

Chapter 2

Background and Motivation

Abstract In this chapter, we aim to present the background and the motivation of this thesis work. We will first give an overview of the Field Programmable Gate Array architecture, which is today the most widely used reconfigurable circuit. After describing its conventional structure, we will detail current trends in architectural organization. Then, we will survey the literature to see how disruptive technologies are used to propose drastic evolutions in the field. We will in particular show how dense nanowires can be used to build logic fabrics in a crossbar organization, and also how the use of carbon electronics allows the construction of interesting logic functionalities. Finally, we will try to formalize the various approaches into a hierarchical representation and compare it to the conventional structure. This representation will help to define the objectives of this work. We mainly intend to propose a digital reconfigurable circuit based on real-life disruptive technologies. This is an important point, since even if a potential technology opens the way towards new phenomena, it is fundamental to work closely with technologists and to keep in mind its feasibility from an industrial perspective. In this context, we will continuously try, in this thesis work, to take into account the technology requirements when designing a circuit.

As introduced previously, reconfigurable logic architectures are generic and highly versatile. This makes them an excellent compromise between costs, development time and performances. Suited for a wide range of application, they offer an intrinsic regularity compatible with the most advanced technological processes.

In this state of the art chapter, we will give an overview of the reconfigurable field. Hence, we will first introduce the conventional *Field Programmable Gate Array* (FPGA) architectures. From the basic FPGA scheme, we will move to the most recent evolutions and discuss the structural issues.

Subsequently, from the emerging technologies perspective in the context of computation, we will survey the nanodevices-based architectures relevant to our field and give a global comparison between the different approaches.

Finally, we will formalize our research methodology, by describing the chosen global architectural template, as well as the position of our work regarding the existing literature.

2.1 Conventional Reconfigurable Architecture Overview

Reconfigurable architectures are today leads by the Field Programmable Gate Array. We will begin this overview with the survey of the homogeneous FPGA architectural scheme and its related sizing. Then, we will see how the homogeneous architecture has been improved to increase the structural performances. Finally, we will discuss the current limits of FPGAs.

2.1.1 The Field Programmable Gate Array Architecture

In this section, we provide an overview of the standard FPGA scheme. We will start with some generalities of the structure. These generalities will deal with a short history/overview of reconfigurable circuits and will present the basements of the FPGA architecture. Then, we will detail the logic blocks architecture. Logic blocks architectures can be based on logic gates or *Look-Up Tables* (LUTs). Interconnection structures will then be detailed, and finally, sizing of the architecture will be presented.

2.1.1.1 Generalities

Field Programmable Gate Array belongs to the family of reconfigurable logic circuits. Its structure is currently the most advanced of the family.

Historically, the reconfigurability has been based on programmable diode logic. Second generation architectures used in *Programmable Array Logic* (PAL)/*Programmable Logic Array* (PLA) architectures [1]. The PAL approach focused on the use of a reconfigurable full interconnectivity pattern for the implementation of the signal routing between macro-logic blocks. Hence, the PAL approach is intimately defined by its large routing array. It is important to note that in such a circuit, the logic is fixed and only the routing part is programmed.

The FPGA breaks this model by using both programmable logic and programmable routing structure. The logic is distributed through the routing structure in an island-style manner. This distribution helps in handling the routing congestions with an optimum number of resources.

The PAL routing architecture is a very simple but highly inefficient crossbar structure. Every output is directly connectable to every input. Connection is made through a programmable switch. The FPGA routing architecture provides a more

efficient routing where each connection typically goes through several switches. In a programmable logic device, the logic is implemented using two-level AND-OR logic with wide input for the AND gates. In an FPGA, the logic is implemented using multiple levels of lower fan-in gates, which is often much more compact than two-level implementations. An FPGA logic block could be as simple as a transistor or as complex as a *Digital Signal Processor* (DSP) block. It is typically capable of implementing many different combinational and sequential logic functions.

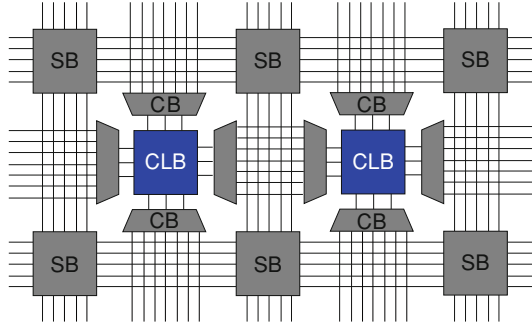
An FPGA structure is defined by fine-grain logic. This name comes from the granularity which is achieved by the logic blocks. Here, each logic blocks is supposed to realize a part of combinational or sequential logic operations but is not able to handle complex logic operation. The granularity is then in opposition to Massively-Parallel Processor Arrays or MultiProcessor System-On-Chip. Another particularity of the FPGA architecture is the hierarchical logic stratification. Indeed, a logic block is build from smaller logic blocks and a local reconfigurable interconnect. This interconnect is generally complete. This means that, like in a PAL, each signals (inputs and outputs) can be routed everywhere.

The routing architecture of an FPGA could be as simple as a nearest neighbor mesh [2] or as complex as the perfect shuffle used in multiprocessors [3]. More typically, an FPGA routing architecture incorporates wire segments of varying lengths which can be interconnected via electrically programmable switches. The choice of the number of wire segments incorporated affects the density achieved by an FPGA. If an inadequate number of segments is used, only a small fraction of the logic blocks can be utilized, resulting in poor FPGA density. Conversely the use of an excess number of segments that go unused also wastes area. The distribution of the lengths of the wire segments also greatly affects the density and performance achieved by an FPGA. For example, if all segments are chosen to be long, implementing local interconnections becomes too costly in area and delay. On the other hand if all segments are short, long interconnections are implemented using too many switches in series, resulting in unacceptably large delays.

The storage of the configuration is in charge of memories. These memories drive logic gates or pass-transistors in order to configure the logic or the routing. Different type of technologies could be used like *Static Random Access Memories* (SRAM), antifuse or flash. SRAM is actually the most standard technology, thanks to its CMOS technological homogeneity. In all cases, a programmable switch occupies larger area and exhibits much higher parasitic resistance and capacitance than a simple via. Additional area is also required for programming circuitry. As a result the density and performance achievable by today's FPGAs are an order of magnitude lower than that for ASIC manufactured in the same technology.

The complexity of FPGA has surpassed the point where manual design and programming are either desirable or feasible. Consequently, the utility of FPGA architecture is highly dependent on effective automated logic and layout synthesis tools to support it. A complex logic block may be under-utilized without an effective logic synthesis tool, and the overall utilization of an FPGA may be low without an effective placement and routing tool.

Fig. 2.1 Island-style field programmable gate array organization



Placement and routing tools mainly depend on the architecture and are specific to manufacturers [4, 5]. Some tools aim to be used as an exploration tool for research exploration of algorithms as well as architectures. As an example, the *Verilog-To-Routing* (VTR) toolflow is dedicated to this purpose [6].

2.1.1.2 General Architectural Organization

FPGAs are built of three fundamental components: logic blocks, I/O blocks and programmable routing, as shown in Fig. 2.1. In Fig. 2.1, CLB stands for *Configurable Logic Block*, which is the combinational and sequential logic block. CB and SB stand respectively for *Connection Box* and *Switch Box*. These circuits form the global routing resources. In Fig. 2.1, Input/Output (I/O) blocks have not been shown. They are found at the periphery of the circuit and are fed by the routing lines.

In an FPGA, a circuit is implemented by programming each of the logic blocks. Each block implements a small portion of the logic required by the circuit. The programmable routing is configured to make all the necessary connections between logic blocks and from logic blocks to I/O blocks, as well. Each of the I/O blocks is configured to act as either an input pad or an output pad.

2.1.1.3 Logic Block Architecture

The logic block used in an FPGA strongly influences the circuit speed and area-efficiency. Many different logic blocks have been used in FPGAs, but it is possible to consider two main families: the gate based FPGAs and the *Look-Up Table* (LUT) based FPGAs. Most current commercial FPGAs use logic blocks based on LUTs.

