Deep Science Publishing
https://doi.org/10.70593/978-81-989050-5-5_11

DeepScience
Open Access Books

Chapter 11: Reprogrammable logic and FPGA-driven adaptability in next-generation systems

11.1. Introduction

In the past twenty years, there was increasing focus towards the development of post-Moore generations of systems. These early Adaptive Architectures were mainly driven by the necessity to move from the typical static processing model towards a new more dynamic paradigm, able to appropriately deal with in-memory processing, heterogeneous approaches, and novel intermediate levels of processing like processing at the edge. During this exploration, an increasing number of possible innovative solution directions emerged. These potential solutions partner along an explicitly interdisciplinary interaction front, by combining architectures and systems, methods and algorithms for the design and its development, and the algorithms and frameworks that guide applications' impact along their life-cycle (Gubbi et al., 2013; Al-Fuqaha et al., 2015; Kumar & Mallick, 2018).

At the same time, the strong experimental definition of the proposed solutions and the acceleration of their deployment towards more impactful applications got a new spin, being able to correctly address the current digitally-enabled Society 5.0 topics. With this broad outlook opening up correlations and dependencies like never before, "completely digital" and more analogic driven paradigms are ultimately converging towards novel intelligence-enabled physical entities. A process that reflects the co-evolution of technology, human, and natural ecosystems while nurturing the inclusive transformation of City 5.0 and Industry 5.0 goals. Within this perspective, the different technological enablers that let Information and Communication technologies become pivotal in the introduction and reinforcement of sharp a technological divide across digital, human, and natural ecosystems opened up an important question: when we talk about Intelligent Services, are we also facilitating human growth development, as the wise use of these new digital services should do (Lee & Lee, 2015; Niyato et al., 2019; Kumar & Mallick, 2018).

The main goal of this chapter is to briefly review and highlight the proposed evolutions in the topic areas, with the aim of sparking discussion and new development ideas to continue the exploration of the Architectural and System Alternatives, Algorithmic-Method and Application Opportunities that correctly address the interdependencies between Autonomous Emotive Machines, Neuromorphic Processing Paradigms, and Interfacing at the Edge, driving adoption and correlation towards improved Service Experiences and enhanced User Experience throughout their life cycle.

