLIR distribution.

2.2 SODA Synthesizer Backend

Bambu (Figure 1c), an open-source state-of-the-art HLS tool from the PandA framework, is the SODA framework's compiler backend. Bambu generates the accelerator designs starting from the LLVM IR produced by SODA-OPT.

Bambu supports several frontends based on standard compilers (GCC or Clang), but can additionally accept LLVM IR inputs through a specific Clang plug-in. This feature allows SODA-OPT to act as an additional specialized frontend to Bambu. Bambu builds an internal IR to perform various necessary HLS steps (e.g., bitwidth analysis, loop optimizations, resource allocation, scheduling, and binding algorithms), and generates the RTL designs in a hardware description language (Verilog or VHDL). In addition to the synthesizable RTL code, it can also automatically produce testbenches for verification. Bambu enables the SODA framework to target FPGAs (from Xilinx, Altera, Lattice, NanoXplore) and ASICs. For ASICs, SODA supports Verilog-to-GDSII generation using both commercial (Synopsis Design Compiler) and open-source (OpenROAD [9]) logic synthesis tools. Bambu ingests LLVM IR generated after SODA-OPT high-level optimizations for HLS, resulting in more efficient accelerators compared to accelerators synthesized starting from C/C++.

Bambu, by default, generates RTL designs following the finite state machine with datapath (FSMD) model, but also integrates methodologies to support parallel accelerator designs. It can, in fact, integrate FSMD accelerators as processing elements in coarse-grained dataflow designs [4], or in high-throughput, dynamically scheduled, multithreaded parallel templates [13]. Bambu also exposes modular synthesis methodologies [14]: differently from other HLS tools, it can generate modules representing functions that may be reused or replicated across an entire design and composed in a complex multi-accelerator system.

MLIR descriptions are naturally parallel and hierarchical, making it possible to trigger Bambu's advanced synthesis methodologies from SODA-OPT. Rather than requiring manual annotations on the input code, we can define the design hierarchy at a higher level of abstraction by exploiting MLIR. This approach demonstrates how clear interfaces and integration between the two tools facilitate hardware design, removing the need to provide input code with hardware information in the form of annotations.

The downstream logic synthesis and physical layout tool can then take the RTL descriptions generated by Bambu as input and, together with physical constraints and design rules, generate the final implementations for FPGAs or ASICs. However, even though the generated RTL code can theoretically work for any type of target device and process technology, several steps of the HLS process can benefit from a detailed knowledge of the targets. In particular, *characterizing functional units and components in the resource library* for specific target devices and technology, and integrating technology-specific interconnect models can improve the quality of the results of the generated designs. The characterization process adds valuable information like area, delay, and power for each element of the resource library. Module binding and scheduling HLS steps can then use this information to optimize for various metrics (e.g., overall latency and area of the accelerator) and meet constraints like the target frequency. For example, if sufficient slack exists, two functional units can be *chained* together to execute in the same cycle.

Bambu provides a specific tool, named *Eucalyptus*, to perform resource characterization. Eucalyptus runs micro-benchmarks with the backend logic synthesis tools and annotates the relevant information for each resource in the resource library. Currently, Bambu

already includes characterization for a variety of FPGA devices and various ASIC technology libraries. However, new targets can be added without recompiling the toolchain. This approach extends the opportunities for design space exploration and allows almost automatic tradeoff evaluations across different target technologies.

3 EVALUATION

We present two case studies to demonstrate the end-to-end capabilities of the SODA framework, i.e., automatically generating specialized hardware accelerators from high-level programming frameworks. We generate ASIC implementations for two different sets of inputs: PolyBench [17] kernels and a LeNet model, leveraging the OpenROAD [9] flow for two technology libraries: ASAP 7 nm [5] and FreePDK 45 nm, respectively.

3.1 PolyBench kernels

In the first case study, we synthesized kernels from PolyBench [17] directly described in MLIR, generating and analyzing the resulting ASIC designs. These kernels are representative of many algorithmic patterns used in scientific computing or high-level data science programming frameworks. The ASIC implementations were generated using the OpenROAD flow with the ASAP 7 nm technology library.

