

## Chapter 2

# Reconfigurable Field Programmable Gate Arrays: Basic Concepts

### 2.1 FPGA Architectures

The first FPGA models have been introduced during the 1980s. The first programmable logic, almost similar to the FPGA, is comparable to the first costly programmable devices called programmable logic devices (PLDs) but able to implement a significantly higher amount of logic. Two first categories of devices have been developed: *antifuse*, consisting of an electrically programmable configuration memory which can be programmed only a single time and FPGA based on a configuration memory with SRAM cells that can be configured. Despite the antifuse devices were initially preferred for the more stability of the configuration memory, at the end of the 1980s, most of the preliminary dependability problems were solved, and the technology based on SRAM has started growing thanks to the volatility of the configuration memory that enables a wide range of applications. The FPGA architecture based on SRAM configuration memory can be configured in a very reduced time with whatever processor, differently from the antifuse FPGA that could be programmed only a single time.

The FPGA architecture consists of a generic matrix of block interconnected by programmable interconnections. The capability of implementing any combinational or sequential function is related to the logic block capabilities. The elementary logic block function is generally called *configurable logic block* and it has the architecture illustrated in Fig. 2.1 The internal components of a configurable logic block may vary among different manufacturers. In the most cases the configuration logic blocks contain a main logical circuit called look-up table (LUTs); an example of a LUT is given in Fig. 2.2. It consists of a static RAM (SRAM) having the following dimensions  $2^m + 1$ . It represents a truth table for a logic function having  $m$  inputs. The input lines connected to the SRAM correspond to the inputs of the truth table, vice versa the output of the SRAM provides the value of the logic function. The LUT provides a high functionality since it can realize any function with  $m$  inputs on a set of possible functions equivalent to  $2^{2^m}$ , with a maximum limit given by the number of configuration memory cells requested for a given LUT with  $k$  inputs, that is, equal to  $2^k$  [7, 8].

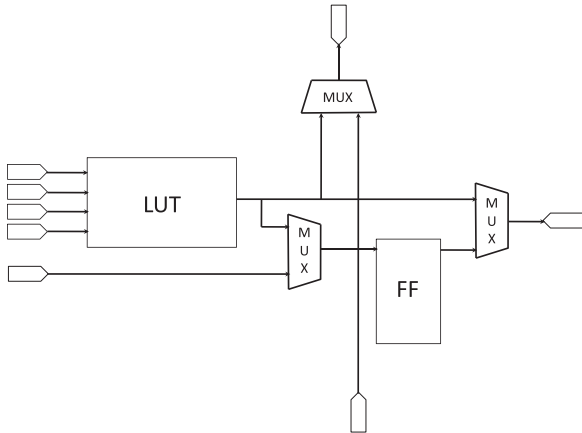


**Fig. 2.1** The main element of the FPGA architecture: the configurable logic block (CLB)

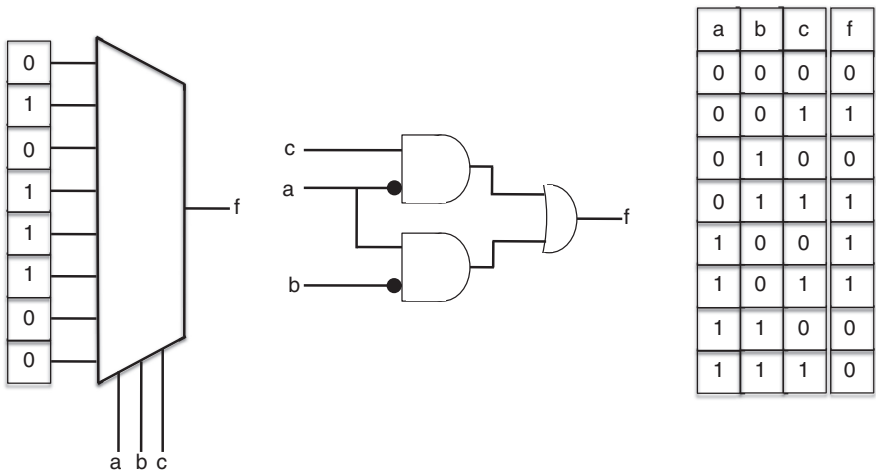


**Fig. 2.2** An example of a three-input LUT

The architectures of existing FPGAs mainly differ in three aspects:

- Type of programming technology used
- Structure of the logic block
- Structure of the interconnection network

Although the manufacturing differs, it is possible to create a generalized model of the internal structure; this model consists of four representation levels:

1. Tile
2. Local routing
3. Multiple tile
4. Context

The first representation level of the model, shown in Fig. 2.3, consists in the tile. The tile representation contains the CLB elements and the interconnections that guarantee their internal connectivity; these interconnections are exclusively engaged for the link of the CLB within a single tile.

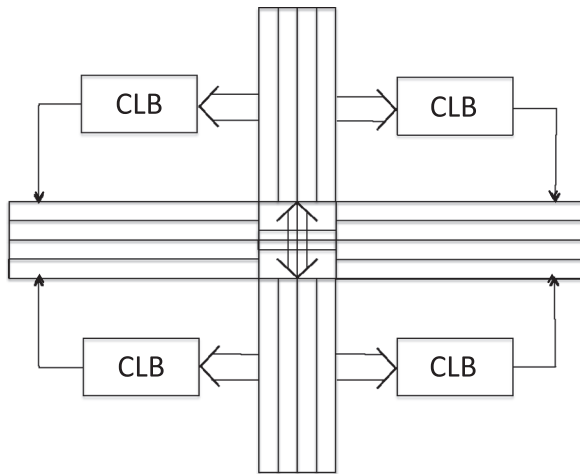


**Fig. 2.3** First representation level of the model: the tile

The second representation level, shown in Fig. 2.4, consists in the local routing among tiles based on the *routing* element called *switch block*. The interconnections between the different tiles are realized thanks to wiring segments embedded in the switching architecture; these wiring segments can be connected through the *switch block*. The interconnections at this level of the model are classified as local; therefore, they characterize the local routing. The *switch block* described in Fig. 2.5 is a component consisting of programmable interconnections able to connect the logic resources of the tile.

The third level of the representation consists in the *multiple tile*: a macro element consisting of a set of tiles and with the respective switch blocks. The representation considers the overlay logic that is implemented by the FPGA architecture. In Fig. 2.6 the third level of the representation is illustrated. In detail, the connections V0 and H0 are the wiring segments for the connection of a generic tile, while V1 and H1 consist of the segments creating the connectivity among tiles. The connectivity channel between the tiles consists of long wiring segments with low resistance that can be traversed by signal in both the directions depending on the direction assessed to the signal by the switch blocks connected to its extreme points. Depending on the traversing direction, several degrees of connectivity may exist. The degree of connectivity is determined by the flexibility of the switch blocks and by the resolution of the routing architecture, it is an important factor for the efficiency of the FPGA architecture.

