IAES Journal of Electrical and Electronics Engineering
Vol. 4, No. 2, December 2017, pp. 21~25
ISSN: 2338-7432, DOI: 10.11591/jeee.v4i2.pp21-25

Beyond conventional digital design: advances in FPGA architectures, programming, and hardware acceleration

Novita Indriani, Andi Martanto
Faculty of Industrial Technology, Universitas Ahmad Dahlan, Yogyakarta, Indonesia

Article Info

Article history:

Received Aug 27, 2016
Revised May 5, 2017
Accepted Sep 9, 2017

Keywords:

Digital design
Field-programmable gate array
FPGA architecture
Hardware acceleration
High-level synthesis
Reconfigurable computing

ABSTRACT

Field-programmable gate arrays (FPGAs) have evolved from configurable logic devices used mainly for prototyping into versatile platforms for implementing application-specific digital systems. Their appeal lies in the ability to combine hardware-level parallelism with post-fabrication reconfigurability, although this flexibility introduces design and programming challenges that are less prominent in conventional processors. This review examines the development of FPGA technology through three connected themes: architectural organization, programming methodologies, and hardware acceleration. It first outlines the principal resources of contemporary FPGA fabrics and discusses the performance, area, and energy trade-offs associated with programmability. It then considers the progression from hardware description languages (HDLs) and register-transfer-level design toward high-level synthesis, emphasizing the productivity gains and quality-of-result limitations of abstraction. The review also examines how FPGA accelerators are integrated into embedded and data-center systems, where workload specialization and parallel execution can improve application performance. Across these themes, the analysis identifies persistent barriers involving memory bandwidth, interconnect overhead, tool maturity, verification effort, and design expertise. The review concludes that FPGA development depends not only on denser logic resources but also on better design automation, software integration, and system-level optimization.

This is an open access article under the CC BY-SA license.

1. INTRODUCTION

Conventional digital system design has traditionally relied on two major implementation approaches: general-purpose processors and application-specific integrated circuits (ASICs). Processors provide flexibility through software execution, but their sequential instruction model can limit performance in applications requiring extensive parallel computation. ASICs offer highly optimized performance and energy efficiency, although their development involves substantial design costs, longer development cycles, and limited flexibility after fabrication. Field-programmable gate arrays (FPGAs) provide an alternative by allowing hardware functionality to be configured after manufacturing, combining a degree of hardware specialization with the possibility of redesign [1].

The increasing complexity of digital applications has strengthened the need for computing platforms that can balance performance, flexibility, and development effort. Signal processing, communication systems, embedded control, and data-intensive computing frequently require parallel operations and predictable processing behavior. Although conventional processors can execute these workloads, software-based implementations may not fully exploit the available parallelism. FPGA technology addresses this limitation by allowing designers to construct customized processing architectures that execute multiple operations concurrently [2]. Despite these advantages, FPGA adoption has historically been constrained by the complexity of hardware design and programming. Designers working with traditional hardware description languages (HDLs) must understand timing, concurrency, resource allocation, and circuit-level

behavior. These requirements create a substantial learning curve, particularly for software engineers who are accustomed to sequential programming models. This review therefore investigates three related research problems: how FPGA architectures have developed to support increasingly demanding applications, how programming methods have attempted to improve design productivity, and how FPGA-based hardware acceleration can provide practical benefits in digital systems.

2. REVIEW APPROACH AND SCOPE

This article adopts a narrative review approach based on established publications concerning reconfigurable computing, FPGA architecture, high-level synthesis, and hardware acceleration. The literature selection emphasizes foundational research and representative developments published no later than 2016. Particular attention is given to studies that explain architectural principles, compare FPGA implementation characteristics, evaluate programming methodologies, or demonstrate FPGA deployment in practical computing environments. The review is intended to synthesize the relationships among these topics rather than provide a quantitative meta-analysis of individual FPGA devices.

3. EVOLUTION OF FPGA ARCHITECTURES

3.1. Fundamental architectural organization

An FPGA consists of configurable logic resources connected through a programmable interconnection network. Its architecture commonly includes configurable logic blocks, lookup tables, flip-flops, routing switches, input/output blocks, and embedded memory resources. Modern devices may also incorporate dedicated arithmetic units, digital signal processing blocks, clock management circuits, and high-speed transceivers. These heterogeneous resources allow an FPGA to support different computational structures without requiring every function to be constructed from basic logic elements.

The lookup table is one of the fundamental components responsible for FPGA flexibility. It implements combinational logic by storing the output values associated with different input combinations, while programmable routing establishes connections among logic elements. Sequential operations are supported by registers and flip-flops, allowing designers to construct pipelines and finite-state machines. The combination of these components enables the implementation of customized datapaths and control structures for application-specific requirements [1].