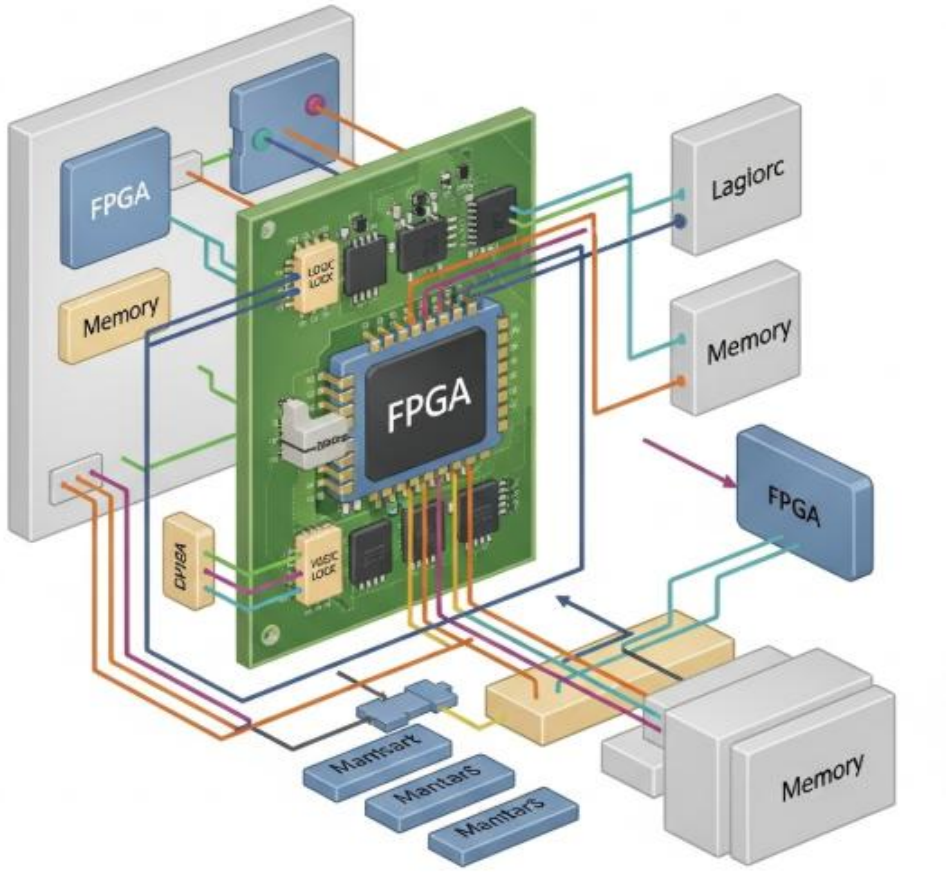

Fig 11.1: Reprogrammable Logic and FPGA-Driven

11.1.1. Background and Significance

Over the last three decades, the workhorse of ubiquitous computation has shifted from the CPU to the GPU because of the acute need for high-performance, data parallel computation for artificial intelligence, graphics, media processing, scientific computing, and machine learning. Throughout, the design and organization of compute resources has focused on finding the right mix and enhancing much-needed specialization, and to

do so; the industry has embraced an innovative approach to modularity that involves building on top of a flexible chip architecture such as the graphic processor, modifying it for special needs, and then fabricating at extreme volumes to amortize the design and manufacturing costs. At the same time, the relentless progress in large systems – and more recently, cloud, scale computing – has raised the need to integrate features and accelerate capabilities that are but too custom for the microprocessor; notably, the acceleration of compute workloads that are elastic with respect to architecture choice and are not well-matched to the core processing engine. This desire for enhancing and evolving the modularity theme for the next-generation microprocessor has manifested itself in the integration of programmable elements alongside fixed-function units – initial attempts to seamlessly integrate FPGAs with CPUs; the development of highly specialized processing engines; and, there is a growing interest in the industry around using custom-enabled resources – such as domain-specialized processors, custom accelerators, or hardware accelerators – for aspects of cloud or hyperscale data centers, and their workloads; including web search, targeted advertising, multimedia and social networking, artificial intelligence, big data analytics, and machine learning. These strong desires for specialization – driven by the performance, efficiency, and flexibility advantages of custom computation – and technology advances that business ASICs in modest combinations, push custom devices further up the performance curve and bring the costs of handcrafted but modifiable accelerators down. These trends have raised the issues of ASIC and methods to partition workloads – the latter the core of adaptable hardware systems and custom computing devices.

11.2. Overview of Reprogrammable Logic

FPGAs have demonstrated considerable technical and conceptual flexibility, continuing to evolve rapidly beyond traditional use as reprogrammable replacement chips for logic functions, or as prototyping platforms for application-specific integrated circuits, to become the most space- and power-efficient versions of the very high integration density and low design-cycle time computers which they are – the embedded microprocessors and DSPs of the next generation of modular reconfigurable imaging systems. Reconfigurability is now recognized as a fundamental enabler of many highly-specified applications, as an essential requirement for high-level system functionality in multisensory situational awareness.

The term reprogrammable logic refers to any array of logic components that can be electrically “programmed,” that is, switched from one logic-function implementation to another, as well as to any system whose functionality can be changed in that manner. Solid-state integrated-addressed programmable logic using fuseablity or antifuse technology first appeared in the form of PROMs in the late 1970s as gate controllers for

packet-switching networks. The first commercial field-programmable gate array was developed in 1985; it began a revolution in application-specific digital system design by incorporating reprogrammable SRAM-based configurable function blocks with programmable interconnects.

11.2.1. Definition and Fundamentals

The term programmable logic refers to some amount of circuit functionality, for example, a flip-flop or look-up table, that can be configured and reconfigured as needed to accomplish a desired task. Programmable logic has enabled an incredible expansion of capabilities in modern electronic systems, from very simple glue logic functions to complex system-on-a-chip elements. The ability to construct circuits that can be changed in the field has allowed increasingly advanced designs, with functions that were originally fixed in hardware being replaced by efficiently programmed functions. Solutions that used to require custom-designed application-specific integrated circuits are now accomplished with one-time programmable logic devices, or even reprogrammable logic device implementations.

