Exploiting Open Source Tools for FPGA Design Flow

Author 1*, Author 2*, Author 3*

***Abstract*—The escalating demands of data processing applications have propelled Field Programmable Gate Arrays (FPGAs) into the forefront of programmable accelerators in modern computing systems. FPGA architectures must be meticulously designed to seamlessly integrate with other computing resources to optimize processing efficiency for specific domains. Nevertheless, developing and deploying customized FPGAs remains a costly and intricate undertaking, even for established industrial firms. This paper demonstrates a fully free, open-source software (FOSS) for the FPGA design flow. Here, all the open-source tools are installed and packaged as a container and integrated with a federated platform for easy access to the users. The results show that the end-to-end resource allocation for reserving the FOSS container is around 11sec.**

***Index Terms*—Docker, Containers, FPGA, Open-source, Jupyter Notebooks, Yosys, Nextpnr.**

I. Introduction

A field-programmable gate array (FPGA) is a semiconductor device that can be programmed after manufacture to perform a specific application design, typically specified as a digital logic system. Since their introduction, there has been rapid growth, and they have become a popular digital circuit implementation. Lately, FPGAs have been extensively used in the field of Artificial Intelligence, specifically in the implementation of Deep Learning and Neural Networks harnessing their acceleration capabilities [1]. FPGAs play a major in next-generation networks (e.g., 5G and beyond), for example, [2] implemented various 5G functions like the Low-PHY functions.

FPGAs are becoming increasingly prominent as programmable accelerators in a wide range of modern computing systems as the demands of data processing applications continue to increase. Modern FPGAs have millions of logic elements that require the use of Hardware Description Language (HDL) and Computer Aided Design (CAD) tools to program. The software flow (CAD flow) takes an application design description in a Hardware Description Language (HDL) and converts it to a stream of bits that is eventually programmed on the FPGA. Converting a circuit description into a format that can be loaded into an FPGA can be roughly divided into five distinct steps: synthesis, technology mapping, input-output (io) mapping, placement, and routing.

It is well known that the FPGA's performance highly depends upon the incorporated design flow. Access to FPGA silicon is generally available through closed-source Computer Aided Design (CAD) tools provided exclusively by the FPGA vendor. In addition, a significant challenge in adopting FPGAs licensing model, unlike other computing devices, FPGA users must pay an ongoing license fee to the vendor in order to program and utilize the hardware. Moreover, this restriction applies even in cloud environments [3], where users must acquire a license before effectively utilizing a rented FPGA cluster. This licensing barrier poses a major hurdle for many potential FPGA users, limiting their ability to leverage the full capabilities of this powerful technology. The current FPGA market is saturated by closed-source vendor-specific tools.

By eliminating these licensing hurdles, FPGAs could achieve greater adaptability and enable a wider range of users to harness their processing power. Transitioning to an open-source design flow presents a promising approach — however, it has its own challenges to overcome. The development of such tools requires a deep understanding of the underlying hardware, which is often kept as a closed loop. Due to this, projects like OpenFPGA [4] are limited to building customizable architectures rather than full-flown CAD tools. Even though the OpenFPGA has the potential to evolve into a complete CAD flow by containing components for all necessary design flow stages, including yosys, Versatile Place and Route (VPR), and a bitstream generator.

As of now, most of the FPGA vendors do not supply an open-source toolchain. The available open-source CAD tools work only well for small designs; any real-world useful design will take an eternity to synthesize with higher resource utilization when compared to vendor-provided solutions. Open-source tools require a lot of optimization for efficient resource utilization. The good thing with Open-source tools is that they are cross-platform and, therefore, release developers from being tied to a single vendor. Their developments may be easily migrated from one hardware to another, even from different manufacturers.

This paper demonstrates a toolchain built from various open-source tools to automate the FPGA design flow. The built toolchain is containerized as a Jupyter docker notebook and integrated into a federated platform. A federated platform that enables its users to log in and reserve the available hardware resources like CPU, GPU, and FPGA for their tasks.

