FABulous: An Embedded FPGA Framework

Dirk Koch, Nguyen Dao, Bea Healy, Jing Yu
[dirk.koch,nguyen.dao,bea.healy,jing.yu]@manchester.ac.uk
Computer Science, The University of Manchester, UK

Andrew Attwood
a.j.attwood@ljmu.ac.uk
Liverpool John Moores University, UK

ABSTRACT

At the end of CMOS-scaling, the role of architecture design is increasingly gaining importance. Supporting this trend, customizable embedded FPGAs are an ingredient in ASIC architectures to provide the advantages of reconfigurable hardware exactly where and how it is most beneficial. To enable this, we are introducing the FABulous embedded open-source FPGA framework. FABulous is designed to fulfill the objectives of ease of use, maximum portability to different process nodes, good control for customization, and delivering good area, power, and performance characteristics of the generated FPGA fabrics. The framework provides templates for logic, arithmetic, memory, and I/O blocks that can be easily stitched together, whilst enabling users to add their own fully customized blocks and primitives. The FABulous ecosystem generates the embedded FPGA fabric for chip fabrication, integrates Yosys, ABC, VPR and nextpnr as FPGA CAD tools, deals with the bitstream generation and after fabrication tests. Additionally, we provide an emulation path for system development. FABulous was demonstrated for an ASIC integrating a RISC-V core with an embedded FPGA fabric for custom instruction set extensions using a TSMC 180 nm process and an open-source 45nm process node.

CCS CONCEPTS

• **Hardware** → *Programmable logic elements*;

KEYWORDS

FPGA, reconfigurable computing, FPGA virtualisation, partial reconfiguration

ACM Reference Format:
Dirk Koch, Nguyen Dao, Bea Healy, Jing Yu and Andrew Attwood. 2021. FABulous: An Embedded FPGA Framework. In *2021 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '21), February 28–March 2, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3431920.3439302

1 INTRODUCTION

Integrating an FPGA fabric into an ASIC provides excellent opportunities for designing systems. It allows taking the full area, performance and cost advantages of the hardened part while providing the flexibility and adaptability of an FPGA fabric exactly *where* and *how* it is needed. However, embedding an FPGA into an ASIC requires taking several difficult design choices which impose a high risk to meet all cost, capacity and performance objectives for the embedded FPGA (abbreviated eFPGA in the following). These design choices add to the overall system complexity, and adding an eFPGA into an ASIC design is therefore imposing a risk to any project.

This is the motivation of the FABulous eFPGA fabric generator that aims at balancing ease of use with the opportunity to customize the embedded FPGA. FABulous is open-source, and it uses several open-source tools to integrate a full eFPGA framework providing the eFPGA ASIC netlist and constraints generation, the eFPGA CAD tools (using Yosys [61], ABC [9], VPR [10], nextpnr [53]), and BitMan [47], and an emulation path to emulate the eFPGA on an FPGA board.

FABulous comes with templates that allow users to generate an eFPGA at reasonable quality by defining the fabric dimensions in terms of logic tiles (e.g., a configurable logic block-CLB [67] in Xilinx terminology and logic array block-LAB [30] in Intel terminology) and the layout of resource columns, which is the sequence of I/O, logic, memory, or arithmetic block columns. While still well supported, this removes the need for excessive design space exploration and the corresponding expertise.

One reason for this approach is that several popular FPGA architectures are becoming 20 years old, which means that original patents have expired. Therefore, most of the architectural features of those FPGAs are now freely usable. Examples of this include

Table 1: Examples of classic FPGA Families.

Vendor, Family	launch year	process
Xilinx, Virtex	1998[65]	220 nm [65]
Xilinx, Virtex-II	2000[23]	150 nm [65]$^{1)}$
Xilinx, Spartan-3$^{2)}$	2003 [29]	90 nm [65]
Altera, APEX II	2001 [18]	150 nm [18]
Altera, Stratix	2002 [5]	130 nm [5]
Altera, Cyclone	2002 [5]	130 nm [5]

$^{1)}$ The Virtex-II Pro series was manufactured in 130 nm [65] and used a fabric with the same FPGA architecture for logic, memory, DSP columns, and routing than Virtex-II FPGAs. Bitstreams for these resources are binary compatible between Xilinx Virtex-II and Virtex II Pro FPGAs.
$^{2)}$ The Xilinx Spartan-3 architecture is very similar to Virtex-II FPGAs.

Spartan-3 and Virtex-II FPGAs from Xilinx and the first generation of Cyclon and Stratix FPGAs from Altera (now Intel) (see also Table 1). While there had been improvements with newer FPGAs generations, these classics are already very sophisticated and well-optimized and therefore very suitable to serve us as a template for eFPGAs. As shown later in this paper, embedded FPGAs are constrained concerning the metalization layers used, and those classics have characteristics that make them well suited for eFPGAs in much smaller process nodes.

To take advantage of this for eFPGAs, FABulous is designed to model virtually any FPGA fabric through simple but very generic means, and we have implemented simplified clones of Xilinx Spartan-3/Virtex-II and Lattice iCE40 UltraPlus FPGAs. We used those models as starting points, and variations of those models will evolve into future templates.

Like in a CPU, where commonly only a small subset of the instructions available is used [68], in an FPGA, many architectural features may not be used for a specific application domain. By simply removing unused FPGA features, we have a rather trivial way to tailor an FPGA fabric to a specific application domain. For instance, FPGA routing is dominating the cost of an FPGA fabric [16, 38, 59], and small eFPGAs may not use some or all long-distance wires, which could consequently be removed. This is consecutively saving logic resources to implement the switch matrices including the corresponding area for configuration bits. With this approach, FABulous aims at area, performance, and power values that can get close to highly optimized general-purpose FPGAs, if the eFPGAs are manufactured in the same process node.

In the following section, we will motivate our approach with related work. Section 3 discusses our design choices with respect to existing work and Section 4 introduces the FABulous framework and its design flow. Section 5 provides a case study of an eFPGA integrated into an RISC-V SoC that is currently manufactured in a 180nm process at TSMC and Section 6 provides further evaluation.

2 RELATED WORK

Building a flexible eFPGA framework relies on A) versatile hardware generators and B) corresponding tools that can harness the hardware features.

2.1 Embedded FPGAs

The most important domains for eFPGAs are telecommunication, industrial applications, military, and automotive [4] and there are several vendors of eFPGA IPs. In addition, there are open-source versions of eFPGAs.

2.1.1 ***Commercial eFPGA Vendors.*** Commercial eFPGA vendors include QuickLogic, Flex Logix, Menta, Achronix, NanoExplore and Efinix [12, 13, 19, 28, 44, 52]. These vendors differentiate in technology, target applications, and usability. For instance, Efinix targets specifically Samsung processes [27], NanoExplore provides radiation-hardened fabrics for aerospace and military applications, and Menta is easily portable to any process as it is entirely based on a standard cell design.

Industry solutions often lag in the level of customization available by predefining fabric primitives, the routing, interfaces to the outside world, and the exact size of the fabric. While standardized eFPGAs are a way to reduce complexity and risk, we should not omit domain-specific customizations. For instance, bit-level operations for cryptographic cores typically require LUTs and more flexibility in the routing, while signal processing applications may benefit from multiplier blocks and word-level routing. Furthermore, systems may define a preferred datapath direction, which could be used to optimize the wires available in the fabric.

2.1.2 ***Open-source and Academic eFPGA Solutions.*** In Section2.2, we will introduce academic open-source CAD tool-chains that had been the base for consecutive open-source (e)FPGA projects. For instance, the VTR flow [42] was used in [32, 33] for building custom FPGAs targeting standard cell technology. This work was extended in [21] 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 and memory blocks in [21] are more sophisticated than what we currently tested in FABulous; and because the frontend tools are the same, those blocks should be directly usable in our framework. VTR was also used in [43] for a top-down FPGA synthesis flow around the Chisel language. Similar work was shown in a Master thesis project at Berkeley with a small homogeneous tile-based FPGA [41]. VTR uses Yosys [61] and VPR [10], and this is the tool-chain enabling the PRGA custom FPGA builder project. [40, 57]. PRGA is designed for ease of use and is hiding several low-level aspects. This goal is shared with FABulous, but in contrast, our tool allows designers to access low-level details in an easy way to customize and optimize fabrics as needed. With this, FABulous allows trading optimization effort for quality of results, while still providing reasonable quality in default mode.

Some early related academic projects include the GILES tools [36], which also included FPGA fabric generation and architecture graph generation for the VPR tool. Like FABulous, this approach supports frame-based reconfiguration, but without support for partial reconfiguration. Furthermore, only a homogeneous fabric was implemented without special tiles like DSPs or memory blocks.