- Gate-based Approach

Gate-based FPGAs closely resemble basic *Application Specific Integrated Circuit's* (ASIC) cells. The finest grain logic block would be identical to a basic cell of an ASIC and would consist of A few transistors that can be interconnected

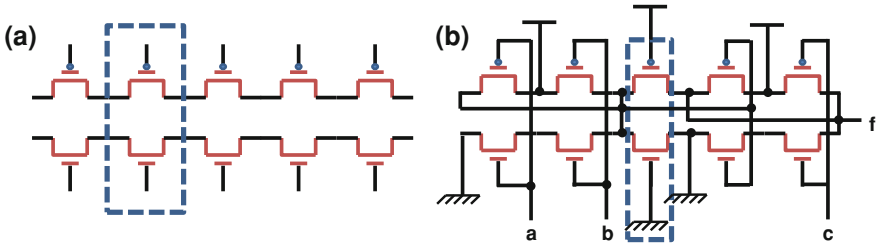


Fig. 2.2 Transistor-level programmability for FPGA [7]. **a** Transistor pair, **b** Isolation transistors

in a programmable way. The finest grain cell solution uses single transistor pairs. In [7], transistors pairs are connected together in rows. Within this pattern, the transistors are programmed to serve as isolation transistors or logic gates. Figure 2.2 illustrates how a function could be implemented. Figure 2.2a shows the transistor pair tiles and Fig. 2.2b shows a programmed function $f = a \cdot b + !c$. The function is programmed by ensuring the correct connections between the different transistors. In Fig. 2.2b, the dashed lines show the transistors that are turned off for isolation. The function is done by the two-input NAND gates formed on the left and right sides.

Instead of using programmability at the lowest transistor level, a two-input NAND gate has been used to realize the combinational block, as depicted in Fig. 2.3 [8]. The expected logic function is formed in the usual way by connecting the NAND gates together.

The main advantage of using fine grain logic blocks is that these blocks are fully utilized. This is because the logic synthesis techniques for such blocks are very similar to those for conventional mask-programmed gate arrays and standard cells. Then, it is easier to use small logic gates efficiently. The main disadvantage of these blocks is that they require a large number of wire segments and programmable switches. Such routing resources are costly in terms of delay and area. As a result, FPGAs employing fine-grain blocks are in general slower and achieve lower densities than those employing coarse grain blocks.

In [9], more complex logic blocks are proposed. Figure 2.4a shows that these blocks are based on the ability of a multiplexer to implement different logic functions by connecting each of its inputs to a constant or to a signal. Reference [10] presents a similar solution. Each input of the multiplexer and not just the select input is driven by an AND gate, as illustrated in Fig. 2.4b.

The alternating inputs to the AND gates are inverted. This allows input signals to be passed in true or complement form, thus eliminating the need to use extra logic blocks to perform simple inversions. Multiplexer-based logic blocks have the advantage of providing a large degree of functionality for a relatively small number of transistors. This is, however, achieved at the expense of a large number of inputs (8 in the case of Actel and 20 in the case of QuickLogic), which when utilized place high demands on the routing resources.

Fig. 2.3 Simple gate FPGA logic block [8]

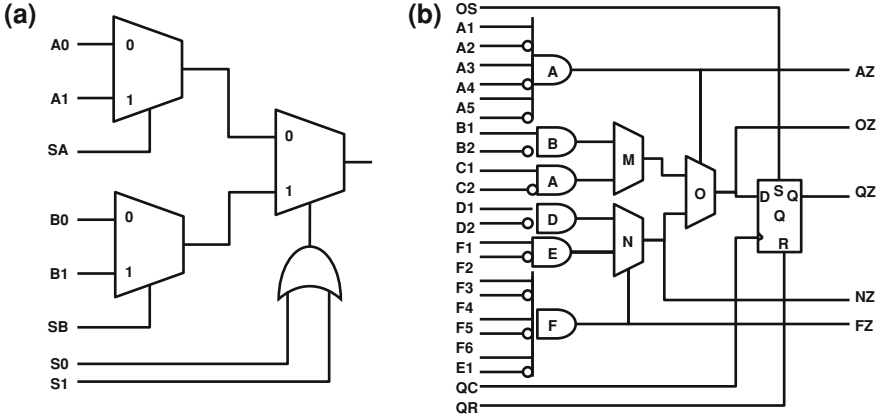
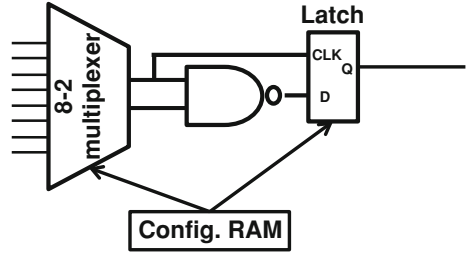


Fig. 2.4 Complex gate FPGA logic blocks. **a** Actel Act-1 LB [9]. **b** Quicklogic pASIC1 LB [10]

- Look-Up Table-based Approach

The other and most used approach is based on LUT. A LUT works thanks to memories driving the data inputs of the multiplexer. The truth table for a K -input logic function is stored in a $2^K \times 1$ SRAM. The address lines of the SRAM function as inputs and the output of the SRAM provides the value of the logic function. For example, we consider the logic function $f = a \cdot b + !c$. If this logic function is implemented using a three-input LUT, then the SRAM would have a 1 stored at address 000, a 0 at 001 and so on, as specified by the truth table. The advantage of look-up tables is that they exhibit high functionality. A K -input LUT can implement any function of K -inputs, i.e. 2^K functions. The disadvantage is that they are unacceptably large for more than about five inputs, since the number of memory cells needed for a K -input lookup table is 2^K . While the number of functions that can be implemented increases very fast, these additional functions are not commonly used in logic designs and are difficultly handled by the logic synthesis tools. Hence, it is often the case that a large LUT will be largely under-utilized.

Most modern FPGAs are composed not of a single LUT, but of groups of LUTs and registers with some local interconnect between them. A generic view for the LUT based logic block is shown in Fig. 2.5.

Fig. 2.5 Configurable logic block architecture [11]

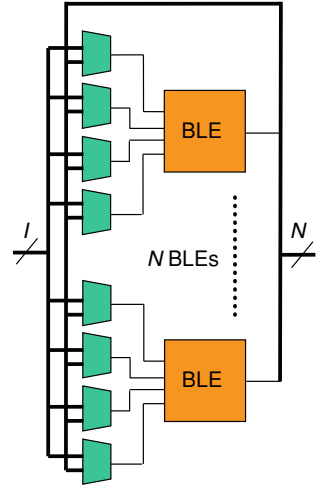
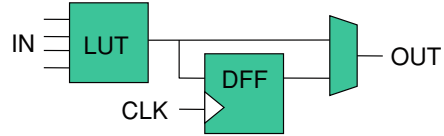


Fig. 2.6 Basic logic element architecture [11]



This CLB has a two-level hierarchy: the overall block is a collection of *Basic Logic Elements* (BLEs) [11]. As shown in Fig. 2.6, the BLE consists of a LUT and a register. Its output can be either the registered or unregistered version of the LUT output.

This is how many commercial FPGAs combine a LUT and a register to create a structure capable of implementing either combinational or sequential logic. The complete logic block contains several BLEs and local routing to interconnect them. Such a generic logic cluster scheme is described by three parameters [11]: the number of inputs to a LUT (K), the number of BLEs in a cluster (N), the number of inputs to the cluster for use as inputs by the LUTs (I). It is worth noticing that not all $K \cdot N$ LUT inputs are accessible from outside the logic cluster. Instead, only I external inputs are provided to the logic cluster. Multiplexers allow arbitrary connections of these cluster inputs to the BLE inputs. They also allow the connection of all the N outputs to each of the BLE inputs. All the N outputs of the logic cluster can be connected to the FPGA routing for use by other logic clusters. It is remarkable that each of the BLE inputs can be connected to any of the cluster inputs or any of the BLE outputs. Logic clusters are therefore internally fully connected. This is a useful feature, as it simplifies *Computer Aided Design* (CAD) tools considerably.

The presented structure is a very generic representation of LUT-based FPGAs. In fact, logic blocks of FPGAs are more complex, as detailed in the next sub-chapter.

2.1.1.4 Routing Architecture

The routing architecture of an FPGA is the manner in which the programmable switches and wiring segments are positioned to allow the programmable interconnection of the logic blocks.

Several routing architectures exist and come most of the time from results on the tradeoff between the flexibility of the routing architecture, circuit routability and density. Commercial routing approaches can be classified into three groups: row-based connections, island-style connections, and hierarchical scheme.

The row-based routing scheme is close to ASIC standard cells routing [9]. Effectively, logic blocks are organized in rows and a large number of horizontal wires are placed between the rows. Less vertical wires are used to connect rows to others.

