Quantifying the gap between open-source and vendor FPGA place and route tools

Shachi Khadilkar
ECE Department
UMass Lowell
Lowell, United States
ShachiVaman_Khadilkar@student.uml.edu

Ahmed Sanaullah
Red Hat Research
Red Hat Inc.
Boston, United States
asanaull@redhat.com

Martin Margala
School of Computing and Informatics
University of Louisiana at Lafayette
Lafayette, United States
martin.margala@louisiana.edu

***Abstract*—The use of Field Programmable Gate Arrays (FPGAs) has increased greatly as a result of their flexibility, power efficiency, and hardware acceleration capabilities. CAD tools needed to map hardware description language (HDL) code to the FPGA are complex and challenging to build. Open-source CAD tools for FPGAs have been developed to facilitate restriction-free customization and have more control over the mapping process. A significant milestone in the development of open-source CAD tools was the ability to target real commercial devices. In recent years, multiple academic CAD tools can map circuits to commercial FPGAs. It is essential to identify and quantify the performance gap between academic and vendor tools targeting commercial devices. To this effect, we compare relevant hardware quality metrics after placement and routing for five tool-flows targeting a commercial FPGA. Our results show the divide between academic and commercial place and route tools for device utilization, run time, and maximum circuit speeds.**

***Index Terms*—Computer Aided Design, FPGAs, place and route, tool comparison.**

I. Introduction

FPGAs have gained popularity after Moore's law ended as they are well suited for a variety of applications like hardware acceleration, machine learning, edge computing, digital signal processing. FPGAs can easily be reconfigured "in the field" and are equipped with parallel compute capabilities, power efficiency and incur lower non-recurring engineering costs compared to Application Specific Integrated Circuits (ASICs) [1]. Digital circuits written in high-level languages like C++ or Hardware Description Languages like Verilog require several processing steps before they can be mapped to the FPGA hardware. CAD tools are needed for this mapping. Commercial CAD tools are proprietary, often require expensive licenses, and provide limited opportunity for customization [2]. Open-source CAD tools have been developed for the past several years to overcome the problems associated with commercial FPGA CAD tools. Older research works show open-source tool flows built for theoretical architectures, possibly because of the heavy dependence of several stages of the FPGA CAD tool flow on knowledge of the device architecture, and these details are not known for vendor devices. In recent times, several tool-flows have been built for commercial FPGAs as well as device architectures of some vendor FPGAs are now available [3]. Open-source CAD tools allow researchers to modify and build on existing work. However, there is a significant gap in hardware quality generated by vendor CAD tools and open-source tools. Prior research work mentions that new academic architectures and tool flows for FPGAs are declining, and the ones that are proposed do not provide major improvements when used with a commercial setup [25]. [25] claim that the large performance gap between open-source and commercial tools is responsible for this. This research work is aimed at comparing the hardware quality of a set of real and synthetic benchmarks placed and routed with commercial and open-source tools, targeting the same commercial FPGA. We compare open-source place and route tools Nextpnr [8] and Versatile Place and Route (VPR) [3], [6] with state-of-the-art commercial tools, Xilinx Vivado 2017.2 and Xilinx Vivado 2023.1 [26]. We use synthesis tools Yosys [12] (open-source) and commercial Xilinx Vivado.

II. Background

The FPGA CAD tool has a series of steps from HDL/high-level code to a bitstream and, finally programming the hardware.

A. FPGA CAD flow

The typical CAD tool flow:

- Synthesis: In this step, the digital circuit described by an HDL like Verilog/VHDL is converted to a netlist, which describes the logical connectivity between the logic/hard blocks on a particular FPGA [4].
- Technology Mapping: Technology mapping accounts for the variations in building blocks across different FPGA architectures (E.g., Block RAMs and DSP blocks may be different for different device architectures). Any generic blocks in the netlist are replaced with specific ones with the help of a technology mapping library [4].
- Packing/Clustering: The basic logic elements like Look Up Tables (LUTs) and Flip-flops (FFs) are grouped into clusters according to resource and timing information [4].
- Placement: The packed netlist has all the clustered netlist elements; the placement algorithm finds the optimal location for all the packed clusters. This is influenced by the placement algorithm (e.g., simulated annealing) and the optimization goals, etc. [4].

- Routing: The routing algorithm finds the optimal route to connect the placed clusters [4]. Routing is the most compute-intensive and time-consuming task in the implementation (place and route) process.
- Bitstream generation: The implemented netlist is then converted to a bitstream, which can be loaded into the FPGA to program it.