Another closely related academic approach is the Open-FPGA project [55]. As provided in FABulous, Open-FPGA is one of the very few approaches that considers both the fabric generation and the FPGA CAD tool-chain, all the way to the bitstream for the eFPGA fabric. Both approaches could probably model most commercial FPGA architectures. However, OpenFPGA models routing hierarchically through separate switch blocks and connection blocks, which is closer to Altera/Intel class of FPGAs. FABulous uses a central switch matrix, which is closer to Xilinx FPGAs. However, FABulous supports stop-over routing, which allows modeling any level of routing hierarchy and consequently modeling Altera devices (see also Section 3.2.1). A further differentiator is that OpenFPGA can only support scan-chain reconfiguration [55], while FABulous supports scan-chain and frame-based reconfiguration (together with partial reconfiguration), as described in Section 3.4. Most importantly, frame-based reconfiguration allows storing configuration data in cheap SRAM cells or latches, while OpenFPGA requires more expensive D-flip-flops. Therefore, FABulous will deliver a better quality of results at the same optimization effort, as shown in Section 6.

The Soft++ eFPGA work [2] is another related project, which proposes tiled floorplanning and the use of "tactical cells" to replace critical standard-cell elements (e.g., high fan-in multiplexers). Fabulous also provides tiled (hierarchical) floorplanning (with an alternative option for flat physical implementations), and it can utilize "tactical cells" given those are provided in the target process technology (e.g., FABulous currently provides pass transistor multiplexers for 180 nm TSMD and 45 nm open-source [45] technology). However, Soft++ is not publicly shared, does not allow user customization, and it considers homogeneous island-style eFPGA fabrics only.

2.2 Open-source FPGA CAD Tools

Research in open-source (e)FPGAs relies on the active development of corresponding CAD tools. The most prominent example is the Verilog-To-Routing framework VTR [42], which uses Odin II [31] for logic synthesis, ABC [9] for technology mapping, and VPR [10] for routing. The recent SymbiFlow framework [62] includes Yosys [61] for logic synthesis, VTR for routing, and Project X-Ray [54] for bitstream assembly. SymbiFlow had been demonstrated for Xilinx 7-series FPGAs. Nextpnr[53] is an entirely new independent place and route tool that uses Yosys and ABC as a frontend. The first target of that tool-chain had been Lattice iCE40 UltraPlus FPGAs Furthermore, nextpnr was used for a self-compilation flow running on Xilinx UltraScale+ FPGAs, where the embedded ARM CPU generated partial bitstreams for the FPGA fabric. [48]. nextpnr is the original CAD tool support used for our FABulous framework.

Both the VTR and the nextpnr tool flows are versatile and can be used for various FPGA targets. For instance, in [25] VTR was used to implement modules on Xilinx Virtex-6 FPGAs. The entire tool flow was based on open-source tools, and the Xilinx vendor tools had only been used for the final bitstream assembly. Both flows (VTR and nextpnr) support different LUT sizes, cascading of LUTs for building larger LUTs, fracturable LUTs, different ways to implement carry chains, and both can deal with memory blocks and DSP tiles. Additionally, VTR supports multiple clock domains, including clock buffer insertion.

The tool ecosystem has reached a high level of maturity and is capable of producing high-quality designs. Currently, FABulous itself does not provide any FPGA CAD tools and relies entirely on Yosys, ABC and VPR/nextpnr. An important aspect of these open-source tools is that they can be easily adjusted to accommodate a broad set of different architectural features. This may include the size of LUTs, the routing fabric, dedicated primitives (like memories and even multiplier and memory blocks) as well as carry chain logic. For instance, it is possible to use the multiply operator to infer DSP tiles directly. In addition, the tools allow to instantiate primitives directly. For example, someone can integrate an ALU primitive to be added to an FPGA fabric, and in the case were the synthesis tool (e.g., Yosys) is not able to infer that primitive directly from an arithmetic expression, it is still possible to instantiate that primitive directly and pass the settings and parameters to that primitive. Those parameters will be passed through to the final netlist, such that the bitstream can be generated for these settings without further user intervention. This is common practice and used in the same way with the FPGA tools of major Vendors (e.g., Quartus-II from Intel and Vivado from Xilinx). FABulous does not put a preference on VPR or nextpnr, which allows choosing the place and route tool based on use-case or personal preference.

3 DESIGN CHOICES

When designing an eFPGA, we have not only to consider the usual cost, power, and performance objectives, but further the implications on integrating, manufacturing, and testing of a larger system. This has led to several design decisions, as discussed in the following sections.

3.1 Fabric Layout

FABulous organizes its eFPGA fabrics into tiles, as shown in Figure 1. A tile can have a switch matrix and an arbitrary number of primitives. To ensure a tight packing of heterogeneous tiles (e.g. arithmetic tiles together with LUT logic tiles), all tiles should have the same height and a width that is related to the tile type. This is the common approach taken by all major FPGA vendors (e.g. Xilinx, Intel, and Lattice). While not a strict requirement, it is common practice to fill full columns with the same tile types.

For simplicity, a logic tile will be used as an atomic unit. For instance, when modeling Xilinx architectures in FABulous, a logic tile would be equivalent to a complex logic block (CLB). However, users can combine multiple (typically vertically aligned) tiles to form more complex blocks, like DSP blocks, AI accelerators or even CPUs, as shown for a DSP column in Figure 1. This is the approach used by all major FPGA vendors. The number of complex blocks can be adjusted by 1) the fabric height (in terms of tiles), 2) the height of specialized blocks (e.g. the number of tiles used for memories or DSPs), and 3) the number of columns dedicated to a specific tile type. As a fourth option, specialized blocks may arbitrarily be used for entire rows or replace some other tiles. However, this may add some heterogeneity to the fabric (i.e. specialized blocks in columns can have different characteristics than the same blocks in rows) and has some implications on the bitstream as different tile types use a different number of configuration bits (see also Section 3.4.

3.2 Tiles and Fabric Primitives

A tile description consists of three sections:

3.2.1 ***Wires.*** are modeled in north, east, south, west direction as directed point-to-point connections. Wires are specified by their direction, a begin and end name, the X and Y offsets of the target tile and wire count. For instance, the example in Figure 2 is modeled as:

```
EAST,  E1Beg,  E1End,  1,  0, 6
WEST,  W4Beg,  W4End, -4,  0, 3
```

Diagonal wires are modeled by composing two wire segments together. For example, 3 quad wires in north-east-direction can be modeled as:

```
NORTH, NE4Beg, NE4Mid,  0,  2, 3
EAST,  NE4Mid, NE4End,  2,  0, 3
```

In the diagonal wire example, it is possible to tap the `NE4Mid` ports as north double wires by using that port in the switch matrix located two positions north. When modeling wires spanning multiple tiles,

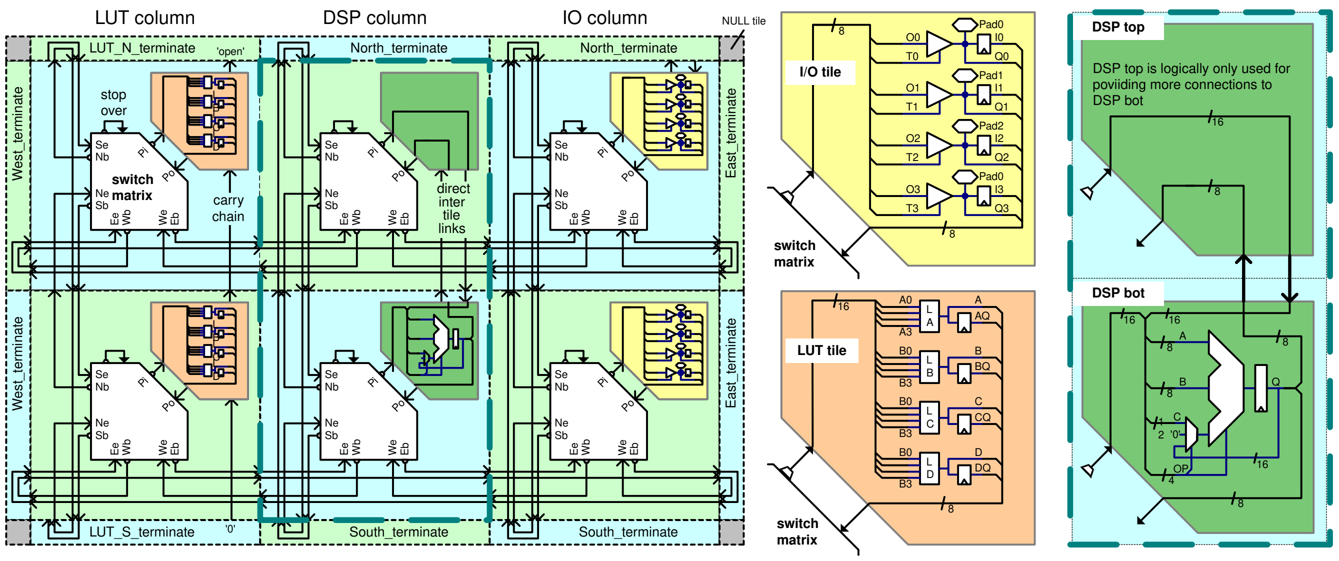

Figure 1: Tiled FABulous eFPGA fabric description. The fabric provides 2 LUTs, one DSP (spanning over two tiles) and two I/O tiles. Each tile defines a type which provides a reference to 1) a list of wires, 2) a switch matrix, and 3) a set of primitives. Termination tiles at the border reflect wires back into the fabric.

