Design and Optimization of Reconfigurable Architectures: The FPGA Family

Hayder Mrabet

Ph.D THESIS OF THE UNIVERSITY PIERRE ET MARIE CURIE

Department : **COMPUTER SCIENCE AND MICRO-ELECTRONICS**

Presented by : **Hayder Mrabet**

Thesis submitted to obtain the degree of
DOCTOR OF THE UNIVERSITY PIERRE ET MARIE CURIE

DESIGN AND OPTIMIZATION OF RECONFIGURABLE ARCHITECTURES: THE FPGA FAMILY

Defended : 25^{th} September 2009

Commitee in charge :

M. Régis Leveugle	TIMA, Reviewer
M. Yves Mathieu	ENST, Reviewer
M. François Anceau	LIP6
M. Marc Belleville	CEA-LETI
M. André Tissot	CEA-DAM
M. Olivier Lepape	Abound Logic
M. Jean-Arnaud François	STMicroelectronics
M. Habib Mehrez	LIP6

Abstract

In the early stages of system design, system architects often choose between FPGAs and ASICs implementations. Such decisions are based on the differences between these implementations in terms of performance, power consumptions and cost, which is related to the silicon area and the target production volume. For circuits containing only combinational logic and flip-flops, the ratio of silicon area required to implement them in FPGA (LUT-based) and ASICs (via standard cells) is on average 40. The ratio of critical path delay is roughly 3 to 4, and approximately 12 times for the dynamic power consumption. This gap is due to the FPGA interconnect network which is the dominant factor in terms of FPGA area (90%) and power dissipation (65%). In order to remain attractive, FPGA fabric must offer a good tradeoff between flexibility, performances and cost. These factors are linked to quality of the architecture, quality of the CAD tools and quality of the physical design. The subject of this dissertation is the exploration of methods and techniques to find the best tradeoff.

The first part deals with the automatic design of domain-specific reconfigurable fabrics with Mesh topology. Developing a domain-specific reconfigurable fabric has taken traditionally too much time and effort to be worthwhile. We are alleviating these design costs by automating the process of creating domain specific reconfigurable fabrics. We develop a technology-independent layout generator which is easily adapted to any standard cell library geometry and to any process rules. We have generated an SRAM Mesh-based Redundant FPGA Core with fine granularity that integrates an error-detector system for SEU mitigation. The design was successfully migrated and taped out in 0.12 um 6-metal layer CMOS process from ST.

The second part focuses on the development and the design of new Multilevel Hierarchical FPGA (MFPGA) architecture based on the Butterfly Fat Tree topology . Since the interconnect is the dominant resource in FPGA, we believe that it is the key to reduce FPGA area, hence to increase performances and decrease power consumption. We explore the effect of different architecture parameters (Rent's parameter, cluster sizes) to satisfy specific applicative constraints of logic density. Thanks to the good balancing between logic and interconnect resources, MFPGA achieves a gain of 54% in term of area compared to the common Clustered Mesh-based FPGA architecture. Finally, we propose a physical floorplanning technique for MFPGA to illustrate layout feasibility, scalability and density.

Keywords: FPGA, Interconnect, Rent's rule, Tree-based architecture, Mesh-based architecture, Physical design, Layout, SEU, CAD.

À mes parents
À ma famille

Remerciements

Je tiens à remercier vivement monsieur Alain Greiner pour m'avoir acceuilli dans son laboratoire et pour l'énergie qu'il a dépensé afin de maintenir des bonnes conditions de travail.

J'adresse mes remerciements les plus chaleureux à mon directeur de thèse, le professeur Habib MEHREZ pour sa grande disponibilité et ses conseils qui m'ont été très utiles dans la préparation de ce travail.

Je tiens également à remercier messieurs Régis Leveugle, professeur à l'INPG, et Yves Mathieu, professeur à Telecom ParisTech (ENST paris), qui m'ont fait l'honneur d'être rapporteurs de cette thèse.

Je tiens à remercier messieurs François Anceau, professeur au LIP6, Marc Belleville, Directeur de recherche au CEA LETI, André Tissot, Ingénieur de recherche au CEA-DAM, Olivier Lepape, vice président de Aboud Logic, et Jean-Arnaud François, IP manager Ã STMicroeclectronics, pour avoir accepter d'examiner mon travail.

J'associe à ces remerciements l'équipe EIM du CEA-DAM et en particulier messieurs André Tissot, Jean Luc Rebourg et Nicolas Fel pour avoir soutenu et encouragé ces recherches.
Je remercie aussi messieurs Franck Wajsbürt et François Pecheux du LIP6 de m'avoir aidé lors de la conception du circuit RedFPGA et la carte de test associé.

C'est aussi pour moi l'occasion d'exprimer ma profonde reconnaissance à monsieur François Durbin pour sa gentillesse et ses talents linguistiques. De nombreux documents et en particulier ce manuscrit auraient été bien mal écrits sans ses aides.

Cette thèse est le résultat d'un travail d'équipe et je salut tous mes collègues Zied Marrakchi, Husain Parvez, Umer Farooq, Emna Amouri, Sompasong Somsavaddy de l'équipe "Arith", Christophe Alexandre, Jean Paul Chaput et Damien Dupuis de l'équipe "Coriolis". Je tiens également à remercier Arnaud Caron, ancien doctorant du LIP6 pour ses conseils et ses encouragements. Je remercie aussi tous les étudiant du Master 2 ACSI ayant contibué à ce travail, ainsi que tous mes camarades du laboratoire.

Enfin je remercie mes parents et toute ma famille pour leur soutien aucours de ces longues années d'études et sans lesquels je n'en serai pas là aujourd'hui.

Contents

List of Figures

List of Tables

Introduction

1 Overview and Synopsys

A Field Programmable Gate Array (FPGA) is a pre-fabricated silicon device that can be reconfigured to implement several applications. The reconfigurability of an FPGA may be derived from reprogrammable Static Random Access Memory (SRAM) cells. By programming the SRAM cells, the functionality of FPGA logic units can be tailored to implement a particular computation. Interconnections between logic units are established by programming SRAM cells to connect prefabricated routing wires together. Thus, any particular application can be mapped onto FPGA by programming the functionality and connectivity of logic units based on the characteristics of the application.

It should be obvious that every application would be best served by specifically targeted custom circuitry. In fact, application-specific integrated circuits (ASICs) are often made in response to special needs. Since the exact features of the application are known beforehand, hardware resources in an ASIC are designed to provide highest performance implementation for the application. But no one can afford to produce a custom chip for every application he wants to develop; even when they are feasible, state-of-the-art ASICs get more expensive every day. Once an ASIC has been fabricated, it is generally impossible to modify the ASIC to implement any applications different from the one it was intended for. Circuit implementation such as Standard Cells, requires that a different VLSI chip be fabricated anew for each design. Further, since the Non-Recurring Engineering (NRE) costs involved in designing and fabricating an ASIC are comparatively high, it is generally infeasible to design and manufacture ASICs by small amounts. As technology improved, a market developed for versatile off-the-shelf parts that can be programmed to emulate arbitrary digital circuits instead of ASICs. FPGAs are one class of such devices, distinguished by their ability to be reprogrammed (reconfigured) any number of times.

The unlimited reconfigurability of an FPGA allows a continuous sequence of custom circuits to be employed, each one being optimized for a well defined task. This is fundamentally different from usual general-purpose microprocessors. An application is implemented on a microprocessor by compiling it into a stream of hardware instructions that are sequentially decoded and executed by fixed, general-purpose logic

resources. Unlike FPGAs, functionality of a microprocessor logic resources cannot be modified on a per-application basis. Instead, each application is compiled to a unique stream of instructions that are executed by the microprocessors. Since it is possible to express almost any application as a sequence of instructions, microprocessors are arguably the most flexible computational devices today. However, microprocessors often incur a performance penalty due to its high flexibility which is its main strong point. To support flexibility, fixed logic resources in a microprocessor are deliberately designed to execute efficiently certain basic computations. Consequently, applications that could benefit from customized, tailor-made logic resources often undergo a performance cut when executed on general-purpose microprocessors.

Since their introduction in the mid eighties, FPGAs evolved from a simple, low-capacity gate array technology to devices [Xilinx, 2008] [Altera, 2008] [Lattice, 2008] [Actel, 2008] providing a mix of coarse-grained datapath units, microprocessor cores, on-chip A/D conversion, and gate counts by millions. Today, FPGAs are installed firmly in the space of computational devices, originally dominated by microprocessors and ASICs. Much like microprocessors, FPGA-based systems can be reprogrammed on a per-application basis. At the same time, FPGAs offer significant performance gains over microprocessor implementations for a number of applications. Although these gains are still generally an order of magnitude less than for equivalent ASIC implementations, the low NRE costs, fast time-to-market, and flexibility of FPGAs make them an attractive choice for low-to-medium volume applications.

The routing network of the FPGA, consuming 90% of the chip area, is designed to suit most circuits types. This is a considerable overdesign for circuits that are not very congestioned. So, there is a need to customize this routing network for various classes of circuits.
The idea of customisation has been considered before, and recently the focus has shifted towards the FPGA routing architecture. For example, early work by Betz and Rose introduced the notion of creating a family of different architectures, each member of which designed to suit better a different type of circuit [V.Betz and J.Rose, 1995]. Betz and Rose demonstrate the usefulness of the approach by using members with different logic blocks. In comparaison, more recent work [S.Phillips and S.Hauck, 2002,S.Phillips et al., 2004, I.Kuon et al., 2005] focuses on customising the routing network for a few pre-specified circuits rather than a general class of circuits.
FPGA vendors currently produce FPGAs with varying amounts of logic, I/O pins, and different speed grades, yet they do not offer FPGAs with different amounts of interconnect for a fixed logic capacity. The main reasons for not offering a variety of interconnect sizes are inventory control, the impact of marketing and sales of seemingly inferior or unroutable devices, and the large amount of engineering effort required to develop a single device. This last reason can be partially addressed by further automating the FPGA design stage with techniques presented in chapter 3 and chapter 5.

