Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2020) 28: 17 – 33

doi:10.3906/elk-1901-193

A review on embedded field programmable gate array architectures and configuration tools

Khouloud BOUAZIZ[1,2,*], **Abdulfattah M. OBEID**[3], **Sonda CHTOUROU**[1], **Mohamed ABID**[1,2]

[1]Computer and Embedded Systems (CES) Laboratory, National School of Engineers of Sfax, University of Sfax, Sfax, Tunisia

[2]Digital Research Center of Sfax (CRNS), Sfax, Tunisia

[3]National Center for Electronics and Photonics Technology, KACST, Riyadh, Saudi Arabia

Received: 28.01.2019 • **Accepted/Published Online:** 07.08.2019 • **Final Version:** 27.01.2020

Abstract: Nowadays, systems-on-chip have reached a level where nonrecurring engineering costs have become a great challenge due to the increase of design complexity and postfabrication errors. Embedded field programmable gate arrays (eFPGAs) represent a viable alternative to overcome these issues since they provide postmanufacturing flexibility that can reduce the number of chip redesigns and amortize chip fabrication cost. In this paper, we present an overview on eFPGAs and their architectures, computer aided design (CAD) tools, and design challenges. An eFPGA must be well-designed and accompanied by an optimized CAD tool suite to respond to target application's requirements in terms of power consumption, area, and performance. In this survey, we studied coarse-grained eFPGAs with customized blocks which are used for domain-specific applications and fine-grained eFPGAs that are used for general purposes but have lower performance.

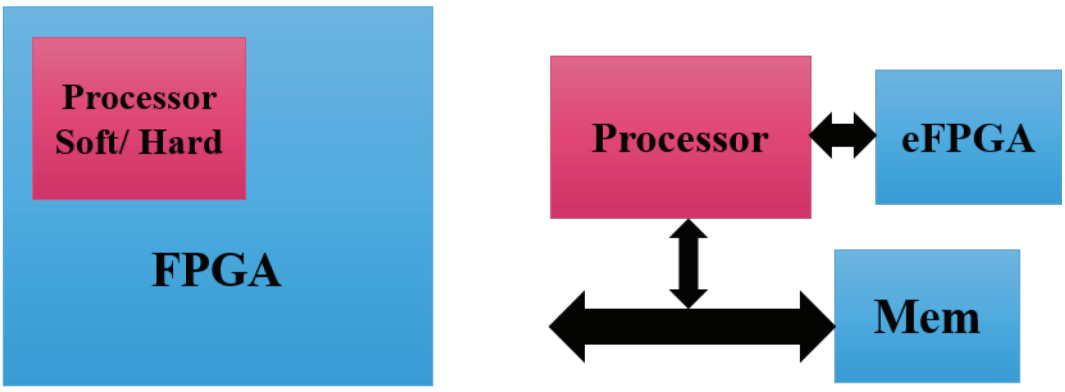
Key words: Embedded field programmable gate array, computer aided design tools, mesh-based architecture, tree-based architecture, field programmable gate array performance

1. Introduction

Current integrated circuits (ICs) exhibit an increasing logic density due to shrinking transistor sizes. This increase in transistor count has led to the emergence of system-on-chip (SoC) design methodologies to manage the increase in design size and complexity. SoC design methodologies, predesigned intellectual property (IP) blocks are combined on a single chip. These blocks may include embedded processors, memories, and dedicated coprocessors to accelerate specific processing functions. Incorporating IP blocks increases productivity gains, interdevice communication bandwidth and removes pin limitations. One major issue in SoC design today is the significant cost of design errors found after fabrication. These errors can be due to design errors not detected by simulation or it may be due to a change in design requirements. Therefore, an SoC must have some postmanufacturing flexibility to reduce the number of chip redesigns and to amortize chip development costs. The best way to solve this problem is to incorporate programmable logic cores into the SoC. The programmable logic core is a flexible logic fabric that can be customized to implement different applications or implement design changes after fabrication. Hence, it can be used to change design specifications and compensate for postmanufacturing errors costs. Field programmable gate arrays (FPGAs) have been known for their flexibility thanks to their reconfiguration capacity. Several companies have already proposed the inclusion of processors

*Correspondence: khouloud.bouaziz@enis.tn

(in soft or hard forms) inside the FPGAs (see Figure 1a). Hence, SoC designers can use the programmable logic cores to design different IPs and connect them to the FPGA's processor. However, compared to application-specific integrated circuits (ASICs), FPGAs pay for their flexibility in terms of area, power consumption, and performance. To illustrate the magnitude of this problem, we refer to the work presented in [1] where authors show that on average, an FPGA consumes 12 times more dynamic power than an equivalent ASIC, it is also 40 times larger and 3.2 times slower compared to ASIC [1]. This gap makes, FPGAs a less ideal choice for high performance, low power SoC designs. In addition, there are many cases where SoC designers would prefer to have small regions of programmable logic closest to the desired IP size, rather than large programmable logic regions in order to save chip area and minimize the area penalty. Therefore, the ideal solution for semiconductor companies wanting to add configurability to their chips is to have SoCs with embedded FPGA (eFPGA) IP [2] rather than stand-alone FPGAs (see Figure 1a). eFPGA can provide future SoC with hardware flexibility in a cost-effective way. In fact, once connected to the rest of the SoC components, the eFPGA can be configured to behave as a high-performance coprocessor or high-speed interface controller. eFPGA IPs can be reconfigured; hence, they reduce time-to-market and amortize chip development costs over several design derivatives. eFPGA can connect to an SoC through standard digital signaling which can be wide and fast, and enables lower latency compared to FPGA interfaces used for communication [1]. Moreover, eFPGAs' response latency is faster than FPGAs because communication bandwidth between eFPGAs and processors can be controlled through data bandwidth while it is limited by the interface bandwidth for FPGA. Hence, eFPGA offers the best tradeoff between the full FPGA and full ASIC solutions in terms of flexibility, performance, and power consumption. In this paper, we aim to introduce commercial and academic eFPGA solutions and discuss the challenges of eFPGA design in terms of power consumption, area, and performance. The organization of the rest of the paper is as follows: In Section 2, we present existing eFPGA architectures and topologies. Then, we detail eFPGA configuration flow used for eFPGA design implementation and layout generation. Section 3 provides an overview on academic and industrial eFPGAs solutions while discussing their advantages and limits. Afterwards, Section 4 recapitulates the main types of eFPGAs based on their granularity.



a) FPGA including processor

b) SoC including eFPGA

Figure 1. FPGA including processor vs. SoC including eFPGA

[1] Menta (2018). eFPGA vs FPGA [online]. Website http://www.menta-efpga.com/efpga-vs-fpga.html [accessed 06 10 2018]

2. Embedded FPGA: eFPGA

In this section, we present existing eFPGA architectures and topologies. Then, we detail eFPGA computer aided design (CAD) tools. Afterwards, we detail the eFPGA design challenges in terms of power consumption, area, and performance.

