Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications

Niccolò Battezzati · Luca Sterpone · Massimo Violante

Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications

Springer

Niccolò Battezzati
Dipto. Automatica e Informatica
Politecnico di Torino
Corso Duca degli Abruzzi 24
10129 Torino, Italy
niccolo.battezzati@polito.it

Massimo Violante
Dipto. Automatica e Informatica
Politecnico di Torino
Corso Duca degli Abruzzi 24
10129 Torino, Italy
massimo.violante@polito.it

Luca Sterpone
Politecnico di Torino
Corso Duca Degli Abruzzi 24
10129 Torino, Italy
luca.sterpone@polito.it

ISBN 978-1-4419-7594-2 e-ISBN 978-1-4419-7595-9
DOI 10.1007/978-1-4419-7595-9
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2010938708

Contents

Chapter 1
Introduction

Field-programmable gate arrays (FPGAs) play an important role in a growing number of applications. Originally devised to implement simple logic functions, FPGAs are today able to implement entire systems on a single chip. The most advanced FPGA devices as the Xilinx Virtex-7 family [3] are now offering up to 2 million logic cells, 65 Mb of embedded memory, and a number of additional features such as high-performance arithmetic functions and high-speed input/output modules. Besides the resource availability, FPGAs offer to designers two additional features that cannot be found in application-specific integrated circuits (ASICs).

By exploiting FPGAs, designers can concentrate all their efforts in the application development, letting to someone else, i.e., the FPGA manufacturer, to deal with the complex task of developing and fabricating a correctly working silicon device. In case FPGAs are used, the actual silicon that will implement the application is already available from the beginning of the application design. On the contrary, in case of ASICs, the silicon is manufactured only after application design and validation have been completed. As a result, by exploiting FPGAs, designers can reduce significantly the time to market of their applications, as application development and its silicon implementation are decoupled. Moreover, FPGAs are general-purpose silicon that can be customized by designers for implementing virtually any application: the very same device can be re-used for a wide range of applications. As a result, the cost of developing a new FPGA device is shared among a larger base of user than for a new ASIC, which is generally targeted to only one specific user. The cost for each FPGA device can hence be kept much lower than that of each ASIC device.

Some FPGAs adopt technologies that make them reconfigurable: the application the device implements is defined by an on-chip memory that can be freely altered by designers. On the contrary, ASICs are not reconfigurable: when the application has been etched in the silicon, it cannot be modified. Reconfiguration capability offers a significant competitive advantage with respect to ASICs in a number of possible scenarios:

- In case of bugs, they can be fixed easily by downloading a new, correct, implementation of the application in the FPGA. This operation can be done without removing the device from the system where it is deployed. For certain type

N. Battezzati et al., *Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications*, DOI 10.1007/978-1-4419-7595-9_1,

of applications, easy reconfiguration for bug-fixing allows for enormous cost savings. For example, in case a bug is found in an electronic apparatus employed in a satellite already placed in orbit, the capability of reconfiguring the FPGAs it embeds can make the difference between saving the entire mission or losing it, and with that substantial amount of money.

- Reconfiguration implies the possibility of changing the algorithm the FPGA implements. New features can thus be added to extend the set of services the application offers when the system is already deployed in the field. By enabling the system to evolve, designers can effectively cope with the obsolescence of apparatus, thus prolonging the useful lifetime of their applications. For example, a telecommunication satellite placed in orbit when a certain standard was not even conceived can be updated to support its years after the satellite entered in service. As new communication standards appear every few years, this example is likely to become quite frequent in the near future.
- Reconfiguration can become an active part of the application. The very same FPGA device can be reprogrammed to implement different functions in different instants of time. As a result, multiple functions can be implemented with a single device, thus saving space and mass. In case of satellite, where the launch cost depends heavily on these two parameters, reconfiguration can save a significant amount of money.

Developer of mission-critical applications like those in the space market already recognized the benefits stemming from FPGAs. A satellite is normally the unique exemplar of its own species, conceived and manufactured for a single customer. As a result, the cost and the time required for developing new ASICs for each new satellite are often not justified and not available. For this reason FPGAs are widely used in the space market. Due to the mission-critical nature of space-borne applications, and the need for operating in a harsh environment, the current design practice is based on not reconfigurable FPGAs (e.g., Actel RTAX family [1]). The adoption of such kind of devices brings only few of the benefits that designers can take advantage of in case reconfigurable devices are used. Today, new reconfigurable FPGAs are available that thanks to adequate design techniques and design tools can find their way in mission-critical applications, offering designers all the possible benefits stemming from their adoption.

