FABulous: an Open-Everything Framework for Embedded FPGAs

Bea Healy, Jing Yu, Nguyen Dao, King Lok Chung, Dirk Koch
Department of Computer Science, The University of Manchester, UK
{jing.yu, nguyen.dao, dirk.koch}@manchester.ac.uk, {tabitha.healy, king.chung@student.manchester.ac.uk}

***Abstract*—This paper presents the open-source embedded FPGA framework FABulous that is designed for ease-of-use and excellent quality of results in terms of logic density, performance, and power consumption. FABulous users can build a fabric from predefined tiles for logic, memory, arithmetic, and various IO. The tiles are stitched together in a Lego-like manner, as needed. Users can add custom tiles or customize the routing resources and most architectural details of the FPGA fabrics. Experienced designers can provide optimized cells for configuration storage or switching (e.g., pass transistor multiplexers, what we currently provide for TSMC 180nm and Skywater 130nm). Alternatively, FABulous can fall back to a standard cell design, which makes FABulous FPGAs easily portable across different technology nodes. Fabulous comes with unique features that are not available in other open-source FPGA frameworks like low power frame-wise reconfiguration and dynamic partial configuration. FABulous integrates several other open-source projects (Yosys, nextpnr, VPR, OpenLane, Verilator) to provide a full and open end-to-end user experience.**

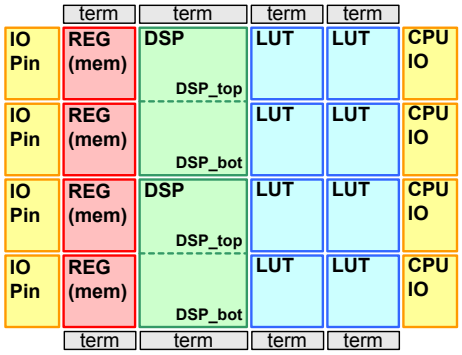
***Index Terms*—FPGA, reconfigurable computing, embedded FPGAs**

I. Introduction

FPGAs are ideal to update chips in the field for enabling hardware maintenance, customization, and even after-sales business. Moreover, FPGA technology is superior in translating an increment in logic density on a chip into corresponding performance. Furthermore, for programmable devices, FPGAs deliver better performance-per-Watt than what can be achieved by CPUs or GPUs.

However, the reconfigurability of FPGAs comes at a substantial cost for making logic functions and connections programmable, and especially in smaller designs, the cost for FPGA hardware programming should only be spent where it is most beneficial. This is the domain where embedded FPGAs (eFPGAs) came into play by integrating a reconfigurable fabric exactly tailored to the needs of the system, while having hardened parts that do not require reconfiguration (e.g., components like memory controllers or a CPU). To support this in an entire open ecosystem, this paper is presenting the FABulous eFPGA framework that provides:

- An easy way to **specify an eFPGA** fabric.
- **An emulation path** for testing entire systems (SoC + eFPGA fabric) on commercial FPGA boards.
- **A full ASIC backend flow** to implement eFPGAs. This includes both commercial EDA tools (Synopsys Design Compiler and Cadence Innovus) and open-source EDA tools (Yosys and OpenLane).



	A	B	C	D	E	F
1	*FabricBegin*					
2	NULL	N_term	N_term	N_term	N_term	NULL
3	W_IO	RegFile	DSP_top	LUT4AB	LUT4AB	CPU_IO
4	W_IO	RegFile	DSP_bot	LUT4AB	LUT4AB	CPU_IO
5	W_IO	RegFile	DSP_top	LUT4AB	LUT4AB	CPU_IO
6	W_IO	RegFile	DSP_bot	LUT4AB	LUT4AB	CPU_IO
7	NULL	S_term	S_term	S_term	S_term	NULL
8	*FabricEnd*					

Fig. 1: Top: example eFPGA. Bottom: spreadsheet model.

- **An integration with open-source FPGA CAD tools**, including Yosys [1] / nextpnr [2] and VPR [3].