B. Related work

Several academic FPGA CAD tools are based on state-of-the-art open-source FPGA CAD tool VTR (Verilog to Routing), which includes open-source place and route tool VPR [6]. Versatile Place and Route (VPR) was developed several years ago to target a wide range of FPGA devices with circuit design sizes comparable to industrial ones at the time [5]. Since then, several enhancements have been made to the VPR tool. The VTR project integrates Odin II [13], ABC [11], and VPR for synthesis, technology mapping, and implementation (place and route), respectively [6]. VTR facilitates architecture exploration, provides flexibility, and is well-documented [6]. There are a few previous research works that analyze the gap between academic and commercial FPGA CAD tools. In [7], a novel open-source framework, VTR to bitstream (VTB), was proposed to enable researchers to use their innovations and techniques on commercial Xilinx FPGAs. They enhance VTR's capabilities to generate Xilinx bitstreams using Xilinx Design Language (XDL) [7]. One of the major contributions was enabling academic researchers to assess their CAD tools on real vendor FPGAs instead of previously used theoretical architectures [7]. They used VTB to compare the open-source academic tool VTR with its vendor counterpart Xilinx ISE targeting Virtex-6 FPGAs [7]. Their results showed the disparity between Xilinx ISE and VTR with respect to area and delays [7]. The performance gap is attributed to missing architectural components like carry chains and better commercial CAD algorithms [7]. [24] presents Verilog to Bitstream 2.0, which includes capabilities to pack, place, and route (routing is done by Xilinx in earlier VTB version) netlists which can then be programmed to Xilinx devices. VTB 2.0 is then used to analyze the discrepancy between research and commercial performance after each major stage in the CAD tool flow: synthesis, packing placement, and routing targeting a Xilinx Virtex-6 FPGA [24]. [24] show that maximum degradation for the delay gap between research and vendor tools is seen with the synthesis step, followed by routing, and minimum degradation is seen at the pack and place step. According to [24], this points to opportunities to work on and improve the front-end tools and not just place and route tools. [25] compare Xilinx vendor tools with equivalent academic tools. As VTR could not be used to target vendor FPGAs, for the academic implementation, an advanced academic target FPGA was used with process technology close to the Xilinx device being used [25]. Xilinx Vivado is used with 20 nm Ultrascale Kintex FPGAs and compared with VTR targeting a similar 22nm architecture [25]. Their results show significant gaps in speed performance and runtime [25]. [25] also enhanced VTB to target Xilinx Virtex-7 FPGAs to compare Xilinx Vivado and VTB. They reported that Vivado designs occupy less area and operate at higher frequencies [25]. In recent years, more academic tool-flows can be used with real commercial FPGAs. For example, the Symbiflow (f4pga) project [3] uses Yosys, ABC, and VPR for synthesis, technology mapping, and implementation, respectively. They have extended VPR to target commercial devices, and their framework can generate bitstreams for a Xilinx Artix-7 device [3]. Yosys + Nextpnr can target commercial FPGAs from Lattice and Xilinx [8].

III. Method

A. Experimental Setup

For the first tool flow, we have used the open-source tool Yosys for synthesis, and the commercial tool Xilinx Vivado 2017.2 [28] for placement and routing. The second tool flow has Yosys for synthesis and open-source tool VPR/Symbiflow [3], [6], [21] for pack, placement and routing. The third tool flow uses Yosys for synthesis and the open-source tool Nextpnr-xilinx [23] for packing, placement, and routing. The fourth tool flow uses Vivado 2017.2 for synthesis and place and route. The fifth tool flow uses Yosys for synthesis and Vivado 2023.1 for placement and routing. In summary, four tool-flows use the Yosys front-end, implementation (pack,place and route) for two of these four tools is done using open-source tools Nextpnr-xilinx and VPR/SYmbiflow. The other two of these four tools are commercial Xilinx Vivado 2017.2 and Xilinx Vivado 2023.1. For Vivado 2023.1, we use the default mode and not the Intelligent Design Runs [27]. The remaining toolflow is synthesized and implemented by Xilinx Vivado so we can observe the effect of changing the front-end tool. We use a Xilinx Artix-7 FPGA [14] device for our experiments, as all the tools we have discussed above can implement designs on Artix-7.