Figure 2.7 depicts more precisely an island-style FPGA. Logic blocs are surrounded by routing channels of pre-fabricated wire segments on all four sides. A logic block input or output, which is called a pin, can be connected to some or all of the wiring segments in the channel adjacent to it via a connection block [12] of programmable switches. At every intersection of a horizontal channel and a vertical channel, there is a switch block [12]. This is simply a set of programmable switches that allows some of the wire segments incident to the switch block to be connected to others. It is worth pointing out that in Fig. 2.7, only a few of the programmable switches contained by switchboxes are shown. By turning on the appropriate switches, short wire segments can be connected together to form longer connections. In the figure, some wire segments continue unbroken through a switchbox. These longer wires span multiple logic blocks, and are a crucial feature in commercial FPGAs.

The number of wires contained in a channel is denoted by W . The number of wires in each channel to which a logic block pin can connect is called the connection block flexibility, or F_c . The number of wires to which each incoming wire can connect in a switch block is called the switch block flexibility, or F_s .

Inspired from generic programmable logic devices routing schemes, the hierarchical routing scheme try to use some form of locality to obtain better density and performance. This hierarchy could be realized with several different interconnect schemes, like PAL or island-style. This is the most currently used interconnection in FPGAs. For example, considering the structure presented in Figs. 2.1 and 2.5, we could remark that a full-interconnectivity scheme is used for CLB, while island-style organization is used for the global FPGA.

2.1.1.5 Architectural Parameterization and Optimum

The programmable switches introduced for routing purpose impact the performances. Thus, the FPGA architecture is the results of a trade-off between high versatility and cost/performance.

The adverse effects of the large size and relatively high parasitic of programmable switches can be reduced by careful architectural choices. By choosing the

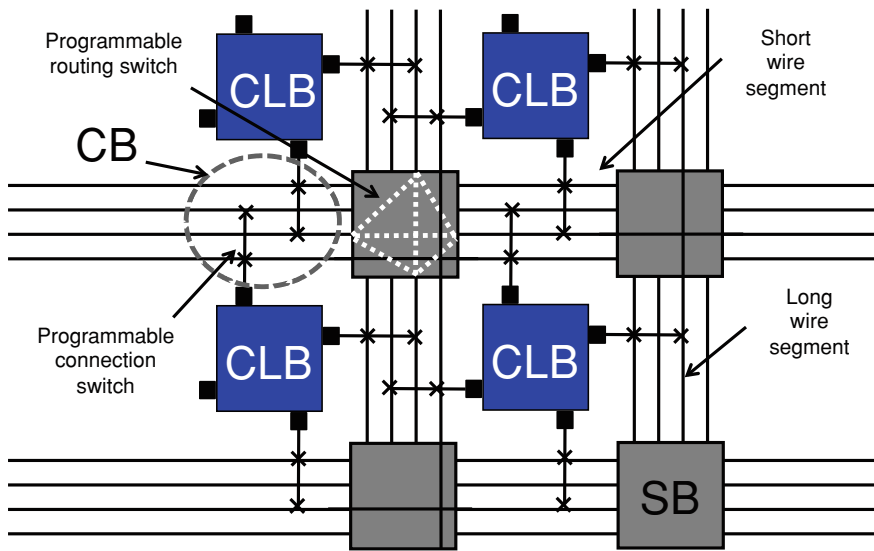


Fig. 2.7 Details on the routing organization in an island-style FPGA [11]

appropriate granularity and functionality of the logic block, and by designing the routing architecture to achieve a high degree of routability while minimizing the number of switches, both density and performance can be optimized. The best architectural choices, however, are highly dependent on the programming technology used as well as on the type of designs implemented, so that no unique architecture is likely to be best suited for all programming technologies and for all designs.

A complete study of architectural parameterization has been conducted in [11, 13, 14]. These studies assume SRAM based homogeneous FPGAs. The principal results are summarized in the following.

- **Combinational Granularity Impact**

As the granularity of a logic block increases, the number of blocks needed to implement a design should decrease. On the other hand a more functional (larger granularity) logic block requires more circuitry to implement it, and therefore occupies more area. This tradeoff suggests the existence of an “optimal” logic block granularity for which the FPGA area devoted to logic implementation is minimized. It has been shown in [13] that the most suited LUT size is reached for K equal to 4.

- **Logic Block Sizing**

Several interesting sizing results can be taken from [11]. Firstly, the number of distinct inputs required by a logic cluster grows fairly slowly with cluster size, N . A cluster of size N requires approximately $2N + 2$ distinct inputs (for $N \leq 20$).

Secondly, because all the input and output pins of a cluster are logically equivalent, one can significantly reduce the number of routing tracks to which each logic cluster pin can connect, F_c , as one increases the cluster size. A good value for $F_{c,output}$ is found with W/N , while $F_{c,input}$ is somewhat higher. Thirdly, logic clusters containing between 4 and 10 BLEs all achieve good performance, so any clusters in this range is a reasonable choice.

- **Routing Organization**

In [11], simulations have been carried out to study the impact of the routing structure. It has been shown that the most area-efficient routing structure is one with completely uniform channel capacities across the entire chip and in both horizontal and vertical directions. The basic reason is that most circuits “naturally” tend to have routing demands, which are evenly spread across an FPGA. Furthermore, it has been shown that it is most important for FPGAs to contain wires of moderate length (4–8 logic blocks) even if commercial FPGAs are using some very short and some very long wires.

2.1.2 Market Trends

In the previous sub-chapter, we focused on the FPGA architecture and especially the generic homogeneous island-style architecture. While the island-scheme is used in most FPGAs, their structures are relatively complex. Indeed, the generic architecture is useful for understanding and research purpose, but it suffers from several performance issues in some application classes. Thus, several improvements have been proposed for modern commercial FPGA. The circuits are enhanced in many ways.

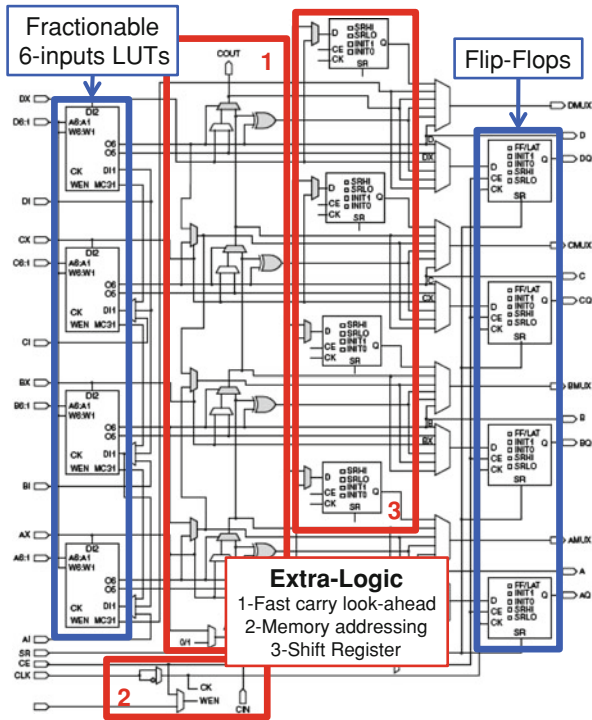
2.1.2.1 Increase of the CLBs Complexity

In order to increase the logic block functionality, the structure has been customized and new features have been added. The logic block of a modern Xilinx Virtex-6 FPGA is shown in Fig. 2.8.

The block is still organized around LUTs and FFs. These elements give the combinational and sequential ability. The first novelty is the fractional behavior of the LUT. A large LUT is useful to realize a large and complex combinational function. However, this situation is not frequent. Instead of wasting a large amount of logic resources when the LUT implements a small function, it is possible to split it into two independent smaller LUTs. It is then possible to optimize the logic fabric to the application, during the synthesis and packing operations.

Within the CLB, extra circuitries have been added to address specific functionalities. We can point out in particular notice the carry lines, the SRAM decoders and the shift-register.

Fig. 2.8 Virtex-6 logic block [15]



The carry lines extend through several LBs, in order to implement logic multipliers efficiently. Effectively, the specific track avoid to implement the carry through the global interconnect. Since global interconnect must be routed, delays for carry will be unknown and in any case larger than with the specific track. This allows significant increase in the performance of such a block without costing too many resources.

