OpenFPGA: Towards Automated Prototyping for Versatile FPGAs

Xifan Tang, Ganesh Gore, Edouard Giacomin, Aurélien Alacchi, Baudouin Chauviere and Pierre-Emmanuel Gaillardon
University of Utah
Email: xifan.tang@utah.edu

***Abstract*—This paper introduces an open-source framework OpenFPGA which aims to automate the design, verification and layout of highly versatile FPGA architectures. OpenFPGA offers a high-level architecture description language for users to customize their FPGA architectures down to circuit-level details. Based on the architecture modeling, OpenFPGA can auto-generate Verilog netlists, with which users can perform verification as well as generate production-ready layouts using modern EDA tools. OpenFPGA includes a generic Verilog-to-Bitstream generator, as a native EDA toolchain for any FPGAs that are prototyped by OpenFPGA. To demonstrate the capability of OpenFPGA, we showcase the <24-hour layout generation of two FPGA fabrics which are based on Stratix-like architecture built with a commercial 12nm standard-cell library and 40nm custom cells respectively.**

I. Introduction

Field Programmable Gate Arrays (FPGAs) are playing a rising role in modern computing systems, particularly as a proxy to implement accelerators, enabling high-performance data processing applications [1]. The applications create a strong need on domain-specific FPGA fabrics where specific types of computing resources, such as BRAM and DSP, are highly demanded more than any existing FPGA product can offer. However, designing an FPGA fabric costs significant human efforts and leads to long development cycles even for industrial leaders, as illustrated in Fig. 1. First, modern FPGAs contains a considerable amount of manual layouts, imposing significant hardware engineering effort when porting to a new technology node. Second, associated *Electronic Design Automation* (EDA) tools, especially bitstream generation, require adhoc software optimization for each FPGA architecture. To achieve hardware-software co-design, architects with strong expertise are needed to coordinate the hardware and software development. Considering the high development cost, general-purpose FPGAs have become the mainstream rather than domain-specific ones, missing the peak efficiency for modern data science applications. To overcome the technical and economical barriers, *embedded*

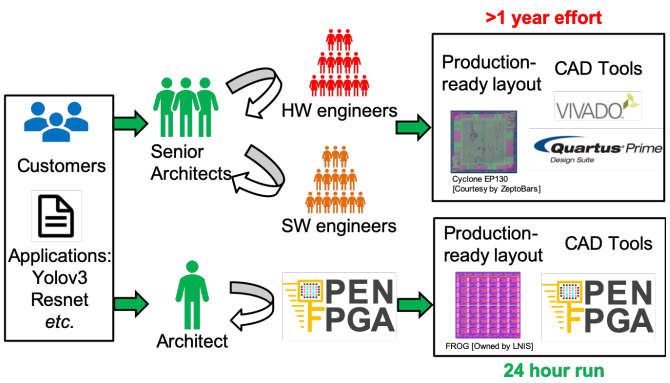


Fig. 1: Comparison on engineering time and effort to prototype an FPGA using OpenFPGA and conventional approaches.

