IAES Journal of Electrical and Electronics Engineering
Vol. 4, No. 2, December 2017, pp. 31~35
ISSN: 2338-7432, DOI: 10.11591/jeee.v4i2.pp31-35

Reconfigurable computing for next-generation embedded systems: a review of FPGA technologies and applications

Tole Sutikno[1], Aiman Zakwan Jidin[2]
[1]Faculty of Industrial Technology, Universitas Ahmad Dahlan, Yogyakarta, Indonesia
[2]Faculty of Engineering Technology, Universiti Teknikal Malaysia Melaka, Malacca, Malaysia

Article Info

Article history:

Received month dd, yyyy
Revised month dd, yyyy
Accepted month dd, yyyy

Keywords:

Dynamic reconfiguration
Embedded systems
Field-programmable gate array
Hardware/software co-design
Real-time processing
Reconfigurable computing

ABSTRACT

The growing complexity of embedded applications has increased the demand for computing platforms that combine processing efficiency, flexibility, and predictable operation. Conventional microprocessors offer programmability, whereas application-specific integrated circuits (ASICs) provide high performance but are difficult to modify after fabrication. Reconfigurable computing, particularly through field-programmable gate arrays (FPGAs), offers an alternative by allowing hardware resources to be adapted to application requirements. This review examines developments in FPGA-based reconfigurable computing, focusing on architectural principles, reconfiguration techniques, hardware/software co-design, programming support, and embedded applications. It discusses how configurable logic, dedicated processing resources, and runtime hardware adaptation can support signal processing, image analysis, encryption, and real-time control. The review also considers practical limitations, including design complexity, reconfiguration overhead, resource management, power consumption, and the lack of standardized software interfaces. Existing operating-system approaches demonstrate that hardware accelerators can be integrated into familiar programming environments, although system-level coordination remains challenging. The review concludes that the future value of reconfigurable embedded systems depends on improving design automation, runtime management, and software integration while preserving the performance advantages of customized hardware.

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

1. INTRODUCTION

Embedded systems are increasingly expected to perform computationally demanding tasks while operating within strict limitations on power, cost, physical size, and response time. Applications such as industrial automation, digital communication, image processing, robotics, and intelligent monitoring require computing platforms that can adapt to different workloads without sacrificing reliability. Conventional microprocessors provide a flexible programming environment, but their general-purpose architecture may not efficiently support every computational task. Application-specific integrated circuits (ASICs), in contrast, can deliver high performance but generally lack the flexibility needed when system requirements change after fabrication.

Reconfigurable computing provides a compromise between these two approaches by allowing hardware functionality to be modified after manufacturing. Field-programmable gate arrays (FPGAs) are among the most established platforms for this paradigm because they combine configurable logic, programmable interconnections, memory resources, and dedicated processing blocks. These resources allow designers to construct hardware architectures tailored to specific applications while retaining the ability to modify their functionality. Compton and Hauck [1] identified this combination of hardware performance and software-like flexibility as a defining characteristic of reconfigurable computing.

Despite its potential, FPGA adoption in embedded systems faces several practical challenges. Hardware development requires specialized knowledge, and the implementation process can be more complicated than conventional software programming. Runtime reconfiguration also introduces additional concerns involving configuration time, resource allocation, task scheduling, and communication between hardware and software components. This review therefore addresses three central questions: how FPGA architectures support reconfigurable embedded computing, how design and system-level techniques improve the usability of these platforms, and which application domains benefit most from hardware adaptability.

2. REVIEW SCOPE AND METHOD

This article presents a narrative review of representative research on FPGA-based reconfigurable computing published no later than 2016. The selected literature covers foundational architectural studies, hardware/software integration, operating-system support, runtime scheduling, and high-level design methodologies. Particular attention is given to publications that provide identifiable architectural contributions or demonstrate practical approaches to embedded implementation. The objective is to synthesize the relationship between FPGA technology and embedded-system requirements rather than to conduct a quantitative comparison of individual commercial devices.

3. FPGA ARCHITECTURES FOR RECONFIGURABLE COMPUTING

3.1. Fundamental FPGA organization

An FPGA is built from configurable logic resources connected through programmable routing structures. Its principal components typically include lookup tables, flip-flops, configurable logic blocks, input/output interfaces, and programmable interconnections. Modern devices may also incorporate embedded memory, digital signal processing units, clock-management resources, and high-speed communication interfaces. This combination enables designers to implement both control-oriented circuits and computationally intensive datapaths on the same device.