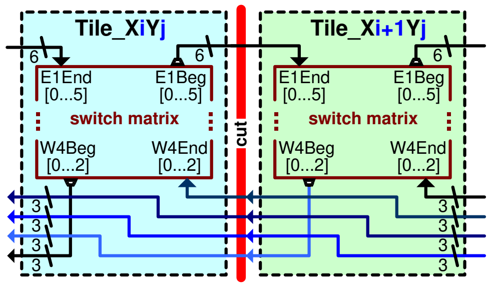

Figure 2: Example of two tiles with their switch matrices. The top wire example shows 6 single wires for routing in eastwards direction, and the bottom routing is for 3 quad-wires in westwards direction.

they are concatenated to one channel and shifted by the wire count, and only the entire wires that start or end at a tile are connected to the switch matrix using the specified names, as shown for three quad-wires in Figure 2. FABulous does not insert buffers on wires, and buffer insertion is left to the ASIC implementation.

An important parameter, when modeling wires of an eFPGA, is the number of physical wires that pass a cut between adjacent tiles. This value is a measure for the congestion that the eFPGA will contribute to in the physical ASIC implementation. In the example in Figure 2, there are $6 \times 1 + 3 \times 4 = 18$ wires passing the cut. As a reference, a Xilinx Spartan-3 FPGA provides 8 single, 8 double, and 8 hex wires in each north, east, south, west direction. This corresponds to $(2 \times (8 \times 1 + 8 \times 2 + 8 \times 6)) = 144$ horizontal wires crossing a horizontal cut between two adjacent CLBs.

eFPGA fabrics should not take more metal layers than the rest of the ASIC. For instance, the 28nm TSMC 28LP, 28HPL, and 28HP processes provide 10 metal layers [58] and are used for ARM Cortex-A53 implementations. For comparison, TSMC produces Altera Stratix V FPGAs using 12 metal layers and Xilinx Kintex-7 FPGAs using 11 metal layers in the same 28nm HP/HPL process [17]. Additionally, when integrating an eFPGA into a system, it should be considered that other parts of the ASIC may have to route signals across the eFPGA, which would put even more stress on the metal stack congestion. For this reason, eFPGA fabrics should not use all available metal layers. For instance, the EFLX 4K eFPGA from Flex Logix uses only 6 metal layers when implemented in the 28nm TSMC processes [20]. See also Section 4.0.2 on optimizing the fabric.

3.2.2 ***Primitives.*** are specified as VHDL modules, which are directly connected to the switch matrix. In the case of a logic tile, a primitive can be a slice featuring multiple LUTs with shared ports (as used by Xilinx) or simply LUTs or LUTs with flip-flops (as used by Lattice). A tile can have multiple primitives. FABulous requires a few simple code conventions as illustrated in the following code example of an I/O primitive:

```
entity IO_1_bidirectional is
Generic ( NoConfigBits : integer := 0 );
Port
(
  I : in STD_LOGIC; --from fabric to ext. pin
  T : in STD_LOGIC; --tristate control
  O : out STD_LOGIC; --from ext. pin to fabric
  Q : out STD_LOGIC; --from ext. pin registered
  PAD : inout STD_LOGIC; --EXTERNAL
  CLK : in STD_LOGIC; --EXTERNAL --SHARED_PORT
  ConfigBits : in STD_LOGIC_VECTOR
             ( NoConfigBits -1 downto 0 )
);
end entity IO_1_bidirectional;
```

At present, FABulous eFPGA fabrics are generated in VHDL. However, the actual tile functionality (like a LUT or specialized primitive) can be provided inside a tile as a Verilog module (or any hardware specification that the synthesis tool accepts). A user has to specify the number of configuration bits used (`NoConfigBits`). The keyword EXTERNAL in a comment guides the tool to export the port name all the way to the top-level eFPGA entity, like for example:

```
Tile_X9Y1_PAD : inout STD_LOGIC; -- EXTERNAL
```

The keyword `SHARED_PORT` directs the tool to combine all ports with the same name, which is used for the clock. Finally, all configuration bits are exported as a single concatenated bitstream into the tile, which provides the configuration storage (see Section 3.4).

Our approach differs from VPR that allows exploring different features automatically in the logic tiles. FABulous follows a different approach where users would normally start from tested templates that can still be adjusted as needed. Moreover, adding custom primitives to the fabric requires a user to follow the interface code convention, and those primitives can be directly instantiated in the Verilog code that is mapped onto the eFPGA fabric (all other steps are automated). In addition, a user has to decide how many adjacent tiles should be used for integrating a custom primitive, which depends on the area expected and the number of pins connected per tile (which should in practice be not more than around 40 inputs and 10-16 outputs for a Spartan-3-like FPGA fabric) and how much area is needed with respect to the basic logic tiles such that all tiles are roughly about the same physical size. In summary, FABulous allows users to start from a well-established FPGA fabric (e.g., a Xilinx or Lattice fabric clone) with the option to add own blocks and use custom ASIC technology cells as needed or as available. All these extensions are possible to use with very little added complexity and risk.

3.2.3 *Switch Matrix.*

Most tiles provide a switch matrix to which all tile wire ports and all primitive pins are connected. The wire end ports and primitive outputs are inputs to the switch matrix. The wire begin ports and the primitive inputs are the outputs respectively. In addition, each switch matrix provides VCC and GND inputs. This is useful for control inputs of primitives (e.g., a tri-state control for an I/O block). It is also possible to define stop-over ports that are inputs and outputs to a switch matrix at the same time. This allows it to build hierarchical routing networks. This is needed when modeling clones of Intel/Altera or newer Xilinx FPGA fabrics.

The switch matrix configures fixed routes only, which are controlled by the configuration bitstream. Any multiplexing at the user circuit level has to be done inside primitives. The complexity of primitives can vary from simple multiplexers all the way to ALUs (or even CPUs).

The adjacency of the switch matrix describes, for each switch matrix output, the configurable input connections. This is equivalent to a physical switch matrix multiplexer. The adjacency can be modeled by a list or by a table (similar as used in [50]). The list can be specified in Xilinx XDL-style where each possible multiplexer setting is a one line entry [8]. For instance, the following two lines would model the adjacency of a 2:1 switch matrix multiplexer:

```
E1Beg0, S2End1
E1Beg0, N2End1
```

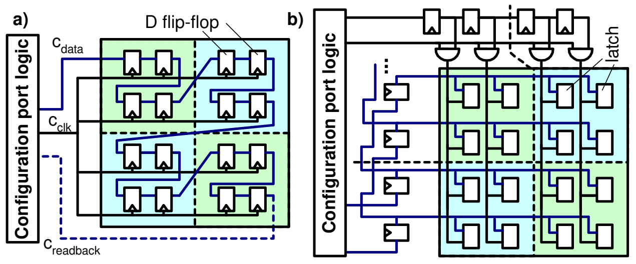

Figure 3: Configuration storage. a) scan-chain reconfiguration using D flip-flops for configuration storage, b) frame-based configuration. In b), a frame is a column of latches (or configuration SRAM cells). In the example, a shift register in the top of b) shifts through a one-hot to select the configuration frame-by-frame, where the frame load register (the chain of vertically aligned flip-flops) is updated after each frame.

Additionally, a compact format is supported that recursively unrolls all elements specified by a list. For instance, the following line sets $4 \times 3 = 12$ connections (`N2Beg0` ⟵ `N2Mid0`, ..., `W2Beg2` ⟵ `W2Mid2`):

```
[N|E|S|W]2Beg[0|1|2],[N|E|S|W]2Mid[0|1|2]
```

This format allows a user to easily import the switch matrix adjacency from Xilinx FPGAs given in the vendor-proprietary XDL format [8].

When generating the eFPGA switch matrices, FABulous is distinguishing different cases; if for an output no connection is set, a warning message is thrown, if one connection is set, we simply set a hard-wired connection, otherwise a multiplexer is generated. If no dedicated multiplexers have been provided, the tool is describing fully encoded multiplexers to be implemented by the ASIC standard-cell backend. If dedicated multiplexers are provided, those are automatically used (and any bitstream encoding is supported).