Table 1 shows the execution clock cycles, the maximum frequency (in MHz) reachable, the design area (in μm^2), and the energy (in nJ) for each generated design. We highlight the impacts of SODA-OPT on the final designs of each kernel. The results for kernels that are directly lowered to LLVM IR and synthesized by Bambu are presented in the *No optimizations* columns, while the results for kernels that first undergo SODA-OPT's high-level optimization pipeline are presented under *With optimizations*. For each kernel, we also report the implementation results considering input/output tensors of different sizes (each tensor has the same number of elements for each dimension), which leads to a different number of operations executed, different optimization opportunities and, consequently, different custom designs. We set both the SODA Synthesizer and the OpenROAD flow to a frequency of 1 GHz.

SODA-OPT's high-level optimizations have a favorable impact on the generated designs: they show significant speedups and are energy-efficient but incur comparatively small area overheads. SODA-OPT's optimizations (e.g., loop unrolling) expose better parallelism, resulting in better performance but larger design area footprints. Table 1 also presents an interesting trend: the 1 GHz design constraint was met by all the non-optimized cases, while the maximum frequency for the optimized cases was typically reduced for larger tensor sizes. The ASIC designs could not be generated for a few optimized cases with larger tensor sizes because of routing congestion during the OpenROAD flow.

3.2 Neural Network: LeNet

In the second case study, we automatically translated a LeNet model trained in TensorFlow to the `linalg` dialect and employed SODA-OPT to search, outline, and optimize different regions of the network. We then generated different specialized accelerators with the SODA framework. In this case, the ASIC implementations were generated using the OpenROAD flow with the FreePDK 45 nm technology library.

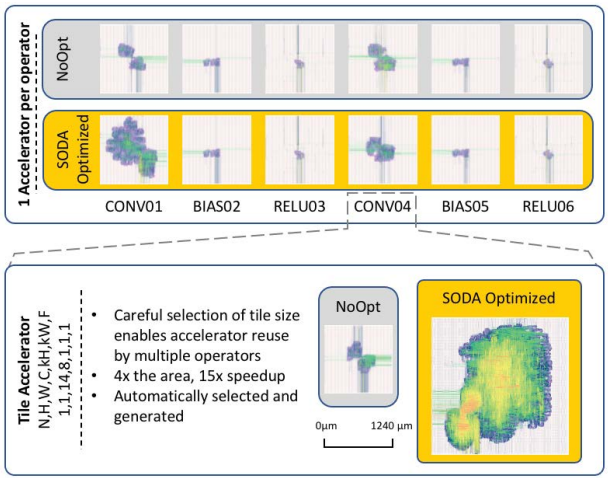

Figure 2: ASIC implementations of LeNet layers.

Table 2 shows the execution clock cycles, the design area (in μm^2), and the power efficiency (in GFLOPS/W) obtained for non-optimized and optimized SODA implementations of accelerators for the different layers from the LeNet convolutional neural network model. SODA-OPT's optimizations provide a speedup of the accelerators proportional to the increase in the area of the accelerators. The power efficiency of the generated accelerators may be slightly reduced due to an increase in power consumption of the faster solutions. Figure 2 shows the floorplans (visualized from the standard GDSII format) of the same accelerators, highlighting the increase in the area of the better-performing optimized designs.

4 RESEARCH OPPORTUNITIES

A modular hardware compiler infrastructure is critical in providing the necessary agility to move from high-level specifications to hardware implementation with minimal or no human intervention. Such infrastructure is essential to an open-source ecosystem that should enable agile hardware design with a quick, fail-fast design cycle. However, the larger open-source ecosystem also provides a variety of near, medium, and longer-term opportunities for applied use and research performed with the tools.

The current tools from the open-source ecosystem have demonstrated interoperability and the possibility of establishing several end-to-end design flows that enable fast prototyping of advanced design concepts. However, there is still the need to strengthen integration among tools from various institutions - with different research focuses - bringing them closer to production quality. As the SODA Synthesizer demonstrates, modular and interoperable open-source compiler infrastructures such as LLVM IR and MLIR are paving the way for integration across various tools. Community initiatives and projects such as CIRCT [6], which leverages the MLIR infrastructure to build hardware compilers, are starting to lead the way. However, significant foundational work is still needed to retrieve and reuse past and current leading-edge research performed in isolated tools.