Designing a mission-critical application aiming at an harsh environment such as space using reconfigurable FPGAs is not an easy task. A number of possible problems can arise, and the appropriate mitigation techniques must be understood and mastered by designers. Tools are available to support designers, but they must be understood and mastered as well.

The purpose of this book is to give an in-depth overview of the problems designers have to face when approaching the design of space mission-critical applications using FPGA devices and describe possible solutions to cope with them. We focused only on the aspect of ionizing radiation, presenting which effects they induce in FPGAs, discussing how to evaluate them and how to mitigate them. Many aspects have been left out of the book, such as the problems related to aging of components,

as well as packaging issues, and the procedures needed to guarantee an adequate quality for space use.

The book is organized into two parts. The first part, "Basic Concepts," describes the concept of reconfigurable FPGA, its failure modes when affected by ionizing radiation, and possible mitigation techniques. In particular:

- Chapter 2 presents the concept of reconfigurable FPGAs, describing the resources that can be found in modern devices, the different technologies available for the configuration memory, as well as a general model that we will use through the book to present the algorithms at the core of tools for mitigating ionizing radiation effects.
- Chapter 3 discusses the impact of ionizing radiation on FPGA devices, from both a physical and a logic level. First, the physical phenomena are discussed to illustrate the interaction mechanisms between radiation and semiconductor. Then, the physical phenomena are modeled at a more abstract level, identifying the so-called fault models. Finally, the effects induced by the considered fault models when hitting the resource of reconfigurable FPGAs are discussed. The chapter ends with an overview of the techniques that can be used to assess the impact of radiation on a certain FPGA technology and of the techniques that can be used for assessing the effects of the considered fault models on applications mapped on FPGA devices.
- Chapter 4 presents the solutions today available for mitigating the effects of radiation. After a review of the design flow needed for implementing an application on an FPGA device, we will address the presentation of the hardening solutions from two different points of view: the point of view of the FPGA manufacturer, by describing how an FPGA device can be made robust against ionizing radiation, and the point of view of the FPGA user, by describing how an application can be designed to tolerate the effects of radiation hitting a non-robust FPGA.

The second part of the book, entitled "Practical Concepts," focuses on reconfigurable FPGAs specifically designed for space use: the Xilinx Virtex-4 QV device [4], the Actel RT ProASIC3 [1], and the Atmel AT280 [2]. In particular:

- Chapter 5 illustrates the characteristics of the considered devices. For each of them, a brief description of the available resources is given, and the data publicly available about their sensitiveness to ionizing radiation are reported and commented.
- Chapter 6 discusses how the mitigation solutions presented in the previous chapters can be implemented on the considered devices. Experimental data coming from realistic benchmarks are presented to allow the reader understand the effectiveness of different mitigation solutions.
- Chapter 7 outlines a possible approach to assess the impact of ionizing radiation on FPGA devices while taking into account the radioactive environment the application is aiming at and the peculiarities of the mission where the application has to be employed. Different solutions are discussed, outlining also the implications they have on the organization of the whole system.
- Chapter 8 draws some conclusive remarks.

References

1. Actel Corporation, *Radiation-tolerant proasic3 low-power space-flight flash fpgas*, 2 ed., November 2009.
2. Atmel, *http://atmel.com/products/fpga/*, 2010.
3. Xilinx, *7 series fpgas*, Tech. report, Xilinx, June 2010.
4. Xilinx, *Space-grade virtex-4qv family overview*, ds653 (v2.0) ed., April 2010.

Part I
Basic Concepts

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].