Embedding an FPGA into an ASIC is difficult and imposes a risk to meet all cost, capacity and performance objectives. The goal of FABulous is to minimize this risk by automating processes as much as possible and by absorbing most low-level details by our framework. However, FABulous allows to customize the FPGA fabrics at any level, as needed.

Figure 1 provides a motivating example. It shows a small fabric and the corresponding spreadsheet specification from which FABulous can generate an eFPGA fabric. This view targets users that have some background in FPGAs and that may have used devices from major FPGA vendors (e.g., Xilinx, Intel, or Lattice) before. In this mode, we provide predefined tiles that are currently a mixture of a Spartan-3-like routing fabric and Lattice iCE40 logic tiles. We decided for this because the Spartan-3 family provides a sophisticated routing fabric but has all patents expired and iCE40 is a popular architecture supported by the major open-source CAD tools (Yosys/nextpnr and VPR). From the original Spartan-3 routing fabric we removed long lines, multiple clock domains, and we used at some points a more hierarchical FPGA architecture graph to save on area.

Initially, FABulous had been presented in [4] and [5]. This

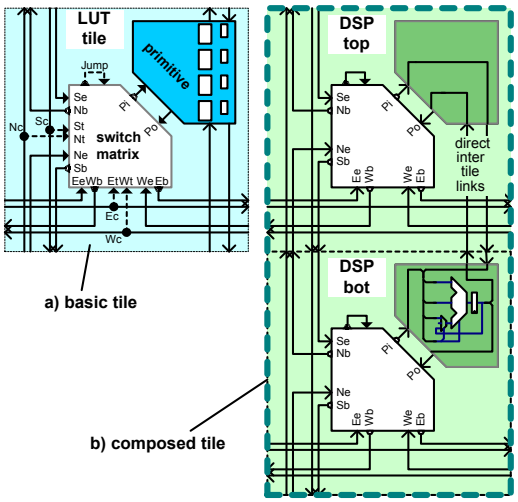


Fig. 2: a) Basic tile consisting of a switch matrix, wires to neighboring tiles, and one or more connected primitives. b) Composed tile taking the size of two basic tiles.

paper is focusing on the open-source integration and usage of FABulous. Novel contributions include a better integration into VPR, an improved FPGA fabric architecture, better support for tiles of different size, support for the open Skywater 130nm PDK, support for the OpenLane ASIC tools, and a Docker container with the framework for ease-of-use.

II. Specifying an eFPGA in FABulous

The specification in Figure 1 uses descriptors to tiles. While FABulous could use this to generate the required ASIC RTL code, a simulation model, and the model files for Yosys, nextpnr, and VPR, this section will provide further details on how to specify custom fabrics.

A. Tile-based Fabric Definitions

Figure 2a) shows a FABulous tile with its central switch matrix, wires, and one or more primitives that are connected to the switch matrix. The primitives could be just look-up tables (LUTs) or somehow more complex blocks like the slices in Xilinx (where multiple LUTs share some clock and multiplexing infrastructure). There are no restrictions to the logic tiles and any simple LUT model (like Lattice FPGAs), composable LUTs (like Xilinx Spartan-3/Virtex-II), and fracturable LUTs (like all modern Xilinx and Intel FPGAs) are supported. To this end, FABulous relies heavily on the abilities of Yosys to make use of the different tile types. Luckily, Yosys is well maintained and steadily growing and it is becoming the GCC frontend equivalent for open-source (FPGA) CAD tool flows.

VPR models are strongly influenced by Altrera FPGA fabrics where a tile uses different cascaded switch matrices. In these fabrics, a first level switch matrix is picking which signals from the routing channels will go into a switch matrix; and from there, a signal may eventually be routed through another second level switch matrix to another tile or a primitive input. Xilinx FPGAs traditionally used a direct approach where the channel wires are directly connected to the switch matrix