The use of FPGAs keeps increasing to implement complex SoCs. Thus, the requirement in terms of embedded memories is growing. LBs embed a large amount of SRAM memories for the LUT configuration. It is then of high interest to give the possibility to use them directly as standalone memories. In this configuration, the LUTs multiplexers are used as the address decoders, and a specific extra circuitry deals with the read management of the read/write control signals.

Finally, shift register behavior is also achievable, thanks to a dedicated module. Indeed, SRAMs can be cascaded and the writing is driven by a shifting register clock signal. This is obviously precious for specific applications.

2.1.2.2 Transition from Homogeneity to Heterogeneity

Associated with the complexity and diversity trends of the CLB architectures, the FPGA scheme tends to add more dedicated blocks and to heterogenize its structure. Indeed, the homogeneous FPGA is known to be slow and area costly for

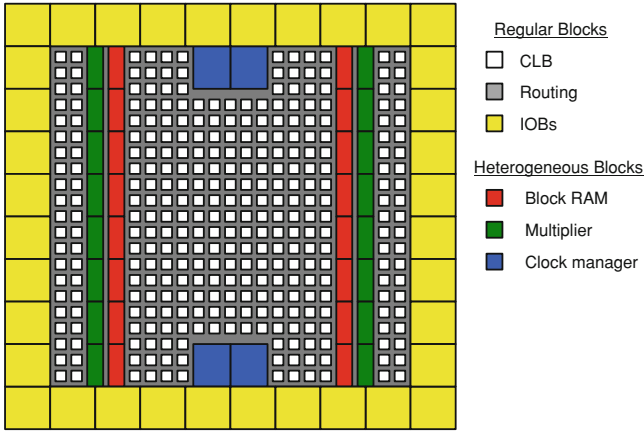


Fig. 2.9 Xilinx Spartan-3A architecture organization [16]

mathematical computation or floating point arithmetic. Hence, vendors have added dedicated co-processing logic units. These units are distributed through the logic grid. Figure 2.9 depicts the internal organization of a commercial Xilinx Spartan-3A FPGA. It is then possible to find multiplier for *Digital Signal Processing* (DSP) blocks or *Multiplier-Accumulator* (MAC) units, specific blocks for complex clocking domain generation or even USB controller, Ethernet controller for communications.

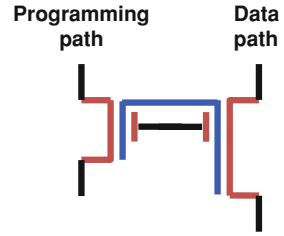
Furthermore, requirements on memories are wide in current applications. FPGAs then integrate dedicated standalone RAM blocks directly reachable by the routing part. This allows the creation of complex SoC and the efficient instantiation of soft microprocessors on the structure.

2.1.2.3 Non-Volatility Features

In high production FPGAs, the configuration is stored by SRAM memories. These memories are found distributed through the entire circuits to store the information as close as possible to the data path logic. SRAMs circuits use the same integration process than the other logic parts. Thus, they are the simplest solution to implement distributed configuration logic. Nevertheless, they suffer from their large size and, in particular from their volatile behavior.

Due to the volatility, the configuration must be reloaded into the FPGA circuits at each power-up. This obviously leads to a large loss of efficiency in terms of delay and power consumption. Furthermore, a non-volatile storage is still required outside of the chip. In current circuits, standalone flash memories are used to store the configuration bitstream, and specific programming circuits have in charge the programming sequence. This is highly area and power consuming. Then, it appears suited to use on-chip non-volatile memories directly.

Fig. 2.10 Schematic of a merged programming-data path floating gate transistor [20]



- Flash-based Approach

The current dominant non-volatile memory technology is the flash technology. Flash requires several technological steps in addition to CMOS. To take into account the complexity of flash/CMOS co-integration, the simpler solution is to co-integrate in the same package or directly on the same die the SRAM FPGA and its configuration flash memory. This solution is used in [16]. Such integration decreases the extra-chip requirements. Nevertheless, the most important hurdles still exist. Information is still duplicated in the flash and in the FPGA's SRAMs. This means that power consumption remains the same with a large wasted power in the distributed RAMs and a long power up time.

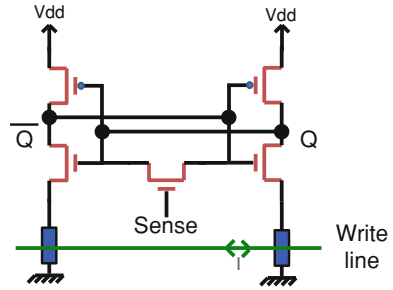
Thus, solutions have been envisaged to distribute the non-volatile memories through the logic grid. Flash based solution is proposed in [17–20], while emerging solutions are shown in [21].

In [17], co-integration of flash and MOS transistors yield in a compact configuration memory nodes. The structure uses a non-volatile pull-up (pull-down respectively) network and resistive pull-down (pull-up respectively) network. This voltage divider arrangement allows the storage of the configuration and the drive of logic gates. This enables the fabrication of a non-volatile base LUT, as introduced in [18]. Complex co-integration permits to create compact non-volatile switches for FPGAs. In fact, flash transistor could be seen as a programmable switch. The programming of a flash transistor needs specific voltages and circuitries. Traditionally, these circuits would be introduced into the data path. In [19, 20], a simple circuit composed of two flash transistors is presented (Fig. 2.10). The particularity is that the floating gate is shared in order to connect a dedicated programming transistor with the data path one. This is of high interest for logic compactness, and distinct separation between programming and logic path.

- Magnetic Memory-based Approach

In [21], magnetic RAMs are used instead of flash. Indeed, emerging resistive memories are expected to reduce the production costs of the circuits. Resistive memories could be integrated after the back-end process. This means that the resistive devices are realized only after the costly CMOS process. This is obviously interesting for cost reduction and process simplification. Figure 2.11 depicts the circuit that is used to store the information. The circuit is unbalanced flip-flop. Two magnetic tunnel junction memories store complementary data that are used to

Fig. 2.11 Magnetic tunnel junction unbalanced flip flop [21]



start the flip-flop in a good configuration at power up. The programming of the magnetic memories is in charge of special writing lines. The magnetic memories are placed above the IC as explained in [21]. Nevertheless, we should remark that this solution is area consuming due to the presence of front-end transistors in addition to the size of the back-end memories.

2.1.3 Limitations of FPGAs

Figure 2.12 presents the area/delay/power breakdown of the various components of a baseline Xilinx Virtex island style FPGA.

The impact of area of each FPGA part has been objectively studied in [22]. This evaluation has been realized with a precise methodology using a stick diagram of the implemented circuits, instead of using a simpler minimum-width equivalent size transistor count. It is worth noticing that the configuration memories occupy roughly half of the area in both the logic blocks and the routing resources. The logic blocks occupy only 22% of the whole area including its own configuration memory. Only 14% is then used for actual computation.

In addition to consuming most of the die area, programmable routing also contributes significantly to the total path delay in FPGAs. In [23, 24], interconnect delays are estimated and found to account for roughly 80% of the total path delay. Programmable routing also contributes to the high power consumption of FPGAs. This problem has recently become a significant impediment to the FPGA adoption in many applications. The power consumption measurements of some commercial FPGAs have shown that programmable routing contributes more than 60% of the total dynamic power consumption [25–27]. As a result of these performance degradations, FPGA performance is significantly worse in terms of logic density, delay, and power than cell-based implementations. Finally, it is commonly admitted that FPGAs are more than ten times less efficient in logic density, three times larger in delay, and 3 times higher in total power consumption than cell-based implementations [28].

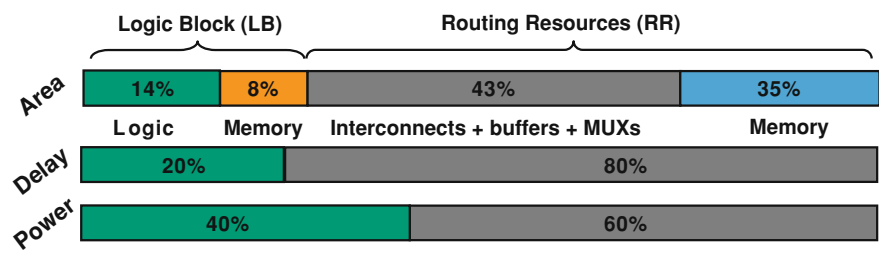


Fig. 2.12 Field programmable gate arrays area/delay/power repartition per block [22]

2.2 Emerging Reconfigurable Architectures Overview

While we expect that reconfigurable architecture represent the future of computation architectures, we will assess the ways in which emerging devices could improve or even break the current FPGA model. This represents a difficult challenge because in many cases, circuit-level models and/or architecture-level models for these devices and their interconnect systems either do not exist or they are very primitive. Moreover, the applications under considerations for these new devices can take many forms; i.e.