II. Related Work

This section describes the currently available open-source EDA tools:

(i) *Synthesis:* The Odin II tool[1] is used for Verilog synthesis. Odin II accepts a synthesizable subset of Verilog and supports hard-IP such as the block-RAMs, carry-chains, and multipliers

[1] github.com/verilog-to-routing/vtr-verilog-to-routing/tree/master/odin_ii

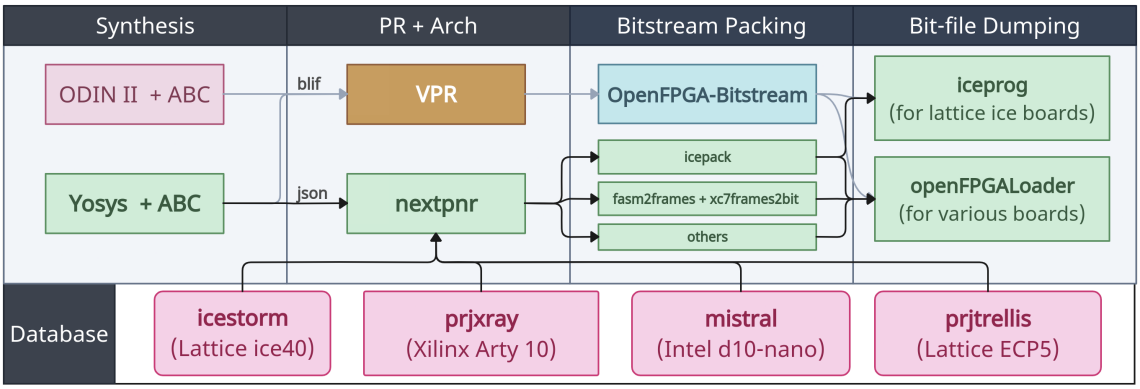

Fig. 1. FPGA Design Flow using Open Source Tools

present in VTR architectures, but can only produce Berkeley Logic Interchange Format (BLIF) netlists primarily used in research. In addition, the yosys [5] tool is also used for verilog synthesis, however, GHDL plugins are used to make VHDL synthesis possible.

(ii) *Place and Route:* Versatile Place and Route (VPR) has been a mainstay of academic research, which is a part of the VTR [6] stack. VTR architectures are described theoretically using the XML format, detailing the (proportional) makeup and layout of soft and hard blocks on the targeted FPGA. It uses architectural parameters such as the number of LUT inputs and their local and global routing connectivities.

(iii) *OpenFPGA:* OpenFPGA [4] is also an opensource tool which can be used for end-to-end FPGA flow but it uses VTR stack under the hood, making it unsuitable for Commercial Off The Shelf (COTS) FPGAs. It follows an agile approach for prototyping and offers Verilog-to-bitstream generation and verification.

The following section elaborates more on the open-source tools available for FPGA design flow.

III. Open Source Tools for FPGA Design Flow

Figure 1 depicts the built open-source flow. The components that we have used are marked in light green. The step-wise component's description is as follows:

(i) *ABC, Yosys —- Synthesis:* ABC [7] is a powerful and versatile technology mapping software designed for synthesizing and verifying binary sequential logic circuits commonly encountered in synchronous hardware designs.

Yosys [5] is a freely available verilog elaborator tool for designing digital circuits. It supports a wide range of Verilog features and can be used to create both FPGAs and ASICs. Yosys combines various optimization techniques, including ABC for logic optimization and cell mapping, to efficiently transform Verilog code into optimized circuits. A typical FPGA design flow using Yosys involves coarse-grain optimization, generic cell inference, and architecture-specific technology mapping. Yosys currently supports synthesis for the Xilinx 7-series, Lattice iCE40, Lattice ECP5, Intel (Stratix, Max, Arria series) families and many more.