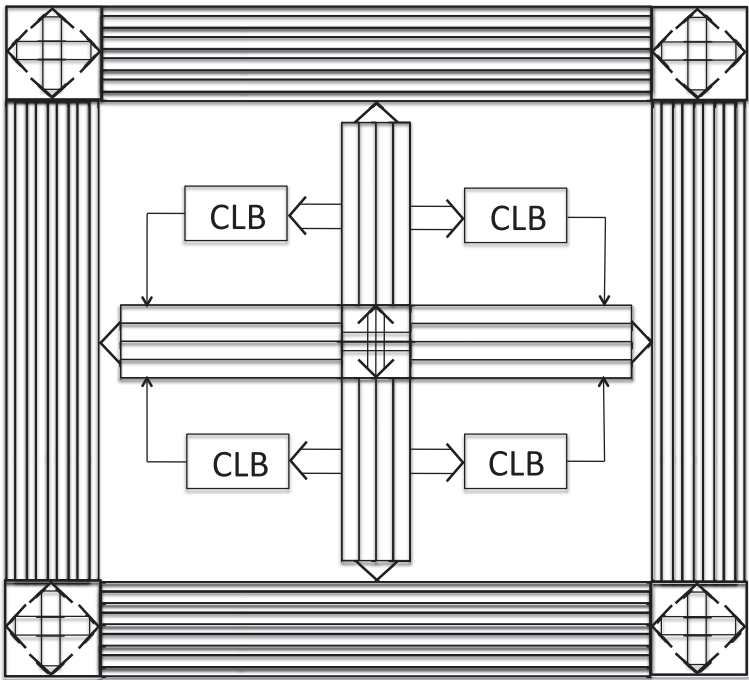


Fig. 2.4 Second representation level of the model: the local routing

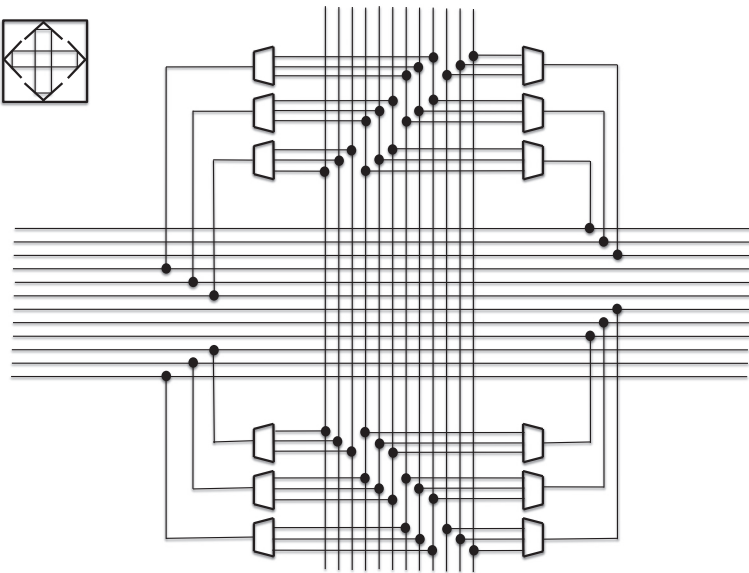


Fig. 2.5 The switch-block element

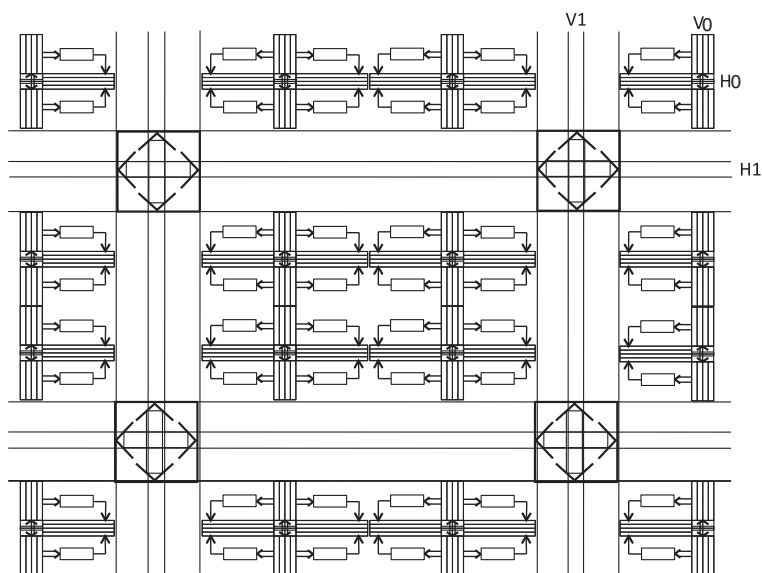


**Fig. 2.6** The third representation level: the multiple tile

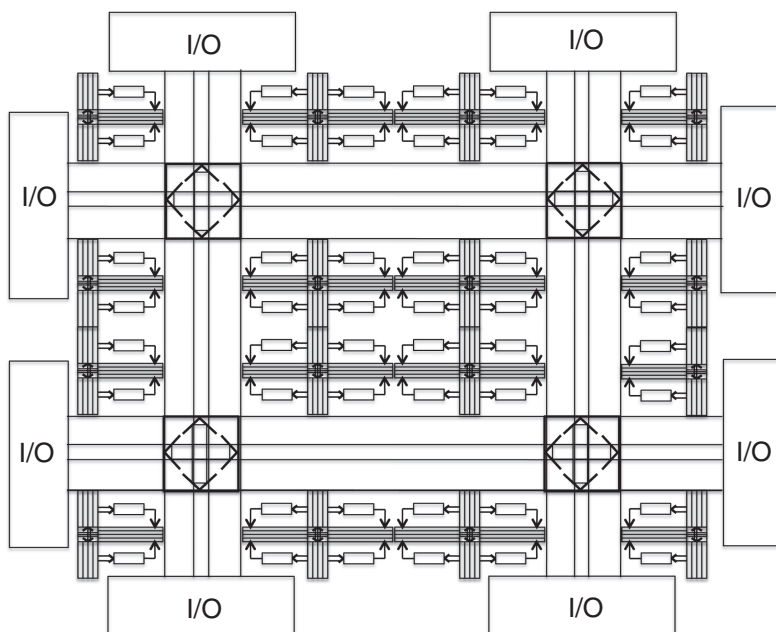


**Fig. 2.7** The fourth representation level: the context

The fourth level completes the representation of the FPGA *context* architecture introducing the *input/output blocks* dedicated to the communication with the FPGA architecture external word. The model is illustrated in Fig. 2.7 where the input/output modules are connected to the principal FPGA interconnection, indicated in Fig. 2.7 as V2 and H2. The interconnections of the third and fourth levels are classified as global and they characterize the global routing of the FPGA architecture.

## 2.2 FPGA Configuration Technology

An FPGA architecture is composed using electrically programmable switches, where the dimensions, the capacity, and the resistance characterize the various models. In this section, we describe the principal manufacturing technologies, principally focusing on the volatility, reprogrammability, and complexity of the manufacturing process.

### 2.2.1 Floating Gate Technology

The *floating gate* technology is realized as an early technology based on the erasable programmable read only memory (EPROM) that consists of memory cells that can be erasable using ultraviolet ray or based on the electrically erasable programmable read only memory (EEPROM) that are electrically erasable. In particular, this kind of approach is used in the model manufactured by Actel. The programmable switch illustrated in Fig. 2.8 consists of a *floating-gate avalanche-injection MOS FAMOS* that can be permanently disabled by injecting a charge into the *floating gate*. By applying a high voltage difference between the gate and the drain of the FAMOS transistor, it is possible to obtain a transition of high-energy electrons from the transistor junction to the isolated floating gate. At the end of the transition, the charge remains indefinitely in the floating gate and the transistor remains permanently polarized since it does not exist an electrical connection with the gate. The FAMOS transistor, if not programmed, is used in order to bring at low level the bit line when the word line is at high level. This approach can be further used to make connections between the word and the bit line; for example, it is especially used in order to implement the wired logic functionalities such as the Wired-AND. A Wired-AND logic consists in the connections of two or more wires with the supply voltage source through a unique resistance. The output corresponds to the product of the single signals with the same functionalities obtained using an AND gate. For this reason, the FAMOS transistor can be used for both the logic and the routing resources into an FPGA architecture.

The best advantage of the EPROM technology is its non-volatility. With respect to the SRAM technology, it is not requested any permanent memory outside the FPGA chip in order to store the programming data. However, the EPROM technology requires three additional manufacturing processes, in order to insert the elevated

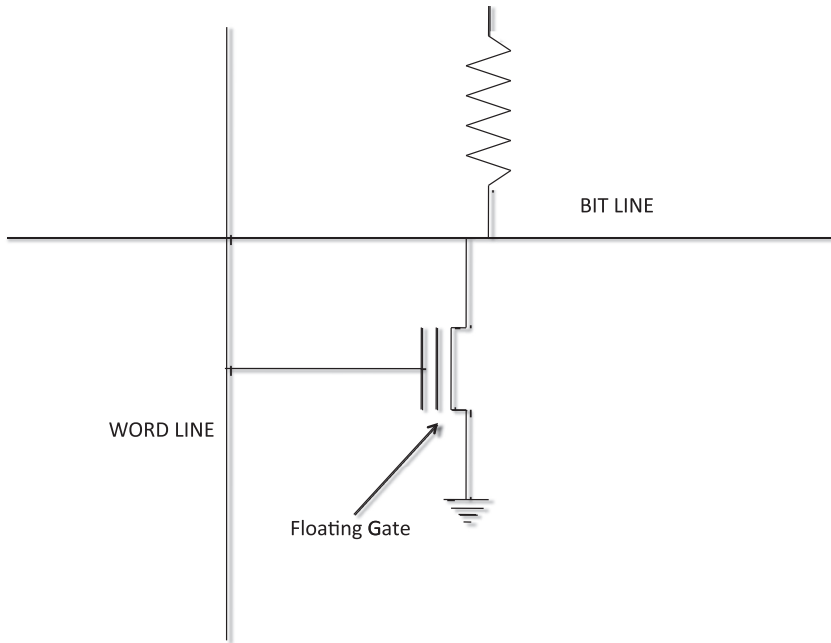


**Fig. 2.8** The floating gate programming technology

*pull-up* resistors. Another drawback is the high static power consumption, due to the pull-up resistance of each EPROM cell transistor [11].

### 2.2.2 Antifuse Technology

An *antifuse* device consists of two terminals with, in between, a high resistance for the not-programmed state. If a high voltage (i.e., a voltage comprised between 11 and 20 V) is applied between the two terminals, the resistance is melted and creates a permanent connection between the two terminals. Programming an antifuse requires an external circuitry providing high-level voltages and currents.

The best advantage of the antifuse technology resides in the small cell dimensions. However, this advantage is generally reduced by the incurred dimensions of the transistors necessary to program the device, since they must manage elevated voltage levels.

### 2.2.3 SRAM Technology

The main characteristic of a static RAM (SRAM)-based FPGA is a configuration memory built with SRAM cells able to control both the multiplexers and the *pass transistors* used for the connections as illustrated in Fig. 2.9. The functional

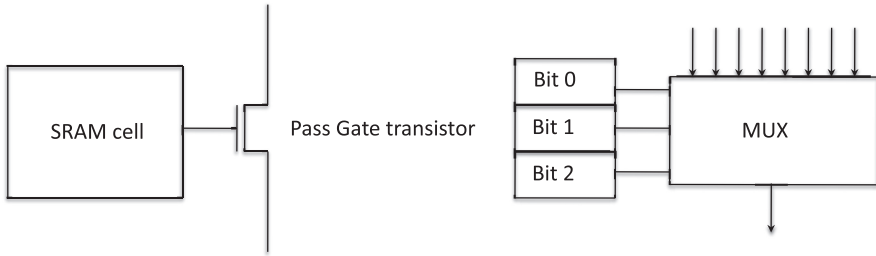


**Fig. 2.9** The Static RAM configuration technology

principle is based on the logic value stored by the SRAM cell: when a logic “1” is stored, the pass transistor creates a connection between two hardwired segments; vice versa when a logic “0” is stored into the SRAM cell, the pass transistor is open and it presents a high resistance between the two hardwired segments. The state of the SRAM cell connected to the select line of the multiplexer illustrated in Fig. 2.9 controls which input of the multiplexer is connected to the output [14].