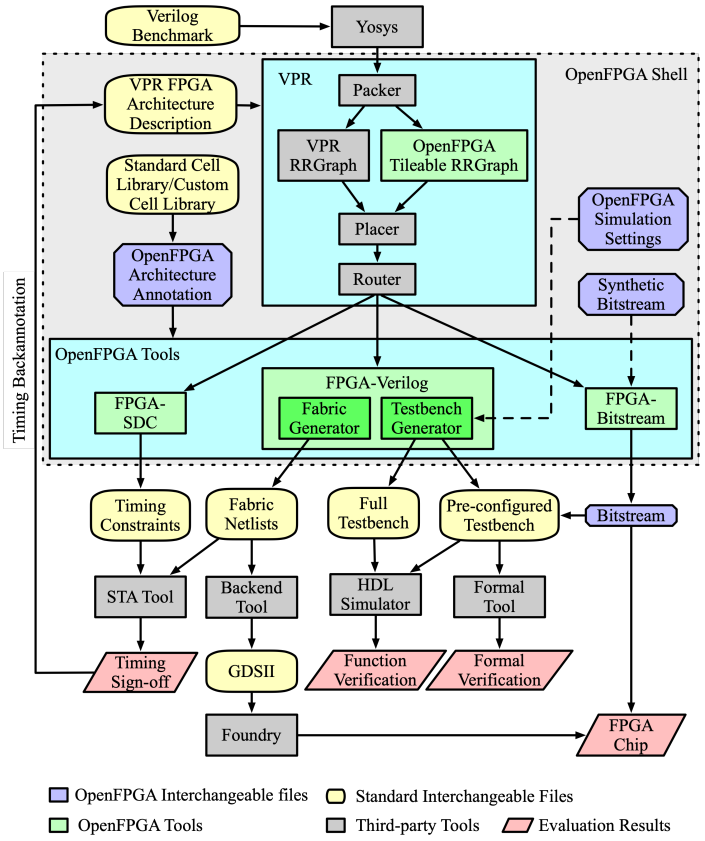


Fig. 2: OpenFPGA tool suites and design flows.

FPGA (eFPGA) industrial players and academic researchers have committed automated methodologies [2]–[7], through modeling FPGA fabrics as Verilog netlists and generating layouts using ASIC semi-custome design suites. However, these pioneering works are mostly closed source and also limited to a small set of architectures.

In this paper, we introduce OpenFPGA [1], an open-source framework that enables automated prototyping for modern and versatile FPGA architectures. To enable various design purposes, OpenFPGA integrates several tools to i.e., FPGA-Verilog, FPGA-SDC and FPGA-bitstream (highlighted green in Fig. 2), with other popular open-source EDA tools, i.e., VPR [8] and Yosys [9]. OpenFPGA offers an XML-based architecture description language for users to customize their FPGA architectures down to circuit-level details. Physical implementations of a customizable FPGA fabric can be implemented using a simple production flow in Fig. 2. An XML-based FPGA architecture description is translated to gate-level Verilog netlists of the whole FPGA fabric, and then a physical design back-end flow can be launched to obtain a complete