2 Research Goals and Motivations

An FPGA can be programmed in seconds, and any bug found once the chip is tested in a system can be corrected in minutes simply by reprogramming the FPGA.
A circuit implemented on an FPGA is typically 40 times larger, roughly 4 times slower and consume 12 times more power than the same circuit implemented via standard cell in an equivalent process [I.Kuon and J.Rose, 2007]. This makes FPGA implementations more expensive than ASIC for high volume production designs. These limitations require further research in new FPGA architectures, in order to reduce power, speed and density penalities.
From the general survey above, we see that conventional FPGAs represent one challenging target in the IC world. This domain is large enough for new interesting achitectures and derivatives such as domain specific FPGA and embedded-FPGA. Figure 1 shows how FPGA derivatives can be better placed than standard FPGA towards standard ASIC.

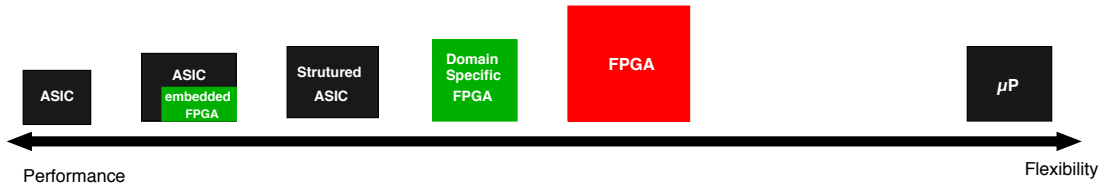


Figure 1: Flexibility Vs. Performances

This work can be viewed as a contribution to the development of reconfigurable cores that can be a viable option for specific-domain applications. The research presented here deals with 2 topics. The first deals with the development of a Mesh based FPGA-layout generator that adapts easily the target technology process with a large spectrum of architecture constraints. The second topic focuses on the development of new hierarchical FPGA topology that provides better logic density and performances than traditional FPGAs.

The thesis features:

- *A high level characterization* in the field of reconfigurable architectures, specially FPGAs. This characterisation helps us understand the key characteristics of reconfigurable devices, including the level of performance we can obtain from various architectural features.
- *Empirical relations* on the key building blocks in CMOS VLSI taken from existing designs in the literature and from our own experimental designs, including relative feature sizes (e.g. interconnect versus logic) and modeling of key area factors.
- *Architecture and Physical designs* which explore new points in FPGA design based on identified needs.

  - Techniques producing reconfigurable hardware additional advantages to integrate large spectrum of domain-specific design (e.g embedded FPGA, Radiation Hard FPGA).
  - Architectures which exploit the identified cost structure to provide greater functional density for reconfigurable devices.

- *Observations and perspectives* for future FPGA architects and system designers.

The major contributions of this thesis include:

An automatic Layout Generation for Mesh-Based FPGA

An automated FPGA physical planning methdology; a complete implementation flow of this methodology that generates an optimal layout for Mesh-Based topologies under different design constraints. This approach has significant advantages in flexibility and portability while preserving topology regularity.

A novel FPGA routing Topology: Multilevel Hierarchical FPGA(MFPGA)

The characteristics of silicon technologies create new opportunities for FPGA Design. We propose a new routing scheme that achieves significant performance improvement and area saving compared with conventional methods. Exploiting the properties identified above and current device topologies described in section 1.4, we develop a new general purpose architecture. Through an efficient well balanced allocation of device resources, this architecture offers high global gain in terms of area over a wide range of applications.

A physical planning of the MFPGA

Silicon implementations of Mulilevel Hierarchical FPGA must map the interconnect and switch components onto the two-dimensional floorplan. Physical floorplanning is particularly critical for FPGA architectures using hierarchical network topologies. The floorplan requires preserving the network regularity while minimizing total interconnect wire length to save power and reduce delay.

3 Thesis Organization

This dissertation is organized as follows. Chapter 1 first introduces the state-of-the-art of FPGA architectures and CAD tools. In particular, we focus on the interconnect topologies used in industrial and academic FPGA architectures. Chapter 2 presents an implementation and floorplanning methodology for Mesh-based FPGA design. In particular,

it is shown that mesh network can be automatically mapped onto silicon floorplan under various constraints. A tool implementation of this floorplanning and layout generation methodology is also introduced. As a proof of the concept, we describe in chapter 3 the generation of a complete SRAM based FPGA chip named REDFPGA, which includes hardware support for mitigation of SEU.
Chapter 4 introduces a new Multilevel Hierarchical network architecture that can reduce routing area with much smaller switches and buffer space requirement than for traditionnal mesh-based architectures. This routing scheme is particularly useful for FPGA design because it minimizes both area and power consumption. We focus on Rent's Rule to compare the new architecture named MFPGA with Mesh architecture. Experiments performed in this chapter are also used to compare switch and wire needs for both architectures.
Chapter 5 focuses on the physical MFPGA network implementation issues. Finally, conclusions are provided along with future perspectives on FPGA research.

1 Background

This chapter presents a technical background on reconfigurable hardware, beginning with an FPGA architectural overview. Sections 1.3 and 1.4 are in-depth case studies of the most common reconfigurable devices and a survey of interconnection strcutures. The 5th section is a brief outline of the software used to implement circuits onto reconfigurable hardware. The final section exposes general FPGA metrics and comparison with ASIC.

1.1 Introduction

Hardware logic devices can be divided into two broad classes, namely fixed logic devices and reconfigurable devices. Fixed logic devices, either for application specific integrated circuits (ASICs) or board solutions containing various components, are optimized at design time to perform efficiently a specific task or a group of tasks. On the other hand, reconfigurable hardware is designed with ability to handle multiple tasks and can be changed, or reconfigured, at any time.
Fixed logic is usually the preferred device type; when the application domain is well known in advance and the design is targeted to a large production where high performance, low power, small device size are important design goals. However, while fixed logic devices are used in a wide range of application domains, there are several drawbacks associated with these devices. One of the main drawbacks of fixed hardware is the fact that once the device is manufactured, it inherently cannot be modified anymore. Consequently, if the device does not work as expected or if there is a change in the requirements, then it must be redesigned, which can be extremely costly. Another

drawback of fixed logic devices is a long and costly time-to-market. It can take several months or years to design and verify a fixed logic device, depending upon the device size and complexity. Moreover, the upfront costs (NonRecurring Engineering, NRE) can range from hundreds of thousands to millions dollars.
Reconfigurable logic devices are typically standard off-the-shelf components, providing a wide range of logic capacity, I/O capabilities, performance, and power characteristics. With the device specific software tools provided by the logic vendor, designers are able to prototype and test their designs quickly on a working circuit, knowing that the reconfigurable logic device which they are testing their design on is the same device that will be embedded into the final system. In addition, in case of design evolution, even after the final system has been produced, this reconfigurable logic can be modified to reflect these changes. This removes almost all NREs costs for a design, and allows an extremely short time-to-market and a reduced bug fix or upgrade path.
With increasing adaptability and performance by following a general trend towards lower cost devices, reconfigurable hardware is used in a wider range of applications than ever before.

1.2 Field Programmable Gate Array

Today's most popular reconfigurable device is the Field Programmable Gate Array (FPGA). It is a collection of Logic Blocks(LB) embedded in a reconfigurable interconnection network. Figure 1.1 shows the basic Island-style model (we can also use the term Mesh), wich was popularized first by Xilinx in 1984.
A global view of an FPGA reminds closely the aspect of a network of processors (Net-

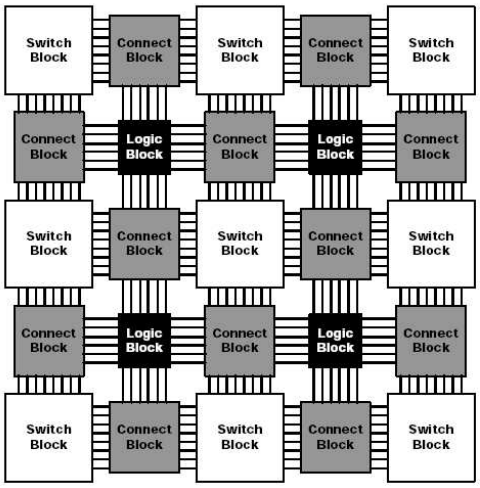


Figure 1.1: Island-style FPGA architecture

work On Chip: NOC). A conventional FPGA, however, differs from a conventional NOC

in several ways:

- **Granularity** - conventional FPGAs have single output bit logic blocks which are controlled independently.
- **Logic Block** - In conventional FPGA, the Logic Block (LB) can perform several operations, but the function cannot be changed from cycle to cycle. Once the function is configured, it is not changed until it is reconfigured.
- **Interconnect** - On conventional FPGA, interconnect is purely static. Connecting sources and sinks is achieved by reserving a path through the reconfigurable switching network controled by the configuration memory.

The most comprehensive collection of mesh architecture studies can be found in the book by Betz, Rose and Marquardt [V.Betz et al., 1999]. Much of the information from this section can be found in this reference as well as papers cited herein. This book gives the major architectural parameters for FPGA created with the mesh-model. VPR, Versatile Place and Route tool is used to experiment this typical model which includes clustered logic block with a full cross bar inside.

Routing Architecture

Conceptually, an island-style FPGA (which is the most popular type of FPGA manufactured today) is an array of LBs. The routing structure can be parametrized to interconnect any LB with any other LB within the array. Every LB which implements the user's logic, has inputs and outputs connected to horizontal and vertical routing channels (figure 1.1). Between rows and columns of an array of LBs, the routing channels contain Switch blocks (S_block) and Connection Blocks(C_blocks). C_blocks enable to connect input and output signals of LBs to the routing channel and vice versa. Once signals are in the routing channels, S_blocks carry them throughout the FPGA by enabling corner turning and in some cases switching between routing tracks. S_blocks, like C_blocks, are made up of user programmable switch points.
Every routing channel contains W parallel wire tracks, where W is called *channel width*. The logical length of a wire segment, L_{wire}, is equal to the number of LB it spans.
To minimize switch counts, FPGAs employ more restricted connection schemes to limit the interconnect ressources which represent the dominant area in FPGA devices. Rose and Brown [J.Rose and S.Brown, 1991] established that S_blocks should contain 3 connections per wire and C_blocks should have a switch at 70-90% of all possible switch locations. They summarised these details by the flexibility parameters $F_s = 3$ and $F_c = 0.7$ to 0.9, respectively.