N. Battezzati et al., *Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications*, DOI 10.1007/978-1-4419-7595-9_2,

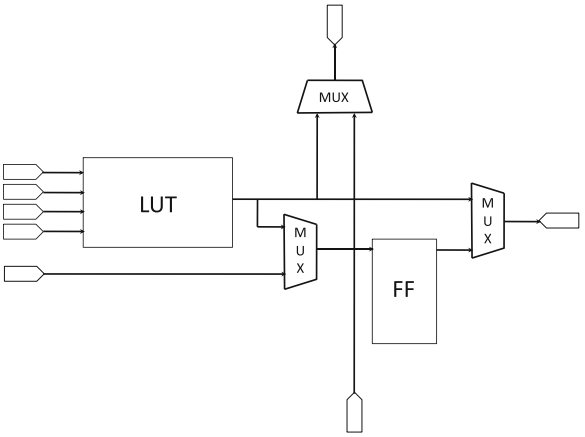
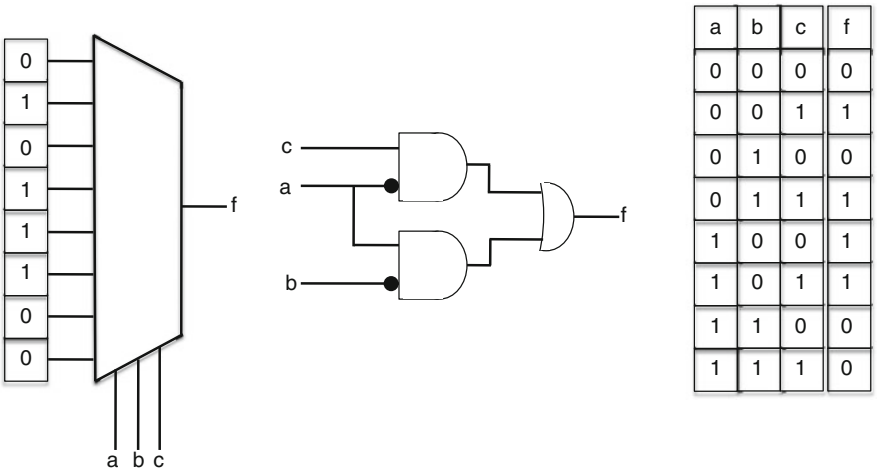


Fig. 2.1 The main element of the FPGA architecture: the configurable logic block (CLB)



a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

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.

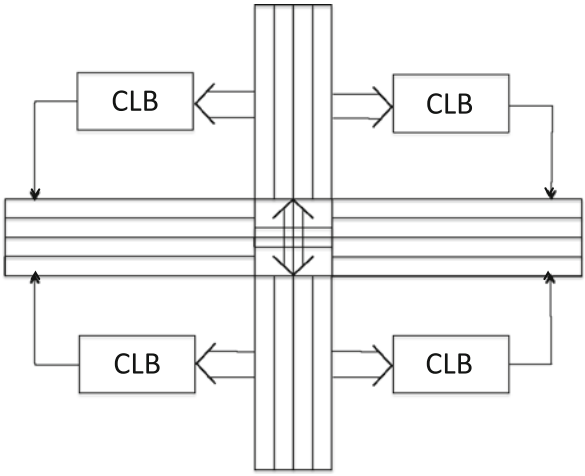


Fig. 2.3 First representation level of the model: the tile

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.