- as a drop-in replacement for CMOS,
- as additional devices that complement and coexist with CMOS devices, or
- as devices whose unusual properties can provide unique functionality for selected information processing applications.

The ITRS—Emerging Research Devices [29] and especially the *Emerging Research Architectures* section aims to identify possible applications for emerging logic and memory devices. Two main families of architectures are defined: the morphic architectures and the heterogeneous architectures. We will survey the principal architectures that have been devised for computing at the nanoscale.

2.2.1 Morphic Approach

As indicated by the etymology, the morphic approach aims to develop architectures that are inspired from other systems. They are often inspired from biology as far as biological systems are highly efficient due to very limited power consumption and high system reliability. In this class of application, neural networks immediately spring to mind. An *Artificial Neural Network* (ANN) is a model inspired by the human brain [30]. Its goal is to reproduce some properties found in the biological organ. Globally, ANNs are used in the following applications which benefit from its properties related to memories and/or learning: sorting, associative memories, compression... The conventional implementation of neural networks is

based on analog neuron and a digital control of the interconnectivity and edge weighting [31]. Such an implementation is far from the biologically inspired models. On the contrary, some implementations copy to emulate the behaviour and the structure of complex biological neural systems [32, 33]. They are called neuromorphics. Emerging devices are in this topic highly adapted for the hardware realization. In particular, [34] proposes to create reconfigurable hybrid CMOS/nanodevice circuits, called CMOL. In such a circuit, a CMOS subsystem with relatively large silicon transistors is used for signal restoration, long-range communications, input/output functions, and testing/bootstrapping. An add-on nanowire crossbar with simple two terminal nanodevices at each cross-point provides most of information storage and short-range communications.

The emergence of new devices and integration paradigms will certainly lead to several improvements of neuromorphic circuits. In this book, we are focusing on standard computation paradigms. Thus, we will consider that architectures suited for morphic applications are outside of our scope.

2.2.2 Heterogeneous Approach

Emerging technologies are expected to supersede CMOS in terms of functionality and performance metrics. Nevertheless, the transition to a new disruptive technology will not occur abruptly. In the near future, it seems reasonable to consider that standard CMOS circuits will be improved by new technologies. Two different means of improvement can be pointed out: the improvements coming from the increase of the device functionality (i.e. the use of devices with more functionality in the same area) and an improvement coming from the increase of integration density (i.e. more devices in the same area).

2.2.2.1 Regular Architectures

Mono-dimensional devices improve the performance of transistors channel. Nevertheless, the ultra-scaled dimensions represent a real challenge for the integration of complex circuits. In particular, photolithography unreliability requires the use of high regularity. Regularity can be envisaged at the transistor level. The use of micro-regularity is of high interest to reduce the size of the circuits drastically. Furthermore, regularity is compatible with bottom-up fabrication techniques. These techniques open the way towards complex arrangements at the nano-scale and lead to the emergence of crossbar circuits. A crossbar is defined by the regular arrangement of devices in the array. This leads to the most integrated structures achievable by the technology.

The first crossbars realized from emerging devices were proposed in [35–37]. The basic function that those circuits are implementing is information storage. Subsequently, the use of dense crossbars computational units has been conceptually

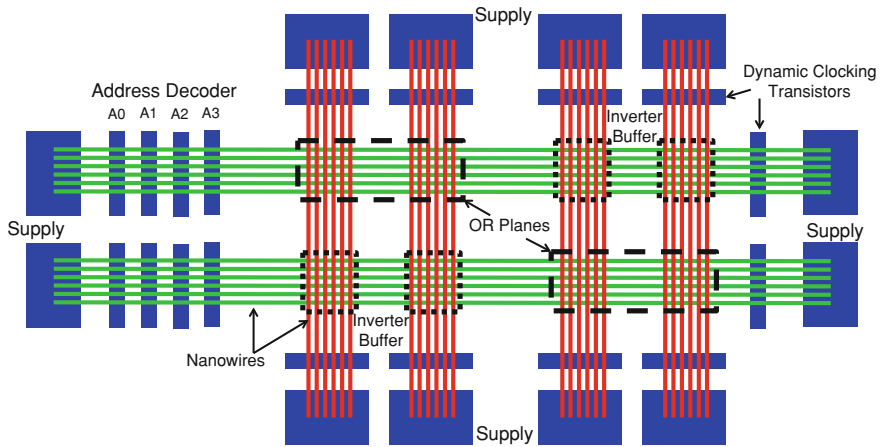


Fig. 2.13 NanoPLA architecture [41]

proposed in [38–40]. More precisely, such computation fabrics are based on semiconducting *Silicon NanoWires* (SiNWs) organized in a crossbar fashion. Active devices are formed at the cross-points. In [41], a Programmable Logic Array is proposed. The cross-points are realized by molecular switches. The switches can be programmed in order to perform either signal routing or wired-OR logic function. The structure is arranged in several sub-crossbars and presented in Fig. 2.13.

The input of the structure is a decoder interface between the micro/nano worlds. The decoder is used to address every nanowire independently of the others. The decoder design assumes that the nanowires are differentiated by a given doping profile [42]. The output of the crossbar is routed to a second crossbar. The signals can be inverted by gating the nanowires carrying the signals. A cascade of these two planes is equivalent to a NOR plane. It is worth noting that, due to diode logic, a logic restoration stage is required. Indeed, a more than unity gain is required to ensure a good cascade. At the end, the structure is duplicated many times and several stages are connected to each other in order to perform complex logic functions.

Instead of using diode logic, [43] uses FETs realized at the cross-points. While the previous approach implements a reconfigurable PLA circuit, this technology based on FETs address specific application-driven designs. Within this organization, a *Nanoscale Application Specific Integrated Circuit* (NASIC) is introduced in [44]. A NASIC tile consists of basic circuits such as adders, multiplexers and flip-flops. Circuits are realized using a dynamic logic style. Two clock transistors are placed between the power lines and a stack of transistors, which realize the logic function. It is possible to implement a standard AND/OR functions and their inverted counterparts. The implementation of the different logic functions is depicted in Fig. 2.14.

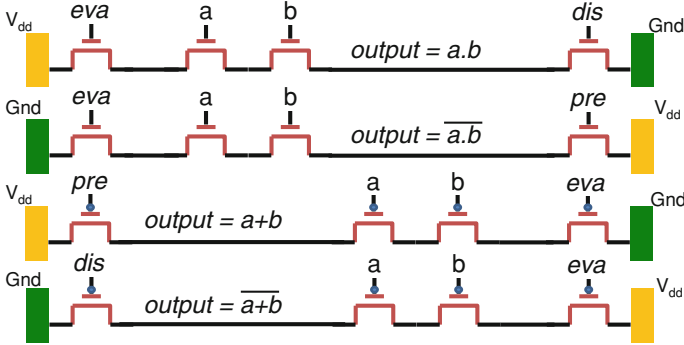


Fig. 2.14 Dynamic logic implementation of AND, NAND, OR and NOR functions [44]

From this logic organization, two-plane are cascaded (AND/OR, NAND/NAND...). Figure 2.15 depicts the complete NASIC tile. A first set of AND logic functions is realized in the horizontal direction. Nano-scale wires are connected to micro-scale power lines and to the other blocks that are surrounding the crossbar core. The second horizontal AND functions are driving transistors into the vertical orientation. The vertical line set implements OR functions. As a complete illustration, the circuit, presented in Fig. 2.15, implements a 1-bit full adder.

The doping of nano-grid strips, the size of the NASIC tiles, the use of certain nano-scale (i.e. sub-lithographic) wires as interconnect between tiles and the micro-level interconnects are chosen in an application/architecture-domain specific manner. These aspects determine a *NASIC* fabric and are the key differentiators between PLA type of nanoscale designs [41] and NASICs.

The most sensitive issue in these crossbar proposals remains the fabrication assessments. In fact, realization of 1D structures and their alignment over long distance with a good aspect ratio is extremely difficult to perform.