Another key aspect is interoperability with commercial solutions. While open-source tools have been mainly research-focused,

Kernel	Size	No optimizations				With optimizations				Trade-offs	
		Cycles	Max. freq. (MHz)	Area (μm^2)	Energy (nJ)	Cycles	Max. freq. (MHz)	Area (μm^2)	Energy (nJ)	Speedup	Area Overhead
atax	2	118	1164.28	1,658	1.63	38	986.65	2,539	1.13	3.11	1.53
	4	463	1384.20	1,970	8.20	63	1182.30	8,211	5.20	7.35	4.17
	8	1,819	1371.31	3,056	33.03	113	634.15	22,677	34.21	16.10	7.42
bicg	2	113	1778.91	1,605	1.26	24	1206.61	4,425	0.72	4.71	2.76
	4	458	1532.42	1,881	4.00	39	1098.86	12,015	3.87	11.74	6.39
	8	1,810	1314.11	2,974	34.98	78	400.86	30,744	21.01	23.21	10.34
gemm	2	161	1401.78	3,100	4.66	27	1411.24	5,067	1.09	5.96	1.63
	4	1,258	1286.68	4,597	37.74	51	782.38	21,089	16.36	24.67	4.59
	8	10,450	1451.15	2,590	143.30	139	–	–	–	75.18	–
gemver	2	246	1250.11	3,778	9.01	66	1374.54	5,832	2.78	3.73	1.54
	4	974	1031.52	7,835	68.27	91	652.62	18,595	43.50	10.70	2.37
	8	3,833	1121.75	7,547	292.83	141	–	–	–	27.18	–
gesummv	2	142	1554.66	2,336	2.25	35	1235.27	4,866	1.12	4.06	2.08
	4	514	1032.78	2,468	12.09	50	1128.52	10,431	5.10	10.28	4.23
	8	1,922	1279.50	3,614	44.16	101	638.13	25,487	52.07	19.03	7.05
mvt	2	114	1583.37	1,737	1.09	24	892.39	4,408	1.07	4.75	2.54
	4	450	1355.68	4,760	20.15	41	1125.35	13,366	3.59	10.98	2.81
	8	1,819	1216.38	3,544	60.56	81	559.19	32,757	16.08	22.46	9.24
three_mm	2	340	1155.26	3,577	9.42	42	1176.22	9,424	3.50	8.10	2.63
	4	2,719	1106.28	7,994	163.69	75	305.15	40,428	90.69	36.25	5.06
	8	22,130	1243.23	5,614	656.84	231	–	–	–	95.80	–
two_mm	2	274	1234.98	4,082	11.51	45	1337.67	6,915	0.78	6.09	1.69
	4	2,163	1019.49	7,063	128.57	75	422.71	34,326	20.94	28.84	4.86
	8	17,762	1281.06	4,251	508.85	–	–	–	–	–	–

Table 1: PolyBench results with OpenROAD and the ASAP 7nm technology library

Kernel	No Optimizations			With optimizations			Trade-offs	
	Cycles	Area (μm^2)	Power eff. (GFLOPS/W)	Cycles	Area (μm^2)	Power eff. (GFLOPS/W)	Speedup	Area Overhead
CONV_01	10,262,618	29,073	4.43	4,627,982	124,255	2.68	2.22	4.27
BIAS_02	251,694	10,395	11.48	40,826	60,048	9.01	6.17	5.78
RELU_03	151,342	7,385	41.55	38,446	35,695	38.39	3.94	4.38
CONV_04	85,380,948	36,814	3.32	83,380,180	37,556	3.34	1.02	1.02
BIAS_05	62,932	10,409	11.00	10,222	60,007	8.41	6.16	5.76
RELU_06	37,844	7,464	41.75	9,620	35,950	37.04	3.93	4.82