[1]Github: https://github.com/LNIS-Projects/OpenFPGA

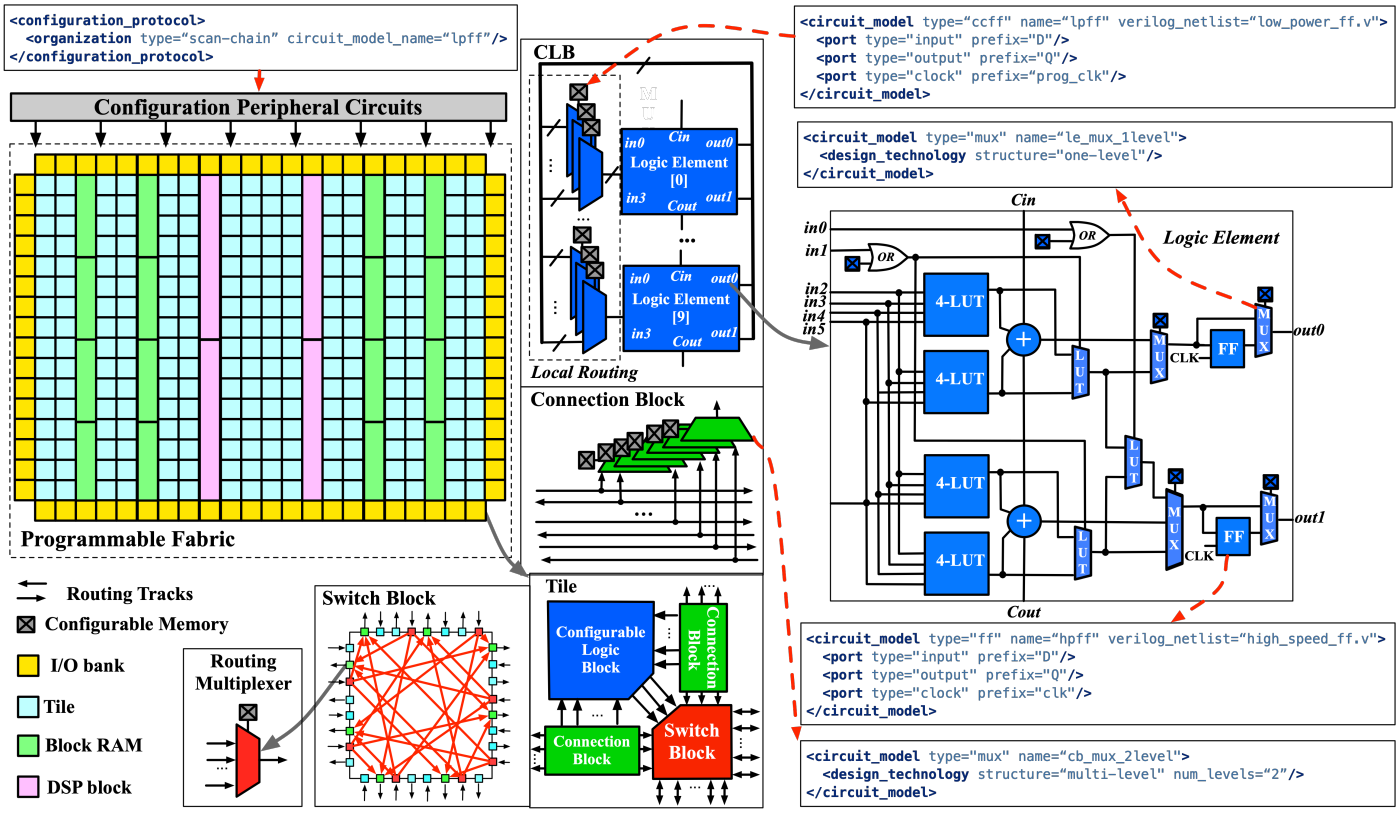


Fig. 3: OpenFPGA architecture annotation enabling fully customizable FPGA architecture and circuit-level implementation.

TABLE I: Supported primitive blocks in netlist generation

Resource Type	Auto-generation	External Netlist	Description
LUT	✓	✓	Support any input size, fracturable structure and any intermediate buffer location
Multiplexer	✓	×	One-level and any multiple-level structure (include tree structure)
Hard IP	×	✓	Include adder, flip-flop, DSP, BRAM and any other IP types that VPR can support
Configurable Memory	×	✓	Support I/O, SRAM, latch and scan-chain flip-flop

GDSII layout. To assist with sign-off, OpenFPGA is capable of auto-generating testbenches to perform pre- and post-layout verification, as well as *Synopsys Design Constraint* (SDC) files to enable timing-driven back-end flows and conduct post-layout timing analysis. FPGA developers can transcript applications written in Verilog to configuration bitstream and implement them on the FPGA fabric, by following the end-user flow in Fig. 2. OpenFPGA accepts and outputs in standard file formats, and therefore can interface a wide range of commercial and open-source tools, as listed in Table II. We demonstrate the capability of OpenFPGA by prototyping two medium-size FPGA fabrics resembling the Stratix IV architecture but different in circuit topology, backend strategy and technology nodes.

The rest of the paper is organized as follows. Section II introduces OpenFPGA's architecture description language. Section VI showcases OpenFPGA with post-layout results. Section VII concludes the paper and discusses future work.

II. OpenFPGA Architecture Description Language

XML-based FPGA architecture language is a key feature of VPR, which allows users to define versatile programmable fabrics down to point-to-point interconnection [11]. OpenFPGA leverage VPR's architecture description by introducing an XML-based architecture annotation, enabling fully customizable FPGA fabric down to circuit elements. As illustrated in Fig. 3, OpenFPGA's architecture annotation covers a complete FPGA fabric, including both the programmable fabric and the configuration peripheral. Circuit implementation of each primitive block, e.g., *Look-Up Table* (LUT), routing multiplexer and configurable memory, at any location of the architecture, can be customized. For example, the routing multiplexers in

TABLE II: Supported commercial and open-source EDA tools

Usage	Tools
Backend	Synopsys IC Compiler™ II Cadence Innovus™
STA	Synopsys PrimeTime™ Cadence Tempus™
Verification	Synopsys VCS™ Synopsys Formality™ Mentor ModelSim™ Mentor QuestaSim™ Cadence NCSim™ Icarus iVerilog
PDK/Cell Library	ASAP 7nm TSMC 40nm TSMC 180nm GF 130nm GF 12nm