121	*TILE*	LUT4AB					
122	#direction	source	X-offset	Y-offset	destination	wires	
123	NORTH	N1BEG	0	1	N1END	8	
124	NORTH	N2BEG	0	2	N2END	4	
125	NORTH	Co	0	1	Ci	1	# carry
126	EAST	E1BEG	1	0	E1END	8	
127	EAST	E4BEG	4	0	E4END	2	
128	SOUTH	S1BEG	0	-1	S1END	8	
129	SOUTH	S2BEG	0	-2	S2END	4	
130	WEST	W1BEG	-1	0	W1END	8	
131	WEST	W4BEG	-4	0	W4END	2	
132	JUMP	J_BEG	0	0	J_END	42	
133	BEL	LUT4.vhdl	LA_				
134	BEL	LUT4.vhdl	LB_				
135	BEL	LUT4.vhdl	LC_				
136	BEL	LUT4.vhdl	LD_				
137	BEL	MUX8LUT.vhdl					
138	MATRIX	LUT4AB_switch_matrix.vhdl					
139	*EndTILE*						

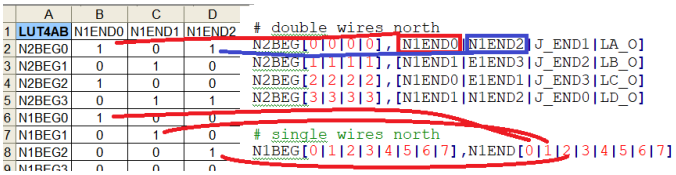
Fig. 3: Example of a tile description.

and where it is possible to reach a primitive input through a single switch matrix multiplexer level.

For FABulous, we decided for a simple, but yet flexible way to model virtually any routing and switching fabric. We connect all wires that end at a specific tile to the corresponding central switch matrix. If a user wants to build an Altera-style hierarchical routing fabric, this is modeled with the help of jump wires (i.e. wires that begin and end at the same switch matrix). Users can specify arbitrarily jump wires for modeling any hierarchy. Jump wires are also needed to model latest UltraScale FPGA fabrics from Xilinx. In original Xilinx FPGAs, multiplexers had been implemented in multiple levels of passtransisitor multiplexers (two levels in Virtex II and three levels in Spartan-3 FPGAs [6]). While UltraScale devices share many similarities with classic Xilinx FPGA fabrics, the two-level monolithic multiplexers are now exposed to the users as two separate levels of multiplexers in UltraScale fabrics.

It is common practice to organize FPGAs in columns of identical resource type (see Figure 1). All tiles will share the same physical height, but the width is adjusted to the individual resource requirements (e.g., a DSP column maybe wider than a LUT logic column). This scheme results in a tight packing even if different resource types are used.

Some tiles will require more resources than what would fit sensibly into a basic tile and/or may need more wire connections to the routing fabric. For this, we can combine multiple adjacent tiles to a composed tile as shown for a DSP example in Figure 2b). In a composed tile, we typically specify one tile to contain all the primitives while using the other tiles to provide extra connectivity to the otherwise homogeneous routing fabric. This means that the tile-to-tile routing fabric is commonly the same regardless of the the particular resource type (except for some exceptions like carry chains etc.). This is fundamentally the same scheme as used in Xilinx FPGAs. Note that this is a logical view, and the physical implementation will use the area of all the basic tiles to implement the composed tile. The functionality (like the DSP module in our example) is simply provided as RTL code (Verilog or VHDL) and connected to a switch matrix (a DSP ALU connected to the bottom switch matrix in the example).



	A	B	C	D
1	LUT4AB	N1END0	N1END1	N1END2
2	N2BEG0	1	0	1
3	N2BEG1	0	1	0
4	N2BEG2	1	0	0
5	N2BEG3	0	1	1
6	N1BEG0	1	0	0
7	N1BEG1	0	1	0
8	N1BEG2	0	0	1
9	N1BEG3	0	0	0

```
# double wires north
N2BEG[0|0|0|0],[N1END0|N1END2|J_END1|LA_O]
N2BEG[1|1|1|1],[N1END1|E1END3|J_END2|LB_O]
N2BEG[2|2|2|2],[N1END0|E1END1|J_END3|LC_O]
N2BEG[3|3|3|3],[N1END1|N1END2|J_END0|LD_O]

# single wires north
N1BEG[0|1|2|3|4|5|6|7],N1END[0|1|2|3|4|5|6|7]
```

Fig. 4: Switch matrix adjacency can arbitrarily be defined in matrix form (left) or in list form (right).