Reprogrammable logic devices allow the intended behavior of an element to be overwritten in the field as the system operates. On the chip level, this functionality is provided by a structure of general circuit resource types, such as memories and processing blocks, all interconnected by programmable routing resources. Across the chip level, routing resources allow each type of device to be interconnected, forming a multi-device heterogeneous chip capable of supporting complex, computation-intensive applications. Because configurations are typically stored in memory elements on chip, these resources span fast and slow, large and small, temporary and permanent memory types, and high and low bandwidth communication channels.

11.2.2. Historical Development

This chapter describes insights into FPGA-driven adaptability in modern computers. Reprogrammable logic supports diversity in system design. It overcomes difficulties due to increasing cost and increasing risk from economies of scale and increasing risk to just a few dominant markets. Specific architectures should be reflected in newly designed systems without needing to resort to a more general distinct architecture. Also, the system must offer greater adaptability. So, using portable IP in reprogrammable logic is an answer to achieving greater adaptability and diversity.

In order to describe the wave of the future, a few facts about the recent past may prove useful. Reprogrammable logic has had an interesting interaction with diverse sections of

the electronics and computer industry. In the mid-1980s, electronic types "circuit designers" were drawn to programmable logic in relatively large numbers. Circuit design engineers, not limited to one section or other, in search of a better, quicker, less costly, more efficient approach to custom applications. A roaring success! In the mid-1980s, systems engineers, applying computers to computers, have become, and remained, the driving force for engineering and involving computers for leading edge applications. Logic entered their radar screens and very rapidly became a definite must on their products.

They were intrigued by the promise of the use of custom logic to help these devices to offer greater functioning flexibility. But as the dust settled, they turned their attention elsewhere, discouraged by the pain of interfacing a rather dumb block into a rather smart, demanding and concise digital computer. Through most of the 90s, reprogrammable logic was turned by a fulcrum technology. Data implemented generic chips for specific application systems. Overhead cables connected decoder chips to chips. System experience while of huss, most engineers designing systems were warned to use carefully structured languages, lest simulation drive them out.

11.3. Field-Programmable Gate Arrays (FPGAs)

FPGAs provide hardware programmability capability in order to be able to adapt the hardware functionalities of the circuits that comprise the system without requiring any modification of their underlying physical structure. With this capability, the target application may be changed after manufacturing and even during the life cycle of the application. This capability is very useful in fast-changing environments. The software programmability capability of microprocessors is achieved by a general-purpose instruction set, and so the software is, in general, executed faster in hardware than in software, but both concepts are capable of delivering the same functionalities. In some cases, the power consumed by hardware circuits implementing the requested function is smaller than that consumed by a microprocessor executing that logic functionality, obtaining logic equivalent results.

But specifically relating to microprocessors and FPGAs, there are some key differences. The software program is written in a high-level programming language and is compiled, linking and assembling before being executed. For FPGAs, high-level programming models are available too and can be used to support the hardware programming capability. The key difference relies on how the hardware is programmed. In microprocessors, the execution of instruction codes will automatically allow modifying the internal architecture connections to control the data path and define the data stored in registers and the functionalities of any existing logic cell in a pure architecture/software complexity paradigm. FPGAs, in turn, are programmed because

their architecture has been modified in a way that provides that capability. Currently, three different techniques are used in order to configure the FPGA architecture: Static RAM-based FPGAs, Flash-based FPGAs, and Antifuse-based FPGAs.

FPGAs are very good for rapid prototyping because they allow quick modifications of the architecture, which is not possible with ASICs until the next silicon version is manufactured. But it is necessary to speed up product development time and minimize product development cost, since they are more expensive and slower in volume production than ASICs.

11.3.1. Architecture of FPGAs

The architecture of a field-programmable gate array (FPGA) or a Complex PLD (CPLD) can be thought of as being composed of a set of resources and a set of switches or interconnections that are used to access these resources. The resources consist of programmable logic blocks (PLBs) connected to a programmable interconnection network that is used to distribute signals among the logic blocks and I/O pads, which connect the internal logic and routing network to the external logic and memory devices. Both PLBs and I/Os are configured or programmed to perform a specific logic function during design time, while the general function of the remaining resources is determined during run time by configuring the interconnection network.