LB Architecture

All LBs contain N Basic Logic Elements (BLEs) grouped together. The BLE inputs are chosen among a set of I shared LB inputs and the N outputs of BLEs in this LB (called

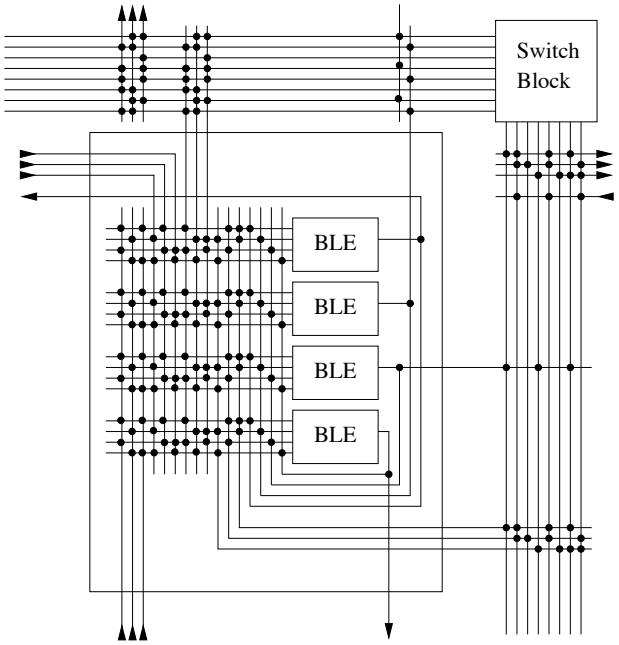


Figure 1.2: Details of Logic Block architecture

feedbacks). These connections are provided by the local cross bar. The local interconnect can be populated fully or sparsely (full or sparse cross-bar). Figure 1.2 shows the contents of an LB with $N = 4$, with a local sparse cross-bar as adopted in [G.Lemieux and D.Lewis, 2004].

In fact, BLEs are simple functions with a small number of inputs and one output. The most general function used in BLE is the *k*-input Look-Up Table (*k*-LUT). As shown in figure 1.3, a typical BLE consists of a lookup table (LUT), a state holding element such as a flip-flop, and multiplexers. LUTs are small memories in which the truth table for the desired function can be written when the FPGA is configured. Multiplexer within the BLE, in the form of bypassing multiplexer, is controlled by a programming bit that enables the sequential or the combinatorial mode within the BLE, depending on the application which is being mapped onto the BLE. By using LUTs, flip-flops, and multiplexers, FPGAs can implement arbitrary logic functions, any BLE handles functions with several inputs typically (between 3 and 6) and one output. Research at the university of Toronto [E.Ahmed and J.Rose, 2000] shows that 4-LUTs yield the most area efficient designs over a collection of circuit benchmarks. A 4-LUT can implement any logic function with 4 inputs and one output. Figure 1.3 shows the canonical 4-LUT inside a 4-inputs BLE.

FPGA configuration

Configuration of the FPGAs is commonly saved in Static RAM (SRAM) cells distributed across the chip. By placing the configuration bits in SRAM cells, FPGAs can be repro-

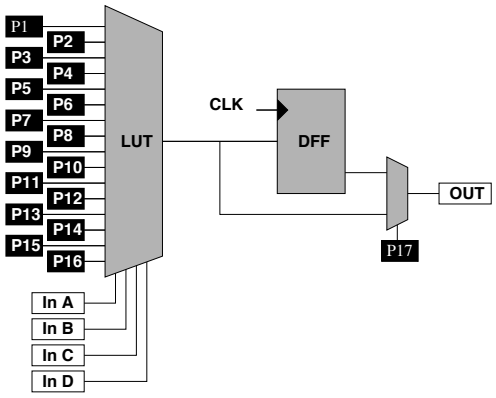


Figure 1.3: Basic Logic Element with a 4-LUT, a Flip-Flop and a bypass multiplexer

grammed many times over their life. Thus, an FPGA can implement many different configurations, just like general-purpose processors can run many different programs. The ability of an FPGA to run such a wide range of programs is only limited by the number of LBs in the array and by the quality of the routing resources.

FPGA Evolution

FPGA appeared in the mid 1980's. In late 80's and early 90's reconfigurable computing engines appeared, based on these new devices. The first FPGA on the market in 1985 was the XC2064 from Xilinx which contained 64 configurable logic blocks (CLBs), all exhibiting two 3-LUTs. Subsequent generations reached 4-LUTs, because they offered a more optimal balance concerning logic utilization and minimization of the number of logic levels compared to similar designs. Reconfigurable FPGA from Xilinx [Xilinx, 2008], Altera [Altera, 2008] and Actel [Actel, 2008] use 4-LUTs as their basic constituent LBs. Recent FPGAs use 6-LUTs and LBs are more complex than a single Look-Up table. Throughout the thesis we use 4-LUTs, as the canonical FPGA logic grain for discussion and comparaisons purpose in forthcoming chapters.

The island-style FPGA presented so far is a simplified version of a modern FPGA, and therefore does not adequately characterize the complexity and functionality of today's devices. Modern devices available from vendors such as Xilinx and Altera are blurring the line between the fine-grained island-style device described above, and a more medium-grained device. This change in the granularity has been brought about by the inclusion of coarse-grained components into the reconfigurable fabric, like multipliers and embedded general-purpose processors.

Today, FPGA companies including Xilinx, Altera and Lattice Semiconductor, are trying to move the emphasis away from SOCs by providing systems-on-a-programmable-chip (SOPCs). The increase in the transistor count in devices has enabled FPGA designers to create reconfigurable devices providing the functionality that was typically the province of SOCs.

An example of such FPGAs is the Virtex family from Xilinx. The Virtex II Pro has many interesting features, including up to four PowerPCs [Xilinx, 2008], 24 embedded Rocket IO multi-gigabit transceivers, 12 digital clock managers, 556 18x18 multipliers, and ten megabits of block RAM. This is an expensive and large device, but it is quite likely to handle a wide range of applications that were previously reserved to SOCs or systems-on-a-board.

Altera, like Xilinx, has a device which can emulate many types of SOCs, specifically its Stratix family [Altera, 2008]. Lattice Semiconductor decided to distinguish their product by enabling designers to pick specific ASIC cores that can be embedded with the reconfigurable fabric. Lattice called these devices as Field Programmable System Chips (FPSC) [Lattice, 2008]. While the reconfigurable fabric is generic in nature, ASIC macrocells range from bus interfaces, high-speed line interfaces to high-speed transceiver cores. All current macrocells belong to the networking domain, with the possibility for Lattice to create more macrocells in the future if designer demand is high enough.

1.3 FPGA structures

In the following sections we present recent FPGA succesful achievement in industry, then we highlight the most common academic architectures. An evaluation of the Stratix III and Virtex-5 family devices, which are developed by the Altera Corporation and Xilinx Corporation respectively, are presented.

1.3.1 Case Study: Altera Stratix III

Based on 65 nm all-layer copper SRAM process, the Stratix-III FPGA family was developed by Altera Corporation [Stratix-III, 2008]; it offers two family variants, optimized to meet different application needs:

- The Stratix-III*L* family provides balanced logic, memory, and multiplier ratios for mainstreams applications.
- The Stratix-III*E* family is rich in memory and multiplier, for data-centric applications such as wireless, medical imaging and military applications.

The Stratix-III FPGA is an island-style device containing logic array blocks (LABs), embedded memories, embedded DSP and multipliers blocks, multi-function I/O elements (IOEs), and up to 12 phase-locked loops (PLLs). All these functional elements are connected to each other by a two-dimensional multi-track interconnect fabric.

Logic Array Blocks and Adaptive Logic Modules The Logic Array Block (LAB) shown in figure 1.4 is composed of basic building blocks known as Adaptive Logic Modules (ALMs) which can be configured to implement logic, arithmetic, and register functions.

Each LAB consists of ten ALMs, carry chains, shared arithmetic chains, LAB control signals, local interconnect, and register chain connection lines.
The LAB of Stratix-III has a by-product called Memory LAB (or MLAB), which adds SRAM memory capability to the LAB. MLAB is a superset of the LAB and includes all LAB features. MLABs support a maximum of 320 bits of simple dual-port Static Random Access Memory (SRAM). Each ALM in an MLAB can be configured as a 16x2 block, resulting in a configuration of 16x20 simple dual port SRAM block. MLAB and LAB blocks always co-exist as pairs in all Stratix III families, allowing up to 50% of the logic (LABs) to be traded for memory (MLABs).
The basic building block of logic in the Stratix III architecture which is the Adaptive