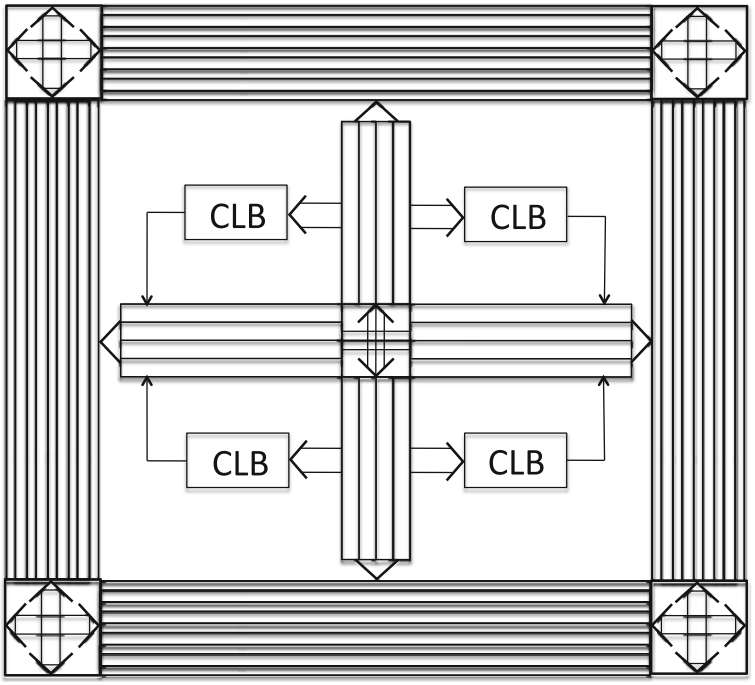


Fig. 2.4 Second representation level of the model: the local routing

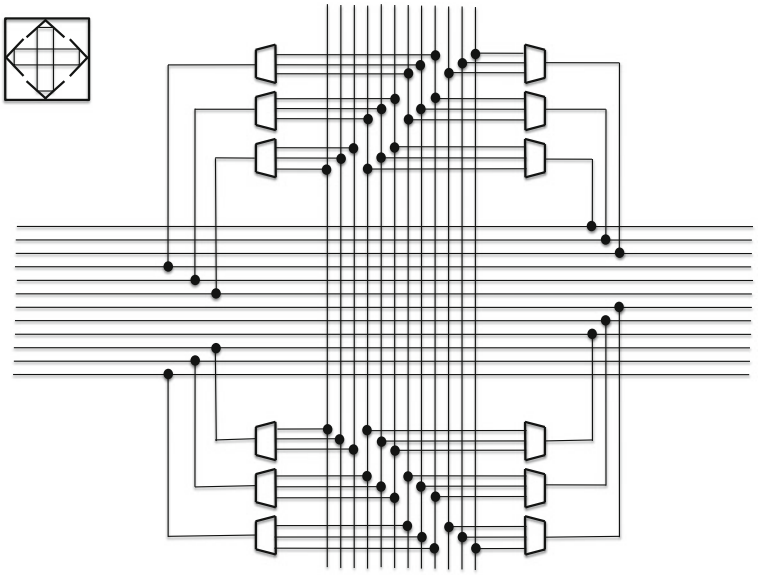


Fig. 2.5 The switch-block element

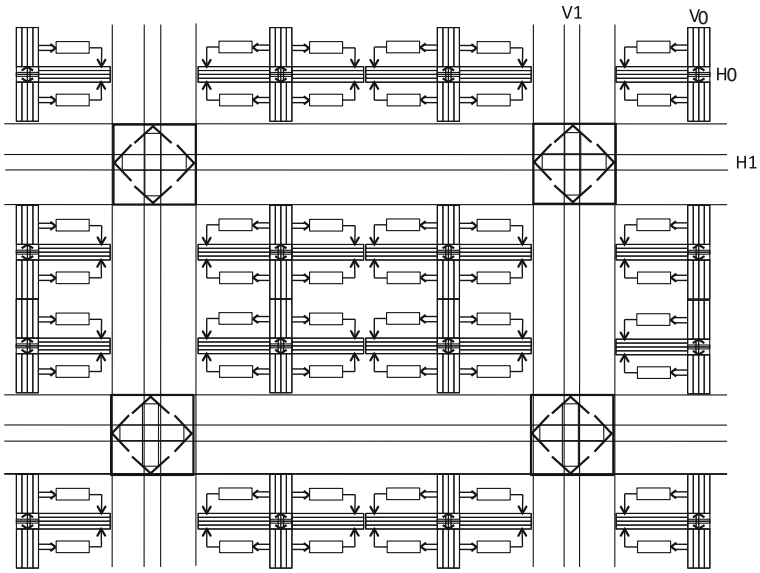


Fig. 2.6 The third representation level: the multiple tile

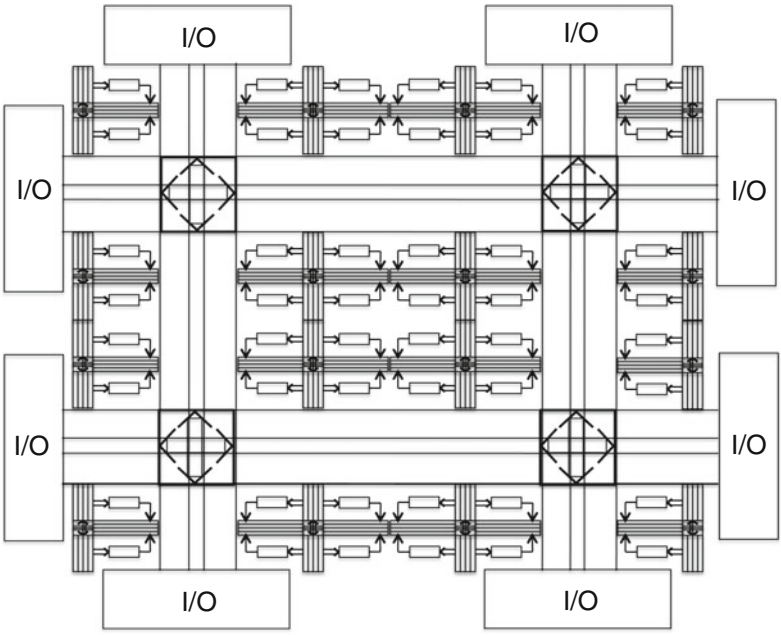


Fig. 2.7 The fourth representation level: the context