Several works have been published in order to assess the technological credibility of the structure [45]. Recently, a simple programmable crossbar-based processor has been fabricated. In [46], the authors have demonstrated a programmable and scalable architecture based on a unit logic tile consisting of two interconnected, programmable, non-volatile nanowire transistor arrays. The transistor structure is depicted in Fig. 2.16. S, D and G correspond to source, drain and gate, respectively. On left, the hole concentration in a *p*-type Ge/Si *NanoWire* Field Effect Transistor (NWFET) for two charge-trapping states is presented. This charge accumulation is in charge of the hysteresis behavior of the conductance.

Hence, each NWFET node in an array can be programmed to act as an active or an inactive transistor state. This is done by charge trapping into the floating layer. By mapping different active-node patterns into the array, combinational and sequential logic functions including full adder, full subtractor, multiplexer, demultiplexer and D-latch can be realized with the same programmable tile. Cascading this unit logic tile into linear or tree-like interconnected arrays is

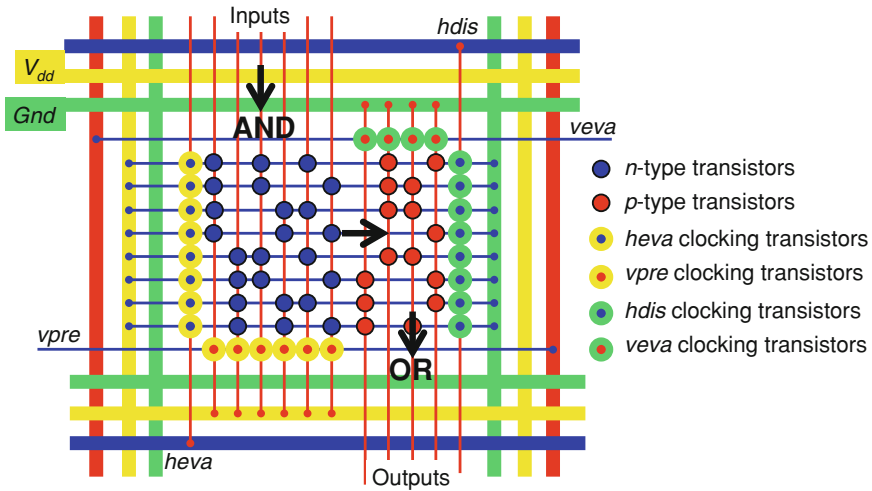


Fig. 2.15 NASIC structure (implementing a 1-bit adder) [44]

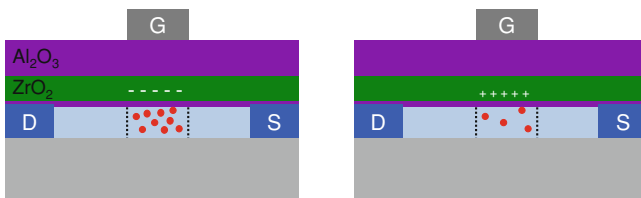


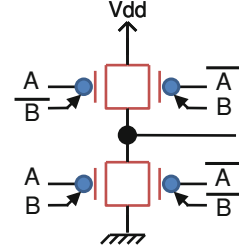
Fig. 2.16 Structure of a programmable NWFET [46]

possible given the demonstrated gain and matched input–output voltage levels of NWFET devices. This provides a promising bottom-up strategy for developing increasingly complex nanoprocessors with heterogeneous building blocks.

2.2.2.2 Enhanced Reconfigurable Architectures

In the previous part, emerging devices have been used to realize crossbar circuits. In fact, 1D structures have been mainly envisaged to create dense interconnection networks or dense substrates to build active devices at the nanoscale. Nevertheless, some 1D materials could be used efficiently to obtain new functionalities at the device level. For example, *Carbon NanoTubes Field Effect Transistor* (CNFET) exhibits an ambipolarity property [47]. This means that the same device could be controlled between *n*- or *p*-type, only thanks to the voltage applied to back-gate electrode. This property of CNTs is an opportunity that does not exist in CMOS technology. In [48–50], the benefit on logic circuit design is assessed.

Fig. 2.17 XOR function example built with four ambipolar transistors [49]



In [48, 49], the main novelty is to leverage the ability of performing logic operations between the signals feeding both gates of ambipolar CNFETs. The design with such operations is demonstrated in a set of static logic families including combinations of transmission-gate/pass-transistor on the one hand and complementary/pseudo logic on the other hand. This yields to a natural, simple and efficient implementation of the XOR function in ambipolar CNT technology with almost no cost, as shown in Fig. 2.17. The basics of the proposition are to configure the polarity of the input signal. The polarity choice is obviously used by the configuration voltage and the transistor type.

2.2.3 Global Comparisons and Discussions

In order to compare the different architectures presented above, we will use five global comparison metrics: the area, the performance, the power consumption, the technological maturity and the fault tolerance ability. Table 1 presents the global results.

It is worth noting that a solution based on sublithographic nanowires seems to increase drastically the area of the final circuits. This seems obvious considering that the integration density of the elementary devices is enhanced. Nevertheless, these solutions are quite worse in terms of other metrics. In fact, NW-based solutions require the use of complex clocking signals to restore the logic levels. This means that a large part of performance and power are wasted only for the periphery. In this context of intense power consumption, it is remarkable that carbon electronics is promising due to the good intrinsic properties of the devices. Thanks to carbon technology, it is possible to build architectural solutions that are globally efficient in all the proposed metrics, even in terms of fault tolerance and reliability. Carbon electronics have been widely studied, to enhance the reliability of the fabrication processes, and to provide complete methodologies for robust designs [51, 52]. Finally, we should consider that morphic approach provides naturally a high robustness. Effectively, these approaches are implementing neural networks. This computation scheme is well known for its versatility and performance regarding faulty tolerance.

Table 1 State-of-art architecture global metrics comparisons

	Reference publications	Area	Performances	Power consumption	Maturity	Fault tolerance
CMOL	[34]	+	+	+	---	+++
NanoFabric	[38, 39]	++	—	—	—	—
NanoPLA	[41]	++	—	—	—	—
NASIC	[43]	++	+	—	—	—
DG-CNFET	[49]	+	+	++	—	+

2.3 Architectural Formalization and Template

In this book, a fast and global evaluation methodology is presented and used to evaluate technologies from an architectural perspective. The proposed method requires the definition of a generic architectural template. From the existing state-of-the-art, it is possible to extract some global comments, regarding the design strategy and methodology.

2.3.1 Global Statement

Several works have been conducted on the fabrication technology process, reliability, circuit design, architectural definition and associated tools. Design with emerging technologies suffers from several unknown parameters. It is challenging to evaluate the expected performance of a whole circuit. Only a few works actually merge more than two aspects. Nevertheless, it is desirable to evaluate the potentiality of the technology within a complete architecture. The term complete covers the questions regarding the technological credibility of assumptions, the design of the computation part itself, the design of all the peripheral circuitries and the associated methodologies. This is required addressing all the previously defined aspects. As an illustration, the design of a compact logic gate does not make sense if the gate is not robust enough or requires significant additional circuitry.

Along the same lines, it is worth highlighting that the work on global architectures is often completely uncorrelated from the technological assumptions. This is standard architecture design with mature CMOS process. In the case of an advanced process, it is hard to handle the design of architecture without credible technological assumptions. As an example, while the use of bottom-up aligned SiNW is credible, it is difficult to imagine the wires aligned over several micrometers at only a pitch of some nanometers. This will leads to an angle deviation less than 10^{-5} degrees and is hard to handle today, as well in the near future. This makes that the architecture study must not be done under unrealistic assumptions, to ensure its credibility.

Finally, while it is recognized that variability and technological issues will increase drastically in the future, only a few works actually consider the questions of reliability. Conventional robustness techniques could be used, such as defect avoidance [53], error correction coding [54], or redundancy. Generally, unreliability in systems is overcome by duplicating the unreliable resource. Such spare circuits will then be used as a replacement part if the primary circuits are defective. However, a robust design approach is preferred to increase the reliability of a circuit early in the design [51, 52]. This allows to directly correct the misperformance at the level where it occurs, instead of a much higher level. Indeed, the correction will have a much larger and detrimental impact on the whole circuit if it is performed far from the origin of the issue. Even if the question of reliability must be tackled in future systems, we consider this specific point as out of the scope of this book.