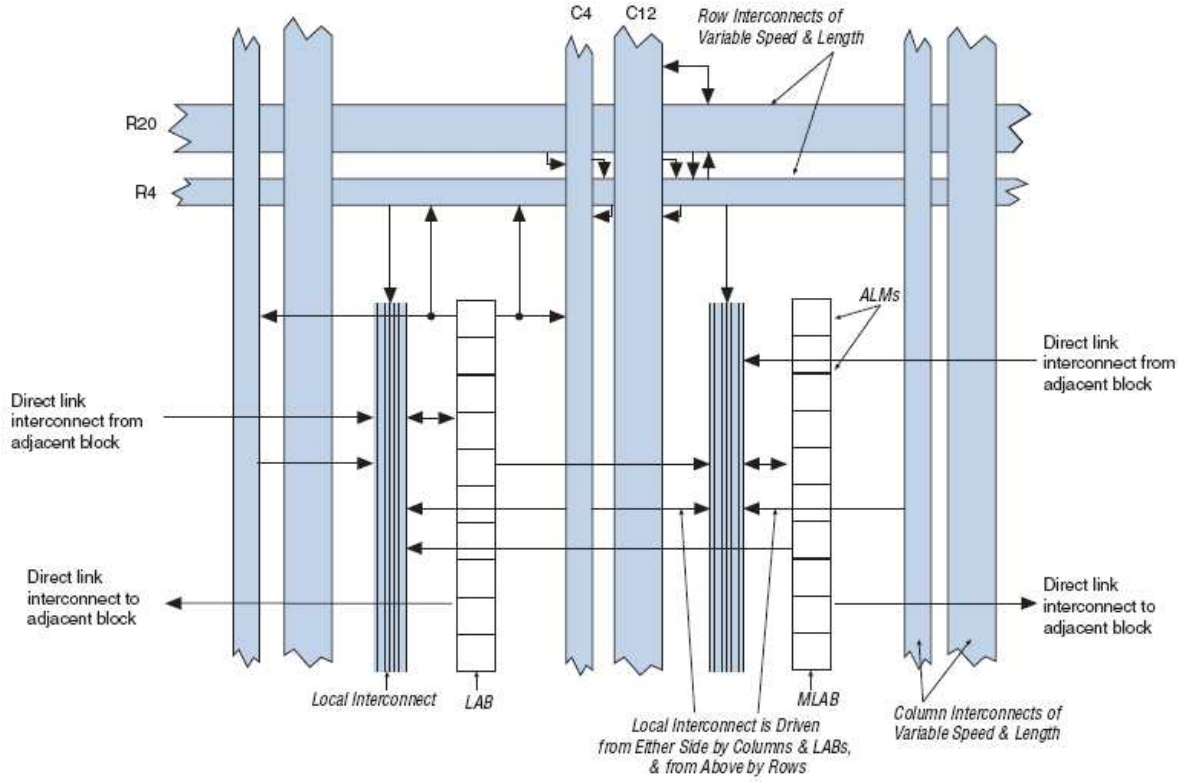


Figure 1.4: Stratix-III LAB structure

Logic Module (ALM), provides advanced features with efficient logic utilization. ALM shown in figure 1.5 expands the traditional 4-input look-up table architecture to 6 inputs, increasing efficiency by reducing LUTs, logic levels, and associated routing. Each ALM includes two Adaptive LUTs (ALUTs) with a total of 64 bits of logic programming space and 8 shared inputs. One ALM can implement any function with 6 inputs and certain seven-input functions. It can also implement some combinations of completely independent functions(4LUT+4LUT, 5LUT+3LUT) and various combinations of functions with common inputs(5LUT+5LUT with 2-common inputs, 6LUT+6LUT with 4-common inputs, 5LUT+4LUT with 1-common input). The logic capacity of Stratix3

can reach 240.000 equivalent 6-inputs LUT.
In addition to adaptive LUT-based resources, each ALM contains 2 programmable registers, 2 dedicated full adders, a carry chain, a shared arithmetic chain, and a register chain. Through these dedicated resources, an ALM can implement various arithmetic functions and shift registers efficiently. Each ALM drives all types of interconnects: local, row, column, carry chain, shared arithmetic chain, register chain, and direct link interconnects.
The LAB local interconnect can drive several ALMs in the same LAB. It is driven by

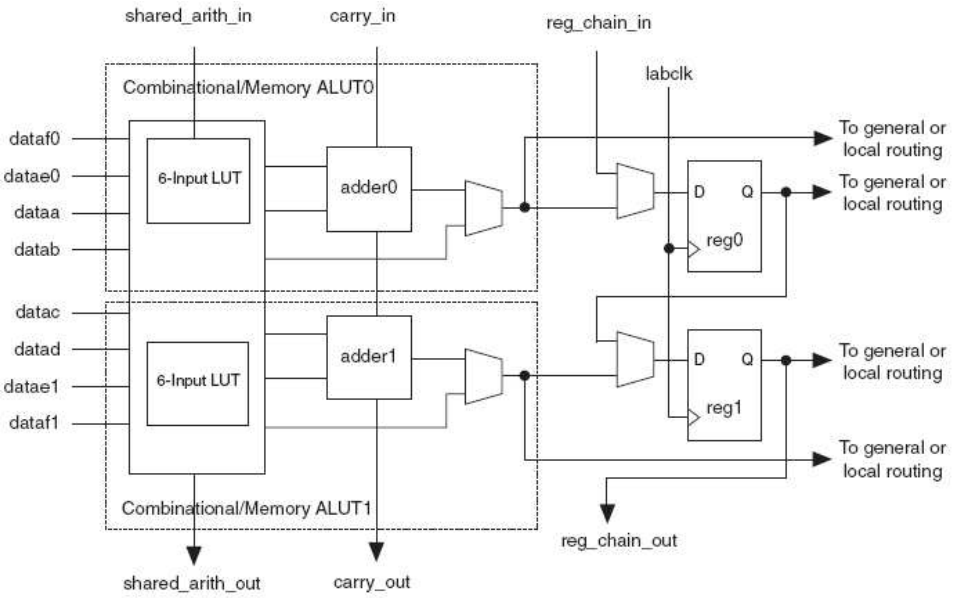


Figure 1.5: Block Diagram of the Stratix-III Adptive Logic Module

column and row interconnects and by ALM outputs in the same LAB. Neighbouring LABs/MLABs, M9K RAM blocks, M144K blocks, or DSP blocks from the left and right can also drive a LAB's local interconnect through the direct link connection. The direct link connection feature minimizes the use of row and column interconnects, providing higher performance and flexibility. Each ALM can drive 30 ALMs through fast local and direct link interconnects. LABs include additionally an enhanced interconnect structure for routing-shared arithmetic chains and carry chains for efficient arithmetic functions. The register chain connection allows the register output of one ALM to connect directly to the register input of the next ALM in the LAB to obtain fast shift registers. These ALM-to-ALM connections bypass the local interconnect.

Embedded Multipliers

Stratix III devices include dedicated high-performance digital signal processing (DSP) blocks optimized for DSP applications requiring high data throughput. Complex sys-

tems such as WiMAX, 3GPP WCDMA, CDMA2000, voice over Internet Protocol (VoIP), H.264 video compression, and high-definition television (HDTV) require high performance DSP blocks to process data. Typically, these system designs use DSP blocks to implement finite impulse response (FIR) filters, complex FIR filters, infinite impulse response (IIR) filters, Fast Fourier Transform (FFT) functions, and discrete cosine transform (DCT) functions.
Stratix III devices have up to 112 High-speed DSP blocks that provide dedicated implementation of 9x9,12x12, 18x18, and 36x36 multipliers (550 MHz maximum frequency), multiply-accumulate functions, and finite impulse response (FIR) filters. Multipliers can optionally feed an adder/subtractor or accumulator in the block depending on user configuration. This option saves ALM routing resources and increases performance, because all connections and blocks are inside the DSP block. Stratix-III provides up to 896 18x18 multipliers.

TriMatrix Embedded Memory Blocks

TriMatrix embedded memory blocks provide 3 different sizes of embedded SRAM to efficiently address the needs of Stratix III FPGA designs. TriMatrix memory includes the following blocks:

- 950 to 6,750 MLAB(320-bit) blocks, optimized to implement filter delay lines, small FIFO buffers, and shift registers.
- 108 to 1040 M9K(9-Kbit) blocks that can be used for general purpose memory applications.
- 6 to 48 M144K(144-Kbit) blocks that are ideal for processor code storage, packet and video frame buffering.

Every embedded memory block can be configured independently to make a single or dual-port RAM, ROM, FIFO or shift register. Multiple blocks of the same type can also be grouped together to produce larger memories with minimal timing penalty. TriMatrix memory in Stratix-III provides up to 20,491 Kbits of embedded SRAM at up to 600 MHz operation.

MultiTrack Interconnect

Connections between all functional units in the Stratix-III are provided by the MultiTrack interconnect structure with DirectDrive technology. This interconnect contains a two-dimensional row- and column-based architecture providing 1-hop connection to 34 adjacent LABs, 2-hop connections to 96 adjacent LABs and 3-hop connections to 160 adjacent LABs. A series of column and row interconnects of varying length and speed provides signal interconnects between logic array blocks (LABs), memory block structures, digital signal processing (DSP) blocks, and input/output elements (IOE). These

blocks communicate with themselves and with all other ones through a fabric of routing wires.
DirectDrive is a deterministic routing technology that ensures identical routing resource use for any function, regardless of its placement in the device.The MultiTrack interconnect and DirectDrive technology simplify the integration stage of block-based designing by eliminating extra optimization cycles that typically follow design changes and additions.
Row interconnect consists of both R4 and R20 connections, as well as direct link connections between LABs and adjacent blocks; R4 interconnects span a combination of 4 LABs, memory logic array blocks (MLAB), DSP blocks, M9K blocks, and M144K blocks. Figure 1.6 shows R4 interconnect connections from a LAB.
R4 interconnects can drive and can be driven by DSP blocks, RAM blocks and row IOEs. R4 interconnects can drive other R4 interconnects to extend the range of LABs they drive. They can also drive and can be driven by C4 and C12 (column interconnects) for connections from one row to another. Additionally, R4 interconnects can drive R20 interconnects.
R20 row interconnects span 20 LABs and provide high-speed resources for row con-

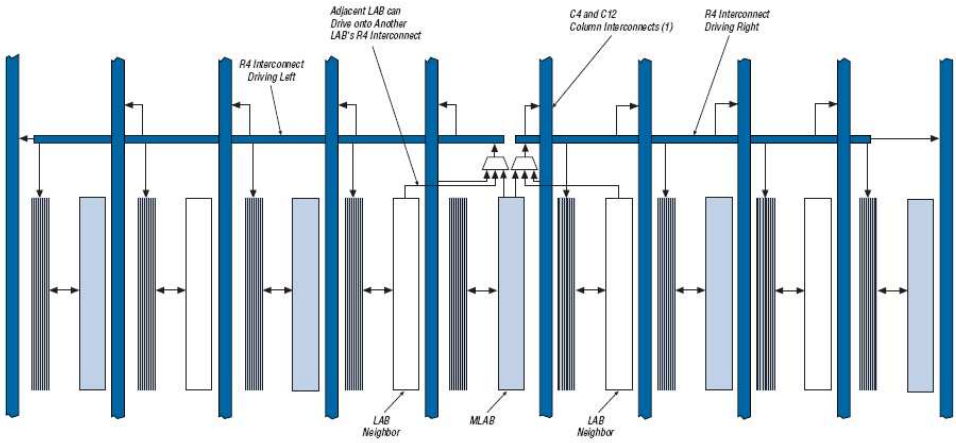


Figure 1.6: R4 Interconnect Connections

nections between distant LABs, TriMatrix memory, DSP blocks, and row IOEs. R20 row interconnects drive LAB local interconnects via R4 and C4 interconnects and can drive R20, R4, C12, and C4 interconnects. R20 can only be driven by another R20, R4 or column C12 interconnect, not directly by LABs.
The column interconnect operates similarly to the row interconnect. It routes signals vertically to and from LABs, TriMatrix memory, DSP blocks, and IOEs. A column of every LAB is served by a dedicated column interconnect. These column interconnect resources include C4 and C12 vertical interconnects crossing.
The C4 interconnects span four adjacent interfaces in the same device column. Figure 1.7 shows the C4 interconnect connections from a LAB in a column. The C4 interconnects can drive and can be driven by all types of architecture blocks, including DSP blocks,

TriMatrix memory blocks, and column and row IOEs. For LAB interconnection, a primary LAB or its LAB neighbour can drive a given C4 interconnect. All C4 interconnects can drive one another to extend its range; they can also drive row interconnects to obtain parallel column connections.