2.1. eFPGA architectures and topologies

An eFPGA is an FPGA core that, rather than being sold as a packaged chip, is sold as a block of semiconductor IP that can be integrated into an SoC. In this section, we presented popular FPGA architectures and interconnect topologies since they are the basis for many eFPGAs used today. FPGA architectures are generally based on a clustered architecture where several look-up-tables (LUTs) are grouped together to act as a configurable logic block (CLB), also called a cluster. FPGA routing interconnect assures different connections among FPGA components. In this section, we present the two existing types of FPGA routing interconnect topologies which are mesh-based and tree-based interconnects (see Figure 2):

- Mesh-based FPGAs, also called island-style FPGAs, are used in most academic and commercial SRAM-based FPGA architectures [2] [3–6]. In mesh-based FPGA architecture, LBs are placed in a 2D grid and surrounded by routing channels as illustrated in Figure 2. Generally, in mesh-based FPGA, the routing interconnect is composed of connection block (CBs), switch block (SBs), and wire segments. The group of programmable switches used to connect LB inputs and outputs to an adjacent routing channel is the CB. Similarly, the group of programmable switches used to connect horizontal and vertical channels intersection is the SB. The SB controls all connections between horizontal and vertical channels. Each vertical and horizontal routing channel contains W parallel wire segments.

- Tree-based or also called multilevel hierarchical FPGAs are used in a number of academic and commercial FPGA families [7–10]. This kind of architecture groups LBs into separated clusters recursively connected to form a hierarchical structure. Only architectures with more than 2 levels of hierarchy can be considered multilevel hierarchical interconnect.

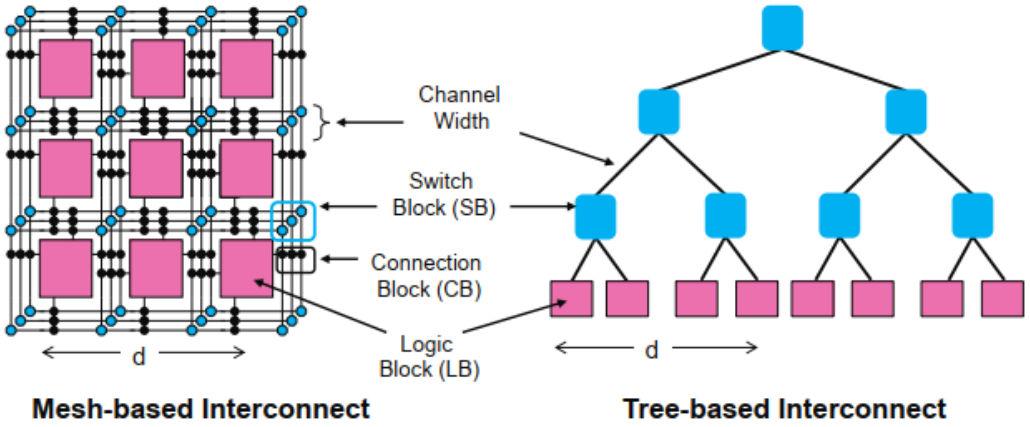


Figure 2. Mesh-based vs. tree-based FPGA interconnects.

[2]Virtex (2019). Xilinx Support [online]. Website http://www.xilinx.com/support/documentation/data [accessed 02 01 2019]

Mesh-based FPGAs are generally designed to maximize logic utilization whereas tree-based FPGAs are designed to increase interconnect utilization. The hierarchical FPGA architecture exploits the locality of connections to provide better performance and more predictable timing behavior. In the tree-based structure, the number of switches in series used to connect two LBs increases as a logarithmic function of the Manhattan distance d whereas the number of segments in series increases linearly with d for mesh structure. However, the wires delay increases exponentially with higher hierarchical levels and then the use of high hierarchical levels can impede the tree-based FPGA performance. In addition, the mesh model is more scalable than tree-based model since it has a regular structure. Hence, each architecture has its advantages. In order to use architecture resources to their optimal potential, the development of efficient CAD tools is of great importance.

2.2. eFPGA CAD tools

CAD tools are crucial for creating, exploring, programming, and validating different eFPGA architectures, as illustrated in Figure 3. They optimize architecture parameters such as LUT size, cluster size, and channel width. Figure 3 presents the diagram of eFPGA Analyser and Creator [11], providing the global overview of the tool suite including exploration and creation flows. Exploration phase enables to create efficient customized eFPGAs based on the set of eFPGA library components. The available components are pre-designed, pre-verified and sometimes silicon validated by test chips. Creation flow achieves the architectural exploration part which helps to create comprehensive libraries including a set of best cases and types which can either directly be suitable for target eFPGA or a good start point to narrow down the design space exploration direction. The building elements of eFPGA Creator are library and architecture Managers, hardware description language (HDL) generator, eFPGA programmer and analyzer. The library manager provides an infrastructure to create a central database of all components/blocks of eFPGA architecture. It consists of components like LBs and SBs with different architectural parameters (i.e. LUT size, cluster size, channel width, SB topologies, etc.). Thus, this flow procures a database of customized components and their silicon implementation to make architectural decisions. Architecture manager builds the architecture of eFPGA core using library manager data base components. Besides, it automatically creates lacking components required for core generation (i.e. components at the boundary of core, inputs/outputs etc.) if they are not provided by the library. Afterwards, architecture manager stores obtained components for future reuse. This tool also conceives the core's hardware for silicon implementation. The HDL generator creates HDL files along with front-end and back-end scripts to implement eFPGA RTL. eFPGA programmer (Figure 3) maps applications on the target eFPGA. It requires corresponding architecture files (including silicon information) to retrieve all the architectural and timing information needed to map applications on the created architecture. The first stage of this flow is synthesis. It converts a circuit description, typically written in an HDL, into a gate-level representation. This representation is a network composed of Boolean logic gates and flip-flops (FFs). The second stage is mapping where the gate-level presentation is further optimized then transformed into LBs relying on the available eFPGA technology. Subsequently, clustering enables to group sets of n LBs into clusters to be easily mapped into the eFPGA afterwards. Placement determines which LB within an eFPGA will implement the LB used by the circuit. Finally, routing determines how connections of LBs are realized within the prefabricated routing interconnect. The analyzer tools enable to determine the possibility to implement an application on the target architecture. It decides whether the architecture specifications meet the application requirements or not. And if the architecture does not possess enough resources, it determines the better tuning direction to optimize the architecture.