The elements of an FPGA interconnection network, which interconnect the inputs and outputs of various components in a design, are programmed during run time. If a given interconnection resource does not exist in a general-purpose FPGA, the interconnections between different circuit elements must be done by adding wires or connecting pins. The advantage of eliminating interconnection resources from FPGAs is that the circuit can be packed tightly, leading to a smaller area size. The disadvantage is that the chip must be redone when a change is made in the interconnections. In the case of FPGAs, a part of the performance advantage of hardwired connections is recovered via the use of faster programmable interconnections to connect the different circuit components. Each FPGA device has the same type of switch at each location in the device, and the architecture of a device may be characterized by the logic inputs to the switch, including the number of inputs, and the flexibility of each switch, including the number of connected wires, their widths, and so on.

To include the function of the logic blocks and I/Os along with the interconnection network, a runtime or configuration or switching technology is needed. A configuration technology describes how the interconnection or switching fabric is programmed and switched to connect different signal lines while the design and implementation of architecture defines the kind of logic circuitry that is connected by programmable

interconnections. Note that the internal logic blocks may have multiple configurations, but an individual logic block is configured to perform only one specific function at any instant during operation.

11.3.2. Programming Techniques

FPGAs are inherently different from the conventional CPUs and GPUs in terms of their programming abstractions and characteristics. There are a myriad of programming techniques available to utilize the reconfigurable fabrics. In fact, this myriad of choices is a double-edged sword.

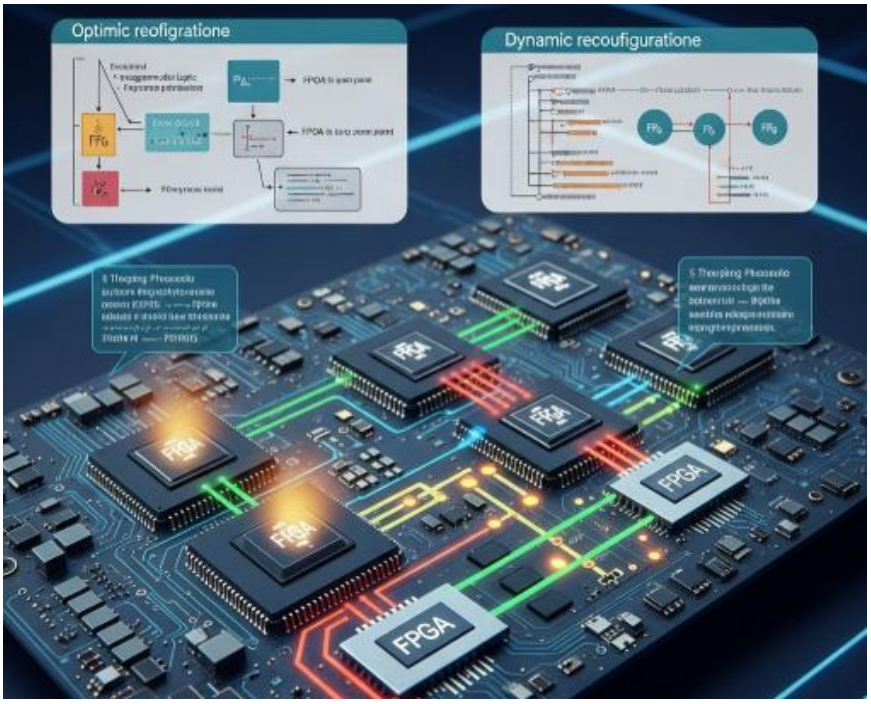

Fig 11.2: Programming Techniques

On one hand, the technology enables them to be capable of very diverse applications – not only the building blocks of compute accelerators and coprocessors, but also MPSoCs, application (or domain)-specific processors, application specific instructions and processing elements, simulators / emulators for hardware / software codesign, communication accelerators, soft and hard real-time control implementations, heterogeneous systems, low-energy designs, and adaptive systems. This application diversity is only possible because of the gradient of choices for their programming, designs, and implementations, which range from the higher end of programming with C and C-like compilation driven transforms and synthesis to the lower end device-level

netlist descriptions and rapid circuits sandwiched between hardware accelerator macros and coarse-grained architectures.

