Learnings from a HLS-based High-Productivity Digital VLSI Flow

Thierry Tambe
Harvard University
USA

David Brooks
Harvard University
USA

Gu-Yeon Wei
Harvard University
USA

1 INTRODUCTION

The twilight of Dennard scaling has activated a global trend towards application-based hardware specialization. This trend is currently accelerating due to the surging democratization and deployment of machine learning on mobile and IoT compute platforms. At the same time, the growing complexity of specialized system-on-chips (SoCs) is levying a more laborious tax on ASIC companies' design and verification efforts. High-level synthesis (HLS) is emerging as a foremost agile VLSI development methodology gaining increasing adoption in the hardware design community. However, concerns over Quality of Results (QoR) remain a key factor inhibiting more mainstream adoption of HLS. Obtaining optimal PPA outcomes can sometimes be an elusive or challenging task and strongly correlates with the syntactic approach of the high-level source code.

In this paper, we aim to share the proven HLS practices we employed to raise the level of confidence in the post-silicon functional and performance expectations from our accelerator designs. In doing so, we recount some of the main challenges we encountered in our HLS-based hardware-software co-design journey and offer a few recommendations cultivated from our learnings. Finally, we posit on where the research opportunities to further improve design QoR and HLS user experience lie.

2 HLS-BASED SOFTWARE-HARDWARE CO-DESIGN INFRASTRUCTURE

In the development of machine learning accelerators, we relied on an objected-oriented HLS-based methodology (OOHLS) for fast SystemC-to-RTL prototyping [9]. Our OOHLS adaptation, shown in Fig. 1, effectively closes the loop between the application's software modeling and the backend hardware implementation being abstracted inside the HLS environment. Using the software ML framework (e.g., PyTorch, TensorFlow) as a golden reference, the HLS environment facilitates making, with relatively fast velocity, hardware tweaks and ECOs until *i)* the hardware and software output activations from the neural network returned matching bit-level results, *ii)* the post-HLS verification is functionally correct, and post-HLS PPA results are satisfactory.

Thanks in great part to this infrastructure, we were able to design, verify, and tape out edge AI SoCs [11, 15] with HLS-based accelerators in a span of about four months. This agility is made

This work was supported in part by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA.

LATTE '22, March 1, 2022, Lausanne, Switzerland

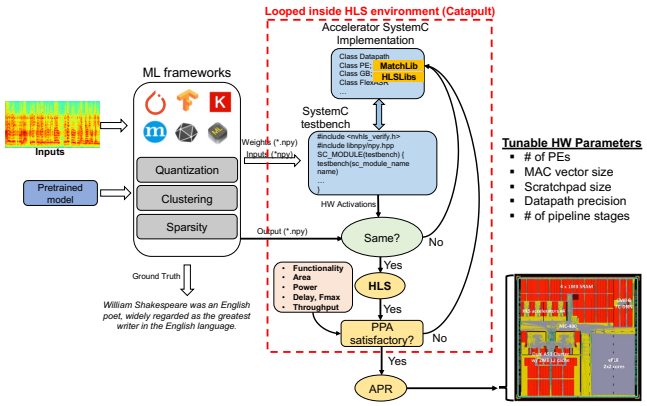

Figure 1: HLS-based HW/SW co-design and co-verification methodology we adopted during tapeouts of edge AI SoCs.

possible by the following characteristics which we benefited from (similarly expressed in [9, 10]):

- **Modularity.** Well-maintained libraries such as MatchLib [9], and HLSLibs [2] boost HLS design productivity by offering a slate of synthesizable, configurable, and common hardware components such as scratchpad memories, latency-insensitive channels, arbitrated crossbar arrays, datatype classes, etc. Notably, MatchLib also provides AXI channels enabling the HLS design to be connected with the rest of the SoC. Therefore, hardware engineers need not "*reinvent every wheel*" as these libraries help modularize the design in a correct-by-construction fashion.
- **High-throughput verification.** C++ based simulation is innately orders-of-magnitude faster than an equivalent RTL-based simulation. And, the reuse of the C++/SystemC testbench in the verification of the HLS-generated Verilog promotes the troubleshooting of the hardware to occur at the higher abstraction level. The overwhelming majority of design bugs, in our experience, were caught prior to the HLS execution, in large part, due to enabling random stalling of communication channels, which help unearth elusive bugs that would have otherwise been detected only during the post-HLS phase.
- **Facilitation of design space exploration.** The parameterization of key hardware features (e.g., number of processing elements, MAC vector size, memory size, etc) in the source code is highly encouraged as these parameters can be conveniently discretized in verification and synthesis Makefiles via external JSON files, enabling rapid high-level investigations.
- **Easy handshake with application software.** We have used the NPY library [1] to great success in order to convert and sync Numpy-based DNN activations and weights over to the HLS environment for verification purposes. Alternatively,

given the high-level implementation already bears some resemblance to software C++ models, the migration of ML data can be further streamlined by plugging in the HLS design directly into the C++ API of the ML framework [6].