The SRAM is volatile; therefore, the FPGA architectures using SRAM cells require to be configured at each power-on. This is a relevant constraint, since it is mandatory that an SRAM-based FPGA system adopts an external and permanent memory, like a programmable ROM (PROM), or a microprocessor-based system that allows to program the FPGA’s configuration memory. However, the main advantage of this technology is the possibility to program the device an infinite number of cycles also in a short period.

## 2.3 The Logic Block

The logic blocks of the FPGA architectures, also known as *configurable logic block*, principally differ for their dimensions and their functional implementation capabilities. The difference of the several logic blocks, as in part described in the previous sections, can be classified referring to the *granularity* of the data. The *granularity* could be defined as the number of boolean functions implemented by the logic or with the total number of transistor. In several FPGA architectures it is difficult to distinguish between the interconnections and the logic blocks given their connectivity; for simplicity we classify the FPGA model into two possible categories: logic blocks with a fine granularity, also called *fine grained*, and logic blocks with a coarse granularity, or *coarse grained*.

### 2.3.1 Fine-Grain Logic Blocks

The fine-grain logic blocks consist of few interconnected elements. An example of a fine-grain logic block is the one manufactured by Plessey [13] as illustrated in

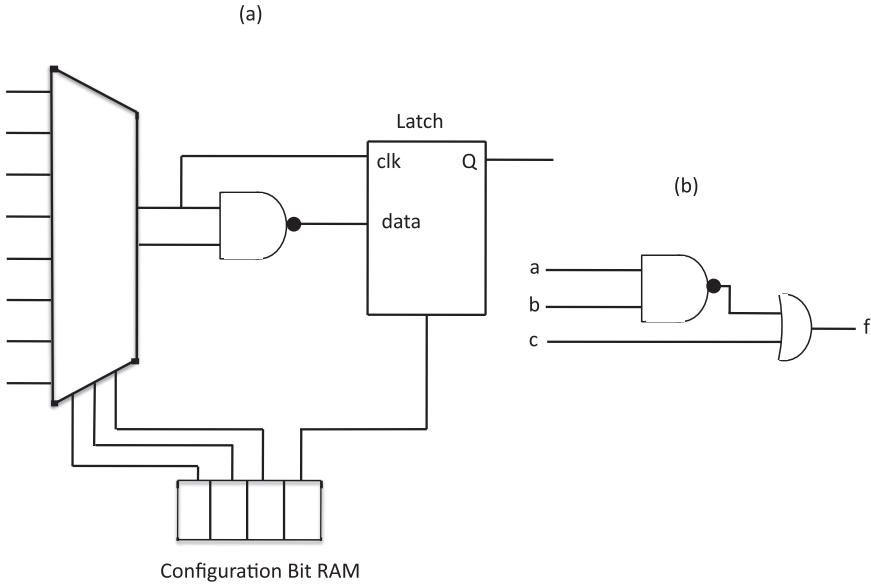


**Fig. 2.10** Fine-grain logic block manufactured by Plessey

Fig. 2.10. The logic is formed connecting the NAND gate to the multiplexer in order to create the desired logic function (combinational or sequential). The SRAM cells controlling the configuration memory bits consist of four bits, three of them configure the multiplexer while the last bit is used to control the latch. For example, in case the logic block is configured in order to implement the logic function  $f = ab + c$ , in this case the latch is not necessary; hence, the correspondent configuration bit will be programmed to a logic value capable to make the latch unused. The main advantage of the fine-grain logic block is the full usability of all the components. However, nevertheless, it is easy to efficiently use few logic gates, it results drastically disadvantageous managing a high number of routing segments and switching elements: These components generate delays and creates a huge increase in the device size.

### 2.3.2 Coarse-Grain Logic Blocks

The coarse-grain logic blocks consist of a higher number of components with respect to the fine-grain logic blocks, principally including multiplexers, NAND gates, and LUTs. The Xilinx company is one of the top manufacturer of FPGA architecture embedding coarse-grain logic blocks. In Fig. 2.11 is illustrated the configurable logic block (CLB) of a generic Xilinx device. The basic components

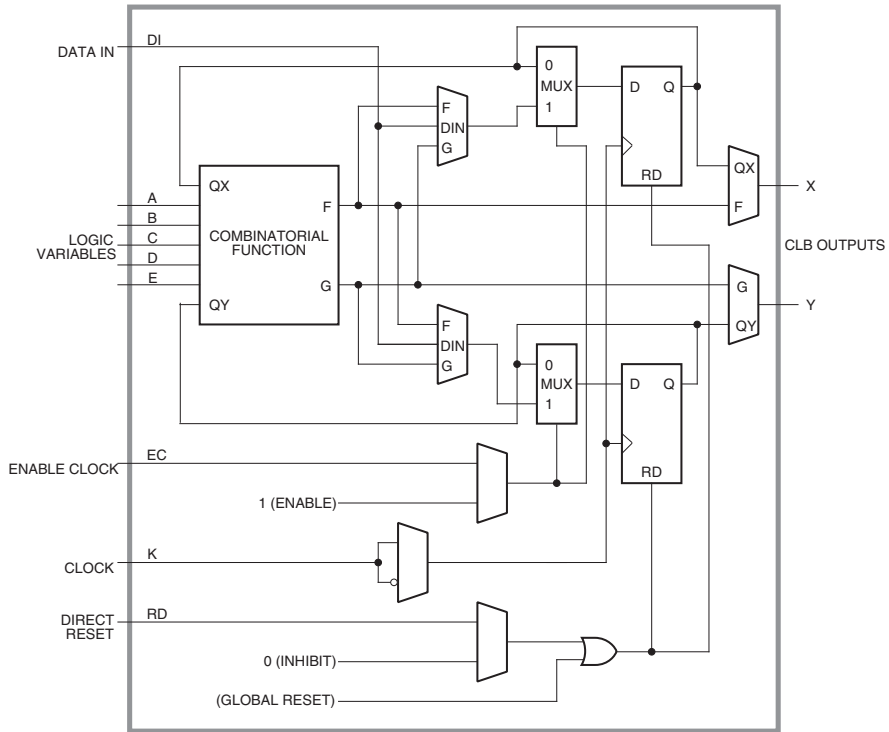


**Fig. 2.11** Coarse-grain logic block manufactured by Xilinx

contained in the CLB are the *generator of combinational logic functions* and two D-type flip-flops whose outputs can be directly connected to the inputs of the generator of combinational logic functions by using the internal routing existing in the CLB. The generator of combinational logic functions consists of two look-up tables (LUTs) with four-inputs that can be used separately or combined into a unique logic function. The partition of the inputs could be automatically made through a partitioning tool generally used during the logic synthesis or manually performed in case particular constraints would be implemented. On the other hand, all the sequential components have a common clock signal; besides the flip-flops that cannot be used as latches have common signal of *clock-enable* and an asynchronous reset. Any asynchronous preset can be obtained using the asynchronous reset if the data are stored in the form of active low logic level.

The principal advantage of the coarse-grain logic block is the possibility to implement complex logic functions with few elements, reducing the need for a high number of programmable interconnections. However, it is difficult to obtain a high efficiency for the used resources, even if the decrease in the functional density is not a prohibitive factor.

## 2.4 The Routing Architecture

The routing architecture of an FPGA device defines how the programmable switches and the wiring segments that made the interconnections are placed in order to realize the complete logic functionality of the FPGA device [12]. To properly describe the FPGA routing architecture we introduce the following definitions:

*Wire segment* It is a programmable connection through a *switch*. One or more switches can be connected to the same wire segment. Each access point of a wire segment has at least one switch connected to itself.

*Track* It is a sequence of one or more *wire segments* that creates a connection between two points of the routing topological structure.

*Routing channel* It is a group of parallel tracks.