A lot of programming models, methodologies, optimization flows, and languages have been developed and borrowed from other processing modules to aid the various kinds of programmers. Different vendors provide different levels of abstraction for a single or a family or support for a variety of APIs for different computing needs in their FPGAs.

11.3.3. Comparison with ASICs

Unlike an ASIC, whose functionality is designed and fixed for mass production and which must be fabricated for a fixed cost, an FPGA is field-programmable, and both the design and the process to implement the design are flexible at low cost. FPGA-based adaptability is often associated with product lifecycle extension. Most designs done using FPGAs performed poorly in comparison to those done using standard ASIC processes. Nevertheless, a wide variety of products, and some that are financially large, have suddenly become feasible because no ASIC process exists for them. Even if functions are to be done in volume of 100,000 or more assuming Moore's law, the fact is that new adaptation possibilities opened up by flexibility are very appealing. Rapid turns in design time and technology help offset all of the disadvantages of FPGAs in high volume. FPGAs can incorporate new improvements in performance and function much faster than hard-wired ASIC components which are micro architected. In these circumstances, even though their performance is limited, FPGAs can be successfully employed in clusters that provide sufficient overall performance at a fraction of the time to market of standard custom implementations. In addition, the technology is likely to allow a demand-mode pricing similar to that of other commodity ICs, unlike the very costly special ASIC tools and processes. Perhaps one-third of contemporary designs being implemented with FPGAs are for specific, non-bulk products such as specialized testing functions, network interfaces, and a variety of custom functions embedded in data path systems unable to support any modification. Many products, like workstations and telecommunications systems depending on ASICs to fasten processors, are starting to use FPGAs for the very high-speed interchip commutation previously requiring expensive, custom-machined multi-chip modules.

11.4. Adaptability in Modern Systems

Adaptability is not a luxury but a necessity in future systems. Computing and Communication technologies are being integrated at all levels. Embedded and cyber-physical systems perform real-time monitoring and control functions for most systems ranging from industrial applications, smart grids, automotive control to health

monitoring. These systems acquire and process data using various sensors and generate control commands to interact with the physical world. Such control functions are expected to sustain physical signals which are highly irregular and unpredictable in nature. These signals can originate from multiple modalities and are of low amplitude and fast-varying temporal characteristics. In addition, the duration and frequency of such control tasks cannot be determined a priori. This presents significant challenges on the capability requirement of Signal-aware Cyber-Physical systems especially in the context of design parameters such as form-factor, weight, power budget and overheating.

Discrete-event systems such as communication and control, deadline-driven real-time systems that are common in modern embedded applications are becoming more ubiquitous. These discrete-event systems and the temporal characteristics of signals that govern them are becoming ever more probabilistic in nature due to increased reliability and safety requirements coupled with user-driven and environment-related stochastic uncertainties. Increased demand for reprogrammable logic that provides task-driven flexibility and adaptability without compromising real-time performance are seen in various domains: From embedded applications in computing, communication and control systems to high-end medical instruments such as MR scanners, imaging and processing applications in Space and Intelligent Infrastructure.

11.4.1. Need for Adaptability

Emerging opportunities in the Internet-of-Things (IoT) scenario catalysts the increasing need for configuring and merging heterogeneous distributed services, including open source pools accessible through networks. Need for a flexible application scenario, where applications are easily composed or merged through appropriate services calls during their overall lifecycle can offer the desired realization of these opportunities. Future computing models, concretely IoT ecosystems, will need to define strategies to build in and offer adaptivity to their services. Building such properties in the basic service functionalities of these distributed networks will allow systems to search for and use the best service available in their environment, reutilizing efforts on composition of each routine task. Moreover, motion capabilities will be a characteristic needed to achieve performances that meet expected goals. Previous automata applications constraints on design time and with expressiveness do not meet required needs.

Services dynamics, from very abstract ones decided by automatic composition software, to others that provide more complex functionalities programmable or configurable, influenced by user decisions or some automata rules, are discussed to justify the necessity of adaptive-based programs, even for purely resource implementations. Flexibility as well as adaptivity components will be offered as part of design services. Nevertheless, low design time and need for data on the resource design process and

service pool behavior becomes important requirements in the use of distributed based adaptable solutions for the macrosystem design. This factor as well as the need to establish strictly deterministic protocols, incorporating for example access with priorities to some services, and defining metrics, even consolidate, to compare different available services, or signaling and adaptation mechanisms, to reduce residual time on demand changes will be critical for their feasibility.