C12 column interconnects span a length of 12 LABs and provide fast resources for

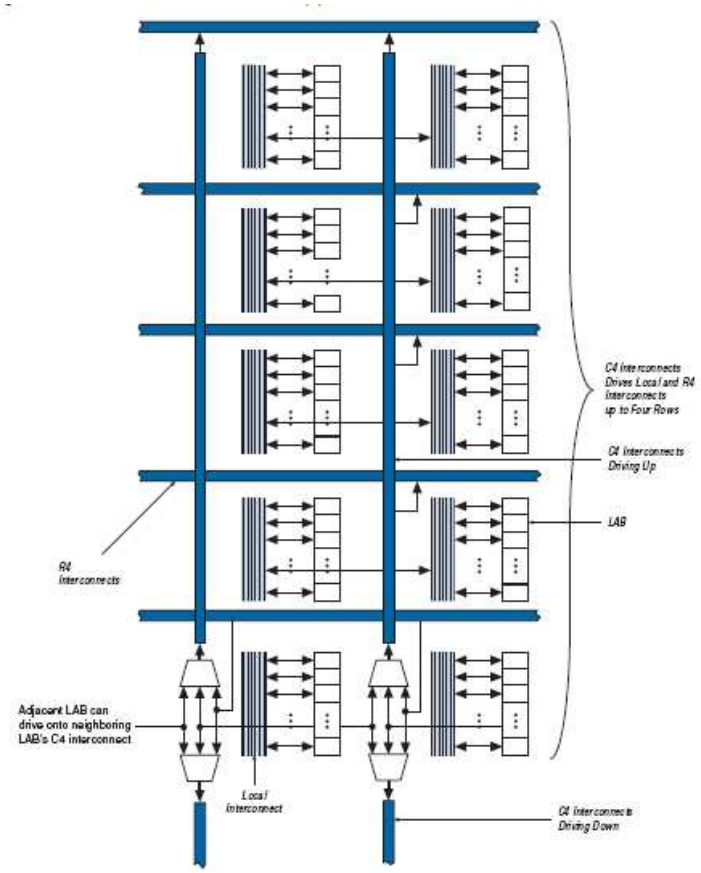


Figure 1.7: C4 Interconnect Connections

column connections between distant LABs, TriMatrix memory blocks, DSP blocks, and IOEs. C12 interconnects drive LAB local interconnects via C4 and R4 interconnects but do not drive LAB local interconnects directly. C20 can be driven only by R4 and R20 and other C20 interconnects.

In this brief outline of the Stratix-III family of devices, several features and details have been left out, since they are beyond the scope of this dissertation. Please see [Stratix-III, 2008] for more information.

1.3.2 Case Study: Xilinx Virtex 5

Virtex-5 devices introduced in 2006 by Xilinx were the world's first FPGAs manufactured with the 65nm technology. The Virtex-5 is based upon an architectural approach that Xilinx created to reduce the cost of developing multiple FPGA platforms, each one with different combinations of feature sets. Xilinx has dubbed this architectural approach as the Advanced Silicon Modular Block (ASMBL) architecture. For each application domain, such as digital signal processing, Xilinx has determined the optimum mixture (ratio) of logic, memory, DSP slices, and so forth. Next, for each application domain, Xilinx creates a set of components, all based on the same "mix" but with varying proportions. The initial release of the Virtex-5 includes devices that offer a choice of 4 new platforms, each one delivering an optimized balance of high-performance logic, serial connectivity, signal processing, and embedded processing:

- **LX** : Optimized for high-performance logic
- **LXT**: Optimized for high-performance logic with low-power serial connectivity
- **SXT**: Optimized for DSP and memory-intensive applications, with low-power serial connectivity
- **FXT**: Optimized for embedded processing and memory-intensive applications, with highest-speed serial connectivity

FPGA designs made considerable progress over recent years. Today's designs often feature wide data paths, especially in the case of digital signal processing (DSP) applications. Implementing these designs using 4-input LUTs can require many levels of logic, thereby reducing performance. In order to address this issue, the ExpressFabric employed by the Virtex-5 family features LUTs with 6 independent inputs, which can significantly reduce the number of logic levels required to implement large functions. Virtex-5 devices are also based on a new diagonal interconnect architecture that facilitates shorter, faster routing. An overview of some of the more significant Virtex-5 architectural features is as follows:

Configurable Logic Block

The Configurable Logic Blocks (CLBs) are the main logic resources for implementing sequential or combinatorial circuits. Each CLB element is connected to a switch matrix to access the general routing matrix (shown in Figure 1.8). A CLB element contains a pair of slices. These two slices do not have direct connections to each other, and each slice is organized as a column. Each slice in a column has an independent carry chain.
Every slice contains 4 logic-function generators (or look-up tables), 4 storage elements, wide-function multiplexers, and carry logic. Each of these logical functions can be used as a true 6-input LUT or as two 5-input LUTs that share five of their inputs. In addition to its four 6-input LUTs, a Virtex-5 slice also includes fast flip-flops to speed pipelined

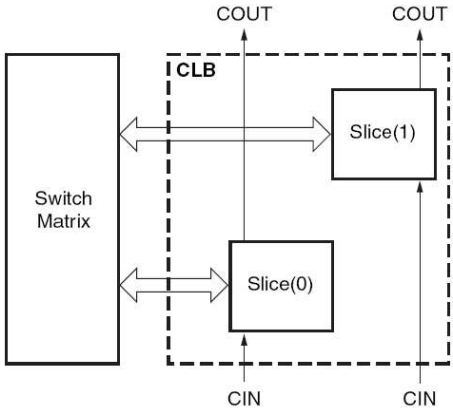


Figure 1.8: Arrangement of Slices within the CLB

designs and an improved carry chain architecture to speed arithmetic operations. Moreover, some slices support two additional functions: storing data using distributed RAM and shifting data with 32-bit registers. Slices that support these additional functions are called SLICEM; others are called SLICEL. Overall, Virtex-5 family can provide up to 207,360 6-inputs LUT equivalent to 330,000 logic cells (Industry defines a logic cell as a 4-input Look-up Table and a Flip-Flop).

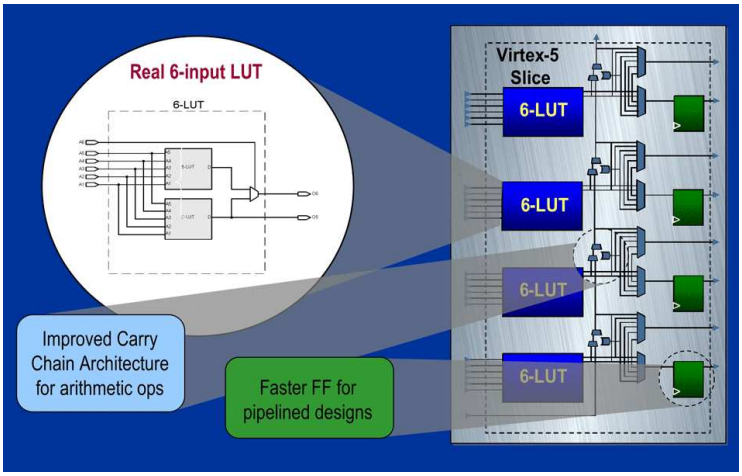


Figure 1.9: Virtex-5 slice

Routing Architecture:

Like Aletra, the general routing topology of the Virtex family was a Mesh architecture [D.Tavana et al., 1996] based on horizontal and vertical wires connecting GRMs between them as shown in figure 1.10. No details are currently available about the Virtex-5 routing architecture. The only available information is about the ExpressFabric

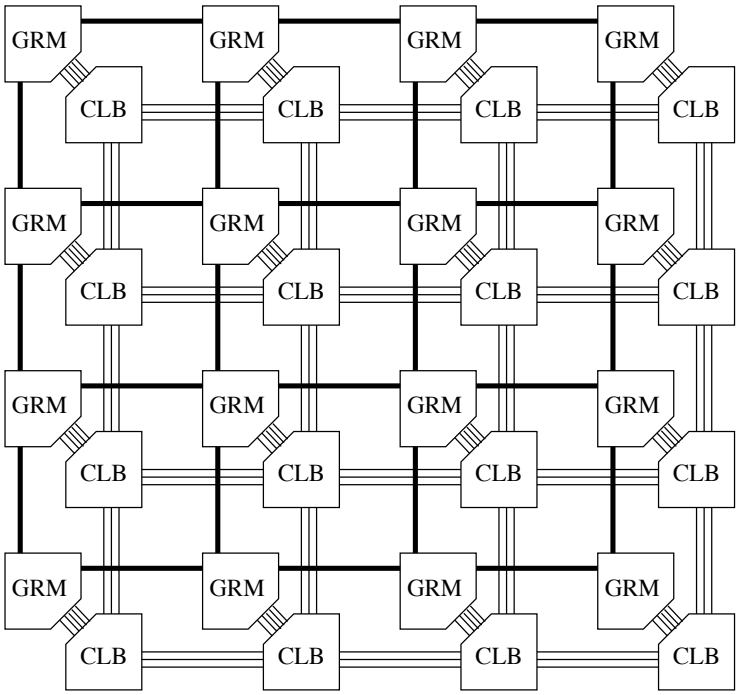


Figure 1.10: General Xilinx Virtex Mesh Topology

feature that is a radically new form of diagonal interconnect. Thanks to their 12-layer metallization (11 copper layers and 1 aluminum layer) including an advanced diagonal interconnect fabric, the Virtex-5 family uses diagonally symmetric interconnects, thus minimizing routing hops (the number of interconnects required from CLB to CLB) to realize major performance improvements over the previous Virtex-4 devices. With fewer hops, the diagonally interconnect patterns result in an average increase of logic performance of 30% over the previous Virtex-4 generation of devices, yielding up to 2 speed grades.