3.3 eFPGA I/O

FABulous itself provides the infrastructure for I/O, and it is up to the designer to add special primitives. For example, this could include gigabit transceivers or dedicated buses (e.g., AXI). In our case study 5, we included bidirectional I/O tiles directly into our fabric. This was integrated by instantiating the TSMC foundry pads inside the I/O tiles, as shown in Figure 1. However, for the physical ASIC implementation, the foundry pads are located in the top level hierarchy, and FABulous is exporting the connections from within the I/O tiles to the top level. The connection with the CPU was implemented the same method and the CPU I/O tiles basically only provide a switch matrix and buffers.

3.4 Configuration Storage

FABulous can use different memory technologies for storing the eFPGA configuration, including SRAM cells, DFF, or latches. In future research, we will examine using alternative technologies such as memristors for configuration storage. Configuration bits can be accessed in *scan-chain mode* or *frame mode* (see Figure3).

Scan-chains are the default when using DFFs. This mode is beneficial for small fabrics (below 100 LUTs) as the configuration logic is very light-weight, as shown in Figure 3a).

Frame mode is similar to the configuration modes used by the major FPGA vendors Xilinx, Lattice, and Intel. This mode uses a long load register arranged vertically over the full height of the fabric. From the configuration port, the bitstream is shifted into the load register and from there into the fabric, as shown in Figure 3b). The data in that load register will be called a *frame* (similar to the configuration frame definition used by Xilinx) and multiple frames, each addressed via a particular frame select signal, are required to write the configuration for a tile (e.g., a CLB).

Let n be the number of frame bit lines per tile and m be the number of frames select lines per tile; then we can use up to $n \times m \leq c_{tile}$ configuration bits, with c_{tile} being the total number of configuration bits. With this, the cost for wiring the configuration bits is proportional to $n + m$ and therefore n and m should be about the same to reduce the overall cost for wiring the configuration cells together. The n frame bit lines span horizontally across the entire eFPGA fabric and are shared for all tile types regardless of the specific tile configuration bits used C_{tile}^{i} for a tile i. Therefore, n will be constant, and the number of frames m^i is depending on the tile type i.

As a reference, a Xilinx Virtex-II CLB with 8 LUT-4 uses 1,600 configuration bits (for the LUT content, internal configuration and all routing). Correspondingly, a Xilinx Virtex-6 FPGA with 8 LUT-6 uses 2,194 configuration bits [34]. However, these architectures are more complex than a typical eFPGA (which will have fewer clock options and less long-distance routing), and they use one-hot coded pass transistor routing (see [39] for details on pass transistor routing). FABulous can use one-hot pass transistor routing multiplexers, but this requires a designer to provide corresponding full-custom cells (see Section 6) and typically dedicated configuration storage primitives. In the case of implementing the FPGA fabric using standard cell libraries, fully encoded multiplexers are the better choice, and the amount of configuration storage needed for an eFPGA will be about half. For instance, a 20 input one-hot encoded pass transistor multiplexer takes $4 + 5 = 9$ configuration bits on Xilinx Virtex-II but only 5 bits when using fully encoded multiplexers. We estimate that a typical eFPGA logic CLB will take roughly about 1Kb of configuration bits and we consequently set the number of frame bits per tile pragmatically to $n = 32$ (i.e. $\sqrt{1024}$).

The signal lines from the frame load register as well as the frame select signals contribute substantially to the overall wire resources needed to implement the eFPGA fabric on the ASIC. Therefore, just the frame configuration bits will add $n = 32$) wires. In the case of an assumed Xilinx Spartan-3 clone with 144 local wires (see Section 3.2.1), this is adding an extra 22% more horizontal wires per tile to the metal stack.

3.5 Reconfiguration Modes

This section discusses the different configuration modes available with FABulous Fabrics.

3.5.1 ***Partial Reconfiguration***. Section 3.4 mentions that scan-chain configuration is beneficial for tiny eFPGA fabrics. However, as a tiny fabric will likely not be able to host multiple modules in parallel, partial reconfiguration was not considered as useful; and in scan-chain configuration mode, only full configuration is available. Partial reconfiguration may be possible to implement through bypass paths that preserve certain regions of the FPGA, but that was not considered necessary for FABulous.

In the frame-based reconfiguration mode, partial reconfiguration is supported similarly as available in Xilinx Virtex-II FPGAs. By overwriting an existing configuration with exactly the same configuration content, no glitches will be caused to the user logic. This allows arbitrary two-dimensional partial reconfiguration. For instance, this allows selectively changing individual LUT functions or switch matrix multiplexer settings if only the corresponding LUT or switch matrix multiplexer bits are changed in the corresponding configuration frames.

3.5.2 ***Configuration Addressing & Multicast Configuration***. In scan-chain configuration mode, all configuration bits are concatenated into one long chain, and no specific addressing has been foreseen. In this mode, the entire configuration is shifted tile-by-tile row-wise (see Figure 3). This mode can consume excessive power during configuration because 1) we are shifting thousands of bits simultaneously and 2) during configuration, we are setting semi-random configurations until all bits have reached their final position. Those intermediate configurations may implement ring-oscillators, which can add further to dynamic power.[1] However, because we consider scan chains only for tiny fabrics, the extra power consumption during configuration is anticipated to be acceptable. While not yet implemented in FABulous, the long shift register could be broken into several parallel and shorter chains for faster configuration. As DFFs for implementing the scan-chain are expensive, configurations would normally be fully encoded such that short-circuits [7] are not a concern.

In the frame-based configuration mode, we address configuration data in three sections: 1) the resource column (i.e. the horizontal index of a column of tiles), 2) the configuration frame inside each column, and 3) the position of configuration bits inside a frame. This is very similar to the addressing scheme of older Xilinx FPGAs [66] and the models used in [34] and [51] for two-dimensional reconfiguration on classic Xilinx FPGAs.

However, we differentiate to the Xilinx approach in the final implementation by the method used for decoding the different configuration frames. Depending on the size of the fabric, we use one or two 32-bit configuration words at the beginning of each frame to provide a bitmask for 1) the resource column and 2) all frames. By activating more than a single column or frame, multiple frames will be written simultaneously. This is in particular useful for blanking an eFPGA region or the entire fabric. Alternatively, a shift register will be used to sequentially enable frames, as shown in Figure 3b).

Xilinx FPGAs provide a multiple frame-write command [66] that allows writing the content of the frame register consecutively to multiple frames. However, this requires sending a small command sequence for each frame such that the overall benefit is only saving

[1]The intermediate ring oscillators resulted in an interesting situation where the functional simulation of the fabric crashed due to an infinite loop while the eFPGA emulation on an FPGA board was working as expected.

about a third of the time whenever multiple frame-write is applied [11]. Our configuration mask shares some ideas with classic Xilinx XC6200 FPGAs [64], which used bitmasks for configuration compression, which could be applied to our bitstreams.

3.5.3 ***Configuration Readback.*** was not considered because this would require dedicated configuration cells or extra wires or logic, which we consider to be too complicated and/or too expensive. Additionally, most of the configuration data (if not all) is user logic invariant such that the fabric configuration data can be reconstructed from a local copy of the original configuration bitstream. Moreover, configuration readback could be seen as a security issue as it may be used to reveal the currently running circuit and the corresponding user states.

However, configuration readback can be useful for testing the fabric after fabrication. With this, the eFPGA fabric can be tested through the configuration port without the need to involve other parts of the system, which would be a design choice for a design for testability (DFT). As an alternative, we are considering to add a dedicated debug primitive to the fabric in future versions of FABulous. This would be similar to the JTAG primitives available in Xilinx and Altera FPGAs.

3.5.4 ***Encrypted Reconfiguration.*** All major FPGA vendors provide sophisticated security features allowing users to configure the FPGA securely with encrypted bitstreams. This option was not considered for our eFPGAs because it is complex to implement and relatively expensive because the cost for security is about the same regardless of the size of the FPGA fabric. Therefore, adding security IP would be over-proportionally expensive, considering rather small eFPGA fabrics.

However, secure configuration can be added through the surrounding SoC. For instance, the configuration may only be exclusively accessible by an on-chip CPU, and secure reconfiguration could then be implemented through that CPU.

3.6 Simplifications

The major FPGA vendors, like Intel and Xilinx, have huge design teams, and they use a substantial amount of third-party IP in their devices. Obviously, FABulous cannot compete with this and this section lists some of the major simplifications that we decided for. Nevertheless, due to the ability to customize an eFPGA to application requirements, we anticipate still to deliver a competitive quality of results (see also Section 6).

3.6.1 ***Single Clock Domain.*** While the VTR tool-chain does support multiple clock domains, we have currently not considered this for FABulous. We decided for a single clock domain because for our first release, we are only testing relatively small fabrics (e.g., our tapeout will only provide 384 LUTs, due to cost constraints for the tapeout). However, there is no principle obstacle to add this feature to our framework. When using a single clock domain, we use the automatic clock tree compilation in the backend tool (here Cadence Innovus). For supporting multiple clock trees in the future, it requires additional constraints in order to ensure that switching between different global clock trees will not incorporate significant extra skew. Furthermore, multiple clock domains typically require gateable clock buffers to limit power on clock trees.