3.2. Architectural flexibility and performance trade-offs

FPGA flexibility introduces an important trade-off between programmability and implementation efficiency. Programmable switches and configurable logic require additional circuitry compared with fixed-function ASIC implementations. Consequently, FPGA designs may experience higher area requirements, longer signal paths, and greater energy consumption than equivalent ASIC circuits. Kuon and Rose [3] examined these differences and demonstrated that the performance, area, and power gaps between FPGAs and ASICs are significant considerations in evaluating reconfigurable hardware.

Nevertheless, FPGA architectures have continued to improve through the integration of specialized hardware resources. Embedded memory blocks reduce dependence on external memory for suitable data-storage operations, while dedicated arithmetic units support computationally intensive tasks. High-speed input/output interfaces also enable FPGAs to communicate with external processors, sensors, and peripheral devices. These architectural developments make modern FPGAs more capable of supporting complete digital systems rather than merely implementing isolated logic circuits.

3.3. Reconfigurable computing as a design paradigm

Reconfigurable computing extends FPGA functionality by treating hardware organization as something that can be adapted to application requirements. Instead of relying exclusively on fixed processor instructions, a reconfigurable platform can implement specialized hardware structures for particular computational tasks. Compton and Hauck [2] described the broader principles of reconfigurable computing and examined the interaction between reconfigurable hardware, software systems, and supporting architectures. This approach is particularly relevant when applications require a combination of computational flexibility and specialized processing performance.

4. FPGA PROGRAMMING AND DESIGN METHODOLOGIES

4.1. Hardware description languages and RTL design

Traditional FPGA development commonly uses HDLs, particularly VHDL and Verilog. These languages describe the structure and behavior of digital circuits, including concurrent processes, signal

assignments, and sequential logic. Designers can specify precise hardware behavior and control implementation details through register-transfer-level (RTL) descriptions. This level of control is valuable when timing, resource utilization, and circuit architecture must be carefully optimized.

However, RTL development requires designers to think in terms of hardware operations rather than ordinary sequential software instructions. A software algorithm must often be reorganized into concurrent processing stages, memory structures, and control logic before it can be implemented efficiently. The designer must also consider clock domains, timing constraints, and signal dependencies during implementation. These requirements can increase development time and make complex FPGA projects difficult to maintain.

4.2. High-level synthesis

High-level synthesis (HLS) attempts to reduce this complexity by translating algorithmic descriptions written in languages such as C or C++ into hardware implementations. Instead of describing every register and logic connection manually, developers specify computational behavior at a higher level of abstraction. The HLS tool then performs operations such as scheduling, resource allocation, and hardware generation. This approach can make FPGA development more accessible to software-oriented engineers while reducing the effort required for initial implementation [4].

HLS does not automatically guarantee an efficient hardware design. The resulting circuit depends on the source-code structure, compiler capabilities, memory-access patterns, and optimization directives. For example, loop unrolling and pipelining can increase parallelism, but they may also increase logic utilization and resource demand. Nane *et al.* [4] evaluated several FPGA HLS tools and highlighted the importance of comparing both performance and resource utilization when assessing their effectiveness.

4.3. Comparing FPGA programming approaches

The main programming approaches differ in their abstraction levels, implementation control, and development requirements. Table 1 summarizes their principal characteristics and illustrates why no single methodology is optimal for every FPGA application. The comparison also shows that increasing abstraction can improve accessibility, but it does not remove the need to understand hardware behavior. In practice, designers may combine high-level descriptions with lower-level optimization to balance development productivity and implementation quality.

Table 1. Comparison of FPGA programming and design approaches

Design approach	Abstraction level	Main advantages	Main limitations
VHDL/Verilog RTL	Low-level hardware description	Precise control over timing and resources	Requires specialized hardware knowledge
High-level synthesis	Algorithmic or behavioral	Faster development and improved accessibility	Results depend on code structure and tool quality
IP-based integration	Functional hardware modules	Reuse of verified components and faster integration	Compatibility and interface constraints
Hardware/software co-design	System level	Combines processor flexibility and FPGA acceleration	Requires careful partitioning and communication design

The comparison in Table 1 indicates that FPGA programming has progressively moved toward higher abstraction levels while retaining the importance of hardware-aware optimization. RTL remains valuable when precise control over circuit implementation is essential, whereas HLS can accelerate development for suitable algorithmic workloads. IP-based integration provides another practical route by allowing designers to reuse existing functional components. Hardware/software co-design complements these methods by assigning tasks to processors or FPGA logic according to their computational and timing requirements.

5. FPGA-BASED HARDWARE ACCELERATION

5.1. Parallel processing and application-specific hardware

Hardware acceleration is one of the most important motivations for using FPGAs in modern digital systems. Unlike a conventional processor that executes instructions through a general-purpose architecture, an FPGA can implement a datapath specifically designed for a target workload. Multiple processing stages can operate concurrently, and pipelining can improve throughput when data are available continuously. These characteristics are useful in applications such as digital signal processing, image processing, data compression, and communication systems.