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

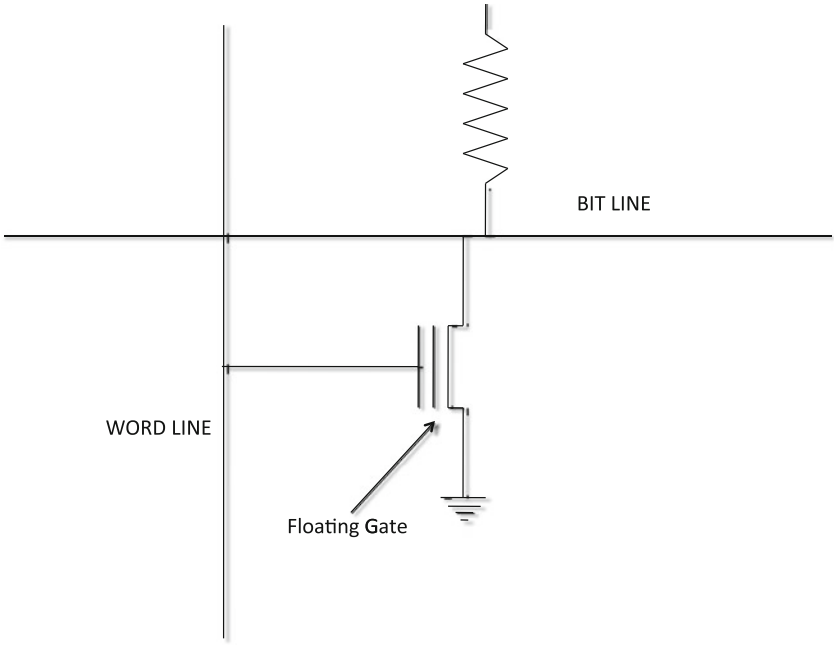


Fig. 2.8 The floating gate programming technology

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

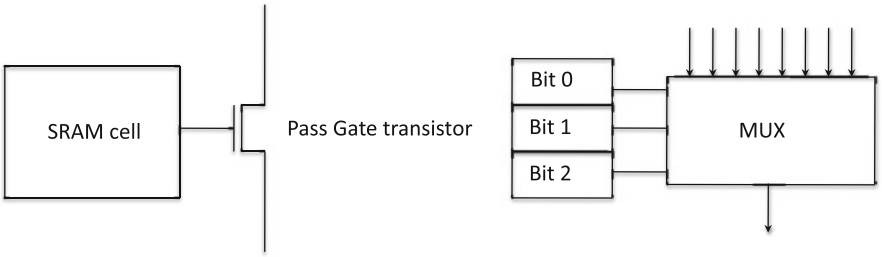


Fig. 2.9 The Static RAM configuration technology

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