B. Benchmarks

We use a few real benchmarks from typical CAD evaluation benchmark suites and also build a set of synthetic benchmarks for our experiments. Our synthetic benchmarks are similar to hardware patterns described in [22]. Our synthetic benchmarks include simple circuits of connected basic logic elements like Look Up Tables (LUTs), Flip-flops(FFs), and carry chains [22] that can be varied to achieve higher utilization. Our synthetic benchmarks also include circuits with over 70% slice register utilization and slice LUT utilization. Example synthetic benchmarks can be seen in Figures 2,3 and 4.

C. Standardization and constraints

We take reasonable measures to standardize our experiments across all the tool flows. We have used the Xilinx xc7a50tfgg484 [19] device across all tool flows. Since several benchmarks need more I/O than what is available on the FPGA, we use BRAMs for getting additional input signals. For the synthetic benchmarks, we prevent the synthesis tools from over-optimizing the circuit. We use equivalent physical constraints for clocks and I/O across all tool flows. We have

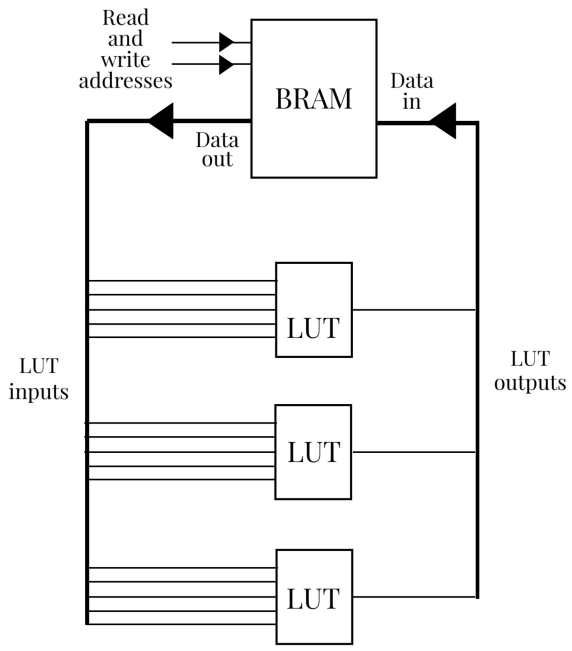

Fig. 1. Synthetic benchmark: LUT chain

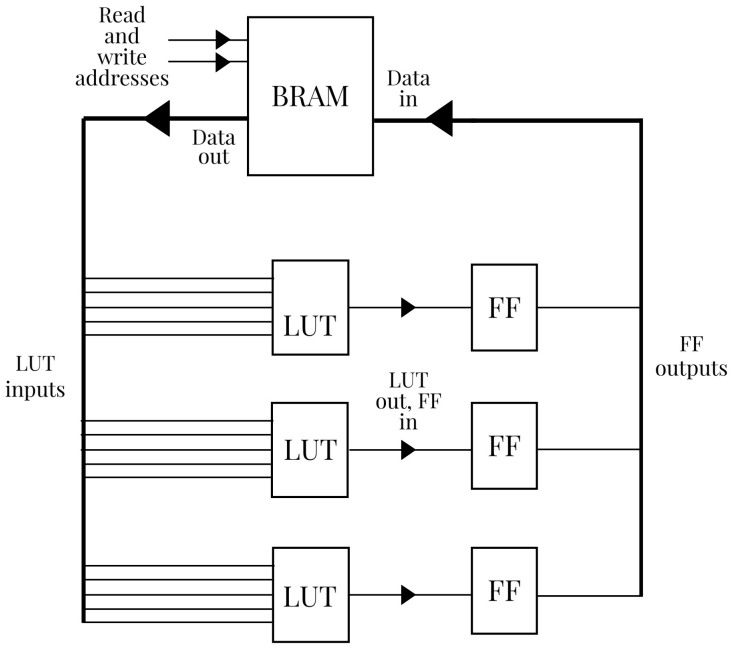

Fig. 2. Synthetic benchmark: LUT FF chain

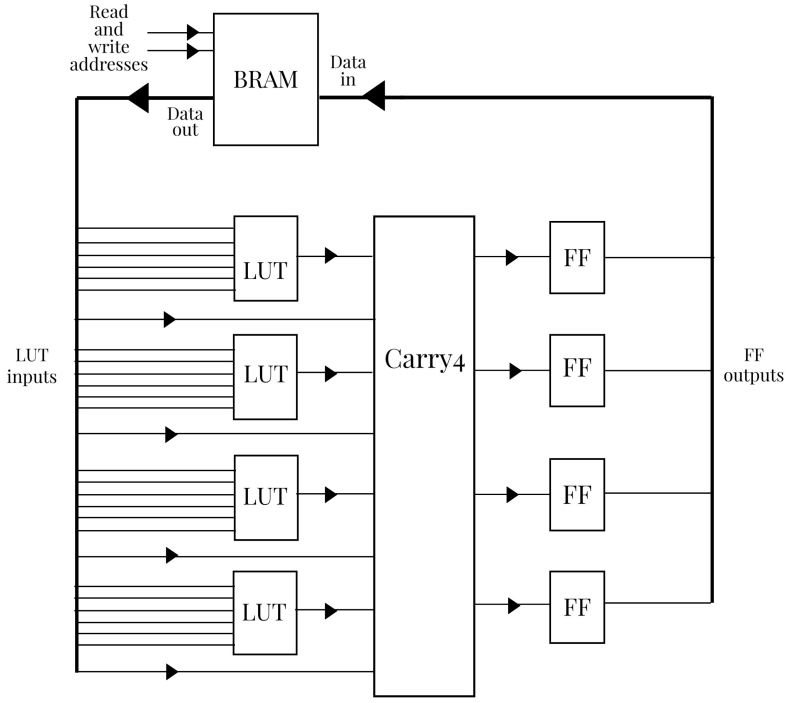

Fig. 3. Synthetic benchmark: LUT Carry FF chain