B. Modeling Routing Fabrics in FABulous

While FPGA users look at their FPGA in terms of the resources available, it is the routing fabric that is mostly defining the area, speed, and power characteristics of an FPGA. The multiplexers with their configuration storage take about 80-90% of the die area and the routing wires contribute most to the overall congestion (use of metalization layers) [7].

Figure 3 provides an illustrative example of the tile description of a configurable logic block (CLB). The description defines three sections: 1) the wires (given by their direction, names for begin and end ports, and the offset to which the wire routes), 2) the primitives or basic elements – BELs (that will be connected to the switch matrix, and 3) an optional switch matrix. We can adjust the direction, amount, and length of wires by editing the wire entries of the tile descriptions. For instance, smaller fabrics do normally not require long distance wires as those are more suited for inter module routing. Branching and diagonal wires can be modeled by concatenating multiple wires segments together.

While users can customize the routing fabric, the wires on horizontal direction should match across the different tile types and all tile types must have a corresponding number of wires that directly match for stitching together a fabric.

All switching is done in the central switch matrix. The adjacency is defined either by a matrix description or a list (see Figure 4), and FABulous can translate arbitrarily between both. When custom switch matrix multiplexers are available, FABulous will use them in the physical fabric implementation. In our present examples, using custom multiplexers can reduce implementation cost by 30%. Here it is not required to provide all used multiplexer variants. For instance in our test chips, we provided a custom 4:1 multiplexer, and FABulous uses that cell to build other (larger) multiplexers.

Defining a good routing fabric in terms of wires and adjacency that allows routing of highly utilized fabrics without overprovisioning routing resources is hard and normally needs expert knowledge. While FABulous would allow exploring this design space, our recommended approach is to use a default fabric and then remove features that are not needed. This is straightforward to use: we can start from a full fledged FPGA fabric, implement our target or some benchmark circuits on that fabric and check the statistics if some resources are underutilized to be removed. This does not require the physical FPGA fabric but just yosys/nextpnr or VPR. We then remove those resources and reimplement the circuits to confirm that they can be implemented on the down-stripped fabric.

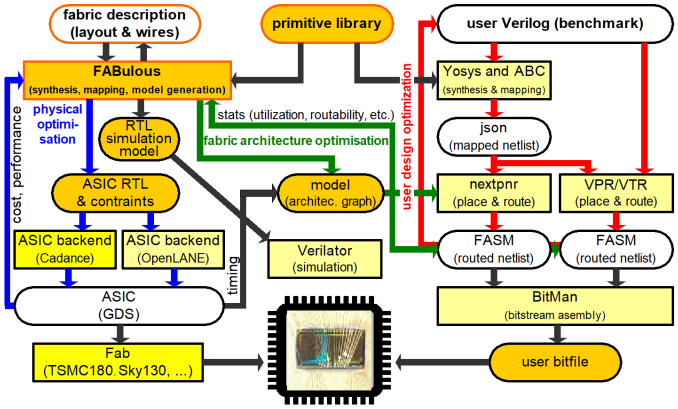


Fig. 5: The FABulous Framework.

III. The FABulous Framework

Figure 5 shows the entire FABulous framework. It integrates tools for the ASIC syntheses flow (left), the eFPGA CAD flow for implementing Verilog to bitstream (right), and paths for simulation (using Verilator [8] and emulation (using Xilinx vendor tools). The FPGA CAD flow is based on SymbiFlow [9], which aims at providing a GCC equivalent for FPGA compilation. A distinct feature of FABulous is that it can generate models for both Yosys/nextpnr and VPR. For any task there is at least one open-source option available. More details on the framework and the tool interplay is in [4].