11.4.2. Use Cases in Various Industries

Several use cases have emerged in recent years showing the onboarding of programmability, reprogrammability and variability in next-generation systems. However, with the introduction of smartness in vertical domains, the interest in adaptable systems has regained momentum. We focus here on several novel industrial use cases that show both the necessity but also the advantages of this concept to novel systems. From the Automotive sector, it has been shown that over-the-air firmware update services and the addition of features that were not planned in the beginning increase both the customer experience as well as the overall lifecycle of a vehicle. On the contrary, challenges have been described regarding the evolution of smart sensors driving innovation in in-vehicle network infrastructure technology. This was mainly possible because of the built-in flexibility of remote device and application management platforms. On the automotive testing side, a HIL framework for EVs has been developed that enables the possibility of choosing the component to implement or emulate. Their system demonstrates how the heterogeneity of the tools available for use can be compensated with the on-chip adaptability of prototype systems designed with FPGAs. All these solutions are based on the technological innovation that has enabled the development of smart, cross-cutting infrastructures and ecosystems. On the other hand, in Wearable and Medical devices, several solutions are being adopted. They mainly belong to two categories, security and sensing energy-efficient management. To date, medical devices have been equipped with standard solutions that use hardware-based security measures to resist unauthorized access. A proposed solution aims to eliminate the congested use, from security and latency points of view, of the serial communication channel between the sensor and the connected device using NFC. The access credentials and information for each sensor are contained in a low-power design provided with an energy source and that can be configured to enable intermittent access. The choice of this approach enables the possibilities of updates and much lower selling costs thanks to the scalable production.

11.5. Design Methodologies for FPGA-Based Systems

FPGA devices are semiconductor chips with large numbers of program configurable digital-nature circuit elements. They are used to implement hardware functions in a program configurable manner after fabrication, ideally matching each specific digital application need while going through the product life cycle. Such needs may include the ability to change the operation of the circuit over its lifetime, achieve efficiency savings in power dissipation, area, or operation speed, etc. As a first step towards the implementation of reconfigurable digital systems, we usually define the FPGA device as a digital logic cell array, where the logic cells are semiconductor programmable arrays made by logic element arrays including Multi-Input Lookup Tables, programmable storage elements, and Hard-Wired Logic Functions.

The values of the configuration parameters are adopted at the time of a design development process after fixing the final specification of the digital application. The entire process starting from design conceptualization to mask generation data level is what we defined as design methodology. The design methodology flow for FPGAs differs from the traditional software design methodologies, and from ASIC-based methodologies in many aspects. FPGA-based design methodologies are recognized to be more user-friendly and more capable for small volume application projects.

The advanced applications targeted by FPGAs are typically not the same as those for ASICs, potentially resulting in a different design methodology. The three primary areas for FPGA design methodologies are High-Level Synthesis for behavioral design, HDL for register transfer level and logic design, simulation and testing. In this chip level design process however, the selection of a device cannot be left till the final stages of integrating the netlist description into either a mask or a configuration file, otherwise the primary expected advantages of using FPGAs risk to be lost.

11.5.1. High-Level Synthesis

High-level synthesis is a CAD technology that performs logic synthesis directly from a high-level behavioral description. Strategies used during HLS include estimation of delay and area cost functions, exploration of alternate algorithm mappings, allocation of computational resources and schedule of operations in time, and modification of high-level descriptions to increase the performance or reduce the costs of implementations generated for a given target architecture. The two dominant languages for defining hardware at this level are C and C++, although there are research tools available that use other languages. HLS tools include various compilers and synthesizers. High-level design tools generate RTL designs that are functionally equivalent to C-level designs. Users of HLS tools report that design times are significantly reduced, while logic area

utilization and performance of synthesizer-generated RTL is comparable to hand-designed netlists.