used a trial-and-error approach to set timing constraints that are just met.

IV. Results

All the tool-flows are compared for hardware metrics post routing. We measure maximum frequency, implementation run times, LUT, and FF utilization. Measurements are normalized to Yosys + Vivado 2017. Of the eleven benchmarks we have used, the first five are real, and the remaining are synthetic. Diffeq1, Diffeq2, and sha benchmarks are from the VTR benchmark suite [15], Picosoc is from the Symbiflow benchmarks [21], and the fifth benchmark used is the Murax core [16]. lut5chain-500 and lut6chain-500 are chains of 500 5-LUTs and 500 6-LUTs respectively. ffchain-1k is a chain of 1000 flip-flops. Lut5ffchain-500 and lut6ffchain-500 are chains of 500 lut5-ff pairs and 500 lut6-ff pairs, respectively. The inputs for these five synthetic benchmarks are sourced from cascaded BRAMs. ffchain-12k is a chain of 12000 flip-flops. Synthetic benchmarks also include ffchain-48k with 48000 chained LUTs, lut5chain12k, and lut5chain24k with 12000 and 24000 5-luts. ffchain-48k uses 73.6% slice registers on the device, and lut5chain_24k uses 73.6% Slice LUTs on the device (from Vivado reports).

V. Discussion

From figure 4, average max. circuit frequency (fmax) is highest for Yosys + Nextpnr (2.01x higher than average fmax for Yosys + Vivado 2017). Yosys + VPR 8 has lowest average fmax (0.78x fmax for Yosys + Vivado 2017). Average fmax