(iii) *nextpnr – pack, place, route*: nextpnr [8] is a freely available, timing-driven place-and-route tool for FPGAs. It differs from many existing tools in that it describes architectures using an Application Program Interface (API) rather than formats like XML. In other words, nextpnr is a more flexible and extensible tool than many existing place-and-route tools. This is because the API allows for more nuanced descriptions of FPGA architectures. As a result, nextpnr is able to support a wider range of FPGA devices and architectures.

(iv) *Architecture Database:* Various projects that Yosys or some third party starts have used the aforementioned APIs for various boards: Project IceStrom[2] for Lattice iCE40, Project Trellis[3] for Lattice ECP5, Project Mistral[4] for Inter Cyclone V, Project Xray[5] for Xilinx Arty-7 and many more.

(v) *Bitstream Packer:* Tools like icepack, ecppack, fasm2frames & xc7frames2bit, etc are used to convert the `.asc` file — ASCII format — produced by nextpnr to `.bit` file that can be loaded into the FPGA.

(vi) *openFPGALoader:* OpenFPGALoader [9] is a universal utility for programming FPGAs. It is compatible with major manufacturers' boards, cables and FPGA (Xilinx, Altera/Intel, Lattice, etc).

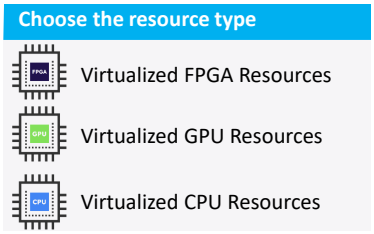

Fig. 2. Available Resources on the Federated Platform

IV. Federated Resources

This section describes the federated testbed designed for users to access the CPU remotely, GPU, and FPGA resources located in the Lab. The FPGAs are connected to servers where the virtualized FOSS containers are created to access the FPGA boards with the help of Ethernet/JTAG. A Federated testbed is a platform that allows users to register and log in

[2]github.com/YosysHQ/icestorm
[3]github.com/YosysHQ/prjtrellis
[4]github.com/Ravenslofty/mistral
[5]github.com/f4pga/prjxray

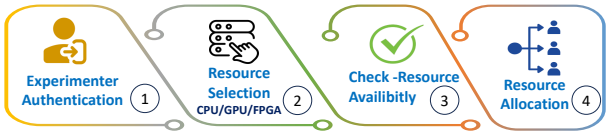

Fig. 3. Federated Platform cycle

from a remote location, reserve available hardware resources, and use them to complete their tasks. Fig 3 shows the available different resources that the users can reserve for their experimentation. The platform is built using the MERN (MongoDB, Express.js, React.js, Node.js). It uses Docker volumes to store a user's work when logged out and restores the storage when logged in. The steps in detail are as follows:

1) *User Authentication*: The registered users can log in using the credentials provided by the administrator. For security, passwords are transformed into a protected, coded format and stored in the testbed's database. The user authentication code is written in Node.js, and all the databases are handled through a local MongoDB cluster.
2) *Resource Selection*: Once a user is successfully authenticated, their name and password are saved in JWT tokens for future use. Later, the user is taken to a page showing available hardware resources such as CPU, GPU, and FPGA (shown in Fig 3). The user can choose the resource they want and is then directed to the available notebooks for the chosen resource.
3) *Confirmation of Resource Availability*: After selecting the type of hardware resource, the user needs to allocate a certain amount of the resource regarding memory and CPUs. The user will be shown the available resources of that type and can only allocate that amount.
4) *Resource Allocation*: To start a Jupyter notebook with specific details like the resource type, notebook type, and resource quantity, the process involves retrieving the user's username and password from the JWT tokens. The user will then use their password to begin the Jupyter Notebook.

V. Experimental Results

This section describes the comparison of the performances of Yosys and the vendor tools while synthesizing a program to dump on FPGA. A basic RISC processor design by the name of `RISC_posedge_clk`[6] is used to evaluate the results. The design is modified by removing Reg Wires as output in the `RISC_core_mem_top.v` to build the design for Lattice ICE40 boards. The default settings were used for all the synthesizers. All the settings in the Yosys synthesizer are defaulted except for the device family. For synth_xilinx, cascading DSPs are turned off (even though no DSP blocks are required for this design), and abc9 [10] technology mapping is used. Fig 4 shows the result obtained by using open-source tools like yosys and vendor tools (Lattice, Xlinix).