The benefits of FPGA acceleration depend strongly on the nature of the workload. Applications with substantial data-level parallelism, predictable computation patterns, and limited branching are often suitable for customized hardware implementation. Conversely, workloads with irregular memory access, frequent control-flow changes, or limited parallelism may be less straightforward to accelerate. Therefore, selecting an appropriate application is as important as choosing the FPGA device itself.

5.2. Integration with embedded and computing systems

FPGAs can operate as independent processing devices or as accelerators connected to general-purpose processors. In embedded systems, this arrangement allows time-critical functions to be implemented in programmable logic while software manages system coordination and less demanding operations. Such partitioning can improve response time and provide predictable execution for selected tasks. It also supports the development of flexible platforms in which hardware functionality can be modified as system requirements change.

A prominent example of large-scale FPGA acceleration was demonstrated by Putnam *et al.* [5], who developed a reconfigurable fabric for accelerating data-center services. Their work integrated FPGAs into a production computing environment and demonstrated the use of reconfigurable hardware to accelerate the ranking component of a web search service. This deployment illustrates that FPGA acceleration is not restricted to laboratory prototypes or small embedded applications. It also demonstrates the importance of system-level integration, including communication interfaces, reliability, and coordination between hardware and software.

5.3. Challenges in hardware acceleration

Although FPGA acceleration offers substantial opportunities, several challenges remain. Data movement between external memory, processors, and programmable logic can become a performance bottleneck when communication overhead exceeds the time saved through parallel computation. Resource limitations may also restrict the size and complexity of an accelerator that can be implemented on a particular device. In addition, power consumption, thermal constraints, and timing closure must be considered when developing high-performance designs.

The development process also requires careful verification and optimization. A functionally correct hardware design may still fail to meet timing requirements or may consume more logic resources than expected. HLS tools can reduce some implementation effort, but developers must still understand the generated architecture and evaluate its performance. These challenges suggest that effective FPGA acceleration requires coordinated optimization across algorithms, architecture, programming tools, and system interfaces.

6. RESEARCH CHALLENGES AND FUTURE DIRECTIONS

The first major research challenge concerns improving FPGA design productivity without sacrificing implementation quality. Higher-level programming environments can reduce the amount of manual hardware description, but abstraction alone cannot guarantee optimal circuits. Future development therefore requires more effective compilers, automated design-space exploration, and improved feedback between software descriptions and hardware implementation results. Such improvements could help bridge the gap between the accessibility of software programming and the precision of hardware design.

A second challenge involves reducing the performance and energy overhead associated with programmable architectures. Improvements in logic density and specialized processing resources are important, but routing efficiency and memory organization also strongly influence system performance. Research into optimized interconnect structures, memory hierarchies, and heterogeneous integration may help address these limitations. The continuing development of FPGA technology should consequently consider the entire computing system rather than focusing exclusively on the number of configurable logic elements.

A third direction concerns the integration of FPGA accelerators into heterogeneous platforms. Combining processors, GPUs, and FPGAs can provide a wider range of implementation choices for applications with different computational characteristics. However, effective heterogeneous computing requires suitable task partitioning, communication mechanisms, programming models, and runtime management. The successful use of FPGA-based acceleration in data centers demonstrates the potential of this direction, while also highlighting the engineering effort required for reliable deployment [5].

7. CONCLUSION

FPGAs have developed into versatile platforms that bridge the gap between general-purpose processors and application-specific hardware. Their configurable logic, programmable interconnections, and specialized resources enable designers to construct parallel architectures tailored to particular applications. At the same time, the flexibility of FPGA devices introduces performance, area, power, and programming trade-offs that must be considered during system development. The architectural evolution of FPGAs has therefore been accompanied by continuing efforts to improve their programming environments and implementation tools.

The transition from traditional RTL design toward high-level synthesis represents an important step in making FPGA technology more accessible. Nevertheless, effective use of HLS still depends on hardware awareness, careful optimization, and evaluation of the generated implementation. FPGA-based acceleration has demonstrated practical value in embedded systems and large-scale computing environments, particularly when workloads can benefit from parallel and application-specific processing. Overall, the future effectiveness of FPGA technology will depend on progress in architecture, design automation, and hardware/software integration, enabling reconfigurable platforms to become more efficient and easier to adopt.

REFERENCES

[1] K. Compton and S. Hauck, "Reconfigurable computing," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, Jun. 2002, doi: 10.1145/508352.508353.

[2] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable computing architectures," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 332–354, Mar. 2015, doi: 10.1109/JPROC.2014.2386883.

[3] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Feb. 2007, doi: 10.1109/TCAD.2006.884574.

[4] R. Nane *et al*., "A Survey and Evaluation of FPGA High-Level Synthesis Tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016, doi: 10.1109/TCAD.2015.2513673.

[5] A. Putnam *et al*., "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceedings - International Symposium on Computer Architecture*, Jun. 2014, pp. 13–24. doi: 10.1109/ISCA.2014.6853195.