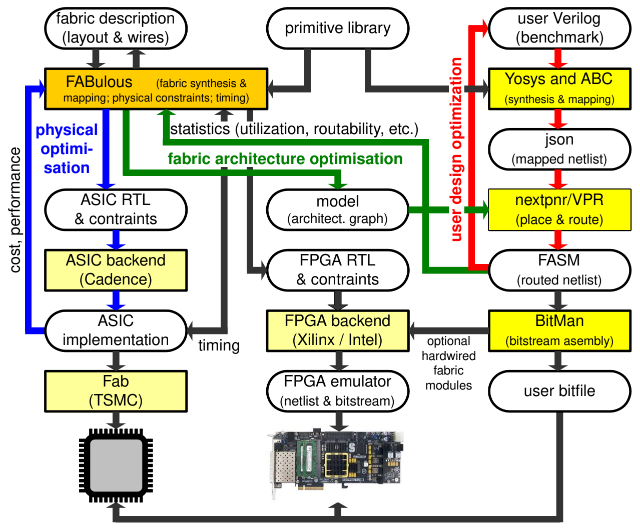
fabric description (layout & wires)
primitive library
user Verilog (benchmark)
FABulous (fabric synthesis & mapping; physical constraints; timing)
Yosys and ABC (synthesis & mapping)
physical optimi-sation
statistics (utilization, routability, etc.)
fabric architecture optimisation
user design optimization
json (mapped netlist)
cost, performance
ASIC RTL & contraints
model (architect. graph)
nextpnr/VPR (place & route)
ASIC backend (Cadence)
FPGA RTL & contraints
FASM (routed netlist)
ASIC implementation
timing
FPGA backend (Xilinx / Intel)
optional hardwired fabric modules
BitMan (bitstream asembly)
Fab (TSMC)
FPGA emulator (netlist & bitstream)
user bitfile

Figure 4: The FABulous ecosystems integrates several academic and industry tools into a complete framework supporting fabric ASIC generation, FPGA prototype implementation and eFPGA configuration bitstream generation.

3.6.2 ***Initial States.*** Xilinx FPGAs allow bringing up a user circuit in a specific state without an extra reset. Moreover, the state of all working flip-flops can be copied into a shadow register for debug purpose. Currently, FABulous omits this feature as this was not considered to be essential for eFPGAs. However, we provide emulation paths on Xilinx FPGAs such that state capturing for debug could be used in emulation mode.

4 FABULOUS FLOW AND FRAMEWORK

The FABulous framework is shown in Figure 4. We integrated industry tools for the eFPGA ASIC implementation, including Synopsis Design Compiler (for synthesis), PrimeTime (for static timing analysis), and Innovus (for the physical implementation). The primitive library consists of a standard cell library (so far we tested FABulous with a 180 nm TMSC library and a 45 nm open-source library). We also provided full-custom pass transistor multiplexers for both libraries. However, providing custom primitives is an optional optimization path and FABulous could always be used with standard cells only for providing portability to virtually any process. The custom cell design used the Cadence tools Spectre (for design), Virtuoso (for layout), and Liberate (for Characterization).

For mapping user circuits onto our eFPGAs, the framework integrates the Yosys, ABC, and nextpnr or VPR (for routing) open-source tools, as described in Section 2.2. FABulous has no limitations on exploiting the features available in these tools to implement eFPGA fabrics. For instance, in our prototype implementation, we used standard 4-input LUTs with carry chain logic, and the tools would support using fracturable LUTs instead (see [26] for a comprehensive study on fracturable LUTs). The final bitstream assembly is performed by BitMan. For simplicity, we omitted the bitstream model generation in Figure 4. The bitstream model links the routed netlist (in FASM format) that is generated by nextpnr (or VPR)

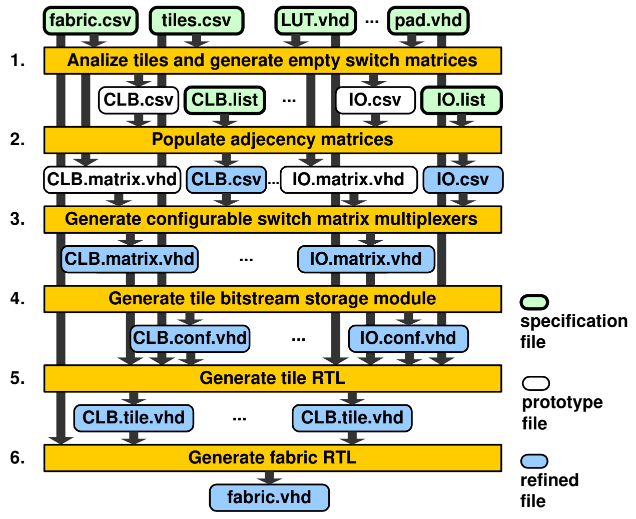

Figure 5: Bottom-up steps of the FABulous flow for building an eFPGA fabric.

back to the exact physical encoding of the primitives. In addition, it provides the mapping of the individual configuration bits into configuration frames.

As a distinct feature, FABulous provides an FPGA emulation path where an entire ASIC can be emulated prior to production. This feature was used for our test chip. The emulation model is cycle-accurate, including (partial) reconfiguration. The bitstreams used for the ASIC and the emulation model are identical.

Figure 4 shows the data that the different tools of the framework are exchanging. Moreover, there are optimization loops shown, which express the complexity of this framework. However, in case of a standard cell eFPGA design using default tiles, this complexity is entirely absorbed through scripts that drive the tools automatically.

4.0.1 ***FABulous Flow.*** Figure 5 shows the FABulous flow that is executed to generate an eFPGA fabric. Fabrics are generated bottom-up through six refinement steps. An eFPGA fabric is described in a CSV file (e.g., `fabric.csv` in the figure) where we define some global variables and modes (e.g., which configuration options and ports we want to support) and the actual fabric in terms of tiles. Taking the fabric in Figure 1 as an example, the fabric would be modeled as a grid being 5 tiles wide and 4 tiles in height (including the termination tiles around the six core tiles). FABulous provides constructs for compact modeling of larger fabrics. The grid expresses the exact fabric layout including the different tile types used (e.g., logic and arithmetic tiles) and it provides references to the specific tile descriptions that are provided in another CSV file (`tiles.csv`). A tile describes the wires to surrounding tiles and the primitives integrated in each specific tile through a reference to a file (e.g., `LUT.vhd` in the figure).

From this model, FABulous will in Step 1 generate prototype files of the switch matrices, which provide the input and outputs. In Step 2, we translate the adjacency lists into adjacency tables (see also Section 3.2.3 with details on switch matrix specifications). The tables are then used in Step 3 to synthesize the multiplexers of the switch matrices. The configuration storage is implemented in a dedicated module in Step 4. This allows using different configuration modes without direct interference with other parts of the fabric. In Step 5, FABulous integrates the different types of tiles (each consisting of switch matrix, configuration storage, and primitives). Finally, in Step 6, the whole fabric is generated. In this last step, we further generate some physical constraints for the Synopsys tools.

Not shown in the figure is a step where we generate the architecture graphs for nextpnr or VPR. This is performed in two phases, where we first generate models without timing information. From this, we generate scripts to query PrimeTime, and the information is annotated in a second phase to the architecture graph model for enabling timing driven routing.

4.0.2 ***Customizations and Optimizations.*** In FABulous, users have many ways to customize their fabrics. However, by providing well-optimized reference tiles, extensive customization is not required to build a working fabric. FPGA vendors hugely optimize their fabric, including hand-optimized physical implementations using full custom cells. Therefore, we expect an area gap of at least 2x when using a standard cell approach [33, 55].

Because the routing takes most of the FPGA resources of an FPGA fabric, we decided to provide custom pass transistor multiplexer cells. Those cells are relatively easy to implement (less than a day for an experienced designer, including all characterizations), and they gain substantial area savings. If provided, we use these multiplexers also for the logic cells (the LUT implementations), which require wide multiplexers for function evaluation (e.g., a 64:1 multiplexer for a LUT-4).

[56] proposes custom configuration D flip-flops for scan chain reconfiguration that are smaller than the original D flip-flops from the standard cell library. Fabulous uses latches instead, which provides about the same area savings for the configuration storage without the need for a custom cell design. Nevertheless, FABulous instantiates each configuration storage element explicitly, which allows it to replace the configuration memory cells with custom ones, if a designer wants to take advantage of a full custom configuration cell.