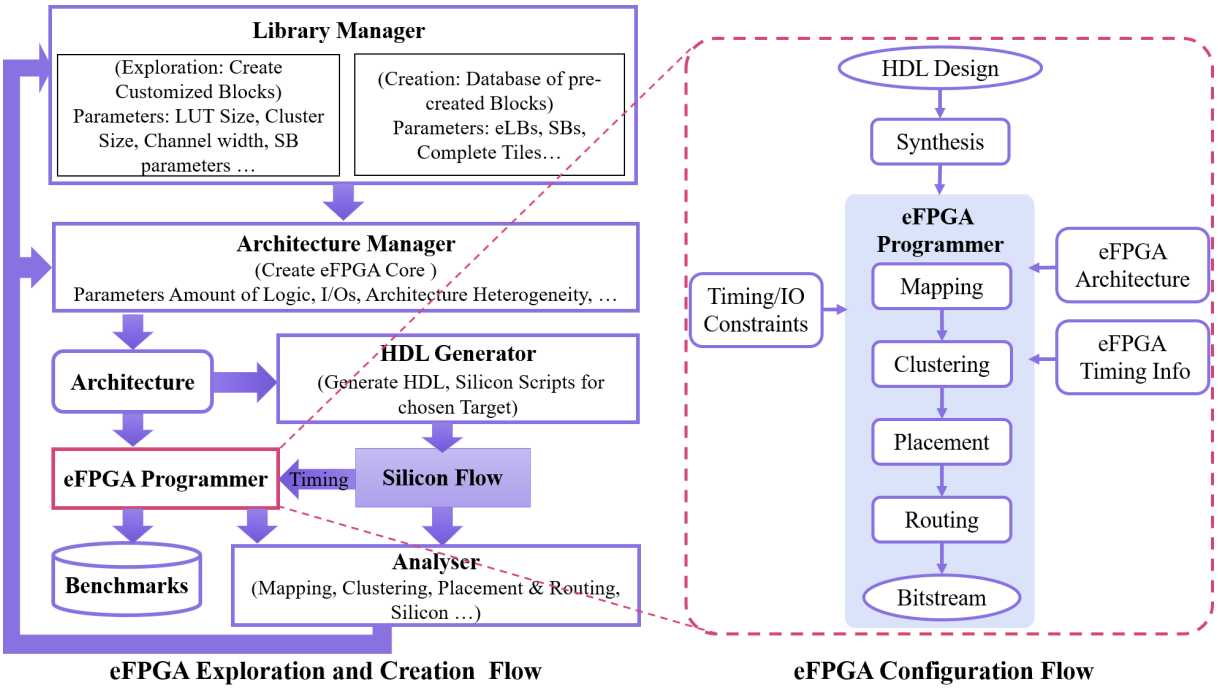


Figure 3. eFPGA CAD tools

2.3. eFPGA design challenges

Since eFPGA has FPGA-like classical architecture, it shares its limits and hence it brings similar challenges. As eFPGAs' capacities continue to grow, there is a considerable demand for eFPGAs with less area, lower power consumption, and less delay. As shown in Figure 4 [12], there is a high interdependency between eFPGA quality metrics which are: area, power dissipation, performance, and cost. For instance, decreasing the number of interconnect switches enables area, power consumption, and manufacturing cost improvement while it can penalize interconnect flexibility and thus affect performance. Hence, the main eFPGA design challenge is to find a good tradeoff between flexibility and performance in terms of area, power consumption, and delay. To do so, both academic and industrial communities have worked on various eFPGA types ranging from fine-grained architecture [3] [4] with almost absolute flexibility to coarse-grained, customized ones with high performance [5] [6]. Moreover, to ensure efficient design implementation and eFPGA integration, they use CAD tools tailored to satisfy eFPGA architecture specifications.

3. Examples of eFPGA solutions

In this section, the main existing eFPGAs and their configuration tools are detailed. There are both academic and commercialized eFPGAs.

3.1. MorphoSys: RC array

In [13], authors presented the MorphoSys project. MorphoSys architecture includes a reconfigurable processing unit, a general processing unit, a general-purpose processor along with a high bandwidth memory interface.

[3] Adicsys (2019). Adicsys: eFPGA Company [online]. Website http://www.adicsys.com/technology [accessed 02 01 2019]
[4] Achronix (2018). Speedcore eFPGA [online]. Website https://www.achronix.com/product/speedcore/ [accessed 01 12 2018]
[5] Menta (2018). Embedded Programmable Logic [online]. Website http://www.menta-efpga.com [accessed 30 11 2018]
[6] EFLEX (2018). Add Flexibility To Your SoC [online]. Website https://flex-logix.com/efpga/ [accessed 30 11 2018]

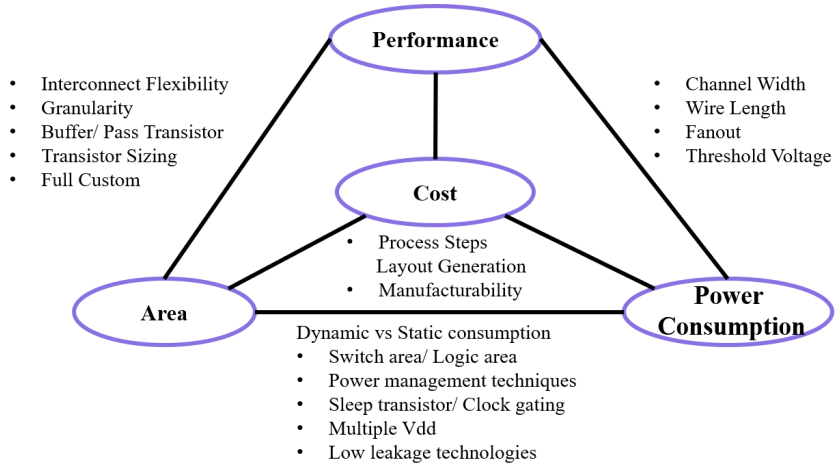


Figure 4. FPGA design challenges [12]

The reconfigurable processing unit, also called reconfigurable cell array (RC array), is an 8 × 8 array of coarse-grained reconfigurable cells (RCs). The general processing unit controls the RC array operation. Implementing applications into MorphoSys is achieved through a comprehensive programming environment represented in Figure 5. This tool suite includes 2 simulators, a graphical user interface, a context parser, and a C compiler. Mulate and MorphoSim are respectively C++ and VHDL simulators. MorphoSim models MorphoSys components and simulates applications using QuickVHDL simulation environment. While Mulate includes a C-based GUI to display and debug the application program during execution. mView is the GUI destined for RC array. It maps applications into RC array then verify and debug simulation runs. Besides, mView enables to study RC array simulation behavior. mView includes 2 run modes: a programming mode and a simulation mode. The programming mode's main role is to place and route applications into the RC array. Afterwards, it generates a context file for the implemented application. Regarding the simulation mode, it uses the context file to display RC states while executing user application. In order to simulate the whole system, mLoad is used to generate processor instructions and RC array context words. Furthermore, a MorphoSys C language compiler, mcc, is required to compile hybrid code for MorphoSys after manual partitioning between processor and RC array. Experimentation results showed that the use of MorphoSys to implement a data encryption application ameliorates performance compared to Pentium II processor and HipCrypto ASIC. Hence, this coarse-grained solution has a high performance for high-throughput, data-parallel applications. Besides MorphoSys [13], other works like [14–16] used reconfigurable solutions including a matrix of computational elements with reconfigurable interconnect targeting DSP applications for traditional SoCs.