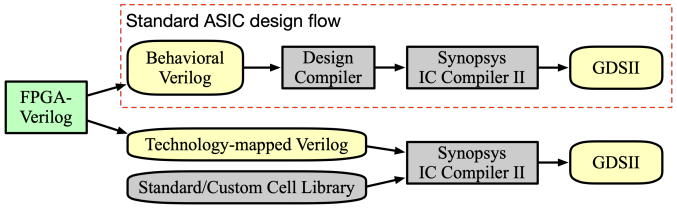


Fig. 4: Flexible netlist format supported by FPGA-Verilog to enable various backend choices.

the connection blocks may adapt a two-level structure while the routing multiplexers in the logic element is built with a one-level structure. OpenFPGA can auto generate the HDL netlists of these circuits or use an existing HDL netlists, such as standard cells or even custom cells that are crafted by users. In the example of Fig. 3, the flip-flops in the logic elements are built with a high-speed standard cell as they are in the datapath, while the flip-flops as configurable memory are built with a low-power standard cell. Table I summarize the supported circuit topology for each type of primitive blocks that may occur in FPGAs. Combined with the VPR architecture description, OpenFPGA offers a large design space for users to customize FPGAs w.r.t their *Power, Performance and Area* requirements.

The architecture annotation is a separated XML file than the VPR architecture description. Being compatible to the VPR XML syntax, architects can first use VPR to perform architecture exploration and then employ OpenFPGA to prototype their FPGA fabrics. OpenFPGA's architecture description language is fully documented at [12].

III. OpenFPGA FPGA-Verilog

FPGA-Verilog consists of two Verilog generators: the fabric generator which converts the XML-based architecture description to Verilog netlists modeling the FPGA fabric, and the testbench generator which outputs Verilog testbenches to validate the correctness of the Verilog netlists.

A. Fabric Netlist Generation

The fabric netlists includes both a programmable fabric with configuration-chain circuits embedded. As shown in Fig. 3, the programmable fabric follows a tile-based organization, where columns of tiles may be replaced by heterogeneous blocks. Note that FPGA-Verilog generates highly repeatable fabrics, which can significantly simplify the backend process. This is enabled by the tileable *Routing Resource Graph* (RRGraph) generator, which can guarantee the minimum number of connection blocks and switch blocks for any FPGA architectures. Therefore, only a few unique tiles are P&Red and then are assembled in a final layout. FPGA-Verilog is designed to output Verilog netlists in flexible format/syntax. The compatibility makes OpenFPGA as an adaptive tool, being capable of interfacing most commonly used EDA tools, as listed in Table II. For instance, FPGA-Verilog support both behavorial-level and technology-mapped fabric netlists, supporting various backend strategies, as shown in Fig. 4. The behavioral Verilog is full compatible for a standard ASIC design flow, starting from synthesis to physical design. The technology mapped Verilog can directly interface the physical design tools, where experienced chip designers can use custom cells (e.g., transmission-gate-based multiplexers) that are well established but not synthesizable using standard ASIC tools. We refer interested readers to [10], [13] for more details.

TABLE III: Auto-generated testbench features

Testbench	Runtime	Test Vector	Test Coverage
Full	Long	Random stimulus	Full fabric
Pre-configured	Short	Random stimulus/ Formal method	Programmable fabric only

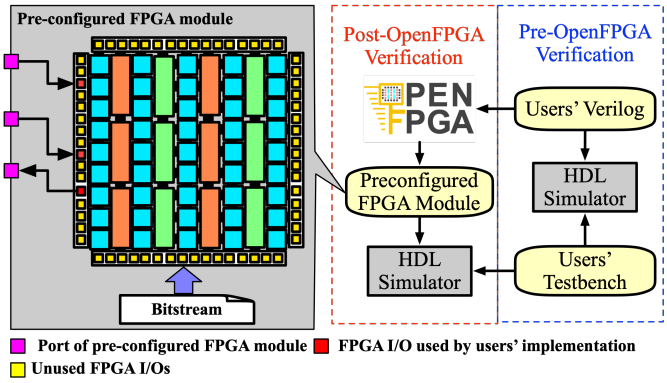


Fig. 5: Auto-generated pre-configured modules enabling testbench reuse.