The model of the routing architecture, as illustrated in Fig. 2.12, has two basic structures: the *connection block* and the *switch block*. Each connection block allows

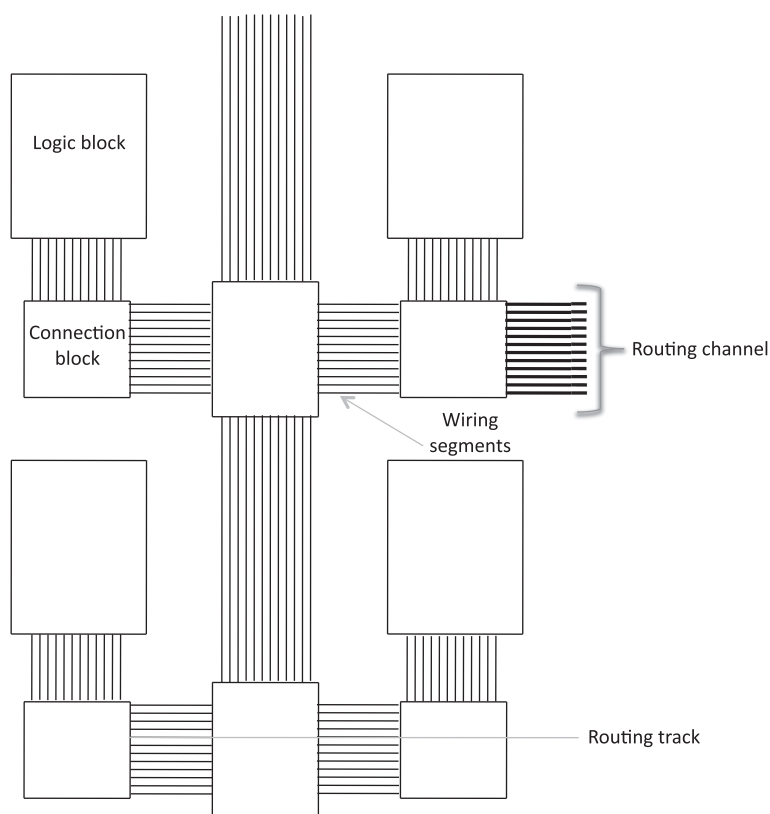


**Fig. 2.12** The model of the routing architecture

the connection between the inputs and the outputs of a logic block and wire segments of a routing channel, while a switch block allows the connection between horizontal and vertical wire segments adopting a connectivity model on the four sides. The connection blocks typically connect only a single part of the whole track passing through a block. The connections on the four sides of the logic blocks are implemented by means of pass transistors for the output pins and multiplexers for the input pins, thus reducing the number of SRAM cells needed of each pin, and therefore reducing the whole dimension of the FPGA device. While the logic block pins are connected through the connection block, the switch block realizes the connection between segments intersecting horizontal and vertical channels.

There are four types of wire segments in an FPGA architecture:

1. *General-purpose* interconnections: They consist in wire segments passing through the switches internally to the switch block.
2. *Direct* interconnections: They consist in the wiring segments connecting the outputs of each logic block directly to the neighboring logic blocks.
3. *Long line* interconnections: They consist in wire segments connecting all the switch blocks in the context of the FPGA architecture.
4. *Clock line* interconnections: They consist in unique wiring segments that connect all the sequential elements of the FPGA architecture. The clock lines are divided into groups depending on the clock domains they belong to.

### 2.4.1 The Switching Elements

Each switching element of an FPGA architecture consists of a variable number of contact called *programmable interconnect points* (PIPs). Considering a switch block, the set of PIPs defines the connection of each wiring segment. PIPs, according to the technology of the FPGA configuration memory, can be implemented in different ways:

- The *floating gate* technology implements the PIP with the FAMOS transistor and pass transistor.
- The *antifuse* technology implements permanent PIP with the antifuse.
- The *SRAM* technology has five transistors that form the SRAM cell and a pass transistor creating the contact: the wiring segments controlled by the pass transistor are connected or not, depending on the logic value contained into the configuration memory cell.

Into the FPGA architecture the net of programmable interconnections consists of a set of wire segments that can be interconnected by means of PIPs, where each PIP can be controlled by some configuration memory cell.

There are four kinds of PIPs:

- *Cross-point* PIP
- *Break point* PIP

- *Multiplexer PIP*
- *Compound PIP*

A cross-point PIP connects wire segments localized in different plans: a horizontal segment to a vertical one and vice versa, or, considering the FPGA model introduced, it can connect a segment of the third level to a segment of the fourth level, or between the second and the third levels and vice versa; see Fig. 2.13.

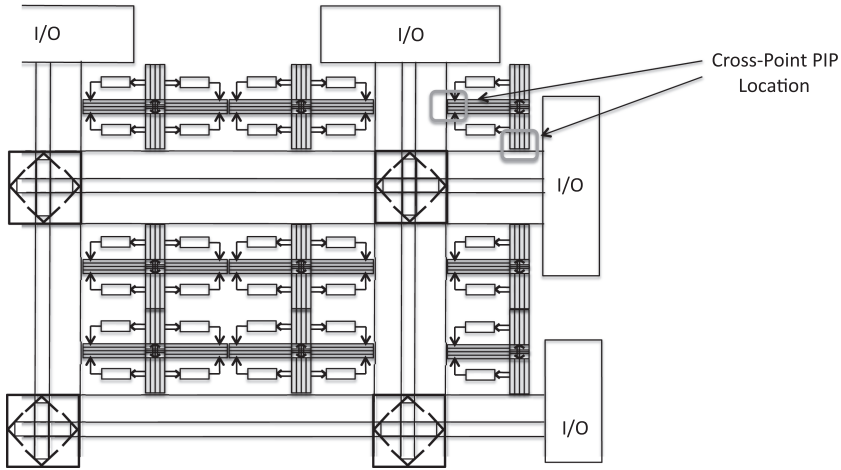


**Fig. 2.13** Location of the cross-point PIP

A break point PIP connects two segments of the same plan: two horizontal segments or two vertical segments. A break point PIP can be located between the third and the fourth levels of the FPGA representation model as illustrated in Fig. 2.14.

A multiplexer PIP (MUX PIP) is classified into two different type: *decoded* MUX PIP and *non-decoded* MUX PIP. A decoded MUX PIP is a set of  $2^k$  cross-point PIPs connected to a common output wiring segment. Each decoded MUX PIP is controlled by  $k$  bits stored in the configuration memory cells; in particular, the input wiring segment addressed by the configuration memory bits is connected to the output segment while the decoding logic is embedded into the bit of the configuration memory and the PIP's pass transistor. A non-decoded MUX PIP consists of a single configuration memory cell for each pass transistor, by this way  $k$  wiring segments are controlled by  $k$  configuration memory bits. As previously underlined, the wiring segments into the FPGA architecture interconnection network include both the local and global routing resources. Where the global routing resources are dedicated to the connection between the CLBs and the input/output blocks, while the local routing, specific for a single CLB, is used to connect a logic block to the global routing resources or to the adjacent logic blocks. The PIPs used for global routing resources are the cross-point PIPs and the break point PIPs, while PIPs of the local routing resources belong to the multiplexer PIP type, as illustrated in Fig. 2.15.

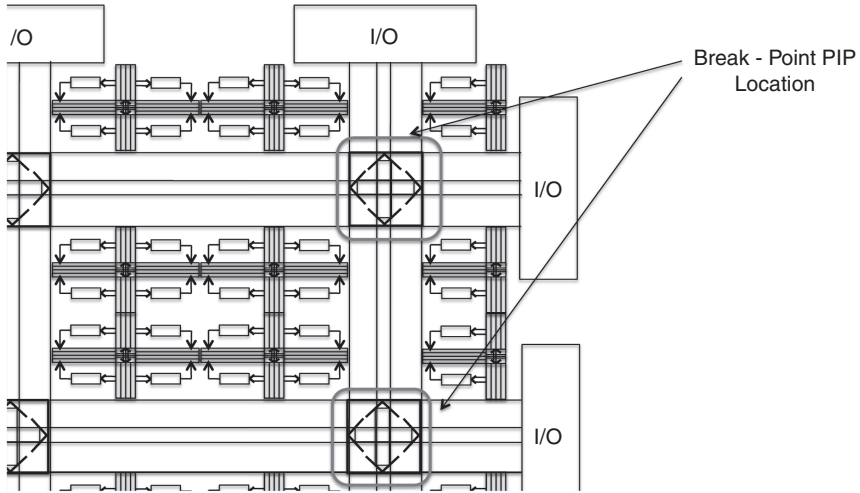


Fig. 2.14 Location of the break-point PIP

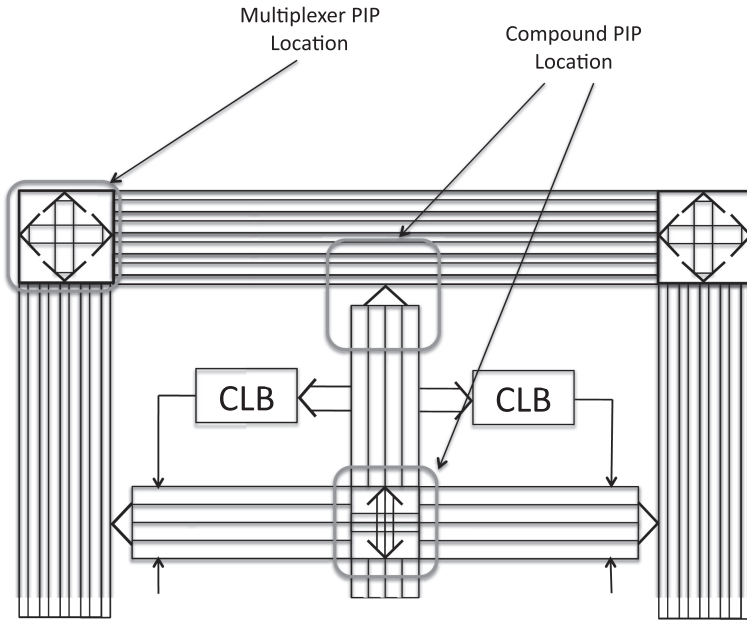


Fig. 2.15 Location of the multiplexer and compound PIPs

Finally, the fourth PIP category consists of the compound PIPs. The compound PIPs are a combination of four cross-point PIPs and two break point PIPs, each one controlled by a particular group of bits into the configuration memory. The most recent FPGA architectures are built with buffered non-decoded MUX PIPs. This

structure is adopted in order to prevent the signal degradation due to the resistance of each pass transistor.