3.2. Totem project: RaPiD

The work in [17] represents the Totem project. This project uses reconfigurable-pipelined datapath (RaPiD) architecture as a reconfigurable structure. RaPiD has intermediate flexibility standing between FPGA and ASIC. It aims to provide ASIC-like performance while maintaining FPGA-like reconfigurability. RaPiD achieves reconfigurability through the use of block components (memories, multipliers, and pipeline registers...). However, unlike commercial FPGAs, RaPiD does not target random logic. However, it mainly uses coarse-grained, computationally intensive functions. Standard cell libraries utilization ensures layout generation process automation while preserving domain-specific flexibility. The CAD flow of this eFPGA start with RaPiD components be-

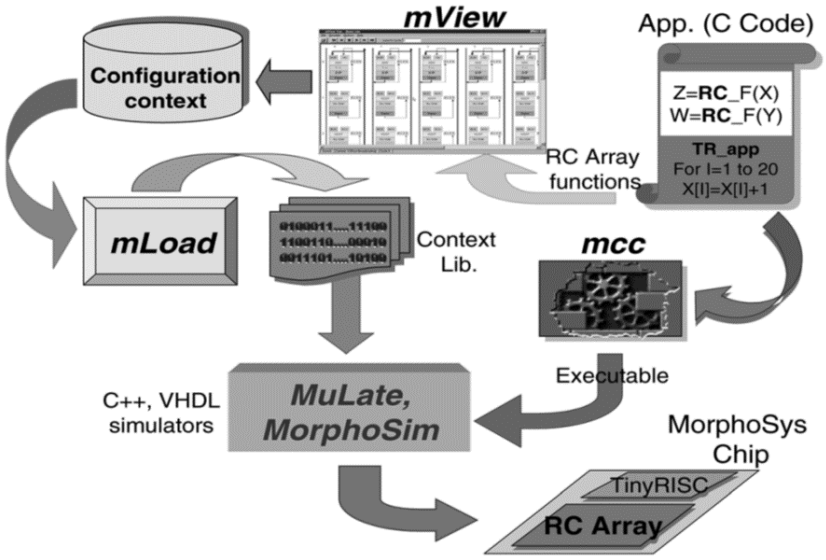


Figure 5. Software and simulation environment for MorphoSys [13].

havioral Verilog generation. The synthesis phase of the obtained output uses Synopsys [7] to produce Structural Verilog based on Tanner standard cells library [8]. Subsequently, placement and routing are achieved using Silicon Ensemble (SE). SE is part of the Cadence Envisia Tool Suite, it is capable of routing multiple layers of metal, including routing over the cells. SE is able to run from macro files which minimize user intervention. Finally, Cadence performs layout generation using the TSMC 0.25 μm design rules for all layouts. Epic Tool Suite, a robust circuit simulator, is utilized for performance analysis. Experimental results showed that if the application domain is known well in advance, FPGA-specific standard cells are 46% smaller and 36% faster than full-custom designs. However, if no reduction from the full functionality is possible, a standard cell approach is 42% larger and 64% slower compared to full-custom designs.

3.3. Hard eFPGA: VariCore

Authors in [18] represented a system including an embedded reconfigurable processor. This processor combines a configurable and extensible processor core and VariCore, an SRAM-based eFPGA. VariCore Embedded Programmable Gate Array is a hard IP core having a mesh-based topology. It consists of an array of PEG blocks surrounded by memory blocks, input/output pins and other interfaces. Each PEG includes an 8 $\times$ 8 array of functional group (FG) blocks along with the routing network to ensure communication between FG blocks. An FG encloses 4 logic units (LUs) joined with hard-wired carry chain for arithmetic use. VariCore can be configured through interfaces like JTAG. The design flow consists of a system-to-RTL and RTL-to-layout design flows. System-to-RTL flow is used for system architecture exploration and integration. It starts with an untimed model of the system. This model is written using C/C++ programming language, it describes the desired application. During this stage, the verification is done through simulations in CoWare N2C environment [19]. This methodology allows users to validate the system specifications. Subsequently, partitioning and interface synthesis enable the verification of the system at a cycle accurate abstraction level. Extensive simulations using profiler helps to group time-consuming segments of codes to be mapped on the eFPGA while implementing the

[7]Synopsys (2018). Silicon Design and Verification [online]. Website https://www.synopsys.com/ [accessed 15 10 2018]
[8]Tanner (2018). Tanner Research [online]. Website http://www.tanner.com/ [accessed 16 10 2018]

rest onto the microprocessor. RTL-to-layout flow includes both silicon implementation and eFPGA configuration flows which are executed at different times. Once the silicon implementation flow has generated the routed database, eFPGA flow can be implemented. eFPGA flow is repeated for each different function built as a soft macro. The RTL code of the CPU core, interface modules and IP blocks is synthesized and integrated with RAM blocks and FPGA hard macro in the floor planning environment. Timing requirements must be considered during synthesis for the logic cells that interfaces eFPGA with the rest of the system. After placement and routing, the final database is verified statically and dynamically against the RTL simulations. Finally, the bitstream and the timing analysis are generated. Implementation of intensive-computing applications using system including eFPGA enables up to 8 times speedups compared to the use of only one microprocessor. Hence, the use of hard eFPGA can increase flexibility and performance. However, its use is consuming in terms of area since it occupies 40% of the system surface.

3.4. Programmable logic cores

In [2], the authors proposed the use of programmable logic core (PLC). PLC is a flexible, fine-grained, logic fabric that can be customized to implement any digital circuit after fabrication. PLC is intended to overcome FPGA's main issues. Two main PLC architectures were explored. The first one is the directional architecture which resembles the standard mesh-based FPGA architecture. However, it is destined only for combinational logic since PLC targets only small and simple circuits. Besides, its interconnect network is directional (from left to right). The second one is the gradual architecture which is more area-efficient than the directional architecture because it includes fewer switches and has more input/output flexibility. Concerning the CAD flow for PLC generation, it starts with partitioning the design into fixed logic and programmable logic written in HDL language and destined to be integrated into the PLC. The programmable logic is then written in RTL description. Afterwards, the designer merges the behavioral descriptions of the fixed part and the programmable logic core to create a behavioral description of the block. Standard ASIC synthesis, place, and route tools achieve the implementation of the soft PLC behavioral description. Hence, both the programmable logic core and fixed logic are implemented simultaneously. Finally, the IC is fabricated. After generating the PLC, the user can configure it for the target application. Circuit placement uses VPR simulated annealing approach with modifications required to adapt to the target PLC architectures: if the directional architecture is used, the cost function depends on the delay of potential connections and distance of Manhattan between pins. Besides, it uses physical layout representation instead of conceptual representation in VPR case to increase estimation accuracy. As for the gradual architecture, the cost function depends on the number of nets demands for each multiplexer as well as multiplexer's capacity according to its column disposition in the PLC. Those modifications were necessary to guarantee routability. That is why routing used VPR negotiated congestion without any modification. Experimentation results showed that the use of the soft PLC requires 6.4 times more area and twice the critical path delay compared to a hardcore. These results prove that the use of fine-grained eFPGA with high flexibility is less efficient compared to a customized eFPGA.

3.5. Tactical eFPGA

The Tactical eFPGA [20, 21] introduced tactical cells as an intermediate solution between ASIC specific cells and FPGA standard cells. Since using ASIC specific cells induces time to market overhead while FPGA Standard cells use causes an area overhead. Tactical cells are created by efficiently modifying NMOS/PMOS distribution inside each cell. Tactical cell eFPGA is based on Island-style FPGA architecture described in VHDL RTL. This

architecture's regular structure improves runtime and Layout generation since it is enough to synthesize one tile and then replicate it to build the whole architecture. Besides, it decreases intertile wire length by an average of 38.5% because a structured layout is firm on placement. Thus, connections between tiles are reduced to short hops. As a result, the critical path delay decreases by 3%. Furthermore, the regular island-style architecture can use diffusion instead of buffering for inputs of multiplexer cell layouts to improve area and density. In order to implement applications into this eFPGA, VPR 4.30 is used for placement and routing. This academic tool has numerous advantages. Since it determines FPGA architecture required parameters to implement just the target circuit. VPR area model also can be used to explore area results. Even though this model gives an optimistic estimation, its values can be considered a lower bound. Besides, delay estimation depends on timing path exception that is generated based on program bitstream extracted from VPR. Experimental results proved that in terms of area, Tactical eFPGA is 58% smaller compared with standard cells eFPGA. However, it is up to 2.8 times larger than full-custom designs. If Tactical eFPGA is compared to hard eFPGA cores where only 1/20 to 2/3 of the area is used, the area's gain ranges from 1/3 to 19/20. However, there is no automatized tool to choose the appropriate eFPGA size to implement on SoC. That is why the designer should interfere to choose an eFPGA that ensures implementing target circuits while taking into consideration an error range. In terms of delay, Tactical eFPGA improves delay by an average of 40% compared to standard cells eFPGA while it is 10% slower than full-custom designs.