Embedded Hard IP

Virtex-5 comes with up to 14.5Mb of block RAM, 640 DSP slices, and 1200 SelectIO. In addition, the FXT platform includes a couple of PowerPC 440 cores, each one providing $1,1000$ DMIPS at 550MHz.
The Virtex-5 DSP48E is based on 25x18 bit multipliers (versus 18x18 in Virtex-4 FPGAs). DSP Slices are cascaded to provide greater multiplier width with 550 MHz clock management, while offering both DCM and PLL. The Block RAM (BRAM) can be used as a 36Kbit, cascadable dual-port block RAM / FIFO with integrated 64-bit ECC (Error Checking and Correction).

1.4 Interconnection Networks and FPGA architectures alternatives

Due to various performance requirements and cost metrics, many network topologies are designed for specific applications. Originally founded for telephony, interconnection networks are adopted in computer systems networks, parallel computing, and graph theory. It is inspiring and informative to look back on the multiplicity of interconnection networks.
Networks can be classified into direct networks and indirect networks [J.Duato et al., 1997]. In direct network terminal nodes are connected directly with all others by the network. In indirect network, terminal nodes are connected by one (or more) intermediate switches. The switching nodes perform the routing. Therefore, indirect networks are also often referred to as multistage interconnect networks.
Direct networks and indirect networks can have different topologies [J.Duato et al., 1997]. It is not the objective of this chapter to discuss functionalities and performance metrics of these different networks. Rather, we give only a brief description of some of the well known network topologies. We use these topologies as examples to formulate the FPGA network problems in later chapters.

1.4.1 Direct Network Topologies

Orthogonal Topology:

Nodes in orthogonal networks are connected in k-ary n-dimensional mesh (k-ary n-mesh) or k-ary n-dimensional torus (k-ary n-cube) formations, as shown in Fig 1.11
Due to simple connection and easy routing provided by adjacency, mesh networks

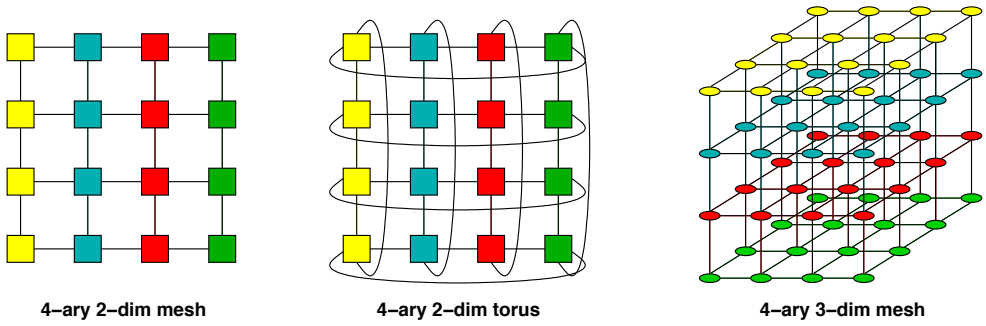


Figure 1.11: Mesh and Torus Networks

are widely used in most FPGA as described in this chapter. Orthogonal networks are highly regular, therefore, interconnect length between nodes is expected to be uniform to ensure performance uniformity of logic nodes.
We can use only local connections within the array between adjacent or close, array

elements. The bisection bandwidth in a mesh topology with n elements is $O(\sqrt{n})$ and hence, never dominates the logical array element size. However, communicating a piece of data between two points in the array requires a switching delay proportional to the manhattan distance between the source and the destination ($O(\sqrt{n})$). This makes distant communication slow and expensive and can make interconnect delay quite high-easily dominating the delay through the logic element.

Cube-Connected-Cycles Topology:

The cube-connected-cycles (CCC) topology is proposed as an alternative to orthogonal topologies in order to reduce the degree of each node [F.P.Preparata and J.Vuillemin, 1981], as shown in Fig 1.12. Each node has 3 degrees of connectivity, compared to 2 degrees in mesh and torus networks. CCC networks have a hierarchical structure: the 3 nodes at each corner of the cube form a local ring.

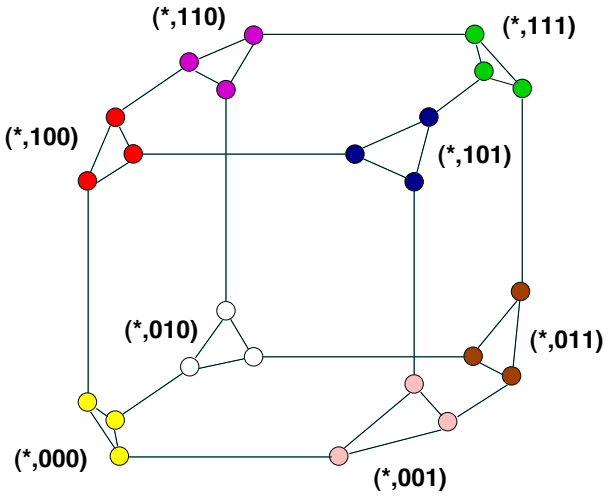


Figure 1.12: Cube-Connected-Cycles Topology

FPGAs with direct network topology

Currently, mesh-based (named also island-style) architecture represent the dominant FPGA architecture style. State of the art shown previously offered by commercial vendors leaders like Xilinx and Altera have 2-dimension mesh-based architecture. A well-known academic mesh topologies are Tryptich [G.Borriello et al., 1995], VPR-style FPGA [V.Betz and J.Rose, 1997]. There has been also some works proposing 3D mesh-based FPGA architectures. Borrowing ideas from multi-chip module (MCM) techniques, Alexander et al. proposed to build a 3D FPGA by stacking together a number of 2D FPGA bare dies [A.J.Alexander et al., 1995]. A straightforward extension of the Tryptich architecture is found in the Rothko 3D architecture, which has routing and logic blocks placed

on multiple layers [M.Leeser et al., 1998]. Recent work of [M.Lin and A.Gamal, 2007] and [C.Ababei et al., 2005] present respectively a routing fabric of monolithically stacked 3D-FPGA and TPR: a placement and detailed routing tool for such 3D mesh-based FPGAs.

1.4.2 Indirect Network Topologies:

Crossbar Switch Fabrics

An NxN crossbar network connects N input ports with N output ports. Any of the N input ports can be connected to any of the N output ports by a node switch on the corresponding crosspoint (Fig 5.16). Such scheme garantees arbitrary full connectivity along elements but the cost is prohibitively high.
For an element array where each element is a k-input function (e.g. k-LUT), the cross-

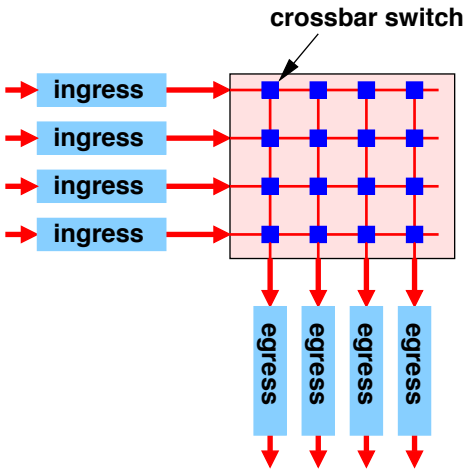


Figure 1.13: Crossbar Switch Fabrics

bar would be an NxkxN crossbar. Arranged in a roughly square array, each input and output must travel $O(\sqrt{N})$ distance. Since interconnect delay is proportional to interconnect distance, this implies the interconnect delay grows at least as $O(\sqrt{N})$. However, the bisection bandwidth for any full crossbar is $O(N)$.

Butterfly Topology

Compared to the crossbar, the Butterfly network (Fig 1.14) can reduce the total number of switches required from$O(N)$ to $O(nlog(N))$. Inside the butterfly fabrics, each source-destination route uses a dedicated path. The delays between any 2 logic nodes are identical, and the delay is determined by the number of intermediate stages on the switch fabrics.

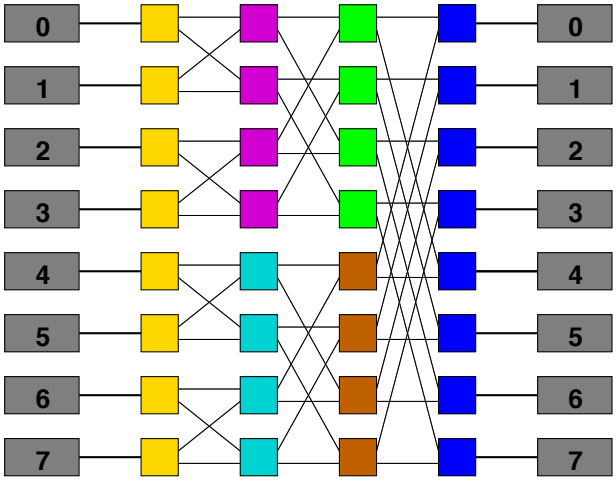


Figure 1.14: Butterfly network

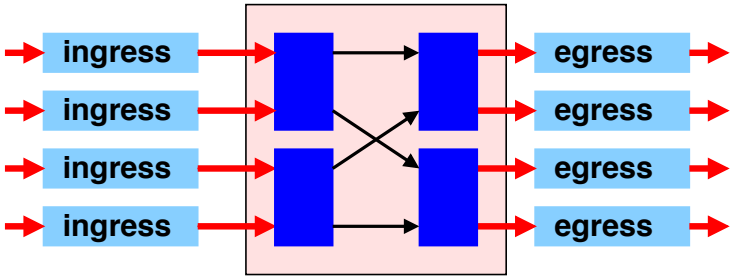


Figure 1.15: Banyan Network

Butterfly topology has many different isomorphic variations, as described as follows:

- Banyan Network (Fig 1.15) is an isomorphic variation of Butterfly topology. It has $N = 2^n$ inputs and $N = 2^n$ outputs, where n is called the dimension of Banyan. It has total of $1/2.N.\log_2(N)$ switches in n stages, each stage is referred as stage i where $0 <= i < n$ [H.J.Chao et al., 2001].