### 2.5 The Input/Output Blocks

The input/output blocks implement the interface between the pins of the *package* and the interconnections and logic blocks structure of the FPGA architecture matrix. This interface provides the other routing resources used on the frontiers of the device with the external word and that allow guaranteeing the connectivity of the FPGA architecture with the external signals. Communication with the outside signals is made by *three-state buffers* whose direction and polarity are determined by specific configuration memory bits. Each driver is configurable in order to have high or low *slew rate* and to manage different activation levels, depending on the voltage or current standard adopted. Besides, the drivers support different kind of communication standards that allow interfacing the architecture in electronic systems of different nature. The *pads*, which are connected to the drivers, are protected against possible electrostatic discharge (ESD) and *over-voltage* transitions. Within the input/output blocks there are several flip-flops, as illustrated in Fig. 2.16, that can be configured as D-type *edge-triggered* flip-flops or *level-sensitive* latches.

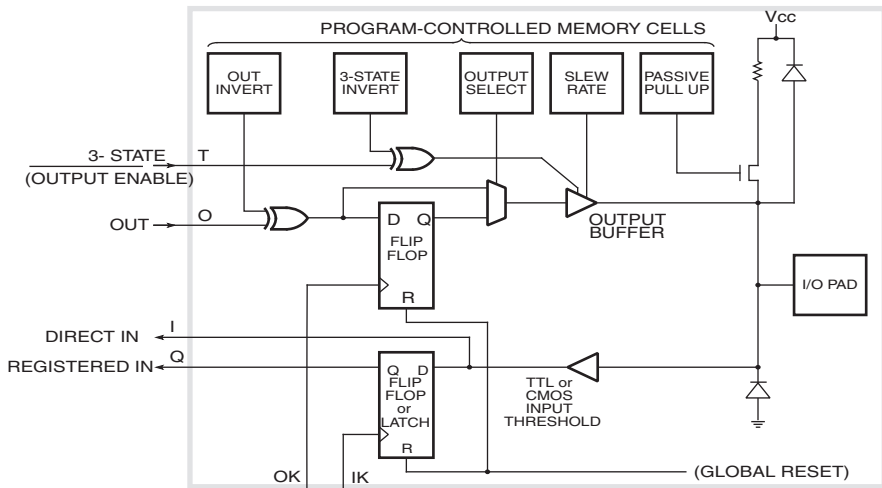


Fig. 2.16 An input/output block

### 2.6 The Configuration Memory

The configuration memory contains all the information needed to program the circuit on the FPGA architecture. This information is also called *bitstream*. As illustrated in the previous sections, each resource within the FPGA architecture is

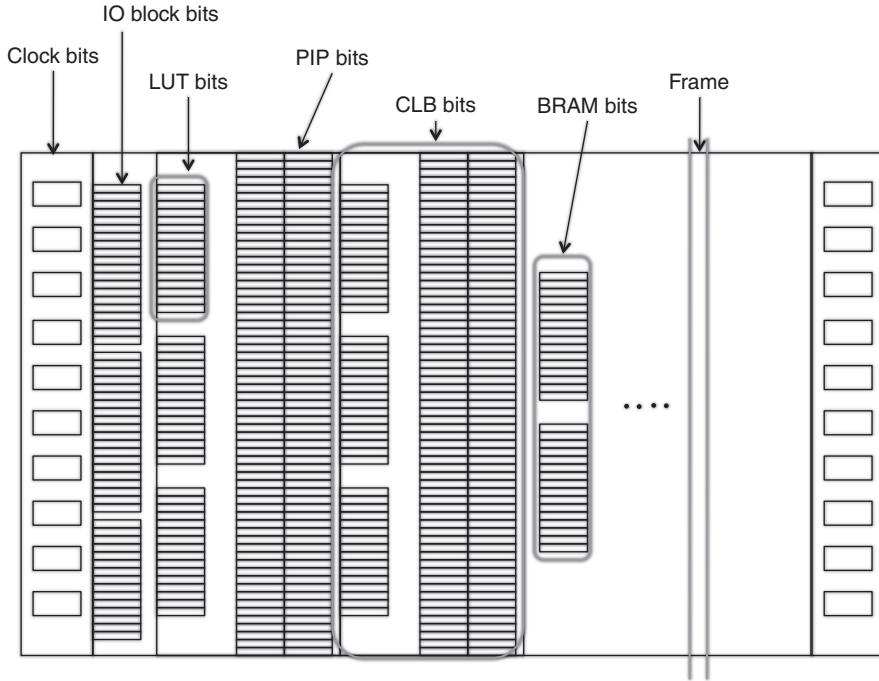


**Fig. 2.17** The general structure of the bitstream

controlled by specific configuration memory bits. The bitstream is generally organized in *frames* and each frame is correspondent to a set of resources localized in a contiguous area of the FPGA architecture, as illustrated in Fig. 2.17. These frames have a regular structure depending on the kind of interested resources. A finite state machine, dedicated to the transfer and localization of each frame in the correct area of the FPGA's configuration memory, controls the bitstream loading. The configuration flow, by which the configuration memory is loaded with the bitstream data, basically consists of four phases:

1. *Power up*: The power supply is applied to the FPGA device.
2. *Device initialization*: The configuration memory is initialized by the configuration state machine dedicated to the bitstream loading.
3. *Configuration load*: In this phase, the device receives the configuration bit logic values that are progressively stored into the configuration memory through the dedicated configuration state machine. All the configuration events happen during the positive edge of the clock signal.
4. *Start up*: When all the configuration memory bits have been received, the configuration state machine brings the device to the *start-up* state: the circuit is actually implemented in the FPGA and it is ready to perform its operations.

## 2.7 An Overview of the Architecture of Modern FPGAs

Across the years, FPGAs underwent a substantial evolution process that resulted in very complex devices, embedding many different resources. The architecture of a typical modern FPGA is depicted in Fig. 2.18, where we can recognize logic and interconnection resources that are used to implement combinational and sequential

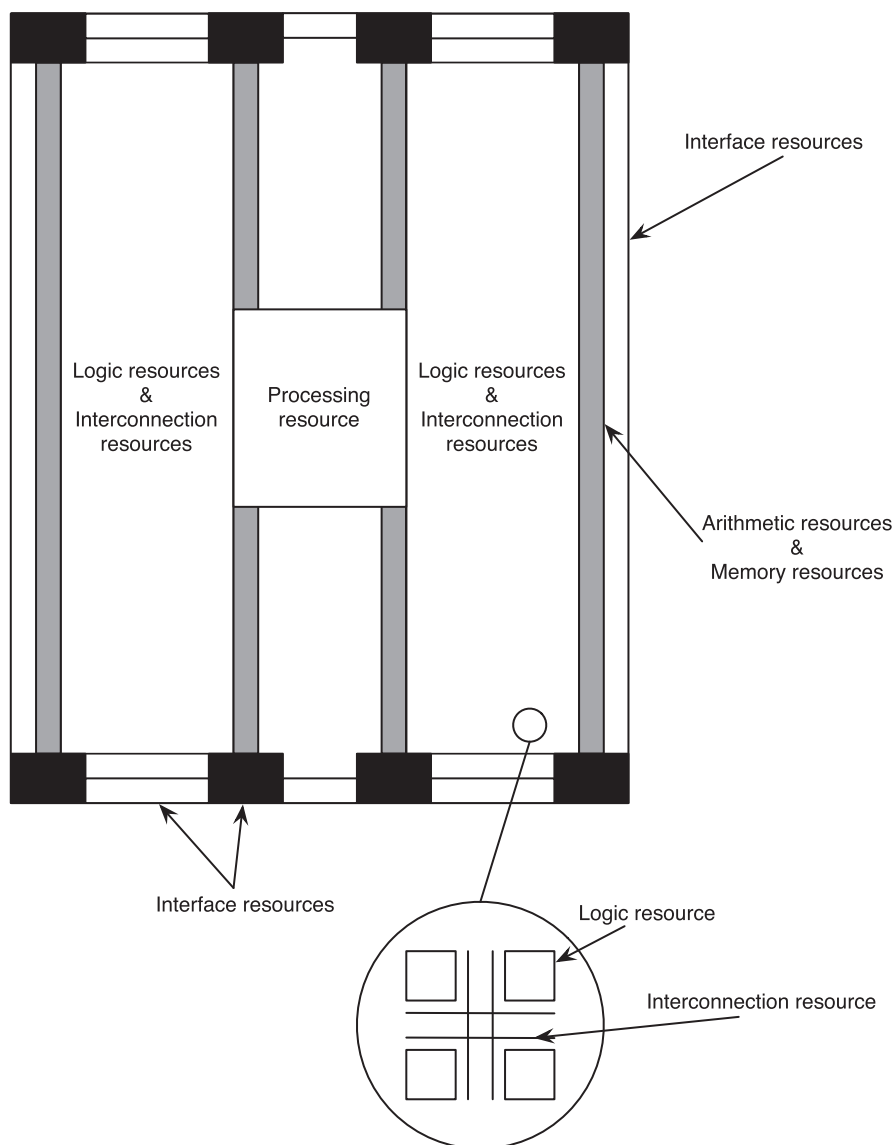


**Fig. 2.18** Architecture of a modern FPGA

functions and to deliver signals among functions; memory resources, that are used to implement random access memories, may be used for data storage by the circuit the FPGA implements; arithmetic resources, like digital signal processing (DSP) blocks, that are used for improving the performance of some computations (for example, multiply and accumulate operations); processing resources (e.g., processor hard cores like the PowerPC found in Xilinx Virtex-II, Virtex-4, and Virtex-5 devices [20]) that are used for building processor-based systems within a single FPGA; and interface resources that are used for delivering data to and from the FPGA using a variety of protocols spanning from the simple buffered short-circuit from the outside and the inside of the FPGA to complex high-speed serial link (like the Xilinx RocketIO [16]). The aim of this section is to outline all these kinds of resources.