When using the default FABulous flow using a tile library, users can immediately explore logic and routing utilization using the Yosys-to-nextpnr/VPR flows. We can provide basic statistics on all resources and individual wires used, and if certain resources are not used over an entire benchmark, a user can decide to remove those resources (including wires) from the model. In the easiest case, it just requires filtering out those primitives and wires from the tile descriptions, which would consecutively result in a shrinking of tiles, which should result in better performance (as wire latency will shrink). However, these optimizations do not require the knowledge of an FPGA architect, and future versions of FABulous will propose users such customizations automatically.

5 CASE STUDY: RISC-V SOC WITH EFPGA

As a case study for testing the FABulous ecosystem in a tapeout, we implemented an SoC that utilizes the lowRISC Ibex core (originally named zero-riscy [15]. The core is a 32-bit microcontroller-class RISC-V CPU core written in SystemVerilog [63]. The eFPGA was integrated directly into the RISC-V core such that reconfigurable custom instructions are mapped into the CPU instruction space. In this system, developers can create hardware designs of custom instructions (in Verilog) targeting the eFPGA that will be used as an assembly instruction (e.g., as used in [6, 24, 35, 60]).

The CPU routes two operand registers from the 32 x 32-bit base registers to the eFPGA, and in the opposite direction, the CPU can select from three possible outputs, provided from three different adjacent regions (slots) on the eFPGA. This allows hosting three small instructions, or respectively fewer larger/more complex instructions by combining adjacent slots. Configurations can be swapped through partial runtime reconfiguration. The slots are designed identical, and the operand and result routing was hardwired in the eFPGA such that reconfigurable instructions are relocatable at bitstream level. A detailed description of the entire system, its programming, and application benchmarks is presented in [14].

[49] presents a chip that is featuring a RISC-V (the ETH PULP Platform) together with a QuickLogic eFPGA. The eFPGA is coupled through an AXI system bus. An AXI interface comes at an extra cost, and we implemented AXI interfaces [3] in order to quantify this cost in terms of fabric CLBs. It takes the area equivalent of ≈1.5 CLBs for a 32-bit AXI lite and ≈6 CLBs for a 32-bit AXI-stream interface. The latter corresponds to 1/8th of the CLBs available on our test eFPGA fabric, and we considered this option as being too expensive for our microcontroller-class of SoC. Therefore, we decided against an AXI interface. The ETH system in [49] is a more loosely coupled approach, which is similar to Xilinx Zynq FPGAs. In that system, the eFPGA is used as an accelerator offload engine. That approach is beneficial, if the amount of work is large (in terms of CPU cycles that can be saved through acceleration) because starting an accelerator commonly involves a driver layer. In contrast, custom instructions in our approach are called directly in user mode with low latency overhead. The system presented here is the first all open-source hardened RISC-V + eFPGA hybrid.

5.1 eFPGA Architecture

For designing and customizing the eFPGA fabric, we used a baseline FPGA routing architecture that is derived from a Xilinx Spartan-3 FPGA model (using a similar number of wires, similar adjacency in switch matrices, and support for wiretaps (MID ports)). However, we removed some long-distance wires as those are less useful for our rather tiny eFPGA. Note that the smallest FPGAs of the Spartan-3 family apply similar optimizations by not providing hex wires.

When examining the Spartan-3 adjacency of LUT input switch matrix multiplexers, we found that these FPGAs provide 8 LUTs (A, B,… H), each with four inputs (A0, A1,… H3). We found that groups of LUT inputs (e.g., {A0, B0, C0, D0} or {A1, B1, C1, D1}) share almost identical adjacency. This means that the LUT input ports of a group connect to (almost) the same wires/ports inside the switch matrix. We used this observation to save on the multiplexer logic by pairing every two LUTs and sharing some of the multiplexer logic. For instance, rather than using two separated 20:1 multiplexers for A0 and B0, we used 5 shared 4:1 muxes in a first bank and separate 5:1 muxes for each of the inputs. With this, we have not seen a degradation in routing but our logic tiles shrank by 7%.

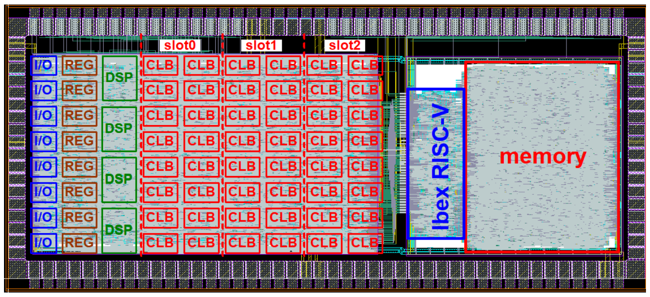

Figure 6: Layoyut of the implemented RISC-V + eFPGA hybrid.

Table 2: Synthesis reports for *e*FPGA and RISC-V

	eFPGA	RISC-V (with memory)
Ports	19270	2750
Cells	157017	41544
Area (um^2)	2283556	418205

For logic tiles, we used primitives that are very similar to Lattice iCE40 CLBs because their carry-chains are supported by Yosys. Furthermore, we used this to demonstrate customization capabilities through FABulous. However, we adopted the F7 and F8 multiplexers from the Xilinx Spartan-3 FPGA family to allow combining multiple adjacent LUTs for implementing wide logic functions. With this, our logic cells combine the baseline LUT-4 logic cells in Lattice with the support for wider logic functions. The eFPGA of the test chip provides 6 columns and 8 rows of logic cells, each featuring 8 x LUT4-FF pairs (348 LUTs/FFs in total).

To demonstrate the ability to support heterogeneous FPGAs, we added one DSP and one register file column, as shown in Figure 6. We fitted 4 DSP primitives, each providing 8x8 multipliers with 20-bit accumulators. The register file column adds 1Kb of memory to the fabric (about 3x more memory bits than available aggregated in all the CLB logic tiles). This was chosen to enable the implementation of a small soft micro-controller in the eFPGA. The register file primitives are also useful for 7 or 8-bit look-up table functions as used for AES S-boxes. With this, the fabric can implement a full AES round as a custom instruction.

We further added a column with bidirectional I/O cells (16 I/Os) such that the eFPGA could be used together with the SoC or standalone through the I/O pins or both. Note that the I/O pins also allow implementing peripherals directly accessible through custom instructions. In this case, the custom instructions could be seen as peripheral I/O move instructions.

In our test chip, we flattened hierarchies to fit more logic into the fabric and to have a more interesting test case. Therefore, relocated modules will have different timing at different positions. We are aware that this method adds complexity and may waste performance when considering module relocation. Therefore, we are not recommending this for commercial settings (for better user experience). However, flattening the physical implementation allowed us fitting the 4 DSP tiles into the fabric without applying other optimizations (our die size was fixed to 2mm x 4.5 mm).

Table 3: Synthesis reports for *e*FPGA building blocks

	CLB	DSP	RegFile	CLB	DSP	RegFile
Technology	180nm			45nm		
Cells	1554	4292	2711	3029	8662	5500
Area (um^2)	39630	52457	37312	10535	20103	13544
Power (mW)	8.23	11.82	8.35	2.46	4.68	3.78

Table 4: Area comparison between standard vs custom implementations for CLB in 180nm technology

Cells	Std+DFF	Std+Latch	+ Custom MUX
Area (um^2)	39630 (100%)	33279 (-16%)	27255 (-31.2%)
Power (mW)	8.23	9.24	15.37

Table 5: CLB'area comparison between this work and previous works

	FABulous	Virtex-E [37]	GILES [37]
Tech. (nm)	180	180	180
Cells	Custom	Custom	Standard
Area (um^2)	27255^3	35462^1 [46]	48282^2
	FABulous	[56]	[22]
Tech. (nm)	45	40	40
Cells	Standard	Standard	Standard
Area (um^2)	8998^3	27160^4	30625^4

$^{1)}$ [46] mentions the area for a Virtex-E tile consisting of 4xLUT-4 in 180 nm. We cannot further comment on how the logic in this area corresponds to our CLB (with 8xLUT-4 and switch matrix).
$^{2)}$ The area is denoted for a tile with LUT-4 and cluster size $n = 4$.
$^{3)}$ 8xLUT-4 Spartan-3 like architecture
$^{4)}$ OpenFPGA [56] and [22]: tiles with 10xfracturable LUT-6

Table 6: Timing reports for *e*FPGA building blocks

Delay (ns)	CLB	DSP	RegFile	CLB	DSP	RegFile
Technology	180nm - Typical corner			45nm - Typical corner		
Min	0.24	0.1	0.1	0.02	0.02	0.02
Max	0.84	0.26	0.28	0.08	0.25	0.25

5.2 ASIC Implementation

The system was implemented in a TSMC 180nm process on a 2mm x 4.5 mm die (Fig. 6)[2]. The RTL codes of eFPGA and RISC-V core with on-chip memory were separately synthesized using Synopsys Design Compiler and implemented by Cadence Innovus. The two parts were then integrated together globally in the top hierarchy. Table 2 reports synthesis results for the eFPGA (48x CLBs (=384x LUT4), 4x DSP and 8x RegFile) and RISC-V CPU.