The architectural flexibility of an FPGA distinguishes it from a conventional fixed-function processor. Designers can configure multiple logic elements to operate concurrently, creating parallel processing structures that match the requirements of a particular application. Dedicated memory and arithmetic resources can further reduce the need to construct every function from general-purpose logic. However, programmable routing and configuration circuitry introduce overhead compared with custom ASIC implementations, affecting area, power consumption, and operating frequency [2].

3.2. Static and dynamic reconfiguration

Static reconfiguration refers to configuring an FPGA before the application begins execution. Under this approach, the hardware architecture remains unchanged during normal operation, although the device can be reprogrammed when the system is restarted or taken offline. This method is relatively straightforward and is suitable for embedded applications with stable processing requirements. Its principal limitation is that the hardware cannot readily adapt to changes in workload while the application is running.

Dynamic reconfiguration allows hardware functionality to change during system operation. Depending on the device and design methodology, an entire FPGA may be reconfigured or only a selected region may be modified while the remaining logic continues operating. This capability can allow different hardware accelerators to share limited FPGA resources at different times. Nevertheless, reconfiguration introduces latency and requires careful management of configuration data, interfaces, and the state of active computations. Partial and runtime reconfiguration can be particularly useful when an embedded platform must support several functions but cannot accommodate all of them simultaneously. A communication device, for example, may use different processing configurations according to its current operating mode. Similarly, an embedded vision platform may allocate hardware resources to different image-processing stages as system requirements change. The benefits of this approach depend on whether the flexibility gained through resource reuse outweighs the time and energy required for reconfiguration.

3.3. Fine-grained and coarse-grained architectures

FPGA fabrics are generally associated with fine-grained reconfiguration because individual logic elements can be configured to implement relatively small digital functions. This provides considerable flexibility but can introduce routing and configuration overhead. Coarse-grained reconfigurable architectures instead use larger processing elements, such as arithmetic units or specialized functional blocks, which can be configured to perform broader operations. These architectures may reduce the complexity of mapping computational workloads onto hardware, although they provide less flexibility at the individual logic-gate level.

The distinction is important when selecting a platform for embedded applications. Fine-grained FPGAs are suitable when the required circuit structure is highly specialized or when detailed control over hardware implementation is essential. Coarse-grained approaches can be attractive for applications involving repeated arithmetic and data-processing operations. In practice, modern FPGA devices increasingly combine general configurable logic with dedicated coarse-grained resources, creating heterogeneous architectures that support a wider range of workloads.

4. HARDWARE/SOFTWARE CO-DESIGN AND SYSTEM SUPPORT

4.1. Hardware/software partitioning

Hardware/software co-design is central to the practical use of reconfigurable computing. Rather than implementing an entire application in programmable logic, developers can assign computationally intensive or timing-critical functions to FPGA hardware while retaining application management and less demanding tasks in software. This division allows the system to combine the flexibility of a processor with the parallel execution capabilities of customized hardware. The quality of the implementation depends on selecting an appropriate partition between the two domains.

A useful partitioning strategy must consider computational complexity, communication overhead, memory access, and real-time constraints. Moving a function into hardware does not necessarily improve overall performance if communication between the processor and accelerator becomes the dominant bottleneck. Similarly, an accelerator that consumes excessive logic resources may prevent other essential functions from being implemented on the same device. System-level evaluation is therefore necessary to determine whether hardware acceleration provides a meaningful advantage.

4.2. Operating-system support for reconfigurable systems

Operating-system support can make FPGA-based systems easier to program and manage. Without a suitable software framework, hardware accelerators may appear as isolated components that require application-specific interfaces and manual coordination. A reconfigurable operating environment can provide abstractions for hardware tasks, communication, synchronization, and resource management. Eckert *et al.* [3] reviewed operating-system concepts for reconfigurable computing and identified recurring approaches to integrating hardware functionality into software environments.