### 2.7.1 Logic Resources

FPGA have been originally intended as a means for implementing digital circuits and, therefore, they have been equipped with resources that allow implementing both combinational and sequential functions; moreover, the resources are programmable in such a way that designers are free to implement any logic function.

To achieve such a goal, different solutions have been pursued by different manufacturers.

#### 2.7.1.1 The Xilinx Configurable Logic Block

The Xilinx configurable logic block (CLB) [19] is the key element for implementing sequential and combinational logic in Xilinx devices. It encompasses a number of slices (each slice includes the configurable resources for implementing boolean functions, flip-flops, and carry-propagation logic) and some interconnection resources. As an example, we report in Fig. 2.19 the architecture for the CLB found in Virtex-4 devices where we can recognize four slices, grouped two by two. Each group is connected to interconnection resources dedicated to fast propagation of carry signals. Moreover, each slice is connected to a switch matrix to route signals toward the rest of the FPGA and to local interconnection resources to route signals toward neighboring CLBs.

Each slice contains several elements:

- At least one  $n$ -input memory that can store  $2^n$  bits. This memory can be configured in several operational modes:
  - Look-up table (LUT) mode: It is used to implement single-output  $n$ -input combinational functions by storing the truth table of the function in the LUT.
  - RAM mode: It is used to implement  $2^n \times 1$  random access memories.
  - ROM mode: It is used to implement  $2^n \times 1$  read-only memories.
  - Shift-register mode: It is used to implement one  $2^n$ -bit shift register.

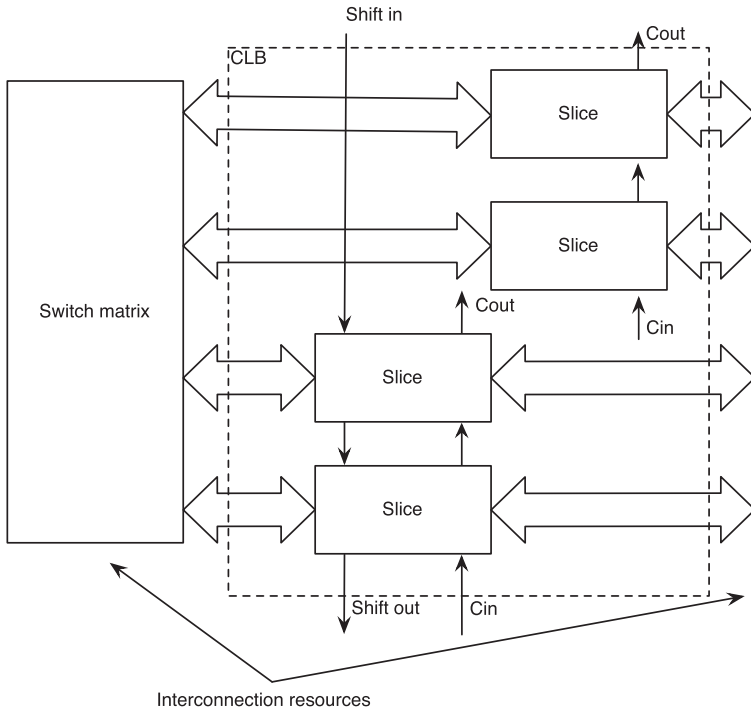


**Fig. 2.19** The Xilinx Virtex-4 CLB

Optionally, the  $n$ -input memory can also be configured to operate as single port of dual-port memory.

- **Carry-propagation logic:** This is used to propagate the carry signals coming from arithmetic operations, using dedicated high-speed lines.
- **At least one flip-flop:** This is used for implementing state register bits for finite state machines or for data storage. Several parameters can be configured for defining the operations of the flip-flop (the operational mode among latch and flip-flop, the reset value, the polarity of the reset signal).
- **Glue logic:** This is used to route signals inside the slice and to compute the complement of control signals (thus making possible to implement positive-edge and negative-edge triggered behaviors).

As an example, the Xilinx Virtex-4 family of devices exploits CLBs with four slices, each offering two LUT and two flip-flops. The CLB can thus be used for implementing eight flip-flops, and up to eight 4-input combinational functions in case all the slices are used in LUT mode, or for storing up to 64 bits in case all the slices are used in RAM/ROM mode, or for working as one 64-bit shift register in case all the slices are used in shift-register mode.

CLBs are arranged in an array, thus replicating the same structure in a regular fashion. The size of the array defines the resource availability of the device. For

example, the Xilinx Virtex-4 XC4VLX device is available in size ranging from  $64 \times 24$  CLBs (offering 12,288 LUTs and flip-flops), up to  $192 \times 116$  CLBs (offering 178,176 LUTs and flip-flops).

### 2.7.1.2 The Actel VersaTile

The VersaTile is the Actel [1] solution for implementing a configurable element able to work as a combinational function or as a sequential element. The VersaTile is available in the IGLOO and ProASIC3 devices and, as depicted in Fig. 2.20, is composed of 4 inputs, 2 outputs, 4 2-input multiplexer, 11 gates (1 NOR and 10 NOT), and a number of programmable switches, and hardwired interconnections among these elements.

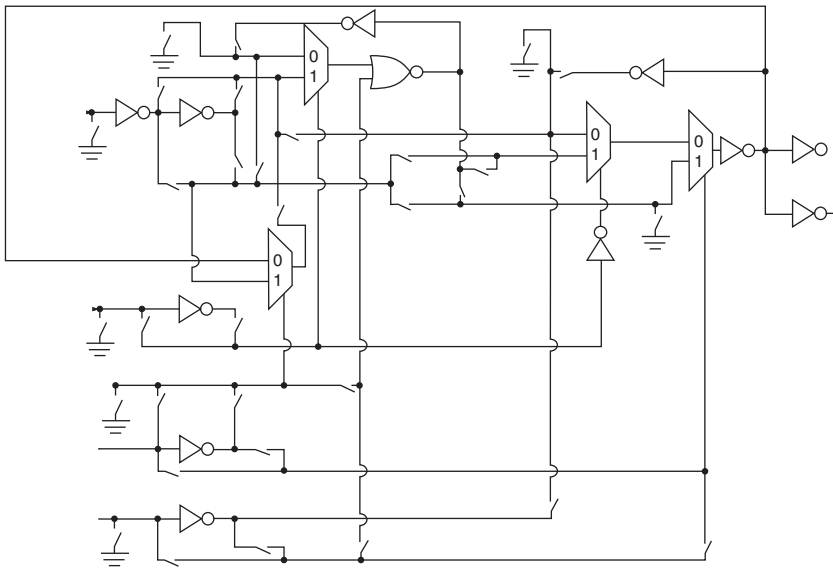


**Fig. 2.20** The Actel VersaTile

By setting the configuration (open or closed) of each programmable switch the designer can use the VersaTile to implement

- any 3-input combinational function;
- one latch with clear or set signal; and
- one D flip-flop with clear or set signal.

VersaTile are replicated in a regular fashion, so that each FPGA carries an array of VersaTiles, whose total number may range from few hundreds (e.g., 384 as in the case of Actel A3P15 device) up to few tens of thousands (e.g., 75,264 in the case of Actel A3PE3000).

### 2.7.1.3 Comments

When comparing the possible different architectures for logic resources, two examples of which have been presented in the previous sections, we can see that there is not a prevailing solution neither among different manufacturers (e.g., Xilinx versus Actel), nor within the same manufacturer (e.g., Xilinx Virtex 4 versus Xilinx Virtex 6).

The solution based on LUTs and discrete flip-flops, as in the Xilinx case, is probably more efficient from the performance point of view as each of these elements can be optimized for one specific function. However, having a huge number of heterogeneous components increases the complexity of the overall design. On the contrary, the solution based on a single, general-purpose, and highly customizable element, as in the Actel case, contributes to simplify the overall architecture, but may have drawback on the efficiency of the obtained device as compromises must be accepted.

## 2.7.2 Interconnection Resources

Interconnection resources are the infrastructure that FPGAs offer to exchange information between logic, memory, arithmetic, processing, and interfacing resources [15]. They can be seen as programmable “wires” that in principle can connect any resources together thus making possible the implementation of any circuit topology.

Figure 2.21 depicts a conceptual model for the interconnection resources, where we can find two types of elements:

- Wires that implement one-to-one hardwired connections between resources. In Fig. 2.21 we can see a number of wires that connect logic resources to router and pairs of routers. Wires are not programmable, as they implement a fixed connection between resources, which was assigned to them by the designers of the FPGA.
- Routers that implement programmable many-to-many connection between resources. In general, each signal connected to a router can communicate via a programmable switch (i.e., a pass transistor) to any other signal of the router. Different configurations are possible, and Fig. 2.21 shows a possible example, where signal on one side of the router (e.g., signal A) is connected to all the signals lying on the other sides of the router.

As a matter of fact, the number of resources that have to be interconnected is constantly growing at each new generation of FPGA. Moreover, when implementing very complex designs on large FPGAs, it is very likely that data have to be exchanged among resources that are physically placed in faraway locations. As a result a hierarchical interconnection architecture is needed. Different solutions have been developed by FPGA manufacturers. Among them, the solutions implemented by Xilinx, Altera, and Actel are the most interesting.