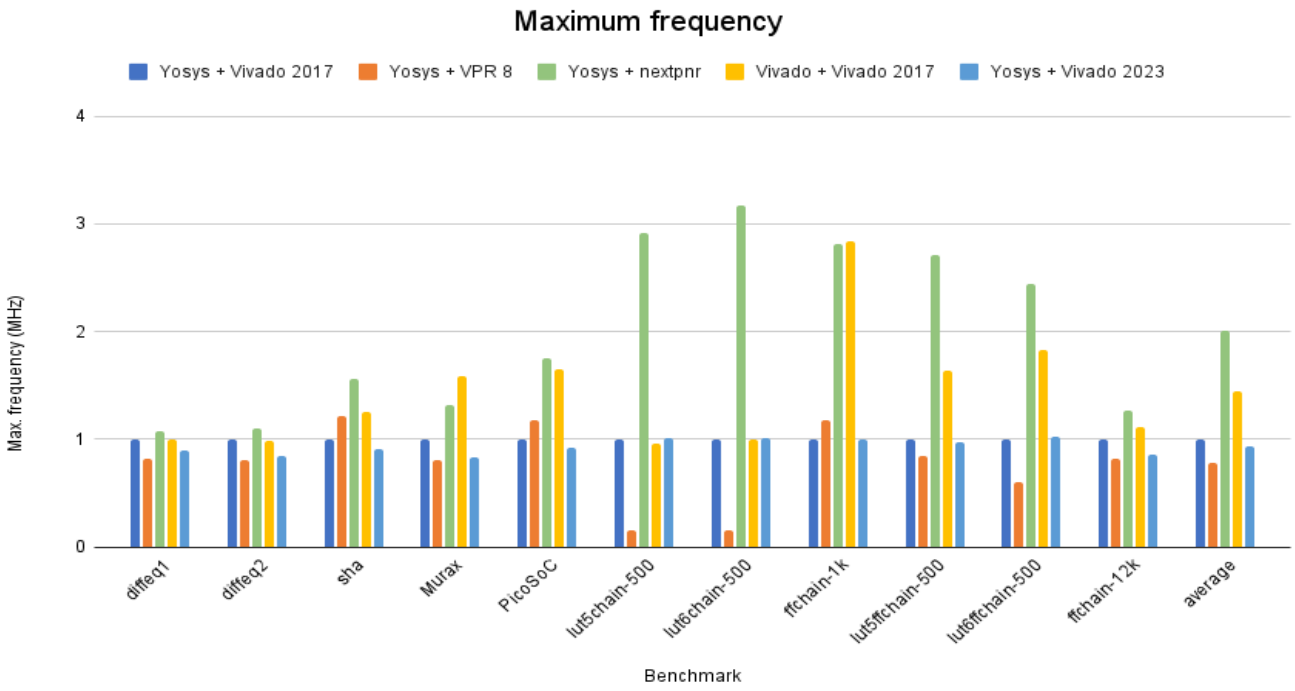

Fig. 4. Maximum frequency across all tool flows

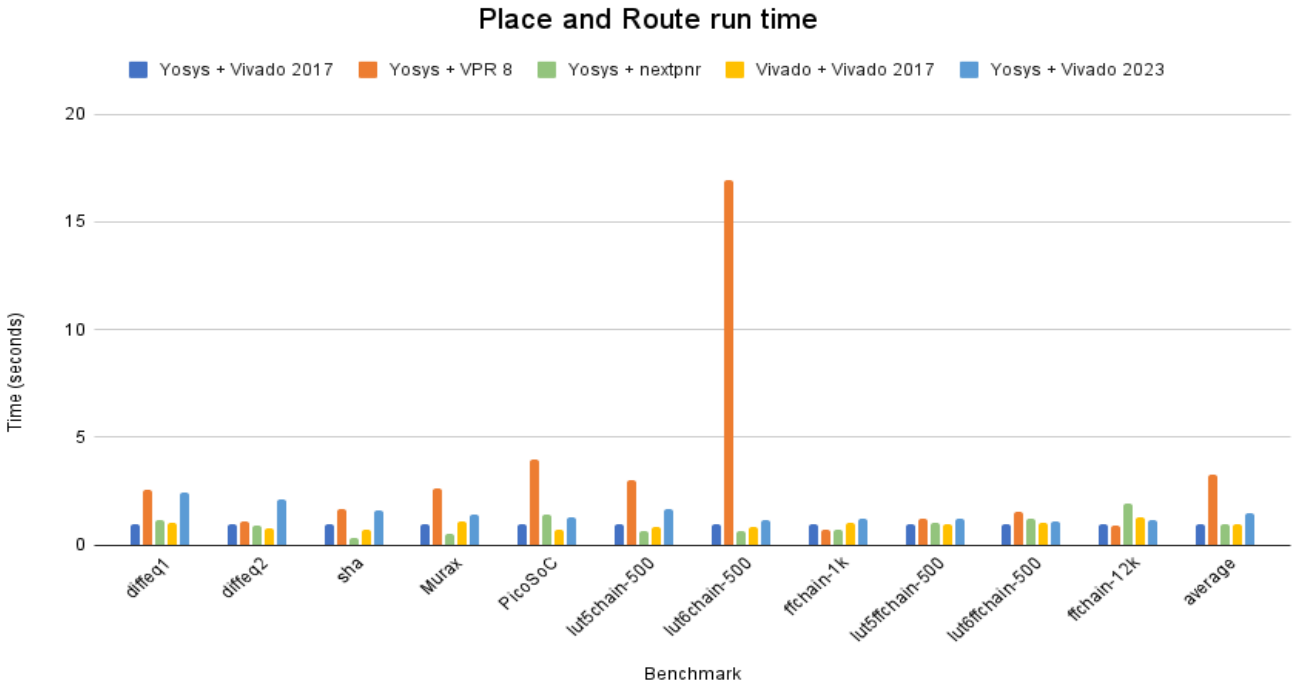

Fig. 5. Implementation run time across all tool flows

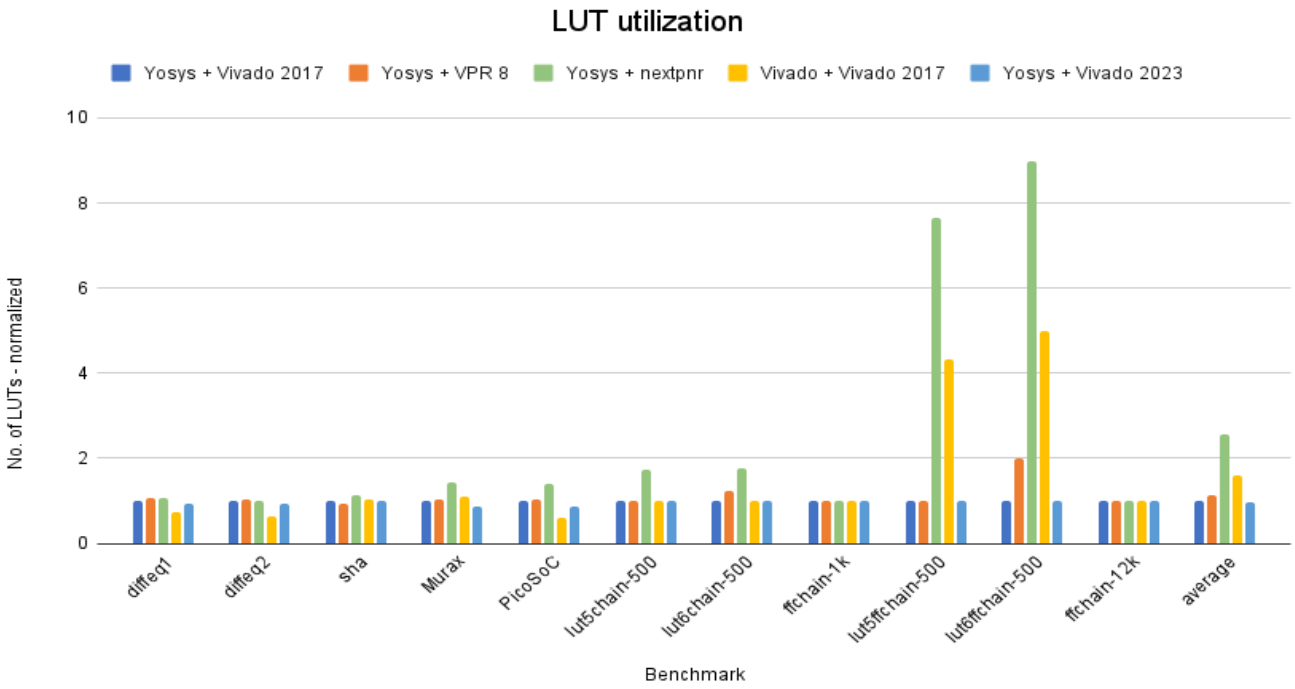

Fig. 6. LUT utilization across all tool flows

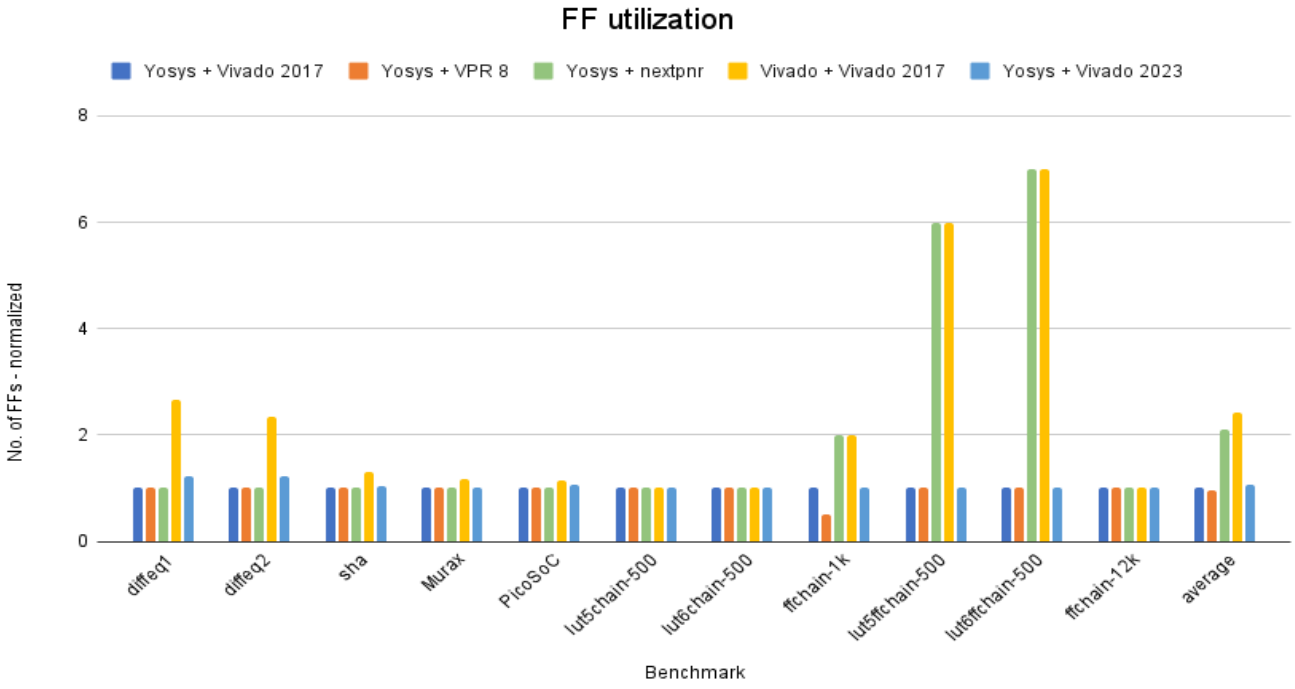

Fig. 7. FF utilization across all tool flows

for Vivado + Vivado 2017 is 1.44x higher than average fmax for Yosys + Vivado 2017; this could point to scope for improvements in the open-source synthesis tool.

Figure 5 shows the place and route run time for the five tool-flows under consideration. VPR has the slowest average implementation run time (3.3x slower than Yosys + Vivado 2017). Yosys + Nextpnr has the fastest average implementation run time (0.95x faster than Yosys + Vivado 2017).