C/C++, like RTL HDL, is amenable to sectioning: that is, a description may include components that are synthesized at different levels. This is advantageous because there are some parts of most designs, especially complex controller areas, for which an RTL description will be superior. SystemC is a wrapper library that allows users to generate C structures and provide operations that tightly match C++ semantics. The C/C++ tools produce Gate level netlists suitable for input to commercial design tools. These netlists can be synthesized directly for FPGAs and also are compatible with gate level place and route tools for ASICs. Because most specialized architectures, including FPGAs, generally do not support high level operations, HLS functionality will be useful for these FPGAs. One available tool that augments the C/C++ design process, in the sense of mixing different representation levels in a single design description, is a specific library.

11.5.2. Hardware Description Languages

Designing with hardware description languages (HDLs) is highly adaptable. If developers decide to code for an undesired speed or resource use, the design fails either

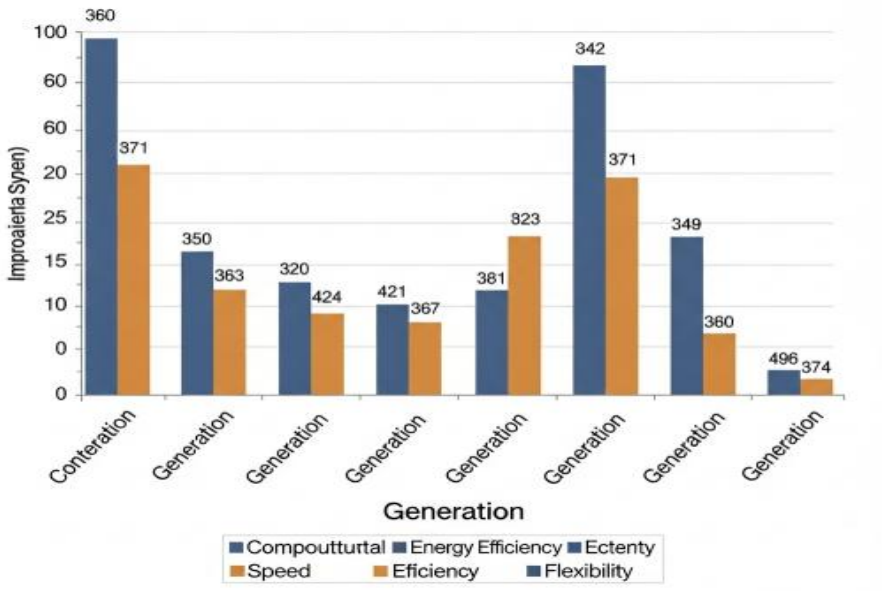

Fig 11.3: FPGA-Driven Adaptability in Next-Generation Systems

on timing analysis or testing of hardware speed. This seems easy to remedy: simply modify the code till synthesis and implementation succeed, possibly through a series of

iterations without resource or timing limits. The flexibility of design expression with HDLs encourages exploration of many more circuits, many of them tested as FPGA implementations. Because HDLs are so generic and widely used, they can quickly accommodate special features of a specific target device.

Additionally, the HDL market has evolved steadily for 40 years, encouraging design reuse, special-purpose tools, and advanced representations. Testing such constructions is a lengthy endeavor, though it is often much rewarded. Due to the ability of synthesis and implementation tools to grasp a larger range of structures developed in this semi-custom methodology, the synthesis tools were heavily updated from initially producing low-quality circuits. The reduced time and cost to get to an FPGA speed implementation encouraged the use of HDLs. Fault simulation requires some additional logic. The variant HDLs require simulation-development resources, but there are many such testers, and in FPGA context the debugging time on the FPGA is often lower than the delay time. Our industry leaders have proceeded along totally different paths but have converged into separate model variants.

11.5.3. Simulation and Testing

In order to test a design at different levels of abstraction, a variety of design simulation environments have been developed. A long-established philosophy is to utilize the same format for the design, which is normally in the form of a netlist, structure the simulator so that one can specify new levels of detail and control, translate to a more general form of internal representation, such as a functional machine configuration, and simulate in different ways for different purposes. This approach to simulation has certain advantages when developing the simulator since timing, resistance, capacitive, and other details may be modeled at whatever level required for the particular application.