[2]The test chip has not yet arrived back from packaging. However, the implementation confirms that our tool flow produces a netlist that is suited for tapeout.

6 EVALUATION AND COMPARISON

As aforementioned, the design was implemented in TMSC 180nm and 45nm open-source [45] technologies. The reports of synthesized eFPGA primitives using standard library cells are listed in Table 3. From this report, we can derive the area gain between flattened versus hierarchical implementation. In this case study, the area gain is about 5.3% due to flattening. Please note that this number does not include the wiring areas between building blocks, which could add up 20% of the design area.

Table 4 compares the area of CLB in different implementations. Replacing DFFs by Latches for reconfiguration bits reduces the CLB area by 16%, while further introducing custom 4:1 multiplexers improves the design area by 15.2%. Our custom 4:1 multiplexer cell is 43% smaller than the standard cell counterpart, but it is not optimized for power. Thus, users can trade-off between area and power when using the custom multiplexers versus standard cell ones.

Timing reports for building blocks in different technology nodes are shown in Table 6. As expected, the delay is significantly improved with smaller technology nodes. Comparison between our work and other works is shown in Table 5.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we introduced FABulous, an all open-source eFPGA framework, which integrates the popular Yosys, ABC, nextpnr/VPR tool-chains. The main differentiator over existing work is that no deep FPGA knowledge is required to drive this framework while still delivering good quality of results. Moreover, simple customizations help reducing the gap between hugely optimized commercial FPGA fabrics and FABulous fabrics, and experienced designers can easily improve quality through further optimizations, like custom cell designs. Moreover, to the best of our knowledge, FABulous eFPGAs are the only academic fabrics that currently support partial reconfiguration (thanks to their frame-based configuration scheme).

The versatility of the entire framework is a starting point for future research. As part of the project FORTE [1], we will use FABulous to build memristor-based FPGA fabrics. Other research directions will include mixed granularity fabrics where bit-level and word-level data path coexist. For instance, instead of routing single-bit wires between DSP blocks, we could provide multi-bit paths which are cheaper to implement for applications (like machine learning applications) that benefit from this. As tools, like nextpnr, work on abstract models of a physical architecture, this is already rudimentary supported when keeping bit and word-level signal paths apart (i.e., if the model does not provide direct connections between bit and word-level signals).
FABulous is available under: `https://github.com/FPGA-Research-Manchester/FabricGenerator`

ACKNOWLEDGMENT

This work is part of the project FORTE [1], which is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grant EP/R024642/1.

REFERENCES

[1] [n.d.]. *The FORTE (Functional Oxide Reconfigurable Technologies) project website.* Retrieved Nov. 15, 2020 from http://www.forte.ac.uk/

[2] V. Aken'Ova and R. Saleh. 2006. A "Soft++" eFPGA Physical Design Approach with Case Studies in 180nm and 90nm. In *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06).*

[3] Alex Forencich. [n.d.]. *Verilog AXI Components.* Retrieved Nov. 15, 2020 from https://github.com/alexforencich/verilog-axi

[4] Allied Market Research. 2017. *Global Embedded FPGA Market by Technology Forecast, 2018-2024.* Retrieved September 1, 2020 from https://www.alliedmarketresearch.com/embedded-fpga-market

[5] Altera Inc. 2002. *Altera 2002 Annual Report.* Retrieved Sep 10, 2020 from https://www.annualreports.com/HostedData/AnnualReportArchive/a/NASDAQ_ALTR_2002.pdf

[6] Peter M. Athanas and Harvey F. Silverman. 1993. Processor Reconfiguration Through Instruction-Set Metamorphosis: Compiler and Architectures. *IEEE Computer* 26, 3 (1993), 11–18.

[7] C. Beckhoff, D. Koch, and J. Torresen. 2010. Short-Circuits on FPGAs Caused by Partial Runtime Reconfiguration. In *20th International Conference on Field Programmable Logic and Applications (FPL).* https://doi.org/10.1109/FPL.2010.117

[8] Christian Beckhoff, Dirk Koch, and Jim Torresen. 2011. The Xilinx Design Language (xdl): Tutorial and Use Cases. In *6th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC).*

[9] Berkeley Logic Synthesis and Verification Group. [n.d.]. ABC: A System for Sequential Synthesis and Verification. \http://www.eecs.berkeley.edu/~alanmi/abc/.

[10] Vaughn Betz and Jonathan Rose. 1997. VPR: a New Packing, Placement and Routing Tool for FPGA Research. In *Field-Programmable Logic and Applications (FPL).* Springer Berlin Heidelberg, 213–222.

[11] C. Claus, F. H. Muller, J. Zeppenfeld, and W. Stechele. 2007. A new Framework to Accelerate Virtex-II Pro Dynamic Partial Self-Reconfiguration. In *2007 IEEE International Parallel and Distributed Processing Symposium.* 1–7.

[12] Menta Corporation. 2018. *Menta eFPGA Website.* Retrieved April 1, 2020 from https://www.menta-efpga.com/

[13] QuickLogic Corporation. 2018. *Evolution of our eFPGA Architecture.* Retrieved April 1, 2020 from https://www.quicklogic.com/products/efpga/efpga-technology/

[14] Nguyen Dao, Andrew Attwood, Bea Healy, and Dirk Koch. 2020. FlexBex: A RISC-V with a Reconfigurable Instruction Extension. In *International Conference on Field-Programmable Technology (FPT).* IEEE.

[15] P. Davide Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini. 2017. Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications. In *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS).* 1–8.

[16] A. DeHon. 2015. Fundamental Underpinnings of Reconfigurable Computing Architectures. *Proc. IEEE* (2015). https://doi.org/10.1109/JPROC.2014.2387696

[17] Sinjin Dixon-Warren. 2012. *A Review of TSMC 28 nm Process Technology.* Retrieved Sep 10, 2020 from http://www.maltiel-consulting.com/TSMC_28nm-Process-Reverse-Engineered.html

[18] EDN. 2001. *Altera Introduces APEX II Family.* Retrieved Sep 10, 2020 from https://www.edn.com/altera-introduces-apex-ii-family/

[19] Inc. Flex Logix Technologies. 2018. *Flex Logix eFPGAs.* Retrieved April 1, 2020 from https://flex-logix.com/efpga/

[20] Inc Flex Logix Technologies. 2020. *Gen2 TSMC 28HPC/HPC+/22ULPEFLX 4K.* Retrieved Sep 10, 2020 from \https://flex-logix.com/wp-content/uploads/2020/02/2020-02-EFLX4K-TSMC28HPCHPC-product-brief.pdf

[21] B. Grady and J. H. Anderson. 2018. Synthesizable Heterogeneous FPGA Fabrics. In *2018 International Conference on Field-Programmable Technology (FPT).* 222–229.

[22] Brett Grady and Jason H Anderson. 2018. Synthesizable Heterogeneous FPGA Fabrics. In *2018 International Conference on Field-Programmable Technology (FPT).* IEEE, 222–229.

[23] Lee Hansen. 2003. *SE 5.2i Delivers Head Start to Spartan-3 Designers.* Retrieved Sep 10, 2020 from \https://www.xilinx.com/publications/archives/xcell/Xcell46.pdf

[24] John R. Hauser and John Wawrzynek. 1997. Garp: a MIPS Processor with a Reconfigurable Coprocessor. In *Proc. of the 5th IEEE Symp. on FPGA-Based Custom Computing Machines (FCCM).* 12.

[25] E. Hung, F. Eslami, and S. J. E. Wilton. [n.d.]. In *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).*

[26] Mike Hutton, Jay Schleicher, David Lewis, Bruce Pedersen, Richard Yuan, Sinan Kaptanoglu, Gregg Baeckler, Boris Ratchev, Ketan Padalia, Mark Bourgeault, Andy Lee, Henry Kim, and Rahul Saini. 2004. Improving FPGA Performance and Area Using an Adaptive Logic Module, Vol. Lecture Notes in Computer Science 3203. 135–144.

[27] Efinix Inc. 2019. *Efinix® Partners with Samsung to Develop Quantum™ eFPGAs on 10nm Silicon Process.* Retrieved September 1, 2020 from https://www.efinixinc.com/company-pr-efinix-partners-with-samsung-for-10nm.html

[28] Efinix Inc. 2020. *Trion FPGA Overview.* Retrieved September 1, 2020 from https://www.efinixinc.com/products-trion.html

[29] Xilinx Inc. 2003. *Xilinx 2003 Annual Report and Proxy.* Retrieved Sep 10, 2020 from http://investor.xilinx.com/static-files/3266328c-a6e6-4048-bf45-486b984e8dbd

[30] Intel Inc. 2020. *Intel Stratix 10 Logic Array Blocks and Adaptive Logic Modules UserGuide.* Retrieved April 1, 2020 from https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/ug-s10-lab.pdf

[31] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon. 2010. Odin II - An Open-Source Verilog HDL Synthesis Tool for CAD Research. In *18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines.* 149–156.