- Fat-tree Topology: Unlike the Butterfly network, a fat-tree network provides multiple paths from source node to destination node. As shown in Fig 1.16, the fat-tree network can be viewed as an expanded n-ary tree network with multiple root nodes. The network delays are dependent on the depth of the tree. SPIN network [P.Guerrier and A.Greiner, 2000] is one design example using 4-ary fat-tree topology for an MPSoC on-chip communication.

FPGA with indirect network topologies

Indirect networks are hierarchical structures; connections between two different blocks at the same level are forbidden, even if the 2 blocks are in close physical proximity. For

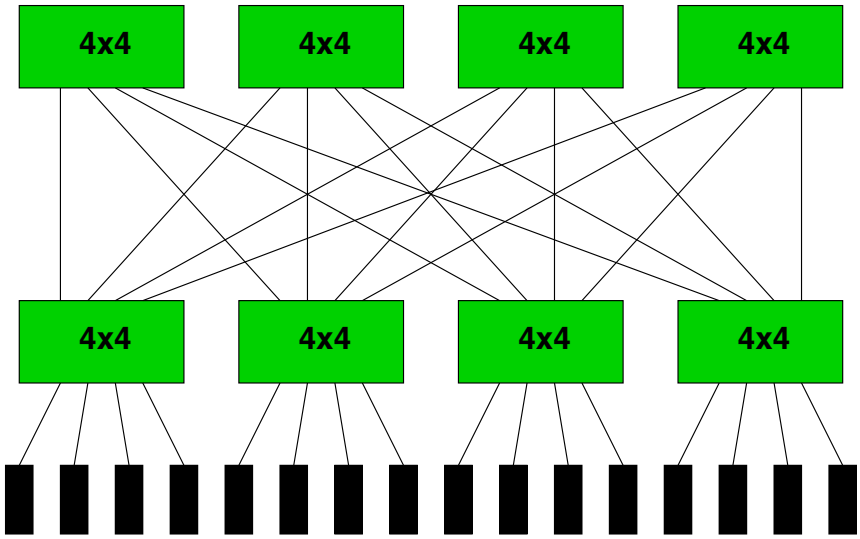


Figure 1.16: Buterfly Fat Tree

this reason, hierarchy was not used at implementing ciruits with many local connections (e.g datapaths). This was particularly important in older technologies where switch delay was more significant than wire delay.
A number of indirect network variations like the Clos, Benes, Baseline, Delta, Omega structures have been explored over the years and used in telephony and computer network research. Such networks have been used in logic emulation systems composed of large number of FPGAs but not really used inside FPGAs. Altera uses hierarchy in many of their CPLD products, like FLEX, APEX and ACEX but they are not strictly hierarchical since direct connections can be made at the lowest level between neighbouring groups [C.Sung et al., 1998].
A well known academic FPGA architecture using indirect network is the Hierarchical Synchronous Reconfigurable Array (HSRA) [W.Tsu et al., 1999]. HSRA has a strictly hierarchical, tree-based interconnect structure (Figure 1.17) where logic and interconnect structures are not closely coupled. The only wire-segments that connect directly to the logic units are at the leaves level, all other wire segments are decoupled from the logic blocks.

1.5 Design Automation for FPGA

Realizing that FPGA performance levels lagged behind ASICs, FPGA architectures were intensely investigated over the past two decades. A major aspect of FPGA architecture research is the development of Computer Aided Design (CAD) tools for mapping applications onto FPGAs. It is well established that the quality of an FPGA-based implementation is largely determined by the effectiveness of its associated collection of CAD tools. Benefits of a well designed, feature rich FPGA architecture might be impaired if

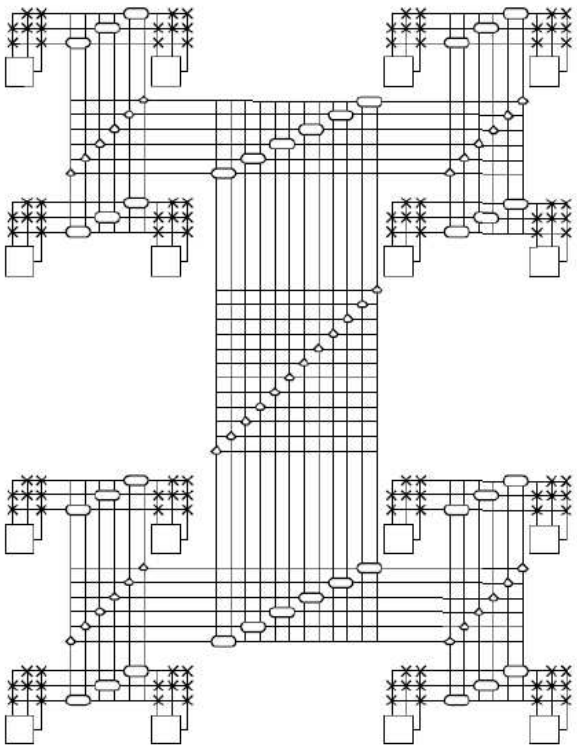


Figure 1.17: Illustration of HSRA's interconnect structure

CAD tools do not take advantage of the specific FPGA features. Thus, CAD algorithmic research is crucial for the necessary architectural enhancement which narrows performance gaps between FPGAs and other computational devices like ASICs.

A typical FPGA CAD tool-flow is shown in Figure 1.18. Initially, a circuit specification of the application is produced either by means of schematic capture, or by a high-level description in a Hardware Description Language (HDL). The appropriate circuit specification is used as input to a Synthesis tool. The Synthesis tool synthesizes the circuit specification into a circuit (we use the terms *circuit* and *netlist* interchangeably from now on) that consists of basic logic gates and their interconnections (hereafter called *nets*). In the Technology Mapping phase, the gate-level netlist is transformed into a functionally equivalent netlist expressed in terms of the logic blocks provided by the FPGA device. The mapped netlist is used as an input to the placement tool, which determines the actual physical location of every netlist logic block in the FPGA layout. When the physical location of each logic block is settled, the routing tool determines the FPGA routing resources needed to route the nets connecting logic blocks already in place. At the end of a successful routing phase, a stream of configuration bits is produced. The configuration bitstream is used to program SRAM cells in the FPGA fabric so that the target application can be implemented.

Each FPGA supplier has its specific CAD Flow with the same successive steps decribed

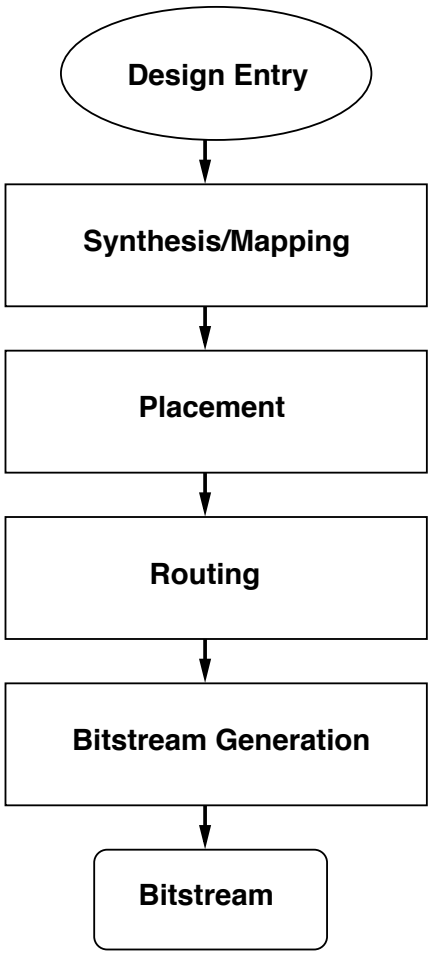


Figure 1.18: CAD flow for FPGA exploration

below. Xilinx and Altera propose their customers the ISE and Quartus software tools respectively.
Considerable work has been done in the academia to develop synthesis, placement and routing tools to evaluate various FPGA architectures.

Mapping

For LUT-based FPGA, mapping is the process of translating the circuit description into a netlist of LUTs and Flip-Flops. The usual cost functions are area and delay. SIS [E.Sentovich, 1992] is an open source tool that can optimize the network and convert it into a netlist of LUTs. Several algorithms can be used for mapping, such as FlowMap [J.Cong and Y.Ding, 1994] and CutMap [J.Cong and Y.Hwang, 1995]. These algorithms allow the size of the LUT to be modified and may optimize delay and area. Recently the University of Berkeley distributes ABC [A.Mishchenko, 2005] the successor of SIS, a growing software system for synthesis and verification of binary sequential logic circuits.

Placement and Routing

Placement problems in both FPGA and VLSI environment are fairly similar. Therefore these techniques were udapted for use in FPGA. Placement algorithms are designed to minimize both routing demands and critical-path delays. Routing demands are reduced by placing highly interconnected logic blocks together closely. Critical-path delays are minimized by moving together logic blocks which are on critical nets. There are several general approaches to placement algorithms: min-cut [D.Huang and A.Kahng, 1997], analytic [B.Riess and G.Ettelt, 1995], and simulated annealing placement algorithms [M.Huang et al., 1986].
Most research work in FPGA CADs has been done for developing the router. Routing consists in connecting different LBs together, after their position have been fixed by the placer. Since the number of tracks in a channel is set and possible connections between tracks are fixed, routing is a critical step for FPGAs. The main goal of routing algorithms is avoiding net congestion while minimizing critical-path delays. Net congestion is avoided by balancing the use of routing resources; critical path delays are minimized by giving higher priority to high criticality nets.
The Versatile Placement and Route (VPR) [V.Betz and J.Rose, 1997] encapsulates the entire placement and routing tools. VPR is targeted at symmetric mesh architecture. The tool allows specification of the various parameters such as logic bloc size, channel width, S block type and flexibility F_c in C blocks. Simulated annealing is used for the placement step. The router is based on the Pathfinder algorithm [L.McMurchie and C.Ebeling, 1995] which focuses on congestion avoidance and delay minimization.

Bitsream Generator

output of placement and routing tools consists in the route taken by each source-sink pair in the netlist. The bitstream generator tool is usally employed to extract the programming bits needed to implement the user-defined circuit.