What is more important is how the user can provide hooks so that the actual design can be mechanically synthesized at various levels of detail and efficiency. It is characterized by the other end applications and models provided by the particular user. Some simulators permit the timing parameters to be initialized during execution so as to convert the general simulated machine configuration into the actual design with every signal transited, but little additional overhead. Also, the more general internal organizational representation can be automatically transformed to the special case but may suffer greater overhead. The compiled approach will actually formulate the detailed timing specification, normalizing it into a functional decomposition sequence of signal transitions and process switches, allowing one to take advantage of greater loads placed on certain signals and those signals infrequently occurring. The transition sequence would lend itself to translation into a set of timing tables for use in a lookup approach.

11.6. Conclusion

Chapter summary – Reconfigurable logic devices, especially FPGAs, brought novel approaches to digital systems design. The highly distributed, flexible, and programmable nature of FPGAs allow new design paradigms and adaptations to become possible. Future trends in next-generation systems will keep emphasizing more and more this adaptability and broader inclusion in a wider range of application domains, including system and application acceleration using heterogeneous, dynamic, and FPGA-based many-core configurations together with their specific tools. To support this development, HW/SW design tools such as Hardware Development Languages, IP/Module Design, Application-Specific Optimizers, Rapid Prototyping, and Embedded Processors are becoming critical issues in order to speed up the design of special-purpose, IP-based devices. To improve such tools, attention must be paid to the specific nature of FPGAs and re-identify many of the mature design and optimization methodologies that were created many years ago for single-task, single-purpose, ASIC devices.

Since the early days of integrated circuits, adaptation to specification changes and responsiveness to new market opportunities have been idiosyncratic economies driving the re-use of special-purpose design techniques. Increasing the design flexibility and functional adaptability of integrated systems has been a constant theme in architecture design. The shifts from configuration of passive components to software programmability; from microprogramming to microcoding; from bus-based control structure to FPGAs; from nano-programming to upload-and-go; all take us along the path towards reducing the up-front cost of design by redirecting the validity of design re-use and parallelizing design effort with post-deployment adaptation. It would appear that no distinction or boundary can be drawn. The next generation of integrated systems will be optimizable in terms of design specifications, application faceting, technology balance, and implementation at deployment and beyond.

11.6.1. Future Trends

The merging of pervasive embedded systems, dynamic and ubiquitous networking, and intelligent self-management capabilities – epitomized in the vision of next-generation systems – represents a largely untapped field for reconfigurable logic applications. Such reconfigurable systems are likely to be characterized by the tradeoffs of small size, low cost, extreme reliability, and low power, while providing the required adaptability in the face of high incidence of failure modes. As for the adaptation requirements systems in that class, these usually exhibit characteristics of (a) complexity of behavior, (b) demand for highly functional components, (c) multi-level, multi-resolution structure, (d) life-

cycle evolution, (e) changing composition, (f) run-time composition/speech, and (g) need for some degree of on-site autonomy.

The increased level of design complexity and specialization of target applications has generated a need to move towards Not Only Synchronous solutions. Traditional synchronous paradigms face fundamental limitations in terms of power and flexibility in the deep submicron design space. Asynchronous design has gained renewed interest in recent years thanks to the convergence of technological, economical, and design environment factors. Flexible Asynchronous Cells technology faces the challenge of enabling the implementation of complex asynchronous modular solutions. While SLA synthesis has been established as a powerful design methodology, the effective and economical support of the technology across the lifecycle is becoming a growing design challenge. Commercial gate-level synthesis tools are still in their infancy, and designing reliable infrastructures for FAC libraries is still a critical issue. Furthermore, the adoption of flexible asynchronous design flows demands the correct balancing of the constraints and requirements posed by each of the design phases. Solutions at that level are not yet mature, and would need more work and research efforts to become accessible in a production design environment. Only with the support of such infrastructure, and with the right design methodologies, scaling-up asynchronous design will be able to provide its claimed benefits.

References

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 29(7), 1645–1660.

Lee, I., & Lee, K. (2015). The Internet of Things (IoT): Applications, investments, and challenges for enterprises. Business Horizons, 58(4), 431–440.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys & Tutorials, 17(4), 2347–2376.

Niyato, D., Kim, D. I., Mastronarde, N., & Han, Z. (2019). Wireless powered communication networks: Research directions and technological approaches. IEEE Wireless Communications, 26(6), 24–31.

Kumar, P., & Mallick, P. K. (2018). The Internet of Things: Insights into the building blocks, component interactions, and architecture layers. Procedia Computer Science, 132, 109–117.