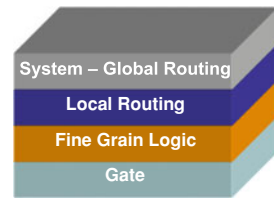
To avoid this lack in methodology, it appears interesting to address the question of the architecture globally. Nevertheless, it is of course, not possible to handle it directly, due to the complexity of the design. Indeed, the problem concerns the handling of a complex mix between gates that leads to computational units, memories and routing with so many unknown parameters. In this book, we will use a generic template for reconfigurable architecture in order to organize the contributions into a hierarchy. The template will be presented in the following. Each layer will then be studied and optimized independently. A correct hierarchy organization ensures that the levels will not impact the others as long as interface requirements remain the same. The optimization of specific parts will be examined in the following chapters. Finally, even if a layer is improved in terms of area or functionality, it is important to assess the impact on the architectural level. Thus, it will be necessary to define a benchmarking tool which is compatible with our architectural template, and which is able to implement much different architecture for each layer. This will allow exploring the architectural design space and permitting fast track optimization for the designs. This will be presented in the following.

2.3.2 Generic Architectural Template

In this book, we are focusing on reconfigurable architectures and the enhancement that could come from emerging technologies.

Reconfigurable architectures such as FPGAs are highly versatile and adaptive in terms of target application. They use regular architecture, with several blocks that are replicated through the circuit. We have previously seen that the FPGA structure is organized hierarchically, which seems a sound approach to manage the large amount of replicated blocks. In this context, it is difficult to see where emerging technologies will be useful to improve the performance of the structure. Furthermore, the FPGA structure has been designed for CMOS-based LUT logic. The structure is optimal for this application, but we could expect that the structure will not be the same in the case of emerging technology. Thus, it is of great interest to propose an architectural

Fig. 2.18 Layered organization of the generic template architecture



template based on a hierarchical organization of configurable and routing blocks, and to generalize each hierarchical level to its global behaviour as well.

The generic template is shown in Fig. 2.18. The template is organized into four levels of hierarchy: the *Gate* level, the *Fine Grain Logic* level, the *Local Routing* and the *System-Global Routing* level.

The *Gate* level corresponds to the elementary gates that are used to perform the computation. By computation is meant the combinational and the sequential operations. Computation in FPGAs is generally realized by LUTs, but it also could be, as we have seen in the literature review, custom logic from the simple transistor to the bigger logic function.

The *Fine Grain Logic* corresponds to the smaller autonomous block that could perform both combinational and sequential logic. We consider also at this level all the configuration memories that are used to program the structure. These memories give the fully programmable functionality of the block. In the FPGA scheme, this level corresponds to the BLE, i.e. a LUT which is used to perform a fine grain operation. The LUT output is routed to a latch but also directly to an output multiplexer, which is in charge of the path selection between the latches and the unlatched version of the signal.

The *Local Routing* level aims to provide an arrangement of Fine Grain Logic and to provide it with a local connectivity pattern. Generally, this connectivity pattern is full and aims to provide a large kind of programmable paths at the lower level. This allows a huge simplification of the packing tools. In the FPGA scheme, this level corresponds to the CLB. Indeed, an assembly of BLEs are fully interconnected by a multiplexer based interconnect. Thus, it is possible to realize with such a block a coarser function with large synthesis simplicity.

At the final *System-Global Routing* level, the circuit is organized with a macro-regularity. Logic blocks defined at the previous level are regularly arranged and interconnected. The organization is generally an island-style scheme and the interconnect resources handle the resource limited interconnection pattern. This means that each logic blocks could be interconnected to others, but it is not possible to realize all the connections at the same time. In the FPGA scheme, we can also find the CLBs surrounded by the programmable interconnect. Several other interconnect patterns can be found with for example a local connection set between the logic blocks, such as those defined in [38].

This generic template represents a real opportunity for design exploration. Indeed, the denomination of a very generic template helps in enlarging each level. In fact, it allows extension of both technologies and architectures of each level,

while keeping the others layer unchanged. This allows each layer to be studied independently, while the assumptions on all the other parts remain unchanged and greatly stimulate the development of original contributions.

2.4 Conclusion and Work Position

In this chapter, we surveyed the state-of-the-art regarding reconfigurable architectures. The FPGA structure is the traditional reconfigurable architecture. The basement of the original FPGA is the hierarchic and homogeneous arrangement of logic blocks. The architecture suffers from the programming circuit in area. Indeed, the computation part of the FPGA structure occupies only a small amount of area, conversely to routing structures. Today, the structure evolves to heterogeneity. Thus, it will reach new application classes and improve the computation/routing ratio. The heterogeneity is found at design level. Adjunction of several specific logic blocks such as memory blocks and DSPs allows increasing the versatility and the computation performance. Also found at the technology level, heterogeneity leads to the use of flash/MOS co-integration to non-volatile store the configuration.

The standard architecture is optimized for CMOS. Emerging technologies could lead to new architectural paradigms for reconfigurable computation. Emerging reconfigurable architectures have been previously envisaged through two approaches: the use of density-increased devices (i.e. ultra-scale devices) and the use of functionality-enhanced devices (i.e. devices with new highly functionality within the same area). Globally, the use of emerging technology leads to the improvement of performance compared to CMOS structures. Nevertheless, their immaturity and the technological hurdles make these solutions prospective.

In this book, we will use the emerging technologies to improve the standard FPGA approach. Two approaches will be used. We will first focus on the improvement of peripheral circuitries, i.e. memories and routing resources ([Chap. 3](#)). These peripheral circuits are expected to improve the standard FPGA scheme by their direct use within the routing resources ([Chap. 4](#)). Then, we will break the logic block paradigms ([Chap. 5](#)) and propose a new seed for architectural organization ([Chap. 6](#)). In addition, we will have in mind two requirements: the proposition of a credible fabrication process and the architectural compatibility with existing FPGAs.

References

1. J. Birkner, Reduce random-logic complexity. *Electron. Des.* **26**(17), 98–105 (1978)
2. W. Carter, K. Duong, R.H. Freeman, H. Hsieh, J.Y. Ja, J.E. Mahoney, L.T. Ngo, S.L. Sze, A user programmable reconfigurable gate array, in *Proceedings of the Custom Integrated Circuits Conference*, May 1986, pp. 233–235

3. H.S. Stone, Parallel processing with the perfect shuffle. *IEEE Trans. Comp. C* **20**(2), 153–161 (1971)
4. Altera Quartus-II: Altera FPGAs Design IDE, <http://www.altera.com/products/software/sfw-index.jsp>
5. Xilinx ISE: Xilinx FPGAs Design IDE, <http://www.xilinx.com/products/design-tools/ise-design-suite/index.htm>
6. Verilog-To-Routing (VTR) Project, <http://www.eecg.utoronto.ca/vtr/>
7. D. Marple, L. Cooke, An MPGA compatible FPGA architecture. *Proceedings of the IEEE Custom Integrated Circuits Conference*, 3–6 May 1992, pp. 4.2.1–4.2.4
8. P. Dillien, Adaptive hardware becomes a reality using electrically reconfigurable arrays (ERAs), in *IEE Colloquium on User-Configurable Logic—Technology and Applications*, 1 Mar 1991, pp. 2/1–2/10
9. A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K.A. El-Ayat, A. Mohsen, An architecture for electrically configurable gate arrays. *IEEE J. Solid-State Circuits* **24**(2), 394–398 (1989)
10. B. Small, The flexibility of the quicklogic FPGA architecture. *WESCON*, 27–29, pp. 688–691 (1994)
11. V. Betz, J. Rose, A. Marquart, *Architecture and CAD for deep-submicron FPGAs* (Kluwer Academic Publishers, New York, 1999)
12. J. Rose, S. Brown, Flexibility of interconnection structures for field-programmable gate arrays. *IEEE J. Solid State Circuits* **26**(3), 277–282 (1991)
13. E. Ahmed, The effect of logic block granularity on deep-submicron FPGA performance and density. Master thesis, University of Toronto (2001)
14. E. Ahmed, J. Rose, The effect of LUT and cluster size on deep-submicron FPGA performance and density. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **12**(3), 288–298, (2004)
15. Xilinx Virtex-6 FPGA Family Overview (2011), http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf. Accessed 24 March 2011
16. Xilinx Spartan-3A FPGA Product Overview (2011), http://www.xilinx.com/support/documentation/data_sheets/ds557.pdf. Accessed 4 April 2011
17. J. McCollum, H.-S. Chen, F. Hawley, Non-volatile programmable memory cell for programmable logic array, U.S. Patent No. 0064484, 2007
18. J. McCollum, G. Bakker, J. Greene, Non-volatile look-up table for an FPGA, U.S. Patent No. 0007293, 2008
19. J. Lipp, D. Freeman, U. Broze, M. Caywood, G. Nolan, A general purpose, non-volatile reprogrammable switch, WO Patent No. 01499, 1996
20. K.J. Han, N. Chan, S. Kim, B. Leung, V. Hecht, B. Cronquist, A novel flash-based FPGA technology with deep trench isolation. *Proceedings of the IEEE Non-Volatile Semiconductor Memory Workshop*, 26–30 Aug. 2007, pp. 32–33
21. N. Bruchon, L. Torres, G. Sassatelli, G. Cambon, New nonvolatile FPGA concept using magnetic tunneling junction, in *Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, 2–3 March 2006, p. 6
22. M. Lin, A. El Gamal, Y.-C. Lu, S. Wong, Performance benefits of monolithically stacked 3-D FPGA. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **26**(2), 216–229 (2007)
23. M. Hutton, V. Chan, P. Kazarian, V. Maruri, T. Ngai, J. Park, R. Patel, B. Pedersen, J. Schleicher, S. Shumarayev, Interconnect enhancements for a high-speed PLD architecture, in *Proceedings of the ACM/SIGDA 10th International Symposium on FPGA* (2002), p. 3
24. D. Lewis, E. Ahmed, G. Baekler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, J. Rose, The Stratix II logic and routing architecture, in *Abstracts of the ACM/SIGDA 13th International Symposium on FPGA* (2005), p. 14
25. E. Kusse, J. Rabaey, Low-energy embedded FPGA structures Paper presented at the International Symposium on Low Power Electronics and Design (1998), p. 155