3.6. Arithmetic eFPGA

In [22], authors focused on designing a coarse-grained eFPGA architecture destined for arithmetic applications. Compared to control-dominated applications (Table 1), arithmetic applications require mostly short connections because of their high locality, few intermediate connections, and some long connections. Hence, in order to optimize architecture for arithmetic applications, logic elements (LEs) are designed to preserve communication locality for arithmetic datapaths like carry chains and to ensure that frequently used logic operations are directly mapped to one LE or LE cluster. Moreover, dedicated logic and interconnect are used to increase area efficiency. Thanks to the regular architecture of arithmetic eFPGA, it is possible to automate layout generation by using flexible datapath generator (DPG). The CAD flow in this work starts with detailing the configuration of single eFPGA components (routing switch (RS), CB, LE). Then it combines the obtained set of configurations to configure a complete macro. Afterwards, it automatically generates a configuration data bitstream and control bitstream which serve as input for the Cadence simulation environment. Arithmetic eFPGA was tested for different arithmetic applications along with Altera Cyclone I and Stratix. Compared to Altera Cyclone I, arithmetic eFPGA offers 5 to 10 times reduction in terms of energy, 3 to 5 times reduction in terms of area and 20% to 80% improvement in terms of delay. As for Altera Stratix, arithmetic oriented eFPGA outperforms it in terms of energy and delay. This reduction is mainly achieved since arithmetic eFPGA uses specific logic block elements destined for arithmetic applications whereas Cyclone I and Stratix contain general-purpose logic block elements. Arithmetic oriented eFPGA can implement control-oriented applications, but it is penalizing compared with commercial FPGAs.

3.7. XiSystem: PiCoGA and eFPGA

The XiSystem SoC, in [23], includes two main reconfigurable devices. The first reprogrammable device is the Pipelined Configurable Gate Array (PiCoGA) [24]. PiCoGA is a coarse-grained array of rows. Each row includes 16 reconfigurable logic cells (RLC). RLC includes 2 4-input 2-output LUTs, 4 registers, and dedicated logic for

Table 1. Arithmetic datapath application vs. control-dominated application.

Arithmetic datapath application	Control-dominated application
Mostly short connections (cause: high locality)	Much less short connections
Few intermediate connection length	High fraction of intermediate segments
Some long connections	Much less long connections

fast carry chain. PiCoGA routing network is a programmable interconnect matrix including switches with 1-bit granularity for better routability. PiCoGA configuration requires a C source code to generate the Dataflow Graph (DFG). Therefore, array core configuration bitstream and control unit configuration bitstream can be generated. The second configurable device is a 2D mesh-based, fine-grained eFPGA. eFPGA logic structure consists of 1-bit granularity Logic Cell (LC). LC includes 4:1 LUT, an FF, and a Multiplexer to choose between registered or asynchronous outputs. eFPGA routing network is a programmable interconnect matrix with 1-bit granularity switches. eFPGA communicates with SoC components through either advanced high-performance Bus (AHB) or input/output pads. Figure 6 illustrates the configuration flow for XiSystem software Toolchain. XiSystem SoC requires a dedicated C-based programming flow in the first place to generate the execution code for the XiRisc processor and the configuration bit-stream for PiCoGA and eFPGA. Configuration data for the PiCoGA are generated starting from a description using Griffy-C [25], a simplified sequential C-based language. The eFPGA configuration flow starts with HDL code synthesis then eFPGA mapping to generate bit-stream. Experimentation results show that PiCoGA can be 2 to 3 times more efficient than eFPGA for DSP applications. Besides, PiCoGA offers up to 89% energy savings, 15 times speedups, and 7 times increase of computational density compared to DSP-like architectures. As for eFPGA case, even if it cannot improve energy consumption or performance, it offers system interfacing flexibility. In addition, it can play the role of an accelerator for time-consuming tasks: for example, LCD display requires RGB pixel format while a MPEG decoder computes frames in YUV format, implementing the necessary conversion in the eFPGA and removing this procedure from the central core achieved 6% energy saving and 10% speedup on the whole decoder application. Moreover, the coprocessor configuration of the eFPGA was used to implement the row processing part of the IDCT algorithm, achieving a further 6% speedup.

3.8. Menta

Menta eFPGA is a standard-cell based eFPGA IP offered by Menta Corporation [9] to ensure flexibility in SOC designs. Its fabric allows modifications to the hardware during both development and postmanufacturing; thus, it enables reducing development time and cost. Menta product is qualified for GLOBALFOUNDRIES' (GF) advanced 14nm Fin Field Effect Transistor (FinFET) and 32nm Silicon on Insulator (SOI) process technologies. GF's advanced 14nm Low Power Process (LPP) can meet the requirements of reliability, power, stability, and size of coprocessing of complex next-generation SoCs. It can be used for defense, aerospace, ADAS, Internet of things, and data center systems applications. These characteristics were recently demonstrated in an eFPGA used by a large aerospace company in the United States. Menta eFPGA Figure 7a has a heterogeneous architecture. It includes embedded logic blocks (eLBs) able to map any Boolean operation and are interconnected as an array using programmable routing resources. A hardwired carry chain that connects eLBs to support high-speed arithmetic functions is included. Besides, embedded custom blocks (eCBs) can be added to increase performance

[9]Menta (2018). Embedded Programmable Logic [online]. Website http://www.menta-efpga.com [accessed 30 11 2018]