[32] Jin Hee Kim and J. H. Anderson. 2015. Synthesizable FPGA Fabrics Targetable by the Verilog-to-Routing (VTR) CAD flow. In *Proceedings of the 25th International Conference on Field Programmable Logic and Applications (FPL).*

[33] Jin Hee Kim and Jason H. Anderson. 2017. Synthesizable Standard Cell FPGA Fabrics Targetable by the Verilog-to-Routing CAD Flow. *ACM Trans. Reconfigurable Technol. Syst.* 10, 2, Article 11 (April 2017), 23 pages. https://doi.org/10.1145/3024063

[34] Dirk Koch. 2012. *Partial Reconfiguration on FPGAs: Architectures, Tools, and Applications.* Springer.

[35] Dirk Koch, Christian Beckhoff, and Jim Torresen. 2010. Zero Logic Overhead Integration of Partially Reconfigurable Modules. In *Proc. of the 23rd Symp. on Integrated Circuits and System Design (SBCCI).* 103–108. https://dl.acm.org/doi/10.1145/1854153.1854181

[36] Ian Kuon, Aaron Egier, and Jonathan Rose. 2005. Design, Layout and Verification of an FPGA using Automated Tools. *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 215–226. https://doi.org/10.1145/1046192.1046220

[37] Ian Kuon, Aaron Egier, and Jonathan Rose. 2005. Design, layout and verification of an FPGA using automated tools. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays.* ACM, 215–226.

[38] G. Lemieux, E. Lee, M. Tom, and A. Yu. 2004. Directional and Single-Driver Wires in FPGA Interconnect. In *Proceedings. 2004 IEEE International Conference on Field-Programmable Technology (IEEE Cat. No.04EX921).* 41–48.

[39] David Lewis, Elias Ahmed, Gregg Baeckler, Vaughn Betz, Mark Bourgeault, David Cashman, David Galloway, Mike Hutton, Chris Lane, Andy Lee, Paul Leventis, Sandy Marquardt, Cameron McClintock, Ketan Padalia, Bruce Pedersen, Giles Powell, Boris Ratchev, Srinivas Reddy, Jay Schleicher, Kevin Stevens, Richard Yuan, Richard Cliff, and Jonathan Rose. 2005. The Stratix II Logic and Routing Architecture. In *FPGA.*

[40] Ang Li and David Wentzlaff. 2019. PRGA: An Open-source Framework for Building and Using Custom FPGAs. In *Proceedings of the 1st Workshop on Open-Source Design Automation (OSDA).*

[41] Hao Jun Liu. 2014. *Archipelago - An Open Source FPGA with Toolflow Support.* Master's thesis. University of California at Berkeley.

[42] Jason Luu, Jeffrey Goeders, Michael Wainberg, Andrew Somerville, Thien Yu, Konstantin Nasartschuk, Miad Nasr, Sen Wang, Tim Liu, Nooruddin Ahmed, Kenneth B. Kent, Jason Anderson, Jonathan Rose, and Vaughn Betz. 2014. VTR 7.0: Next Generation Architecture and CAD System for FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* 7, 2 (July 2014). https://doi.org/10.1145/2617593

[43] Prashanth Mohan, Oguz Atli, Onur O Kibar, and Ken Mai. 2020. A Top-Down Design Methodology for Synthesizing FPGA Fabrics Using Standard ASIC Flow. In *Proceedings of the 28th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA).*

[44] NanoExplore. 2020. *NanoExplore Website.* Retrieved September 1, 2020 from https://www.nanoxplore.com/

[45] NCSU Electronic Design Automation. [n.d.]. *Generic 45 nm UserGuide.* Retrieved Sept. 15, 2020 from \https://www.eda.ncsu.edu/wiki/FreePDK45:Manual

[46] Ketan Padalia, Ryan Fung, Mark Bourgeault, Aaron Egier, and Jonathan Rose. 2003. Automatic Transistor and Physical Design of FPGA Tiles from an Architectural Specification. In *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays (FPGA '03).* Association for Computing Machinery, New York, NY, USA, 164–172. https://doi.org/10.1145/611817.611842

[47] K. Pham, E. Horta, and D. Koch. 2017. BITMAN: A Tool and API for FPGA Bitstream Manipulations. In *Design, Automation, and Test in Europe (DATE).* https://doi.org/10.23919/DATE.2017.7927114

[48] K. D. Pham, M. Vesper, D. Koch, and E. Hung. 2019. EFCAD — An Embedded FPGA CAD Tool Flow for Enabling On-chip Self-Compilation. In *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* https://doi.org/10.1109/FCCM.2019.00011

[49] riscv.org. 2018. *QuickLogic Collaborates With ETH Zurich To Integrate eFPGA Into PULP Platform.* Retrieved April 1, 2020 from https://riscv.org/2018/08/quicklogic-collaborates-with-eth-zurich-to-integrate-efpga-into-pulp-platform/

[50] Herman Schmit and Vikas Chandra. 2002. FPGA Switch Block Layout and Evaluation. In *Proceedings of the 2002 ACM/SIGDA 10th International Symposium on Field-Programmable Gate Arrays (FPGA).* Association for Computing Machinery, New York, NY, USA, 11–18. https://doi.org/10.1145/503048.503051

[51] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght. 2006. Modular Dynamic Reconfiguration in Virtex FPGAs. *IEE Proceedings - Computers and Digital Techniques* 153, 3 (2006), 157–164.

[52] Achronix Semiconductor. 2019. *Speedcore Embedded FPGA IP*. Retrieved April 1, 2020 from https://www.achronix.com/product/speedcore

[53] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanovic. 2019. Yosys+nextpnr: An Open Source Framework from Verilog to Bitstream for Commercial FPGAs. In *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* https://doi.org/10.1109/FCCM.2019.00010

[54] SymbiFlow. [n.d.]. Project X-Ray - Xilinx Series 7 Bitstream Documentation. https://github.com/SymbiFlow/prjxray.git.

[55] X. Tang, E. Giacomin, B. Chauviere, A. Alacchi, and P. Gaillardon. 2020. OpenFPGA: An Open-Source Framework for Agile Prototyping Customizable FPGAs. *IEEE Micro* 40, 4 (2020), 41–48.

[56] Xifan Tang, Edouard Giacomin, Baudouin Chauviere, Aurélien Alacchi, and Pierre-Emmanuel Gaillardon. 2020. OpenFPGA: An Open-Source Framework for Agile Prototyping Customizable FPGAs. *IEEE Micro* 40, 4 (2020), 41–48.

[57] Princeton University. 2019. Princeton Reconfigurable Gate Array. https://prga.readthedocs.io/en/latest/

[58] wikichip.org. 2020. *28 nm lithography process*. Retrieved Sep 10, 2020 from https://en.wikichip.org/wiki/28_nm_lithography_process

[59] Steve Wilton. 1997. *Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memories*. Ph.D. Dissertation. Department of ECE, University of Toronto, Toronto, Canada.

[60] M. J. Wirthlin and B. L. Hutchings. 1995. DISC: the Dynamic Instruction Set Computer. In *Proc. on Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing (SPIE) 2607*, John Schewel (Ed.). SPIE – The International Society for Optical Engineering, Bellingham, WA, 92–103.

[61] Clifford Wolf. [n.d.]. Yosys Open SYnthesis Suite. http://www.clifford.at/yosys/.

[62] Claire Wolf, David Shah, Aliaksei Chapyzhenka, Karol Gugala, and Tim Ansell. [n.d.]. SymbiFlow Open source flow for generating bitstreams from Verilog. https://github.com/SymbiFlow/.

[63] www.lowrisc.org. 2020. Ibex RISC-V Core. \https://github.com/lowRISC/ibex.

[64] Xilinx. 1997. *XC6200 Field Programmable Gate Arrays (Version 1.10).*

[65] Xilinx Inc. 2005. *Xilinx 2005 Annual Report and Proxy.* Retrieved Sep 10, 2020 from http://media.corporate-ir.net/media_files/IROL/75/75919/reports/annual2005/XLNX_AR_05/pdf/Xilinx_Full_2005_AR.pdf

[66] Xilinx Inc. 2011. *UG702 (v13.3)- Partial Reconfiguration User Guide.*

[67] Xilinx Inc. 2017. *UG574: UltraScale Architecture Configurable Logic Block*. Retrieved April 1, 2020 from https://www.xilinx.com/support/documen-tation/user_guides/ug574-ultrascale-clb.pdf

[68] P. Yiannacouras, J. G. Steffan, and J. Rose. 2007. Exploration and Customization of FPGA-Based Soft Processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 2 (2007), 266–277.