3 OBSERVED CHALLENGES

Although our tapeout execution largely benefited from the agile design and verification velocity of the above-described OOHLS HW/SW co-design flow, we did experience a number of challenges which are discussed below. [16] recounts some of the engineering techniques we employed to mitigate these pitfalls.

3.1 Grasping proper semantics for correct Post-HLS functionality

It is relatively easy to write a compiling SystemC/C++ code that fails post-HLS RTL verification. A somewhat appreciable learning curve is required in order to develop understanding of the syntactic idioms that guarantee error-less synthesizability and functionality. These learnings are eventually acquired after lots of trials and errors. A common manifestation would be a compiling high-level code that eventually hangs or freezes in the middle of the post-HLS RTL verification. However, we have observed that newer releases of MatchLib that internally and automatically reset its communication channels, along with the usage of random port stalling have greatly reduced the occurrence rate of these bugs.

3.2 Grasping proper semantics for optimal Post-HLS PPA

We have found it is also relatively easy to write a compiling, functioning, and synthesizing source code that ultimately produces very subpar post-HLS performance. Often, the cycle behavior of the untimed or "loosely-timed" high-level abstraction does not match that of the generated RTL which tends to be more pessimistic. Therefore, an even bigger learning curve is grasping the coding habits that would achieve rigorous post-HLS PPA metrics – as simple tweaks in the source code may affect area, power, and throughput in drastic ways. This learning endeavor also requires intimate understanding of the many pragmas and directives offered by the HLS tool and their impact on circuit behavior.

A common source of inefficiency comes from using overly long, and often very sequential SC_THREADs requiring larger initiation intervals (II) that degrade the datapath throughput. As shown in Fig. 2, multithreading such designs is key to improving performance. However, as independent SC_THREADs contribute to the same design output behavior, orchestrating their timing and event-based properties may be nontrivial.

Moreover, Fmax and delay characteristics assumed during the HLS execution may be different from those observed after gate-level synthesis. Therefore, a struggle is figuring out how much positive or negative margins are baked into the HLS process in order to maximize post place-and-route performance.

4 OPPORTUNITIES FOR HLS ENHANCEMENT

Beyond its heavy adoption in the FPGA world [3–5, 8], HLS has also been proven in many successful academic [11, 13, 15] and industry [17, 18] SoC tapeouts. And it is currently being used to design digital ASICs in a mass production capacity [14].

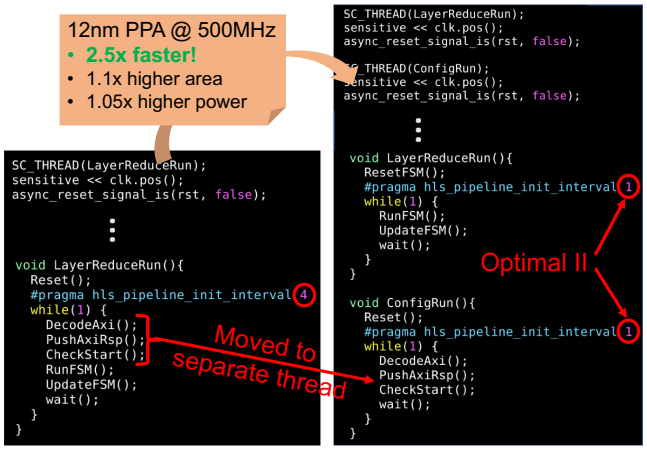

Figure 2: Multithreading complex SC_THREADs can signficantly reduce the required initiation interval of datapaths, improving overall throughput. This opens another challenge of syncing the timing behavior of these independent threads.

Still, QoR concerns are key factors limiting more mainstream adoption in competitive and market-centered business models where exacting PPA demands are combined with an extremely high bar for functional health. Addressing these stringent scenarios warrants raising the level of confidence higher through:

- A mature formal equivalence verification (FEV) or logical equivalence checking (LEC) tool comparing the high-level source code with the HLS-generated RTL representation [9].
- A mature pre-HLS coverage closure tool that examines the comprehensiveness of SystemC/C++ verification testbenches.
- Awareness of floorplan and STA constraints during the HLS execution in order to maximize post place-and-route frequency attainment. We note such solutions are starting to form in the case of FPGA-based HLS [7].
- Improved legibility and decipherability of HLS-generated RTLs for human reviewers.