Figure 6 shows the LUT utilization for the tool-flows. Average LUT utilization over all benchmarks for Yosys + Nextpnr is 2.56x higher than average LUT utilization for Yosys + Vivado 2017. Yosys + VPR has 1.12x higher LUT utilization than Yosys + Vivado 2017. Yosys + Vivado 2023 has the lowest LUT utilization (0.95x lower than Yosys + Vivado 2017).

Figure 7 shows Flip-flop utilization for all tool flows. Vivado 2017 + Vivado 2017 has highest average FF utilization (2.42x higher than Yosys + Vivado 2017). Yosys + Nextpnr average LUT utilization is 2x that of Yosys + Vivado 2017. Yosys + VPR has the lowest LUT utilization (0.95x Yosys + Vivado 2017).

We also run three additional synthetic benchmarks - ffchain48k (0 LUTs, 48k FF), lut5chain12k (12k 5-LUTs, 0 FF) and lut5chain24k (24k 5-LUTs, 0 FF). Vivado 2017 and Vivado 2023 can place and route all three benchmarks. VPR and Nexpnr fail to place and route ffchain48k synthetic benchmark.

VI. Conclusion

In this work, we have compared results from five different tool flows across the Xilinx Artix-7 board and quantified the gap between open-source and commercial implementation CAD tools for a set of hardware quality metrics.

Acknowledgment

We thank Red Hat for their financial support for this project.

References

[1] W. Li and D. Z. Pan, “A New Paradigm for FPGA placement Without Explicit Packing,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 11, pp. 2113-2126, Nov. 2019.

[2] A. Sanaullah, “Rolling your own processor”, Red Hat Research Quarterly, Vol.1:3, 2019.

[3] K. Murray, T. Ansell, K. Rothman, A. Komodi, M. Elgammal and V. Betz, “Symbiflow & VPR: An open-source design flow for commercial and novel FPGAs” IEEE Micro vol. 40, issue 4, 2020.

[4] “F4PGA Documentation,” https://f4pga.readthedocs.io/en/latest/ (accessed: Sept. 9, 2023).

[5] V. Betz and J. Rose, “VPR: A new packing, placement and routing tool for FPGA research,” in Proc. of 7th International Workshop on Field-Programmable Logic and Applications, SpringerVerlag, Berlin, Heidelberg, 213–222, 1997.