IV. Chip Gallery

Figure 6 shows chips that had been built with FABulous:

chip	process	area[1]	LUTs	DSPs	REG	Mem
a) caravel_sky	Sky 130nm	10mm^2	864	6	12	6x1Kb[2]
b) RISCV_sky[3]	Sky 130nm	8.7mm^2	320	10	–	–
c) STRIVE_sky	Sky 130nm	15.5mm^2	1440	–	45	–
d) RISCV_TSMC[4]	TSMC 180	3.4mm^2	384	4	8	–

[1] core area of the eFPGA.
[2] dual-ported with conf. read and write aspect ratios, built by OpenRAM [10].
[3] Dual RISC V system using a shared eFPGA through custom instructions.
[4] RISC V with eFPGA for custom instruction extension. Up to three partially reconf. instructions; can be used standalone (via dedicated I/O tiles) [5].

The chip gallery shows different ways to integrate IP into the FPGA fabric and expresses the versatility of FABulous: 1) by embedding tiles into the fabric (e.g., custom DSPs), 2) by attaching the fabric to external IP, such as a RISC-V core or my attaching external IP to the fabric (the BRAMs in a), which appear logically inside the fabric, but that are actually located at the border of the fabric).

V. Related Work

Commercial eFPGA vendors (e.g., QuickLogic, Flex Logix, Menta, Achronix, NanoExplore, and Efinix [11]–[16]) differentiate in target applications, technology, and usability. For instance, Menta is portable to any process as it is entirely based on a standard cell design and flip-flops for configuration storage. Fabulous is similar to that, but will deliver better quality of results when using custom multiplexers and cheaper latches for configuration storage. Industry solutions often lag in customizing primitives, the routing, interfaces to the outside world, and the exact size of the fabric. While standardized

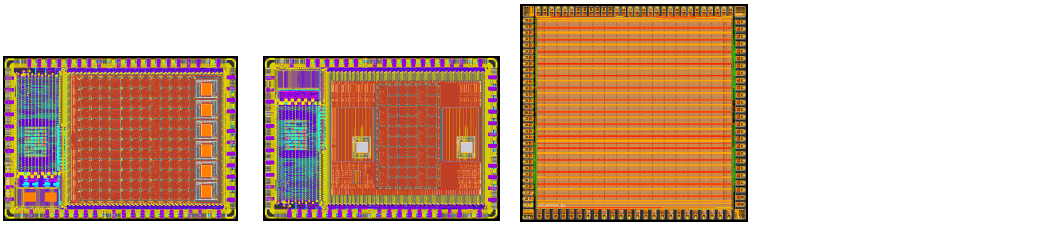


Fig. 6: Fabulous chip gallery.

eFPGAs are a way to reduce complexity and risk, FABulous is not omitting domain-specific customizations.

The open-source FPGA approach in [17], used VTR [18] for building custom FPGAs targeting standard cell technology. This work was extended in [19] with support for heterogeneous FPGA fabrics that support BRAMs and DSPs. FABulous produces better quality results by using frame-based reconfiguration for large fabrics. However, the DSP blocks in [19] are more sophisticated than what we are currently providing in FABulous but we could use those by providing a different model and without changes to FABulous itself. Other related flows include [20]–[23]. Compared to all these approaches FABulous is the only framework integrating both Yosys/nextpnr and VPR as well as supporting partial reconfiguration (PR). Some related approaches claim to support PR, but that is not working with shift register configuration.

Like FABulous, Open-FPGA [24] is one of the few approaches considering both the fabric generation and the FPGA CAD tool-chain for the eFPGA bitstream generation. OpenFPGA supports scan-chain reconfiguration [24] only, while FABulous supports scan-chain and frame-based reconfiguration (and therefore PR). Most importantly, frame-based reconfiguration allows storing configuration data in cheap SRAM cells or latches, while OpenFPGA requires more expensive D-flip-flops and will require more power for configuration. Therefore, FABulous will deliver a better quality of results at the same optimization effort. For direct comparison, OpenFPGA and FABulous have a MPW Skywater 130 tapeout and the CLBs compare as follows:

	area	resources
OpenFPGA	$217x250 = 54250um^2$	1300 MUX2, 530 DFF
FABulous	$210x220 = 46200um^2$	376MUX4/46MUX2/8FF/586Latch

Therefore, FABulous needs only 85% for a CLB of roughly similar complexity (1xMUX4 is about 3xMUX2).