All things considered, HLS is poised to make greater inroads and breakthroughs as Moore's law effectiveness wanes and machine learning becomes more ubiquitous. For example, homogeneous SoC-level HLS design approaches are being introduced – as opposed to traditional manual integration of HLS IPs into a RTL-based chassis [12]. A proposal for furthering HLS democratization would be to invest in a reverse RTL-to-C HLS tool which would be helpful in "softening" handcrafted hard IPs for rapid hardware recalibration to new PPA or process node targets or application-specific constraints.

5 CONCLUDING REMARKS

C-to-RTL HLS flows enable engineering and research teams to develop application-driven SoCs with agility and velocity. However, optimizing HLS designs is often a nontrivial endeavor, which would greatly benefit from the standardization of best known practices in order to improve designer's user experience.

REFERENCES

[1] 2021. *ARM Compute Library.* https://github.com/ARM-software/ComputeLibrary

[2] 2021. *Open-Source High-Level Synthesis IP Libraries.* https://github.com/hlslibs

[3] Yuze Chi, Licheng Guo, Jason Lau, Young-kyu Choi, Jie Wang, and Jason Cong. 2021. Extending High-Level Synthesis for Task-Parallel Programs. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 204–213. https://doi.org/10.1109/FCCM51124.2021.00032

[4] Young-kyu Choi, Yuze Chi, Weikang Qiao, Nikola Samardzic, and Jason Cong. 2021. HBM Connect: High-Performance HLS Interconnect for FPGA HBM. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Virtual Event, USA) *(FPGA '21)*. Association for Computing Machinery, New York, NY, USA, 116–126. https://doi.org/10.1145/3431920.3439301

[5] Jason Cong, Bin Liu, Stephen Neuendorffer, Juanjo Noguera, Kees Vissers, and Zhiru Zhang. 2011. High-Level Synthesis for FPGAs: From Prototyping to Deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 4 (2011), 473–491. https://doi.org/10.1109/TCAD.2011.2110592

[6] M. Fingeroff. 2021. *Machine Learning at the edge: using HLS to optimize power and performance.* https://resources.sw.siemens.com/en-US/white-paper-machine-learning-at-the-edge-using-hls-to-optimize-power-and-performance

[7] Licheng Guo, Yuze Chi, Jie Wang, Jason Lau, Weikang Qiao, Ecenur Ustun, Zhiru Zhang, and Jason Cong. 2021. AutoBridge: Coupling Coarse-Grained Floorplanning and Pipelining for High-Frequency HLS Design on Multi-Die FPGAs. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Virtual Event, USA) *(FPGA '21)*. Association for Computing Machinery, New York, NY, USA, 81–92. https://doi.org/10.1145/3431920.3439289

[8] Licheng Guo, Jason Lau, Yuze Chi, Jie Wang, Cody Hao Yu, Zhe Chen, Zhiru Zhang, and Jason Cong. 2020. Analysis and Optimization of the Implicit Broadcasts in FPGA HLS to Improve Maximum Frequency. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218718

[9] Brucek Khailany, Evgeni Krimer, Rangharajan Venkatesan, Jason Clemons, Joel S. Emer, Matthew Fojtik, Alicia Klinefelter, Michael Pellauer, Nathaniel Pinckney, Yakun Sophia Shao, Shreesha Srinath, Christopher Torng, Sam Likun Xi, Yanqing Zhang, and Brian Zimmer. 2018. INVITED: A Modular Digital VLSI Flow for High-Productivity SoC Design. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465897

[10] Aki Kuusela and Clint Smullen. 2021. Video Coding Unit (VCU) : Hot Chips 2021. In *2021 IEEE Hot Chips 33 Symposium (HCS)*. 1–30. https://doi.org/10.1109/HCS52781.2021.9567040

[11] Sae Kyu Lee, Paul N. Whatmough, Marco Donato, Glenn G. Ko, David Brooks, and Gu-Yeon Wei. 2022. SMIV: A 16-nm 25-mm² SoC for IoT With Arm Cortex-A53, eFPGA, and Coherent Accelerators. *IEEE Journal of Solid-State Circuits* 57, 2 (2022), 639–650. https://doi.org/10.1109/JSSC.2021.3115466