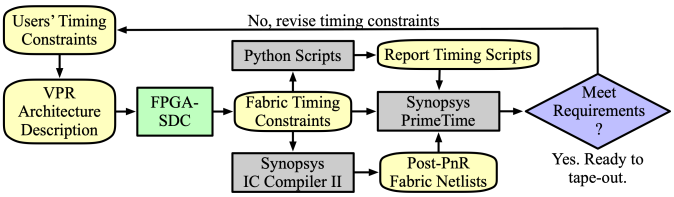


Fig. 6: An example of how iterative timing constrained backend flow can be enabled by FPGA-SDC.

B. Testbench Generation

As shown in Fig. 2, FPGA-Verilog can auto-generate two types of Verilog testbenches to validate the correctness of an implemented fabric: full and pre-configured. Users can customize clock frequencies and number of clock cycles to be used in the testbenches through an XML-based simulation setting file (see Fig. 2). The two testbenches share the same organization with self-testing features Full testbench is designed to validate both the configuration circuits and programming fabric of an FPGA, using a limited number of test vectors. The pre-configured testbench skips the time-consuming configuration phase and focus on applying high-coverage test vectors. As illustrated in Fig. 5, the preconfigured testbench is based on the preconfigured FPGA module, where an FPGA fabric is instantiated with a preloaded user's bitstream. Note that the preconfigured module is encapsulated with the same port mapping as the user's RTL design. Therefore, users can either feed the module to a formal tool for a 100% coverage formal verification, or use their existing testbenches for a post-OpenFPGA high coverage testing. The two testbenches offer different trade-offs between runtime and test coverage, as detailed in Table III. We believe that with a proper use of the two testbenches, the verification process for FPGAs can be significantly simplified or even automated. We refer interested readers to [14] for more details.

IV. OpenFPGA FPGA-SDC

As explained in Fig. 2, FPGA-SDC aims to generate timing constraints in a standard SDC format, which can be used by both backend tools and STA tools. Users can define the timing constraints in the VPR architecture, covering all the

pin-to-pin timing of the programmable logic and configuration circuits. Using the timing constraints, backend tools can force homogeneous delays across the fabric. By exploiting backend and STA tools, users' timing constraints can be automatically checked (see Fig. 6), enabling iterative improvement on the timing convergence of FPGA fabrics. More details are available in the OpenFPGA online documentation [2].

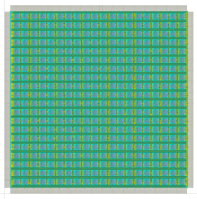
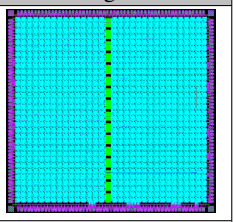
V. OpenFPGA FPGA-Bitstream

FPGA-Bitstream can generate two types of bitstreams: (1) a generic bitstream where configuration bits are organized out-of-order in a database. FPGA-Bitstream read and output the bitstream database in XML format, enabling the creation of synthetic bitstream. Note that the generic bitstream is designed to be an interchangeable database similar to FASM [15], but cannot be directly loaded to the FPGA fabric. (2) a fabric-dependent bitstream, where the generic bitstream is organized in the sequence being loadable to the configuration circuits of FPGAs. FPGA-Bitstream is a general-purpose bitstream generator, natively supporting any FPGA architecture that the XML description can model. As such, OpenFPGA can offer instant EDA support for chip designers once the XML-based architecture description is finalized. More details are available in OpenFPGA online documentation [3].

VI. OpenFPGA Showcase

OpenFPGA has been practiced to generate the full-chip layouts within 24 hours. Table IV showcases two examples using different technology nodes but being similar to the Stratix-IV architecture. We refer interested readers to [14] for more details.