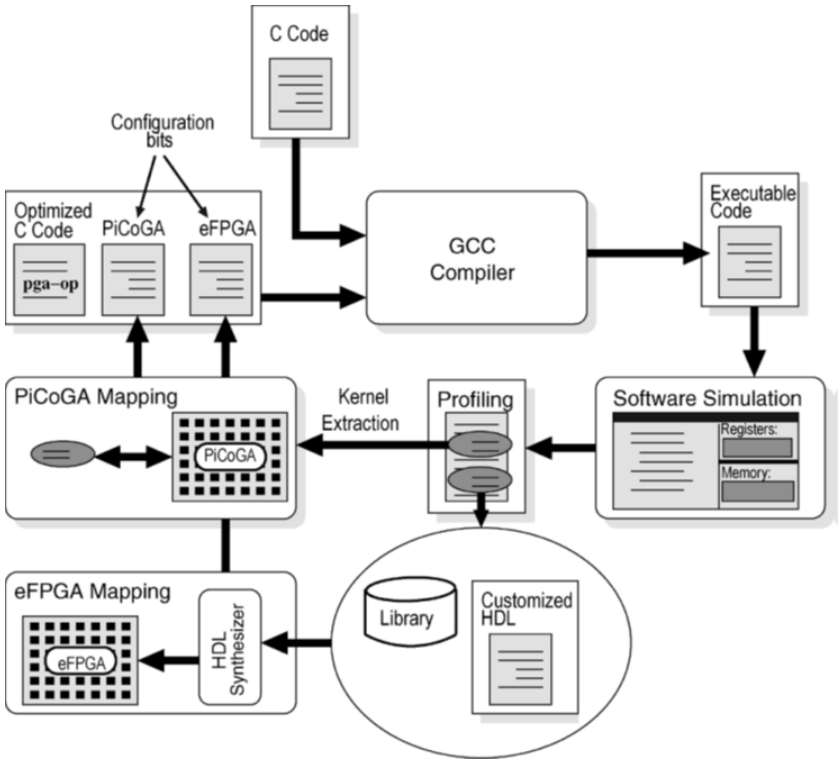


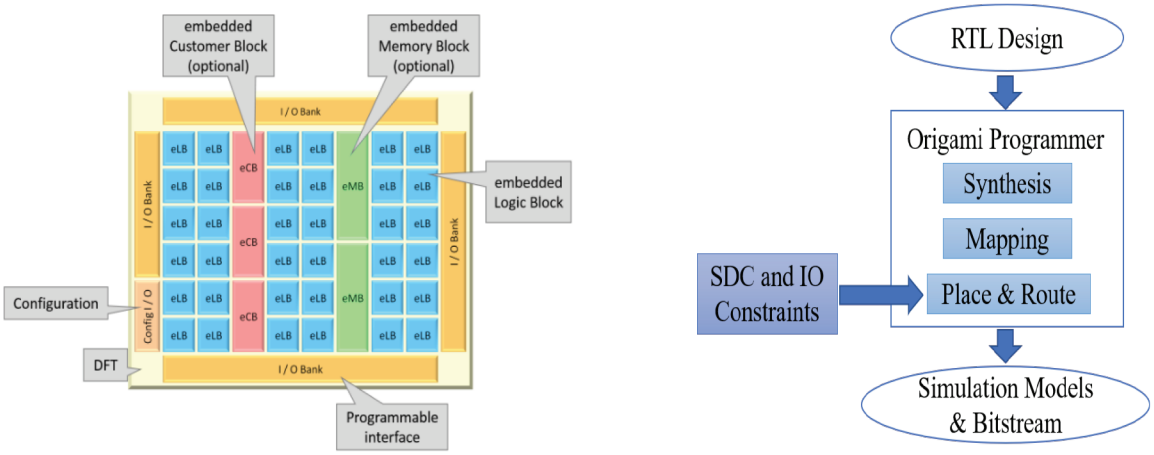
Figure 6. XiSystem software Toolchain [23].

for application-oriented designs. The Core IP can also include memory blocks (eMBs). Designers can choose a predefined IP core from the available preconfigured IP blocks (IPMs) families or can use Origami Designer to customize an eFPGA with the optimal size and perfect mix of blocks (eLBs, eCBs, eMBs). Figure 7b represents Origami Programmer design flow. Origami Programmer includes all the steps of a typical FPGA design flow and has an intuitive graphical interface. Origami Programmer requires an RTL design (VHDL, SystemVerilog, or Verilog), eFPGA architecture and Synopsys Design Constraints (SDC) to compute the programming bitstream file. It also generates speed estimations along with a model that can be used in standard simulation flow and for equivalence check. Menta solution enables customers to take instant benefit of eFPGA integration in their system while obtaining a custom solution for a specific node based on target market constraints. In [26], authors investigated the use of Menta eFPGA as a reconfigurable coprocessor joined with LEON3 processor and its effect on performance. They proved that at the cost of small area overhead, eFPGA enables a gain in terms of power consumption and speed.

3.9. EFLEX

EFLEX eFPGA is a Flex Logix product [10]. EFLEX coarse-grained architecture consists of reconfigurable building blocks (RBBs)and multiplier-accumulators (MACs) for DSP applications. RBB latest generation (second generation) includes 6-input LUTs, which can also be configured as dual 5-input LUTs with two bypassable FFs on the outputs. The choice of 6-input LUTs improves performance by 25% and density by 20%

[10]EFLEX (2018). Add Flexibility To Your SoC [online]. Website https://flex-logix.com/efpga/ [accessed 30 11 2018]



(a) Menta eFPGA Core IP architectural features

(b) SoC including eFPGA

Figure 7. Menta architecture and configuration flow.

compared to the dual-4-input-LUTs used in the first generation [11]. In terms of routing network, EFLEX utilizes a patented mixed radix hierarchical-mesh interconnect [27] inspired by Benes Networks [28]. This interconnect structure improves area efficiency by about 50% compared to traditional 2D FPGAs. EFLEX CAD flow requires an HDL file as input, it uses Synopsys Synplify synthesis tool to generate an '.edif' file. Afterwards, EFLEX Compiler Flex Logix uses the '.edif' file to pack, place, route, and compute the worst-case path timing and then generates the appropriate bitstream. EFLEX proved to be efficient in various applications like software reconfigurable input/output pin multiplexing and flexible input/output for MCU and IoT and even SoCs. In addition, EFLEX enables extending battery life for MCU and IoT since EFLEX can implement DSP applications at lower energy than ARM Cortex M4 [12]. This product also allows fast control logic for reconfigurable cloud data centers, DSP acceleration, debugging, and reconfigurable accelerators.

3.10. Synthesizable programmable core

Synthesizable programmable core (SPC) is a soft eFPGA core commercialized by Adicsys company [13] and it targets ASICs, SoCs, and silicon IPs in general. SPC has a fine-grained, mesh-based FPGA architecture which makes it scalable and available for various sizes. SPC structure is mainly based on a standard cell library to decrease nonrecurring engineering and verification costs. This eFPGA has a LUT-based programmable core. This core ensures mapping generic RTL functions. The CAD flow of SPC starts with RTL checking, elaboration, and synthesis achieved by ADICSYS distributable RTL front end. Then the synthesis of ASIC and FPGA are required to generate special RTL description. Afterwards, ADICSYS technology mapper, placer, and router enable to generate the bitstream. SPC is then fully integrated into RTL SoC design flow. Adicsys product offers 15% to 33% higher FPGA density compared to academic Mesh of Trees [29].

[11] EFLEX (2018). LUT6 [online]. Website http://www.flex-logix.com/6lut-faster-denser [accessed 16 09 2018]
[12] EFLEX (2018). EFLEX for DSP [online]. Website http://www.flex-logix.com/energy-efficient-dsp [accessed 30 11 2018]
[13] Adicsys (2019). Adicsys: eFPGA Company [online]. Website http://www.adicsys.com/technology [accessed 02 01 2019]

3.11. ArcticPro

QuickLogic Corporation [14] created ArcticPro product as an ultralow-power eFPGA technology. It is designed to be easily integrated into SoCs. ArcticPro has a fine-grained architecture which can implement multiple input functions and ensures high logic cell utilization. Logic cells can be configured as two independent 3-input LUTs or one 4-input LUT. ArcticPro flexible and multidrop routing network reduces routing delays and covers an array size ranging from 16 $\times$ 16 to 64 $\times$ 64. Aurora utilizes Mentor Graphics Precision tool for synthesis and a simulation tool compatible with industry standard EDA simulators (NC-Sim, VCS, Questa, ModelSim). The software ArcticPro IP generation requires Aurora Place and Route Tools. To achieve the layout generation of this IP, Borealis Compiler defines the required size of the eFPGA array then generates all necessary files for SoC integration.