ReconOS is an important example of this integration strategy. It provides a unified multithreaded programming model in which hardware and software threads can interact through familiar operating-system mechanisms, including synchronization and message communication [4]. This approach allows developers to treat hardware accelerators as components of a larger concurrent application rather than as independent circuits. Such integration is particularly relevant to embedded systems, where hardware and software must cooperate to meet application-level timing and functional requirements.

4.3. Runtime scheduling and resource management

Runtime resource management becomes increasingly important when multiple hardware tasks compete for limited FPGA resources. The system must decide which tasks should execute in hardware, when reconfiguration should occur, and how available resources should be allocated. Scheduling policies must also consider task deadlines, configuration latency, and the consequences of interrupting an active computation. These decisions are more complex than ordinary processor scheduling because hardware tasks may require substantial physical resources and cannot always be interrupted in the same way as software threads.

Research on preemptive hardware multitasking in ReconOS explored mechanisms for managing hardware tasks during execution [5]. This line of work demonstrates that runtime management is not merely a software concern, since scheduling decisions directly affect the configuration and availability of physical hardware resources. Reliable resource management is therefore essential when reconfigurable devices are used in embedded systems with competing tasks. The challenge becomes especially significant in applications requiring predictable execution and continuous system availability.

5. PROGRAMMING AND DESIGN METHODOLOGIES

5.1. Traditional hardware description

Traditional FPGA development relies on hardware description languages such as VHDL and Verilog. These languages allow designers to describe concurrent processes, sequential behavior, signal relationships, and hardware structures. Register-transfer-level design provides detailed control over circuit implementation, making it useful when precise timing and resource optimization are required. However, the approach requires developers to understand hardware behavior and the consequences of their design decisions.

The complexity of low-level hardware development can limit FPGA adoption among software-oriented engineers. An algorithm that is straightforward to express in a conventional programming language may require substantial restructuring before it can be implemented efficiently as a hardware datapath. Developers must consider concurrency, memory organization, clock domains, and timing constraints. This creates a need for design methodologies that reduce implementation effort without hiding important hardware characteristics.

5.2. High-level synthesis

High-level synthesis (HLS) addresses this challenge by translating algorithmic descriptions into hardware implementations. Developers can describe computational behavior using higher-level languages, while synthesis tools perform operations such as scheduling, resource allocation, and hardware generation. This abstraction can improve productivity and make FPGA development more accessible to engineers without extensive RTL experience. Nane *et al.* [6] evaluated FPGA HLS tools and highlighted the importance of examining both implementation performance and resource utilization.

HLS does not eliminate the need for hardware-aware design. The structure of the source code, memory-access patterns, loop organization, and optimization directives can significantly affect the generated circuit. A design that is convenient to write may produce inefficient hardware if the synthesis tool cannot identify suitable parallelism or data reuse. Consequently, effective FPGA development requires a balance between high-level abstraction and detailed evaluation of the resulting hardware.

6. APPLICATIONS IN EMBEDDED SYSTEMS

Reconfigurable computing has been investigated across several application domains where adaptability and specialized processing are valuable. These include digital signal processing, image and video analysis, communication systems, encryption, and embedded control. The operating-system survey by Eckert *et al.* [3] identifies examples such as video tracking, image processing, signal processing, and encryption among the applications supported by reconfigurable computing systems. Table 1 summarizes representative application areas and explains how FPGA characteristics can address their requirements.

As shown in Table 1, FPGA-based reconfigurable computing is relevant to applications with different computational and operational requirements. Signal processing and image analysis benefit from parallel datapaths, while software-defined radio can use hardware adaptability to support changing communication functions. Cryptographic systems can use specialized logic to accelerate repeated operations, although security management remains essential. Industrial control and robotics place additional emphasis on predictable timing and reliable coordination between hardware and software.

Table 1. Representative embedded applications of reconfigurable computing

Application domain	Typical processing requirements	Potential FPGA contribution	Main implementation concern
Digital signal processing	High-throughput filtering and transforms	Parallel datapaths and dedicated arithmetic resources	Data throughput and precision
Image and video processing	Repeated pixel operations and feature extraction	Pipelined processing and parallel execution	Memory bandwidth and latency
Software-defined radio	Adaptable modulation and signal-processing functions	Reconfigurable communication datapaths	Reconfiguration delay and interface timing
Cryptography	Repeated arithmetic and logical transformations	Specialized hardware accelerators	Security and key management
Industrial control	Predictable response and concurrent monitoring	Dedicated real-time processing logic	Timing determinism and reliability
Robotics and embedded vision	Sensor processing and rapid perception	Hardware acceleration near data sources	Resource limits and system integration

6.1. Signal and image processing

Signal and image processing are natural application areas for FPGA technology because many operations involve repeated computations over streams of input data. Filtering, transforms, image enhancement, and feature extraction can often be organized into pipelined processing stages. FPGA architectures allow these stages to operate concurrently, potentially improving throughput and reducing processing delays. Embedded systems can therefore use programmable logic to process sensor data without relying entirely on a general-purpose processor.