[12] Nathaniel Pinckney, Rangharajan Venkatesan, Ben Keller, and Brucek Khailany. 2021. IPA: Floorplan-Aware SystemC Interconnect Performance Modeling and Generation for HLS-based SoCs. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9. https://doi.org/10.1109/ICCAD51958.2021.9643499

[13] Kartik Prabhu, Albert Gural, Zainab F. Khan, Robert M. Radway, Massimo Giordano, Kalhan Koul, Rohan Doshi, John W. Kustin, Timothy Liu, Gregorio B. Lopes, Victor Turbiner, Win-San Khwa, Yu-Der Chih, Meng-Fan Chang, Guénolé Lallement, Boris Murmann, Subhasish Mitra, and Priyanka Raina. 2022. CHIMERA: A 0.92-TOPS, 2.2-TOPS/W Edge AI Accelerator With 2-MByte On-Chip Foundry Resistive RAM for Efficient Training and Inference. *IEEE Journal of Solid-State Circuits* (2022), 1–1. https://doi.org/10.1109/JSSC.2022.3140753

[14] Parthasarathy Ranganathan, Daniel Stodolsky, Jeff Calow, Jeremy Dorfman, Marisabel Guevara, Clinton Wills Smullen IV, Aki Kuusela, Raghu Balasubramanian, Sandeep Bhatia, Prakash Chauhan, Anna Cheung, In Suk Chong, Niranjani Dasharathi, Jia Feng, Brian Fosco, Samuel Foss, Ben Gelb, Sara J. Gwin, Yoshiaki Hase, Da-ke He, C. Richard Ho, Roy W. Huffman Jr., Elisha Indupalli, Indira Jayaram, Poonacha Kongetira, Cho Mon Kyaw, Aaron Laursen, Yuan Li, Fong Lou, Kyle A. Lucke, JP Maaninen, Ramon Macias, Maire Mahony, David Alexander Munday, Srikanth Muroor, Narayana Penukonda, Eric Perkins-Argueta, Devin Persaud, Alex Ramirez, Ville-Mikko Rautio, Yolanda Ripley, Amir Salek, Sathish Sekar, Sergey N. Sokolov, Rob Springer, Don Stark, Mercedes Tan, Mark S. Wachsler, Andrew C. Walton, David A. Wickeraad, Alvin Wijaya, and Hon Kwan Wu. 2021. Warehouse-Scale Video Acceleration: Co-Design and Deployment in the Wild. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS 2021)*. Association for Computing Machinery, New York, NY, USA, 600–615. https://doi.org/10.1145/3445814.3446723

[15] Thierry Tambe, En-Yu Yang, Glenn G. Ko, Yuji Chai, Coleman Hooper, Marco Donato, Paul N. Whatmough, Alexander M. Rush, David Brooks, and Gu-Yeon Wei. 2021. 9.8 A 25mm2 SoC for IoT Devices with 18ms Noise-Robust Speech-to-Text Latency via Bayesian Speech Denoising and Attention-Based Sequence-to-Sequence DNN Speech Recognition in 16nm FinFET. In *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, Vol. 64. 158–160. https://doi.org/10.1109/ISSCC42613.2021.9366062

[16] Thierry Tambe. 2022. *Effective SW/HW Co-Design of Specialized ML Accelerators using Catapult HLS.* Siemens Webinar. https://event.on24.com/wcc/r/3549953/BDEFF7A2C76676D76EF4C26F6A580A1B?partnerref=

[17] Brian Zimmer, Rangharajan Venkatesan, Yakun Sophia Shao, Jason Clemons, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel S. Emer, C. Thomas Gray, Stephen W. Keckler, and Brucek Khailany. 2019. A 0.11 pJ/Op, 0.32-128 TOPS, Scalable Multi-Chip-Module-based Deep Neural Network Accelerator with Ground-Reference Signaling in 16nm. In *2019 Symposium on VLSI Circuits*. C300–C301. https://doi.org/10.23919/VLSIC.2019.8778056

[18] Brian Zimmer, Rangharajan Venkatesan, Yakun Sophia Shao, Jason Clemons, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel S. Emer, C. Thomas Gray, Stephen W. Keckler, and Brucek Khailany. 2020. A 0.32–128 TOPS, Scalable Multi-Chip-Module-Based Deep Neural Network Inference Accelerator With Ground-Referenced Signaling in 16 nm. *IEEE Journal of Solid-State Circuits* 55, 4 (2020), 920–932. https://doi.org/10.1109/JSSC.2019.2960488