3.12. Speedcore eFPGA

Speedcore IP eFPGA is the product of Achronix corporation [15]. Speedcore has a coarse-grained architecture. Its components are arranged in homogeneous columns with customizable height and number, the mixture of components is automatically defined by Achronix according to user application requirements. The device includes reconfigurable logic blocks (RLBs) consisting of 4-input LUTs, registers, and fast adders. Besides, it has logic RAM (LRAM) blocks and DSP building blocks designed in a modular structure which allows customers to define any quantity of resources required for their end system. Speedcore architecture is decided using Achronix ACE design tool. There are design rules that dictate the minimum and the maximum relative quantities for each of the available resources responding to customer requirements. This ACE generates the standard GDSII IP format which can be used for IC layout. In addition, Achronix ACE integrates Snapshot real-time design debugging tool to enable real-time user design evaluation.

4. Synthesis

Table 2 summarizes the main criteria of eFPGA examples denoted in Section 3. As illustrated in Table 2, eFPGAs can be imported as soft IP or hard IP. Hard IP approach offers highly optimized eFPGA core and predesigned with full-custom layout techniques. Nevertheless, only few variations of the eFPGA core are possible and there are numerous sources of underutilization. To overcome these limits, soft IP approach offers more flexible process since it proposes CAD flow to automatically generate an eFPGA fabric within the ASIC design flow (synthesis, place, and route). The main idea here is that an eFPGA architecture is described in behavioral RTL. Then, the ASIC design flow is used to create the physical IC layout. Moreover, eFPGA can be obtained without the bottleneck of a node-specific hard eFPGA solution.

We can also classify eFPGAs structures into fine-grained, coarse-grained, or heterogeneous architectures. Table 3 classifies eFPGAs into 3 main categories according to their granularity and compares them in terms of flexibility and performance based on the observation of eFPGAs examples in Section 3. Coarse-grained eFPGAs can reach high performance but their flexibility is limited to specific application domains. Meanwhile, fine-grained eFPGAs are generally more flexible and cover a wider range of applications which degrades their performance compared to customized platforms. To achieve a trade-off between both flexibility and performance, a heterogeneous eFPGA with mixed granularity is an appealing solution which needs to be backed up with optimized CAD tool suit to define and utilize eFPGA resources efficiently based on the target application's requirements.

[14] QuickLogic (2019). QuickLogic [online]. Website https://www.quicklogic.com/ [accessed 03 01 2019]
[15] Achronix (2018). Speedcore eFPGA [online]. Website https://www.achronix.com/product/speedcore/ [accessed 01 12 2018]

Table 2. eFPGAs: Creation flow output, configuration flow, and integration into SoC (N. A: not available)

eFPGA	eFPGA Characteristics		eFPGA CAD Tools	eFPGA Integration into SoC
	Soft/ Hard	Granularity		
MorphoSys : RC Array	Hard IP	Coarse-grained	Fully IP-specific: Mview: mapping, verification, debugging	Control through general processing unit
Totem project: RaPiD	Soft IP	Coarse-grained	Partially IP-specific: Synopsys for synthesis, Silicon Ensemble (SE) for placement and routing	N. A
Hard eFPGA : VariCore	Hard IP	Coarse-grained	Fully IP-specific Processor Datapath, Bus (AHB), I/O pins	
Programmable Logic Cores	Hard IP	Fine-grained	Partially IP-specific: Standard ASIC tools for synthesis, place, and route	I/Os
Tactical eFPGA	Soft IP	Fine-grained	Partially IP-specific: VPR 4.30 for placement and routing	N. A
Arithmetic eFPGA	Hard IP	Coarse-grained	Partially IP-specific: automatically generates a configuration data bitstream and control bitstream which serve as input for the Cadence simulation environment	N. A
XiSystem : PiCoGa	Hard IP	Coarse-grained	Fully IP-specific	Datapath
XiSystem : eFPGA	Hard IP	Fine-grained	Fully IP-specific	Bus (AHB), I/Os pins
Menta	Hard IP	Heterogeneous	Fully IP-specific: Origami Programmer	Customized
EFlex	Hard IP	Coarse-grained	Partially IP-specific: Synopsys Synplify for the synthesis	Bus (APB), Data path, I/O Pins
Synthesizable Programmable Core	Soft IP	Fine-grained	Fully IP-specific	Customized according to application
ArcticPro	Hard IP	Fine-grained	Partially IP-specific: Mentor Graphics Precision tool for synthesis, Aurora for Placement and Routing	Borealis Compiler generates all necessary files for SoC integration
SpeedCore eFPGA	Hard IP	Coarse-grained	Fully IP-specific: Achronix ACE design tool	Customized

All presented eFPGAs in this paper are synthesizable. Some of them developed prototypes of their eFPGA and other ones propose a CAD flow that automates synthesis and the process of layout generation of eFPGA. As illustrated in Table 2, eFPGA CAD tools can be fully IP-specific or partially IP-specific. A fully IP-specific flow is tailored to create and program the target eFPGA without relying on or being restricted with other

Table 3. eFPGAs Types and their main criterion.

eFPGA Types	Flexibility	Performance
Coarse-grained	Specific application domain	High performance: close to ASIC performance
Fine-grained	High flexibility	Medium performance: intermediate between processor and ASIC
Heterogeneous	Good/high flexibility	Good/high performance

existing tools, unlike partially IP-specific tools. Creating partially an IP-specific tool is more constrained and complex. However, generating an eFPGA solution compatible with existing environments is indeed beneficial since it enables the test and validation of the output [22] or its integration into other existing systems [16].

Finally, in Table 2 we highlight different techniques used for eFPGA integration into SoC which can be realized through using APB bus, Datapath, or I/O pins. Nevertheless, some eFPGAs were not integrated into SoCs [17, 20–22] mainly because of implementation problems. Moreover, some eFPGA constructors offer the possibility to customize the integration into SoC according to the target application [17] [18].

5. Conclusion

eFPGA is growing to be a trending technology since it offers the best tradeoff between the full FPGA and full ASIC solutions in terms of flexibility, performance, and power consumption. Its reconfiguration capacity as an IP to be implemented in hardware systems like SoCs makes it a very appealing solution for many fields. This paper revolves around eFPGA solutions for SoCs integration. We presented eFPGA architectures and interconnect topologies, associated CAD flow, and crucial challenges they are facing. We surveyed existing academic and industrial eFPGA solutions for SoCs integration. Then, we analyzed and classified these solutions based on their eFPGA design, CAD tools, and different techniques to integrate them into the SoC. In the light of our survey, we studied coarse-grained eFPGAs with customized blocks which are used for domain-specific applications and fine-grained eFPGAs that are used for general purposes but have lower performance. We discussed also the concept, motivation, and limits of hard and soft eFPGAs IPs.