VI. Conclusion

This paper provides a brief intro into the capabilities and usage of the FABulous eFPGA framework. FABulous is designed to serve a wide range of users, from system integrators (by using default templates) to FPGA architects (who have detailed control over the entire flow). Tapeouts have demonstated the applicability of FABulous, and the quality of results can well compete with commercial and academic eFPGA offerings.

FABulous is currently the most versatile framework with respect to modeling capabilities and the flows supported. The framework is integrating many other high-quality open-source projects, including Yosys [1], nextpnr [2], VPR [3], OpenRAM [10], and the Verilator [8]. To empower the open-source community to make widespread use of reconfigurable computing, we are actively maintaining this project. The project is released under Apache 2.0 license and all sources and a docker container is available under:
`https://github.com/FPGA-Research-Manchester/`

Acknowledgment

This work is kindly supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grant EP/R024642/1 under project FORTE (http://www.forte.ac.uk/).

References

[1] C. Wolf, "Yosys Open SYnthesis Suite," http://www.clifford.at/yosys/.
[2] D. Shah and et al., "Yosys+nextpnr: An Open Source Framework from Verilog to Bitstream for Commercial FPGAs," in *27th FCCM*, 2019.
[3] V. Betz and J. Rose, "VPR: a New Packing, Placement and Routing Tool for FPGA Research," in *FPL*, 1997, pp. 213–222.
[4] D. Koch, N. Dao, B. Healy, J. Yu, and A. Attwood, "FABulous: An Embedded FPGA Framework," ser. ACM FPGA, 2021, p. 45–56.
[5] N. Dao, A. Attwood, B. Healy, and D. Koch, "FlexBex: A RISC-V with a Reconfigurable Instruction Extension," in *FPT*, 2020.
[6] C. Beckhoff, D. Koch, and J. Torresen, "Short-Circuits on FPGAs Caused by Partial Runtime Reconfiguration," in *20th FPL*, 2010.
[7] V. Betz, J. Rose, and A. Marquardt, Eds., *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.
[8] Wilson Snyder, Verilator. Online: https://www.veripool.org/verilator/.
[9] C. Wolf, et al., "SymbiFlow Open source flow for generating bitstreams from Verilog," https://github.com/SymbiFlow/.
[10] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "OpenRAM: An Open-Source Memory Compiler," in *ICCAD*, 2016.
[11] Online: https://www.quicklogic.com/products/efpga/efpga-technology/.
[12] Flex Logix eFPGAs. Online: https://flex-logix.com/efpga/.
[13] Menta eFPGA Website. Online: https://www.menta-efpga.com/.
[14] Achronix. Online: https://www.achronix.com/product/speedcore.
[15] NanoExplore Website. Online: https://www.nanoxplore.com/.
[16] Efinix Inc. . Online: https://www.efinixinc.com/products-trion.html.
[17] Jin Hee Kim and J. H. Anderson, "Synthesizable FPGA Fabrics Targetable by the Verilog-to-Routing (VTR) CAD flow," in *25th FPL*, 2015.
[18] J. Luu and et al., "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM TRETS*, vol. 7, no. 2, Jul. 2014.
[19] B. Grady, et al., "Synthesizable Heterogeneous FPGA Fabrics," FPT'18.
[20] P. Mohan and et al., "A Top-Down Design Methodology for Synthesizing FPGA Fabrics Using Standard ASIC Flow," in *28th FPGA*, 2020.
[21] H. J. Liu, "Archipelago - An Open Source FPGA with Toolflow Support," Master's thesis, University of California at Berkeley, 2014.
[22] Princeton University, "Princeton Reconfigurable Gate Array," 2019, https://prga.readthedocs.io/en/latest/.
[23] A. Li and D. Wentzlaff, "PRGA: An Open-source Framework for Building and Using Custom FPGAs," in *1st OSDA*, 2019.
[24] X. Tang and et al., "OpenFPGA: An Open-Source Framework for Agile Prototyping Customizable FPGAs," *IEEE Micro*, vol. 40, no. 4, 2020.