26. L. Shang, A.S. Kaviani, K. Bathala, Dynamic power consumption in Virtex-II FPGA family, in *Proceedings of the ACM/SIGDA tenth International Symposium on. FPGA* (2002), p. 157
27. V. Degalahal, T. Tuan, Methodology for high level estimation of FPGA power consumption, in *Proceedings of the Design Automation Conference*, 2005, p. 657
28. I. Kuon, J. Rose, Measuring the gap between FPGAs and ASICs, in *Proceedings of the ACM/SIGDA 14th International Symposium on FPGA*, 2006, p. 21
29. Emerging Research Devices and Materials Chapters, Updated Editions, International Technology Roadmap for Semiconductors (2010), <http://www.itrs.net/Links/2010ITRS/Home2010.htm>
30. R.C. Eberhardt, R.W. Dobbins, *Neural Networks PC Tools—A practical guide*. (Academic Press Inc., San Diego, 1990)
31. J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. U.S.A.* **79**, 2254–2258 (1982)
32. J. Hoekstra, E. Rouw, in *Modeling of Dendritic Computation: The Single Dendrite*, vol. 517 (The American Institute of Physics, Melville, 2000), pp. 308–322
33. C. Mead, *Analog VLSI and Neural Systems* (Addison Wesley, Menlo Park, 1989)
34. D.B. Strukov, K.K. Likharev, Reconfigurable hybrid CMOS/nanodevice circuits for image processing. *IEEE Trans. Nanotechnol.* **6**(6), 696–710 (2007)
35. W. Wu, G.-Y. Jung, D.L. Olynick, J. Straznicki, Z. Li, X. Li, D.A.A. Ohlberg, Y. Chen, S.-Y. Wang, J.A. Liddle, W.M. Tong, R.S. Williams, One-kilobit cross-bar molecular memory circuits at 30-nm half-pitch fabricated by nanoimprint lithography. *Appl. Phys. A Mater. Sci. Process.* **80**(6), 1173–1178 (2005)
36. Y. Luo, C.P. Collier, J.O. Jeppesen, K.A. Nielsen, E. DeIonno, G. Ho, J. Perkins, H.-R. Tseng, T. Yamamoto, J.F. Stoddart, J.R. Heath, Two-dimensional molecular electronics circuits. *J. Chem. Phys. Phys. Chem.* **3**, 519–525 (2002)
37. J.E. Green, J. Wook Choi, A. Boukai, Y. Bunimovich, E. Johnston-Halperin, E. Deionno, Y. Luo, B.A. Sheriff, K. Xu, Y. Shik Shin, H.-R. Tseng, J.F. Stoddart, J.R. Heath, A 160-kilobit molecular electronic memory patterned at 1011 bits per square centimetre. *Nature* **445**, 414–417 (2007)
38. S.C. Goldstein, M. Budi, NanoFabrics: spatial computing using molecular electronics, in *Proceedings of the 28th Annual International Symposium on Computer Architecture* (2001), pp. 178–189
39. S. Goldstein, D. Rosewater, Digital logic using molecular electronics. *Proceedings of the IEEE International Solid-State Circuits Conference*, vol. 1, 2002, pp. 204–459
40. A. DeHon, Array-based architecture for FET-based, nanoscale electronics. *IEEE Trans. Nanotechnol.* **2**(1), 23–32 (2003)
41. A. DeHon, M.J. Wilson, Nanowire-based sublithographic programmable logic arrays, in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays* (2004)
42. A. DeHon, P. Lincoln, J. Savage, Stochastic assembly of sublithographic nanoscale interfaces. *IEEE Trans. Nanotechnol.* **2**(3), 165–174 (2003)
43. C.A. Moritz, T. Wang, Latching on the wire and pipelining in nanoscale designs, in *Proceedings of the 3rd Workshop on Non-Silicon Computation (NSC-3)*, June 2004
44. T. Wang, P. Narayanan, C. A. Moritz, Combining 2-level logic families in grid-based nanoscale fabrics, in *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, Oct 2007
45. P. Vijayakumar, P. Narayanan, I. Koren, C.M. Krishna, C.A. Moritz, Impact of nanomanufacturing flow on systematic yield losses in nanoscale fabrics, in *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, June 2011
46. H. Yan, H.S. Choe, S.W. Nam, Y. Hu, S. Das, J.F. Klemic, J.C. Ellenbogen, C.M. Lieber, Programmable nanowire circuits for nanoprocessors. *Nature Lett.* **470**, 240–244 (2011)
47. Y.-M. Lin, J. Appenzeller, J. Knoch, P. Avouris, High-performance carbon nanotube field-effect transistor with tunable polarities. *IEEE Trans. Nanotechnol.* **4**(5), 481–489 (2005)

48. M.H. Ben Jamaa, D. Atienza, Y. Leblebici, G. De Micheli, Programmable logic circuits based on ambipolar CNFET, in *Proceedings of the 45th ACM/IEEE Design Automation Conference*, 8–13 June 2008, pp. 339–340
49. M.H. Ben Jamaa, K. Mohanram, G. De Micheli, Novel library of logic gates with ambipolar CNTFETs: opportunities for multi-level logic synthesis. Paper presented at the Design, Automation & Test in Europe Conference & Exhibition, 20–24 April 2009, pp. 622–627
50. I. O'Connor, J. Liu, F. Gaffiot, F. Pregaldiny, C. Lallement, C. Maneux, J. Goguet, S. Fregonese, T. Zimmer, L. Anghel, T.-T. Dang, R. Leveugle, CNTFET modeling and reconfigurable logic-circuit design. *IEEE Trans. Circuits Syst. I Regul. Pap.* **54**(11), 2365–2379 (2007)
51. N. Patil, J. Deng, A. Lin, H.S.-P. Wong, S. Mitra, Design methods for misaligned and mis-positioned carbon-nanotube-immune circuits. *IEEE Trans. Comp. Aided Des.* (2008)
52. N. Patil, A. Lin, J. Zhang, H. Wei, K. Anderson, H.-S.P. Wong, S. Mitra, Scalable carbon nanotube computational and storage circuits immune to metallic and mis-positioned carbon nanotubes. *IEEE Trans. Nanotechnol.* (2010)
53. J.R. Heath, P.J. Kuekes, G.S. Snider R.S. Williams, A defect-tolerant computer architecture: opportunities for nanotechnology. *Science* **280**(5370), 1716–1721 (1998)
54. W. Huffman, V. Pless, *Fundamentals of Error-Correcting Codes* (Cambridge University Press, Cambridge, 2003)

Disruptive Logic Architectures and Technologies

From Device to System Level

Gaillardon, P.-E.; O'Connor, I.; Clermidy, F.

2012, XIV, 190 p., Hardcover

ISBN: 978-1-4614-3057-5