[6] K. E. Murray et al., “VTR 8: High-performance CAD and Customizable FPGA Architecture Modelling,” ACM Trans. Reconfigurable Technol. Syst., 13, 2, Article 9, June 2020.

[7] E. Hung, F. Eslami and S. J. E. Wilton, “Escaping the Academic Sandbox: Realizing VPR Circuits on Xilinx Devices,” 2013 IEEE 21st Annual International Symp. on Field-Programmable Custom Computing Machines, pp. 45-52, 2013.

[8] D. Shah, E. Hung, C. Wolf, S. Bazanski , D. Gisselquist and M. Milanovic´, “Yosys + nexpnr: an Open Source Framework from Verilog to Bitstream for Commercial FPGAs”, in IEEE Field Programmable Custom Computing machines (FCCM), 2019.

[9] C. Lavin and A. Kaviani, “RapidWright: Enabling Custom Crafted Implementations for FPGAs,” 2018 IEEE 26th Annual International Symp. on Field-Programmable Custom Computing Machines, pp. 133-140, 2018.

[10] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson and B. Hutchings, “RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs,” 2011 21st International Conference on Field Programmable Logic and Applications, pp. 349-355, 2011.

[11] R Brayton and A. Mishchenko, “ABC: An Academic Industrial-Strength Verification Tool,” in Touili, T., Cook, B., Jackson, P. (eds) Computer Aided Verification. CAV 2010. Lecture Notes in Computer Science, vol 6174. Springer, Berlin, Heidelberg.

[12] “Yosys Open SYnthesis Suite,” https://yosyshq.net/yosys/ (accessed: Nov.27, 2022).

[13] P. Jamieson, K. Kent, F. Gharibian and L. Shannon, “Odin II - An Open-Source Verilog HDL Synthesis Tool for CAD Research,” 18th IEEE Annual International Symp. on Field-Programmable Custom Computing Machines, pp. 149-156, 2010.

[14] “Artix-7,” https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html (accessed: Sept.9, 2023).

[15] “VTR Benchmarks,” https://docs.verilogtorouting.org/en/latest/vtr/benchmarks/ (accessed: Sept.9, 2023).

[16] GitHub repository for RISC-V implementations in SpinalHDL, https://github.com/SpinalHDL/VexRiscv#murax-soc (accessed: Sept.10, 2023).

[17] “Xilinx Architecture Terminology,” https://www.rapidwright.io/docs/Xilinx_Architecture.html (accessed: Sept. 10, 2023).

[18] “Vivado Design Suite Properties Reference Guide,” https://docs.xilinx.com/r/en-US/ug912-vivado-properties/SITE (accessed: Sept. 9, 2023).

[19] AMD Xilinx page describing xc7a50tfgg484 device package, https://www.xilinx.com/content/dam/xilinx/support/packagefiles/a7packages/xc7a50tfgg484pkg.txt (accessed: Sept. 8, 2023).

[20] F4PGA documentation ”Supported Architectures”, https://f4pga.readthedocs.io/en/latest/status.html (accessed: Sept. 9, 2023).

[21] Github for Symbiflow(f4pga), https://github.com/f4pga (accessed: Sept. 10, 2023).

[22] S. Khadilkar and M. Margala, ”Optimizing open-source FPGA CAD tools,” 2022 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1-4, 2022.

[23] Github page for nextpnr-xilinx, https://github.com/gatecat/nextpnr-xilinx (accessed: May.10 2023).

[24] E. Hung, ”Mind the (synthesis) gap: Examining where academic FPGA tools lag behind industry,” 2015 25th International Conference on Field Programmable Logic and Applications, London, UK, pp. 1-4, 2015.

[25] E. Vansteenkiste, A. Kaviani and H. Fraisse, ”Analyzing the divide between FPGA academic and commercial results,” 2015 International Conference on Field Programmable Technology, Queenstown, New Zealand, pp. 96-103, 2015.

[26] AMD Xilinx page for Vivado ML 2023.1, https://www.xilinx.com/support/download.html (accessed: June 15th, 2023).

[27] Vivado Design Suite User Guide, https://docs.xilinx.com/r/en-US/ug906-vivado-design-analysis/Intelligent-Design-Runs (accessed Sept. 9, 2023).

[28] Vivado 2017.2, https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html (accessed Sept. 9, 2023).