The results show the open-source tools take higher logic elements compared with vendors tools. Here, the open-sources tools are in the early stage of development and are not optimised yet. However, recent advancements with abc9 look promising for the open-source community. Moreover, we observed that open-source tools consume high time for compilation. On the other end, the vendor-supported tools are able to synthesize in very less for the same design. The end-to-end federated platform is tested by allowing 20 users to log in and reserve a container concurrently. The values in table I show each component's execution time, and their low values signify a better user experience.

[6]github.com/tangxifan/micro_benchmark/tree/main/processors

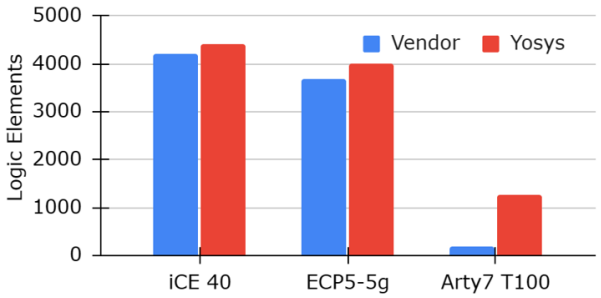

Fig. 4. Logic Elements Utilised (Lower is Better)

Function	Time Taken[s]
Experimenter Authentication	0.820
Resource Selection [CPU/GPU/FPGA]	0.177
Confirmation of Resource Availability	0.045
Resource Allocation	10.431

TABLE I
Time Taken for each Function

VI. Conclusion and Future Work

This paper demonstrates how free, open-source software (FOSS) tools can be utilised for FPGA design flow. The results show the FOSS advanced tools may closely perform to vendor tools. However, there is a strong effort needed to optimize the FOSS tools in order to reduce compilation time and memory utilisation during the design time. The future work will focus on exploiting FOSS tools more FPGA boards and looking towards adopting Open Hardware chips for FPGA designs.

References

[1] K. Liu, E. Börjeson, C. Häger, and P. Larsson-Edefors, "FPGA Implementation of Multi-Layer Machine Learning Equalizer with On-Chip Training," 2023, p. M1F.4.
[2] J. C. Borromeo, K. Kondepu, N. Andriolli, and L. Valcarenghi, "FPGA-accelerated SmartNIC for supporting 5G virtualized Radio Access Network," *Computer Networks*, vol. 210, p. 108931, 2022.
[3] R. Skhiri, V. Fresse, J. P. Jamont, B. Suffran, and J. Malek, "From fpga to support cloud to cloud of fpga: State of the art," *International Journal of Reconfigurable Computing*, vol. 2019, p. 8085461, Dec 2019.
[4] X. Tang, E. Giacomin *et al.*, "OpenFPGA: An Open-Source Framework for Agile Prototyping Customizable FPGAs," *IEEE Micro*, vol. 40, no. 4, pp. 41–48, 2020.
[5] YosysHQ. Yosys Open Synthesis System. https://github.com/YosysHQ/yosy.
[6] "Verilog to Routing – Open Source CAD Flow for FPGA Research," https://github.com/verilog-to-routing/vtr-verilog-to-routing.
[7] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 24–40.
[8] D. Shah, "YosysHQ/nextpnr: nextpnr portable FPGA place and route tool," https://github.com/YosysHQ/nextpnr, (Accessed 11-2023).
[9] G. Goavec-Merou, "Universal utility for programming FPGA," https://github.com/trabucayre/openFPGALoader, (Accessed 11-2023).
[10] B. L. Barzen *et al.*, "Narrowing the Synthesis Gap: Academic FPGA Synthesis is Catching Up With the Industry," in *In. Proc of DATE*, 2023, pp. 1–6.