TABLE IV: OpenFPGA layout generation showcase

Item	Homogeneous	Heterogeneous
Layout view		
Tech. node	40nm	12nm
LUT	4k	9.92k
Backend	Cadence Innovus 19.1	Synopsys ICC2 2019.03
Area	$7mm^2$	$9mm^2$
Runtime	24 hr	12 hr

VII. Summary and Future Work

In this paper, we introduced OpenFPGA, an open-source framework that can prototype a customizable full FPGA fabric through XML-to-GDSII design flow. OpenFPGA also provides a Verilog-to-Bitstream design flow as the associated CAD tools supporting any FPGAs that the architecture description language can model. We showcased two FPGA fabrics whose layouts are generated in $<$24 hours based on Stratix-like architecture and built with a commercial 12nm standard-cell library and 40nm custom cells respectively. In future, research efforts will be spent in achieving the same complexity of commercial state-of-art, such as supporting million-of-LUT device, more versatile architecture and configuration circuitry. OpenFPGA will also embrace more open-source tooling, such as OpenROAD [16], cocotb [17], SymbiFlow [18], SkyWater Open Source PDK [19], to build a vibrant community.

[2]https://openfpga.readthedocs.io/en/master/manual/openfpga_shell/openfpga_commands/fpga_sdc_commands.html

[3]https://openfpga.readthedocs.io/en/master/manual/fpga_bitstream/index.html

Acknowledgments

This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-2-7855. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.

References

[1] Chen Zhang *et al.*, *Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks*, ACM/SIGDA International Symposium on FPGA, 2015, pp. 161-170. 35, No. 1, pp. 16-22, Feb. 2018.

[2] I. Kuon *et al.*, *Design, Layout and Verification of an FPGA Using Automated Tools*, ACM/SIGDA International Symposium on FPGA, 2005, pp. 215-226.

[3] V. Aken'Ova *et al.*, *A soft++ eFPGA Physical Design Approach with Case Studies in 180nm and 90nm*, IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06), 2006, pp. 1-6.

[4] J. Kim *et al.*, *Synthesizable Standard Cell FPGA Fabrics Targetable by the Verilog-to-Routing (VTR) CAD Flow*, ACM Transactions on Reconfigurable Technology and Systems, Vol. 10, No. 2, April 2017.

[5] B. Grady *et al.*, *Synthesizable Heterogeneous FPGA Fabrics*, IEEE International Conference on FPT, 2018, pp. 1-8.

[6] H. Liu, *Archipelago - An Open Source FPGA with Toolflow Support*, Master Thesis, University of California, Berkeley, 2014.

[7] A. Li *et al.*, *PRGA: An Open-source Framework for Building and Using Custom FPGAs*, workshop on Open Source Design Automation (OSDA), 2019.

[8] Kevin 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).

[9] yosys – Yosys Open SYnthesis Suite, https://github.com/YosysHQ/yosys

[10] X. Tang *et al.*, *OpenFPGA: An Opensource Framework Enabling Rapid Prototyping of Customizable FPGAs*, International Conference on FPL, 2019, pp. 367-374.

[11] FPGA Architecture Description, https://docs.verilogtorouting.org/en/latest/arch/

[12] OpenFPGA Architecture Description, https://openfpga.readthedocs.io/en/master/manual/arch_lang/index.html

[13] X. Tang *et al.*, *A Study on Switch Block Patterns for Tileable FPGA Routing Architectures*, IEEE International Conference on FPT, 2019, pp. 247-250.

[14] X. Tang *et al.*, *OpenFPGA: An Open-Source Framework for Agile Prototyping Customizable FPGAs*, in IEEE Micro, vol. 40, no. 4, pp. 41-48, 1 July-Aug. 2020.

[15] FPGA Assembly (FASM), https://github.com/SymbiFlow/fasm

[16] The OpenROAD project, https://theopenroadproject.org/

[17] cocotb: a coroutine based cosimulation library for writing VHDL and Verilog testbenches in Python, https://github.com/cocotb/cocotb

[18] SymbiFlow: Innovate by reaching for the open source FPGA tooling, https://symbiflow.github.io/

[19] SkyWater Open Source PDK, https://github.com/google/skywater-pdk