6.2. Communication and software-defined radio

Communication systems often need to support different signal-processing configurations and operating modes. Software-defined radio provides an example in which selected communication functions

can be implemented through programmable hardware and software. FPGA reconfigurability allows parts of the processing architecture to be adapted as communication requirements change. However, the system must manage configuration time, synchronization, and data continuity to avoid disrupting communication.

6.3. Industrial and robotic systems

Industrial automation and robotics require coordination between sensing, computation, and control. FPGA hardware can implement time-critical functions such as signal acquisition, filtering, motor-control logic, and sensor processing. A processor can then handle supervisory functions, communication, and application-level decision-making. This division can improve responsiveness, although the final system must be verified to ensure that hardware/software interactions satisfy the required timing constraints.

7. RESEARCH CHALLENGES AND FUTURE DIRECTIONS

The first major challenge is improving design productivity while retaining the implementation control that makes FPGA technology valuable. Low-level hardware description provides flexibility but requires specialist knowledge, whereas HLS improves accessibility but may not always generate optimal circuits. Future design environments should provide better performance estimation, automated optimization, and clearer feedback about resource utilization. Such improvements would help developers make informed architectural decisions earlier in the design process.

The second challenge concerns runtime reconfiguration and resource management. Dynamic hardware adaptation can improve resource utilization, but configuration latency and task coordination may reduce its benefits. Embedded systems also need dependable scheduling mechanisms that account for deadlines, power constraints, and the availability of hardware resources. More effective operating-system abstractions and runtime frameworks could make these capabilities easier to use in practical applications.

A third challenge involves power efficiency, security, and reliability. Reconfigurable hardware can provide application-specific processing, but its total energy consumption depends on logic activity, memory access, communication, and configuration operations. The configuration process itself also creates security concerns because unauthorized modification of hardware functionality could compromise system behavior. Future embedded platforms must therefore consider secure configuration, fault handling, and system-level verification as integral parts of reconfigurable computing.

8. CONCLUSION

Reconfigurable computing offers an important approach to developing embedded systems that require both computational efficiency and hardware flexibility. FPGA architectures allow designers to implement application-specific processing structures while retaining the ability to modify hardware functionality after manufacturing. Static and dynamic reconfiguration, hardware/software co-design, and operating-system integration provide complementary mechanisms for using these capabilities in practical systems. Research published through 2016 demonstrates their relevance to signal processing, image and video analysis, communication, encryption, and real-time embedded applications.

Nevertheless, the benefits of reconfigurable computing depend on overcoming challenges involving design complexity, runtime scheduling, resource allocation, power consumption, and system integration. HLS and operating-system support have improved accessibility, but they do not remove the need for hardware-aware optimization and verification. Future progress will depend on more effective design automation, reliable runtime management, and standardized software interfaces. By addressing these issues together, FPGA-based platforms can provide a flexible foundation for increasingly demanding embedded applications.

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] 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.

[3] M. Eckert, D. Meyer, J. Haase, and B. Klauer, "Operating System Concepts for Reconfigurable Computing: Review and Survey," *International Journal of Reconfigurable Computing*, vol. 2016, pp. 1–11, 2016, doi: 10.1155/2016/2478907.

[4] A. Agne *et al.*, "ReconOS: An operating system approach for reconfigurable computing," *IEEE Micro*, vol. 34, no. 1, pp. 60–71, Jan. 2014, doi: 10.1109/MM.2013.110.

[5] M. Happe, A. Traber, and A. Keller, "Preemptive hardware multitasking in reconOS," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9040, 2015, pp. 79–90. doi: 10.1007/978-3-319-16214-0_7.

[6] 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.