Table 2: Evaluation of non optimized and optimized LeNet operators in ASIC technology (FreePDK 45 nm at 500 MHz)

they sometimes lack the production-ready quality of proprietary tools. Conversely, commercial tools are typically difficult to directly integrate into automated flows and require significant manual efforts, since some of their algorithms and interfaces are proprietary. Regarding these aspects, we are investigating the integration of the SODA framework with several different tools of the open-source and proprietary ecosystems. For SODA-OPT, in particular, we have implemented initial support for commercial FPGA synthesis tools

(Vitis HLS) by also generating optimized LLVM IR inputs, extending the work in [20]. SODA-OPT can already reason about system-level design. It performs code partitioning, optimizations specific for custom hardware generation, and composition of a system architecture, generating glue code for control processors or assembling accelerators in dynamically scheduled architectures. A similar approach could be further extended by integrating rapid prototyping platforms in the open-source hardware ecosystem, such as the Embedded Scalable Platforms (ESP) [11]. We are currently working to integrate both SODA-OPT and Bambu with ESP. SODA-OPT can drive the system-level design, leveraging the services offered by ESP to invoke accelerators. Bambu can provide ESP with an open-source HLS backend for custom accelerators, which will be generated from code partitioned, optimized, and mapped on the ESP SoC by SODA-OPT.

Considering the availability of open-source IPs and architectural templates, several of these modules can either become targets for SoC design (e.g., platforms provided with RISC-V cores) or part of the HLS tool resource library. For example, Bambu can integrate templates of systolic arrays in its resource library, allowing it to generate specialized processing elements, similar to the approach presented in [13].

The presented case studies demonstrate our support for an open-source logic synthesis flow, leveraging the OpenROAD flow with different PDKs. With the same resource characterization and integration mechanisms discussed in Section 2.2, it would be possible to support the SkyWater 130 nm and 90 nm PDKs and related toolchains. Such integration would also enable chartered fabrication runs using such technologies. Furthermore, integrating solutions like LS Oracle [15] in the OpenROAD flow, which further optimizes the logic synthesis process, will allow another level of design space exploration across the tools, potentially without even directly exploiting resource characterization. Similarly, we have demonstrated [3] support for commercial logic synthesis tools (Synopsys Design Compiler) and leading-edge proprietary PDKs (Global Foundries 14/16 nm) previously. However, we cannot distribute the information obtained through the resource characterization step due to license agreements and the resource characterization must be repeated every time. Bambu's current resource library for the FPGA targets includes characterization for select commercial FPGAs from AMD/Xilinx, Intel/Altera, NanoExplore, and Lattice.

Finally, an additional opportunity enabled by the open-source ecosystem is supporting domain-specific FPGAs. SODA could integrate with solutions such as OpenFPGA [19], performing high-level analysis to identify patterns that might require additional hard macros in the hardware substrate while still leveraging fine-grained reconfigurability. The HLS tool could perform design space exploration, leveraging the hard macros through the resource library, or even synthesizing the hard macro on the fly. The SODA framework would then be able to automatically provide the domain-specific FPGA organization and generate it using the logic synthesis and physical layout tools.

We believe that creating and strengthening integration between open-source hardware design tools will be critical with the upcoming investments in advanced manufacturing. Providing end-to-end solutions, from high-level specification to fabrication, will allow exploring new computing concepts and make domain-specialized systems viable for many more areas. Integrating with industrial tools will also enable quicker technology transitions, removing research and development cost barriers. Finally, a rich open-source ecosystem will make training the next generation of hardware design workforce and researchers much more effective.

5 CONCLUSIONS

This paper overviews the SODA framework, an end-to-end, multi-level, open-source hardware compiler composed of a frontend based on the MLIR infrastructure and a backend leveraging a state-of-the-art HLS engine. Through its frontend, SODA interfaces with a variety of high-level programming frameworks typically used by domain scientists for the novel "converged" applications. Through its backend, it can generate complete hardware designs targeting FPGAs from different vendors and ASICs. The end-to-end nature of the framework provides the agility needed to go from algorithmic formulation to hardware implementation. We presented case studies showing the SODA framework in the larger open-source ecosystem of hardware design tools, and discussed the integration with other tools, such as the OpenROAD flow with the FreePDK 45 nm and the ASAP 7 nm technology libraries. We also discussed the opportunities that the SODA framework provides and made a case that SODA can be a key technology enabler for fully integrating other frameworks into an ecosystem.