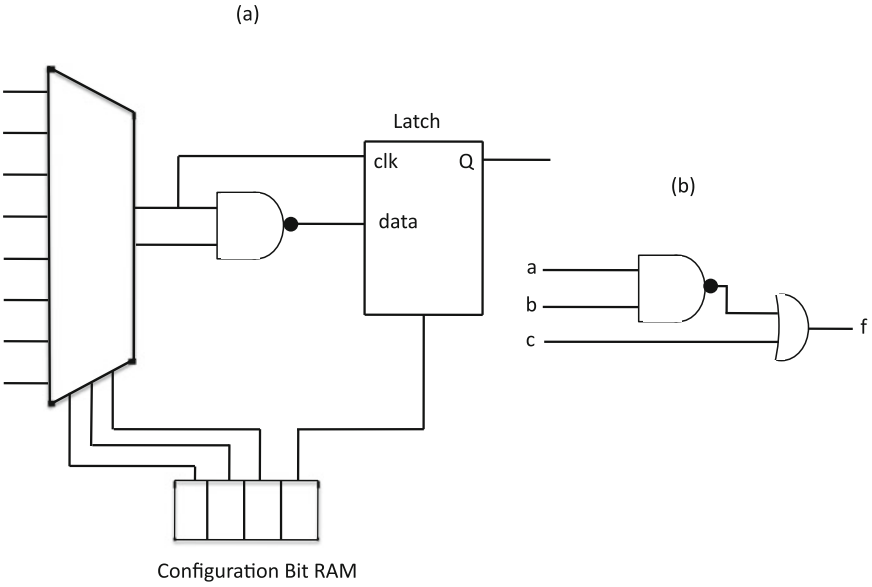


Fig. 2.10 Fine-grain logic block manufactured by Plessey

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

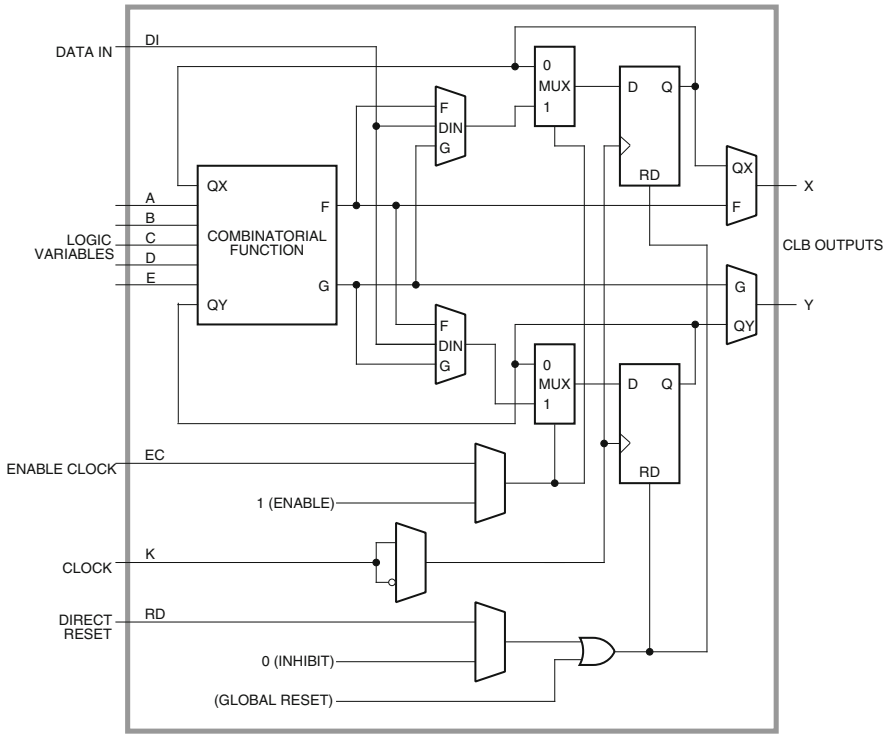


Fig. 2.11 Coarse-grain logic block manufactured by Xilinx

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

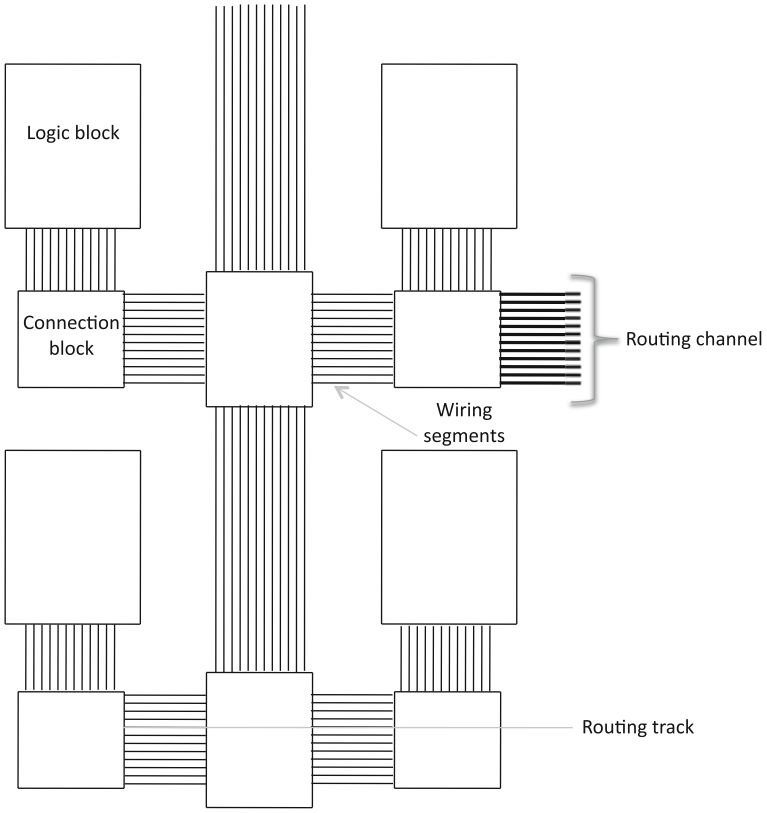


Fig. 2.12 The model of the routing architecture