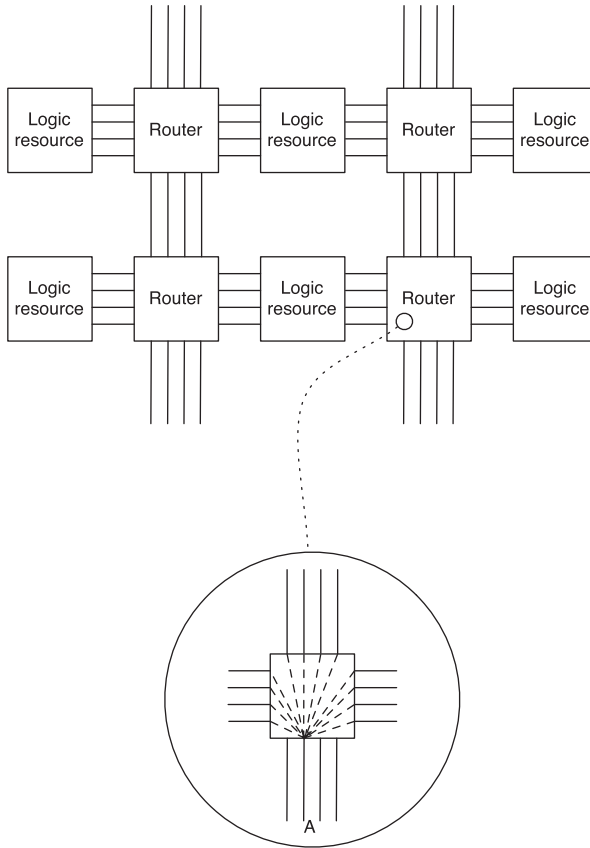


**Fig. 2.21** Conceptual model for the interconnection resources

### 2.7.2.1 The Xilinx Interconnection Resources

The Xilinx interconnection resources [15, 17] offer a hierarchical structure to designers, comprising the following:

- Routers that in the Xilinx terminology are known as switch matrix. Any switch matrix offers a given number of hardwired interconnections to other switch matrix, and in particular assuming a regular array of  $N \times M$  CLBs each connected to a switch matrix:
  - A number of so-called long lines that connect all the switch matrices along the full width and height of the FPGA.
  - A number of so-called hex lines that connect to every switch matrix in position  $(x, y)$  to the switch matrix in position  $(x + i, y + j)$  with  $i \in \{-6, -3, 3, 6\}$  and  $j \in \{-6, -3, 3, 6\}$ .
  - A number of so-called double lines that connect to every switch matrix in position  $(x, y)$  to the switch matrix in position  $(x + i, y + j)$  with  $i \in \{-2, -1, 1, 2\}$  and  $j \in \{-2, -1, 1, 2\}$ .

- A number of direct interconnections between every switch matrix in position  $(x, y)$  and the nine switch matrices surrounding it.
- Direct interconnection between slices and switch matrix, as depicted in Fig. 2.19. Connecting to switch matrix, any CLB can communicate to whichever CLB in the entire FPGA.
- Direct interconnection between slices in the same CLB, as depicted in Fig. 2.19. Thanks to this dedicated, fast interconnection links, it is possible to minimize the communication delays when exchanging information among functions implemented in the same CLB, without wasting the global interconnection resources within switch matrix.

By exploiting the Xilinx architecture, two functions can be interconnected by traversing a number of interconnection resources that depend on the placement of the design on the device. If the two functions are placed in the same CLB, they will benefit from the availability of fast dedicated interconnection links. On the contrary, if the design is very dense, resulting routing is very congested; they may have to resort to multiple links traversing a number of switch matrices. As a result, in case of very dense design, with very congested routing, the performance of the circuit may be limited by the need for traversing several switch matrices before reaching the end destination.

### 2.7.2.2 The Altera Interconnection Resources

Also in the Altera devices we can find a hierarchical organization of interconnection resources [9, 15], which are composed as follows:

- Local interconnections: Altera logic resources are clustered in logic array blocks, each comprising 10 identical logic elements (LEs) able to implement combinational and sequential functions by offering one 4-input LUT and one flip-flop. LEs within a LAB are directly connected using dedicated, fast local interconnection so that functions placed on the same LAB do not have to cross the global interconnection resources to exchange data.
- Vertical/horizontal channels: The channel contains wires of length 4, 8, 16, and 24 that connect to LAB along the entire length of the wire. As LABs are regularly arranged in an  $N \times M$  array, we have that channels offer dedicated wires to each LAB within 4, 8, 16, and 24 columns/rows from a given LAB.

### 2.7.2.3 The Actel Interconnection Resources

The Actel interconnection resources [1] offer hierarchy of four types of interconnections:

- Local resources: They are dedicated interconnections that link the outputs of each VersaTile to the inputs of each of the eight surrounding VersaTiles.
- Long-line resources: They are interconnections intended for supporting delivery of data to longer distances and for higher fanout connections. They have length

spanning one, two, or four VersaTiles; they run both vertically and horizontally and cover the entire device. Each VersaTile can drive long-line resources, which can access every input of every VersaTile.

- Very-long line resources: They span the entire device and are intended for implementing very long or high-fanout interconnections. Their length spans 12 VersaTiles in the vertical direction on each side of a given VersaTile and 16 VersaTiles in the horizontal direction on each side of a given VersaTile.
- VersaNet networks: They are low-skew, high-fanout nets accessible from the outside of the FPGA or from internal logic. They are intended for clocks and resets distribution, and, in general, for any high-fanout nets requiring minimum skew.

### 2.7.3 Memory Resources

As the number of logic and interconnection resources available in modern FPGAs is growing, these devices are becoming a very appealing replacement for ASICs in certain type of application. As a result, the FPGA is no longer intended for implementing just glue logic; it is becoming the device for building an entire system. The availability of on-chip memory resources is boosting this shift. Indeed, any computing system needs to store data and having the possibility of keeping them on-chip contributes to significantly speed up the performance of the system, as most of the task is performed on-chip without the need for massively accessing to outside memories via less efficient interfaces.

As a result, modern FPGAs embed configurable memory resources that can be seen as static RAM arrays distributed among the whole device. The RAM array offers few tens of  $k$  bits of data (e.g., 18 and 36 kb for Xilinx Virtex-6 devices [22], 9 and 144 kb for Altera Stratix IV devices [5]), which can be arranged in several different “aspect ratios.” As an example, the 36 kb RAM available in Xilinx Virtex-6 devices can be configured as  $32\text{ kb} \times 1$ ,  $16\text{ kb} \times 2$ ,  $8\text{ kb} \times 4$ ,  $4\text{ kb} \times 9$ ,  $2\text{ kb} \times 18$ ,  $1\text{ kb} \times 36$ , or  $512\text{ b} \times 72$  in dual-port mode. Moreover, memory resources can be cascaded to implement larger blocks.

Moreover, the memory resources can support at least three operational modes:

- single-port mode;
- dual-port mode; and
- first-in first-out (FIFO) mode.

Other operational modes can optionally support in the following:

- shift-register mode, which is particularly convenient when the memory resource is used for implementing data storage in digital signal processing (DSP) algorithms and
- ROM mode.

### 2.7.4 Arithmetic Resources

As more and more complex applications are demanded to FPGA, manufacturers recognized the benefits on embedding hardwired arithmetic resources in their devices, implementing specific recurring functions that are common to many classes of applications. The arithmetic resources most widely adopted by FPGA manufactures (e.g., Xilinx, Altera, and Actel) is the multiply and accumulate (MAC) resource, which is common to many applications like filtering applications, software-defined radio, and radar systems. By exploiting the arithmetic resources, FPGA user can save logic resources to implement other non-recurring functions, thus making possible building more complex systems on a single FPGA device.

Different FPGA manufacturers followed different implementation paths in developing their MAC resource. In general, the arithmetic resources they developed can provide at least one of the following functionalities:

- Multiply
- Multiply and accumulate
- Multiply add
- Three-input add
- Barrel shift
- Wide-bus multiplexing
- Magnitude comparator
- Bit-wise logic functions
- Pattern detect
- Wide counter

As an example of arithmetic resource we can consider the Mathblock Actel recently introduced with the RTAX-DSP family of devices [2]. The architecture of the Mathblock is shown in Fig. 2.22. The arithmetic resource allows the hardwired logic for implementing the following function:

$$P_n [40 : 0] = P_{n-1} [40 : 0] \pm (A \times B) \quad (2.1)$$

### 2.7.5 Processing Resources

Similar to what happened with arithmetic resources, FPGA manufacturer recognized the importance of embedding processing resources under the form of hardwired processor cores in the most advanced devices. By using hardwired processor cores, designers can implement computing systems on a single FPGA device. Moreover, they can have the possibility of building complex architectures where custom hardware accelerator implemented using FPGA logic, and arithmetic resources cooperate with embedded software running on the processor core the FPGA embeds. As all the modules are integrated in the same chip, a substantial performance