Acknowledgment

This work is supported by Computer and Embedded Systems Laboratory and Digital Research Center of Sfax.

References

[1] Kuon I, Rose J. Measuring the gap between FPGAs and ASICs. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems 2007; 26 (2): 203-215. doi: 10.1109/TCAD.2006.884574

[2] Wilton SJE, Kafafi N, Wu JCH, Bozman KA, Aken'Ova VO et al. Design considerations for soft embedded programmable logic cores. IEEE Journal of Solid-State Circuits 2005; 40 (2): 485-497. doi: 10.1109/JSSC.2004.841038

[3] Li A, Wentzlaff D. PRGA: An open-source framework for building and using custom FPGAs. In: The First Workshop on Open-Source Design Automation; Florence, Italy; 2019. pp. 1-6.

[16] Achronix (2018). Speedcore eFPGA [online]. Website https://www.achronix.com/product/speedcore/ [accessed 01 12 2018]
[17] Menta (2018). Embedded Programmable Logic [online]. Website http://www.menta-efpga.com [accessed 30 11 2018]
[18] Achronix (2018). Speedcore eFPGA [online]. Website https://www.achronix.com/product/speedcore/ [accessed 01 12 2018]

[4] Chtourou S, Marrakchi Z, Amouri E, Pangracious V, Abid M et al. Performance analysis and optimization of cluster-based mesh FPGA architectures: design methodology and CAD tool support. Turkish Journal of Electrical Engineering and Computer Sciences 2017; 25 (3): 2044-2054. doi: 10.3906/elk-1506-51

[5] Gaillardon PE, Tang X, Kim G, De Micheli G. A novel FPGA architecture based on ultrafine grain reconfigurable logic cells. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2015; 23 (10): 2187-2197. doi: 10.1109/TVLSI.2014.2359385

[6] Rose J, Luu J, Yu CW, Densmore O, Goeders J et al. The VTR project: architecture and CAD for FPGAs from verilog to routing. In: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays; Monterey, CA, USA; 2012. pp. 77-86.

[7] Yuan FL, Wang CC, Yu TH, Marković D. A multi-granularity FPGA with hierarchical interconnects for efficient and flexible mobile computing. IEEE Journal of Solid-State Circuits 2015; 50 (1): 137-149. doi: 10.1109/JSSC.2014.2372034

[8] Marrakchi Z, Mrabet H, Farooq U, Mehrez H. FPGA interconnect topologies exploration. International Journal of Reconfigurable Computing 2009; 2009 (6): 1-13. doi: 10.1155/2009/259837

[9] Hutton M, Adibsamii K, Leaver A. Timing-driven placement for hierarchical programmable logic devices. In: Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays; Monterey, CA, USA; 2001. pp. 3-11.

[10] Lai YT, Wang PT. Hierarchical interconnection structures for field programmable gate arrays. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 1997; 5 (2); 186-196. doi: 10.1109/92.585219

[11] Syed Zahid A. eFPGAs: Architectural explorations, system integration and a visionary industrial survey of programmable technologies. PhD, University of Montpellier, Montpellier, France, 2011.

[12] Hutton M. The design of modern FPGA Architectures. In: International Symposium The Future of Configurable Hardware; Gent, Belgium; 2004.pp.1-10.

[13] Singh H, Lee MH, Lu G, Kurdahi FJ, Bagherzadeh N et al. MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications. IEEE Transactions on Computers 2000; 49 (5): 465-481. doi: 10.1109/12.859540

[14] Becker J, Glesner M. A parallel dynamically reconfigurable architecture designed for flexible application-tailored hardware/software systems in future mobile communication. The Journal of Supercomputing 2001; 19 (1): 105-127. doi: 10.1023/A:1017456815823

[15] Zhang H, Prabhu V, George V, Wan M, Benes M et al. A 1 V heterogeneous reconfigurable processor IC for baseband wireless applications. In: IEEE International Solid-State Circuits Conference; San Francisco, CA, USA; 2000. pp. 68-69.

[16] Obeid AM, Qasim SM, BenSaleh MS, AlJuffri A. HyDRA: hybrid dynamically reconfigurable architecture for DSP applications. IEICE Transactions on Electronics 2016; 99 (7): 866-877. doi: 10.1587/transele.E99.C.866

[17] Phillips S, Hauck S. Automatic layout of domain-specific reconfigurable subsystems for system-on-a-chip. Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-Programmable Gate Arrays; Monterey, CA, USA; 2002. pp. 165-173.

[18] Borgatti M, Lertora F, Forêt B, Calí L. A reconfigurable system featuring dynamically extensible embedded microprocessor, FPGA, and customizable I/O. IEEE Journal of Solid-State Circuits 2003; 38 (3): 521-529. doi: 10.1109/JSSC.2002.808288

[19] Schewel J. A hardware/software co-design system using configurable computing technology. In: Parallel Processing Symposium; Orlando, FL, USA; 1998. pp. 620-625.

[20] Aken'Ova VO. Bridging the gap between soft and hard eFPGA design. PhD, University of British Columbia, Canada, 2005.

[21] Aken'Ova VC, Lemieux G, Saleh R. An improved "soft" eFPGA design and implementation strategy. In: International Conference on Application-specific Systems, Architectures and Processors; San Jose, CA, USA; 2006. pp. 125-131.

[22] von Sydow T, Neumann B, Blume H, Noll TG. Quantitative analysis of embedded FPGA-architectures for arithmetic. In: Proceedings of the IEEE 2005 Custom Integrated Circuits Conference; Steamboat Springs, CO, USA; 2005. pp. 179-182.

[23] Lodi A, Cappelli A, Bocchi M, Mucci C, Innocenti M et al. XiSystem: a XiRisc-based SoC with reconfigurable IO module. IEEE Journal of Solid-State Circuits 2006; 41 (1): 85-96. doi: 10.1109/JSSC.2005.859319

[24] Lodi A, Toma M, Campi F. A pipelined configurable gate array for embedded processors. In: Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays; Monterey, California, USA; 2003. pp. 21-30.

[25] Mucci C, Chiesa C, Lodi A, Toma M, Campi F. A C-based algorithm development flow for a reconfigurable processor architecture. In: International Symposium on System-on-Chip; Tampere, Finland; 2003. pp. 69-73.

[26] Syed Zahid A, Julien E, Laurent R, Jean-Baptiste C, Gilles S et al. Exploration of power reduction and performance enhancement in LEON3 processor with ESL reprogrammable eFPGA in processor pipeline and as a co-processor. In: Proceedings of the Conference on Design, Automation and Test in Europe; Nice, France; 2009. pp. 184-189.

[27] Yuan FL, Wang CC, Yu TH, Marković D. A multi-granularity FPGA with hierarchical interconnects for efficient and flexible mobile computing. IEEE Journal of Solid-State Circuits 2015; 50 (1): 137-149. doi: 10.1109/JSSC.2014.2372034

[28] Nassimi D, Sahni S. A self-routing Benes network and parallel permutation algorithms. IEEE Transactions on computers 1981; C-30 (5): 332-340. doi: 10.1109/TC.1981.1675791

[29] Balkan AO, Qu G, Vishkin U. Mesh-of-trees and alternative interconnection networks for single-chip parallelism. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2009; 17 (10): 1419-1432. doi: 10.1109/TVLSI.2008.2003999