ACKNOWLEDGMENTS

This research was partially supported by the Software Defined Accelerators for Data Analytics (SO(DA)^2) project in the Data Model Convergence Initiative under the PNNL's Laboratory Directed Research and Development (LDRD) program and the Defense Advanced Research Projects Agency's (DARPA) Real-Time Machine Learning (RTML) program.

REFERENCES

[1] Mario Christopher Bedrunka, Dominik Wilde, Martin Kliemank, Dirk Reith, Holger Foysi, and Andreas Krämer. 2021. Lettuce: PyTorch-Based Lattice Boltzmann Framework. In *High Performance Computing (ISC'21)*, Heike Jagode, Hartwig Anzt, Hatem Ltaief, and Piotr Luszczek (Eds.). 40–55. https://doi.org/10.1007/978-3-030-90539-2_3

[2] 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.

[3] 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

[4] 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

[5] 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

[6] CIRCT Developers. 2020. CIRCT: Circuit IR Compilers and Tools. Retrieved August 07, 2022 from https://github.com/llvm/circt

[7] 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

[8] John L. Hennessy and David A. Patterson. 2019. A New Golden Age for Computer Architecture. *Commun. ACM* 62, 2 (Jan. 2019), 48–60. https://doi.org/10.1145/3282307

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

[10] 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

[11] Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G. Cota, Michele Petracca, Christian Pilato, and Luca P. Carloni. 2020. Agile SoC Development with Open ESP. In *IEEE/ACM International Conference On Computer Aided Design (ICCAD'20)*. 1–9. https://doi.org/10.1145/3400302.3415753

[12] Omolemo Godwill Matlou and Adnan M. Abu-Mahfouz. 2017. Utilising artificial intelligence in software defined wireless sensor network. In *43rd Annual Conference of the IEEE Industrial Electronics Society (IECON'17)*. 6131–6136. https://doi.org/10.1109/IECON.2017.8217065

[13] 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

[14] 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

[15] Walter Lau Neto, Max Austin, Scott Temple, Luca Amaru, Xifan Tang, and Pierre-Emmanuel Gaillardon. 2019. LSOracle: a Logic Synthesis Framework Driven by Artificial Intelligence: Invited Paper. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'19)*. 1–6. https://doi.org/10.1109/ICCAD45719.2019.8942145

[16] Ebberth L. Paula, Marcelo Ladeira, Rommel N. Carvalho, and Thiago Marzagão. 2016. Deep Learning Anomaly Detection as Support Fraud Investigation in Brazilian Exports and Anti-Money Laundering. In *15th IEEE International Conference on Machine Learning and Applications (ICMLA'16)*. 954–960. https://doi.org/10.1109/ICMLA.2016.0172

[17] 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

[18] Steven Spurgeon, Colin Ophus, Lewys Jones, Amanda Petford-Long, Sergei Kalinin, Matthew Olszta, Rafal Dunin-Borkowski, Norman Salmon, Khalid Hattar, Wei-Chang Yang, Renu Sharma, Yingge Du, Ann Chiaramonti, Haimei Zheng, Edgar Buck, Libor Kovarik, R Penn, Dongsheng Li, Xin Zhang, and Mitra Taheri. 2020. Towards data-driven next-generation transmission electron microscopy. *Nature Materials* 20, 3 (Oct. 2020), 274–279. https://doi.org/10.1038/s41563-020-00833-z

[19] Xifan Tang, Edouard Giacomin, Aurélien Alacchi, Baudouin Chauviere, and Pierre-Emmanuel Gaillardon. 2019. OpenFPGA: An Opensource Framework Enabling Rapid Prototyping of Customizable FPGAs. In *29th International Conference on Field Programmable Logic and Applications (FPL'19)*. 367–374. https://doi.org/10.1109/FPL.2019.00065

[20] Ruizhe Zhao, Jianyi Cheng, Wayne Luk, and George A. Constantinides. 2022. POLSCA: Polyhedral High-Level Synthesis with Compiler Transformations. In *32nd International Conference on Field Programmable Logic and Applications (FPL'22)*. To appear.