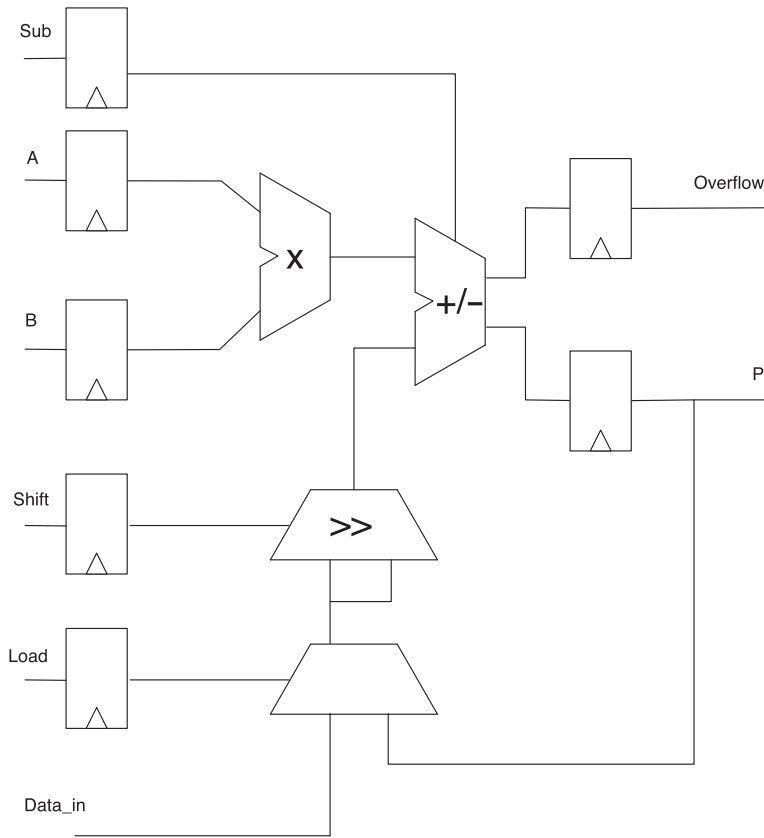


**Fig. 2.22** Architecture of the Actel Mathblock

improvement is achieved with respect to implementations where discrete chips (one for the hardware accelerator and one for the processor) are used.

The two most important example of processor cores embedded in FPGA are the PowerPC processor available in Xilinx Virtex 2-pro [20], and some versions of Xilinx Virtex-4 and Virtex-5 devices [18], and the Cortex-M3 processor available in Actel SmartFusion devices [3].

The Virtex-5 FXT [21] series of devices from Xilinx offers up to two PowerPC 440 cores (Virtex-2 pro and Virtex-4 adopted PowerPC405). Each processor offers the following features, which are schematized in Fig. 2.23:

- A  $5 \times 2$ , 128-bit crossbar switch
- Simultaneous memory bus and processor local bus (PLB) access
- Integrated DMA channels, PLB interfaces, and dedicated memory interface so that FPGA logic resources are not employed to implement embedded computing system

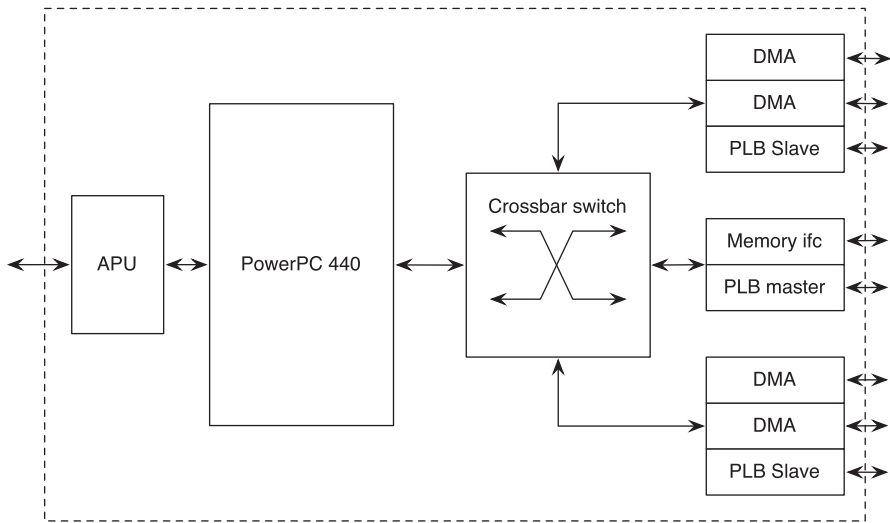


**Fig. 2.23** Architecture of PowerPC 440 adopted in some Virtex-5 FXT devices

- Auxiliary processor unit (APU) controller to integrate hardware accelerators and create custom co-processors
- Non-blocking pipelined point-to-point access to FPGA resources
- Dedicated memory interface port for up to 128-bit data transfer per cycle to offload PLB
- Highly pipelined transmit and receive scatter-gather DMA channels
- User-selectable port prioritization and operating frequencies

The Cortex-M3 processor is an optimized version of the ARMv7-M [6] architecture for low-cost applications. It embeds the following features:

- Hardware single-cycle divide and multiply instructions
- Nested vectored interrupt controller (NVIC)
- Configurable from 1 to 150 physical interrupts; up to 256 levels of priority
- Memory protection unit (MPU)
- Data watchpoint and trace unit (DWT)
- Flash patch and break point unit (FPB) that implements six program break points and two literal data-fetch break points

If we analyze the evolution of the market, we can see that the ARM Cortex architecture is growing interest. Indeed, following the example of Actel with the Cortex M3 in SmartFusion FPGA, Xilinx recently announced a novel architecture based on Cortex A9MP, which offers a dual-core processor each running at 800 MHz [10].

By comparing these examples we can see that there is a great variability in the solutions adopted by FPGA manufacturers, spanning from providing any hard cores at all (like Altera who opted for a soft-processor core, the NIOS II [4], which is implemented using FPGA logic and interconnection resources), to the

microcontroller-like solutions as in the Actel case, to the almost full-fledged high-performance processor as in the Xilinx case.

### 2.7.6 Interfacing Resources

FPGAs are today embedded in a number of high-demanding applications, where moving quickly data inside and outside the device is a crucial issue. To relieve FPGA users from the complexity of implementing standard interfaces using logic resources, and thus also allowing saving device resources, FPGA manufacturers are embedding in their devices hardwired modules providing functionalities like

- PCI Express endpoints
- Ethernet media access controller (MAC) modules
- High-speed serial interfaces (e.g., Xilinx 11.18 Gbps GTH transceivers)

As FPGA can also be appealing for embedded control applications, some device manufacturers are also implementing mixed signals interfacing resources.

## References

1. Actel Corporation, *Fpga array architecture in low-power flash devices*, v1.4 ed., December 2008.
2. Actel Corporation, *Smartgen hard multiplier adder/subtractor handbook*, v1.0 ed., May 2009.
3. Actel Corporation, *Actel smartfusion microcontroller subsystem (mss) user's guide*, 1 ed., May 2010.
4. Altera Corporation, *Nios ii processor reference handbook*, v10.0 ed., July 2010.
5. Altera Corporation, *Stratix iv device handbook*, siv5v1-4.1 ed., March 2010.
6. ARM, *Armv7-m architecture reference manual*, 2008.
7. S. Brown, *Fpga architectural research: a survey*, IEEE Design & Test of Computers **13** (1996), no. 4, 9–15.
8. S. Brown and J. Rose, *Fpga and cpld architectures: A tutorial*, IEEE Design & Test of Computers **13** (1996), no. 2, 42–57.
9. Altera Corporation, *Comparing ip integration approaches for fpga implementation*, Tech. report, February 2007.
10. K. DeHaven, *Extensible processing platform. ideal solution for a wide range of embedded systems*, White Paper WP369 (v1.0), Xilinx, April 2010.
11. H. Haznedar, *Digital microelectronics*, Benjamin/Cummings Pub. Co., San Francisco, 1991.
12. Y. Khalilollahi, *Switching elements, the key to fpga architecture*, Conference Record of WESCON/94. 'Idea/Microelectronics', Anaheim, CA, 1994, pp. 682–687.
13. Plessey, [www.plessey.com](http://www.plessey.com); [www.plesseysemiconductors.com](http://www.plesseysemiconductors.com), Plessey FPGA, November, 1989.
14. J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, *Architecture of field-programmable gate arrays*, Proceedings of the IEEE, vol. 81, Piscataway, USA, 1993, pp. 1013–1029.
15. J. Rose, Abbas El Gamal, Senior Member, and Albert Sangiovanni-Vincentelli, *Architecture of field-programmable gate arrays: The effect of logic block functionality on area efficiency*, Proceedings of the IEEE **25** (1990), 1217–1225.
16. Xilinx, *Rocketio™ transceiver user guide*, ug024 (v3.0) ed., February 2007.
17. Xilinx, *Virtex-ii platform fpga user guide*, ug002 (v2.2) ed., November 2007.

18. Xilinx, *Virtex-4 fpga embedded processor block with powerpc 405 processor*, ds306 (v2.01b) ed., April 2009.
19. Xilinx, *Virtex-6 fpga configurable logic block*, ug364 (v1.1) ed., September 2009.
20. Xilinx, *Powerpc 405 processor block reference guide*, ug018 (v2.4) ed., January 2010.
21. Xilinx, *Virtex-5 fpga user guide*, ug190 (v5.3) ed., May 2010.
22. Xilinx, *Virtex-6 fpga memory resources user guide*, ug363 (v1.4) ed., May 2010.

Reconfigurable Field Programmable Gate Arrays for  
Mission-Critical Applications

Battezzati, N.; Sterpone, L.; Violante, M.

2011, VII, 220 p., Hardcover

ISBN: 978-1-4419-7594-2