1.6 FPGA characteristics and challenges

An overall view of conventional, reconfigurable devices shows that 80-90% of the area is dedicated to the reconfigurable interconnect. The remaining area is dedicated to reconfigurable logical blocks. This 80-90% area includes switches, wires and configuration memory which are reserved for interconnect. The remaining area is dedicated to reconfigurable logic blocks.

To illustrate the magnitude of this problem, an area profile is given in table 1.1 [G.Lemieux and D.Lewis, 2004]. From this table, it can be seen that FPGAs area profiles are approximatively 80-90% for routing and only 10% of the area is used to implement logic directly. Inability to meet timing requirements, power consumption, or logic capacity constraints make it infeasible technically to use FPGA in the most demanding applications. De-

Resource	Details	Proportion of area	
		Range	Average
Logic	flip-flops, lookup tables, lookup table input buffers	**8-16%**	**12%**
Routing	lookup table output buffers and switches	8-10%	9%
	BLE input multiplexers	18-36%	27%
	LB input multiplexers	11-13%	12%
	cluster input buffers (track buffers)	7-11%	9%
	routing switches (switch blocks)	23-39%	31%
	Total routing	**84-92%**	**88%**

Table 1.1: Area profile of a mesh-based FPGA [G.Lemieux and D.Lewis, 2004]

spite their design cost advantage, FPGAs impose large area overheads when they are compared to custom silicon alternatives. The interconnect and configuration overhead is responsible for the 40x density ratio disadvantage [I.Kuon and J.Rose, 2007] incurred by reconfigurable devices, compared to hardwired logic.

According to [E.A.Kusse and J.Rabaey, 1998], FPGA energetic losses come from 3 main sources following the distribution described in Fig 1.19. Clearly the most dissipat-

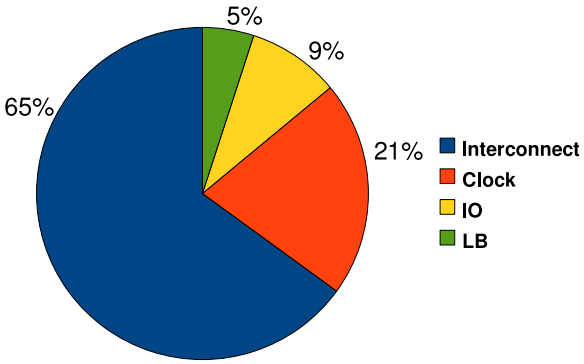


Figure 1.19: Average Power Breakdown for a FPGA

ing element is always the interconnect, due to the great number of wires and switches necessary to maintain high FPGA flexibility as explained in the previous paragraph. Then comes the clock distribution network, a very consuming element in all VLSI devices. Finally, we have the logic part of the FPGA (mostly CLB).

As discussed in [V.George and J.Rabaey, 2001], attempting to design low energy system requires efforts at all levels of design abstraction. An efficient approach to reduce power consumption in FPGA must comply with the following order:

1. Architecture level : Change architecture to decrease the wires number and their loading capacity.
2. Logic level : Use various techniques and devices to decrease the dynamic and static losses.
3. Layout level : Adapt the technology parameters to decrease the leakage current losses.

Anyone seeking to understand the design of an FPGA must delve into the complexities of programmable routing architecture. There is a considerable demand for FPGA with lower area, lower delay and reduced power dissipation. One general method used to make FPGAs more efficient is to improve the numerous algorithmic steps which map a logic circuit into an FPGA. Improvements to the logic synthesis step can reduce the amount and the depth of logic needed. Further improvements to the partitioning, placement, and clustering steps, such as described [V.Betz et al., 1999], can reduce interconnect use and delay by shortening connections. Similarly, improvements to the routing step can enhance map critical delay paths to get faster connections. Other works such as [K.Poon and S.J.E.Wilton, 2002] improve the CAD flow to reduce the overall FPGA power consumption.
The definition of an FPGA architecture consists in determining the logic and routing resources, so that these resources produce the most efficient results possible. Since both the algorithm and the architecture can be defined simultaneously, there is a significant amount of interaction which can influence the final result. The scope of this design problem has motivated a considerable amount of research to improve FPGAs efficiency.

1.7 Conclusion

After we defined clearly what an FPGA is made of, it is important to identify the various FPGA penalizing causes and to try to alleviate them. Three factors allow to determine the characteristics of an FPGA: quality of its architecture, quality of CAD tools and the manufacturing process (including all its electric advantages).
The gap between FPGA and ASIC is illustrated by area, performance and power consumption discrepancies. ASICs are still more efficient than FPGAs but lack some flexibility. To enjoy the benefits of programmable logic, we distinguish 2 alternatives for FPGA that will be explored in the following chapters:

- ASIC designers must add some flexibility to their design by integrating embedded FPGA or by developing Domain-Specific and tailored FPGA (Flexible ASIC). Automating FPGA layout should be quite effective for fulfilling this purpose, and thus getting closer to ASIC design.

- FPGA designers must increase logic density and optimize interconnect since it is the most dominant resource. The most used and studied architecture is the Mesh-based one. New architecture should be developed with better quality of routing network. This would be beneficial for stand-alone, domain specific or embedded FPGAs.

2

Automating Layout of Mesh Based FPGA

The main purpose of our work is to reduce the design cycle time for the developement of an island-style FPGA core layout. We develop a technology-independent layout-generator which can be adapted to any standard cell library geometry and to any process rules. It also offers a garantee of portability and compatibility with the overall ASIC design environment. The proof-of-concept we created is a set of FPGA cores validated in 0.12μ process from STmicroelectronics.

2.1 Introduction

FPGA are getting more prevalent in digital systems and have a wide range of applications, ranging from telecommunications switching systems to wireless interfaces. Currently, the transistor-level design and layout of an FPGA core is done mostly by hand; developing a new FPGA is a time consuming and challenging task. It is reported in [K.Padalia et al., 2003] that a new FPGA design involves approximately 50 to 200 person-years. It is an interesting option to reduce significantly the time-to-market of the product at the expense of limited area penalty. This result can be reached by automating the complete FPGA design process.

A number of previous attempts were made regarding automated generation of FPGAs.
Phillips and Hauck focused on the automatic layout of domain specific reconfigurable systems [S.Phillips and S.Hauck, 2002] [S.Phillips et al., 2004]. The first step of the approach is a high level architecture generation, followed by different methods for au-

tomating the layout process. Their findings was that the automated approach yielded a layout 42% larger and 64% slower than a manual design. By reducing functionality to what is required for a specific domain, the authors successfully demonstrate that smaller and faster layouts can be created automatically. Using standard cell libraries, Cadence tools [Cadence, 2006] are used for the VLSI layout generation. Since it applies a global place and route for a flattened design, this method cannot take advantage of FPGA regularity. Phillips and Hauck do not report any manufactured chips resulting from this work.

Kafafi et al. propose a new architecture and an implementation using standard cells to construct an FPGA from a VHDL input description [N.Kafafi et al., 2003]. Their context is the creation of FPGAs embedded within a non-programmable ASIC. The layout for this logic, called a synthesizable embedded programmable logic core, is easily created using commercial synthesis, placement and routing tools. The authors adopted this approach to allow for easy integration in system-on-a-chip applications and targeted a very specific architecture, suitable only for small embedded cores. Nevertheless, this approach could also simplify the design of standalone FPGAs. Its disadvantage is, as expected, a significant penalty in terms of area. In [N.Kafafi et al., 2003], the authors estimate that an automatically generated layout is 6.4 times larger than the equivalent layout created manually. Like Phillips and Hauck work, this approach does not consider the regularity of the FPGA core which penalizes speed and performances.

GILES [I.Kuon et al., 2005] work made a major contribution in this domain. It demonstrated a complete automation of FPGA creation with significantly reduced manual labor. The GILES tools are used to generate the tile netlist and placement which are exported to Cadence's Virtuso Assembly platform [Cadence, 2006] for the routing. To form the complete FPGA, tiles are then abutted together using Cadence scripting language named SKILL [Cadence, 2006]. GILES uses an appropriate library (which includes basic cells: LUT, buffers, muxes, etc.) that restrict migration to various industrial processes and integration to typical ASIC design flow (Timing, area or power oriented flows).
It is demonstrated that GILES, using automated tools, can produce a layout 36% larger than a commercial FPGA device layout. Kuon et al. [I.Kuon et al., 2005] report that an FPGA tape-out created with this methodology in a 0.18 um process by TSMC did operate successfully.
The flexibility reported above highlights one of the gains of automated design we hope to achieve with our methodology.

This chapter presents the generic method and the total flow to automate the process of producing the transistor-level layout of an FPGA core and all steps involved in creating the FPGA chip. In the following sections, we present a general methodology with the adequate CAD flow allowing designers to create their own FPGA with their own specifications. This method is more suitable for automated FPGA generation because prior work restricts the number of inputs, outputs, or switches; moreover there are re-

strictions on the basic cell library and technology choices too.

2.2 Adaptive VLSI CAD Platform

Over the past decade, research has been conducted at the university of Paris-6 to produce a complete open source VLSI CAD flow named ALLIANCE [Alliance, 2006]. As shown in figure 2.1 ALLIANCE offers open-source CAD tools for simulation, synthesis and VLSI place&route.
More recently, additional research at University of Paris-6 created a new physical design platform named CORIOLIS [C.Alexandre et al., 2005]. This research adopted the approach of creating a circuit design environment, allowing physical CAD tools exploration leading to valid architectural decisions. CORIOLIS provides high level C++ associated to Python APIs, along with a unified and consistent hierarchical VLSI data model through all design steps, from logic down to final layout. It provides an open CAD environment (both for design flows and algorithms) by offering a progressively enhanced set of CAD tools such as Zephyr [C.Alexandre et al., 2006](static timing analyzer), a Python extension language STRATUS [S.Belloeil et al., 2007], a set of core functionalities such as a "lef/def" interface, and a graphical user interface(figure 2.2). We extend ALLIANCE and CORIOLIS platform to take into account physical layout constraints in the context of FPGA-architecture, as well as the automated FPGA layout procedure.
We apply an efficient scheme to integrate designer-guided action in the automated core generation, by using the procedural language STRATUS. In this work the procedural techniques perform power routing, clock distribution, configuration memory placement, interface connector placement, tiles abutment, and route specific nets. ALLIANCE VLSI CAD tools (OCP for placement, NERO for routing) are used to complete automatically the remaining layout.

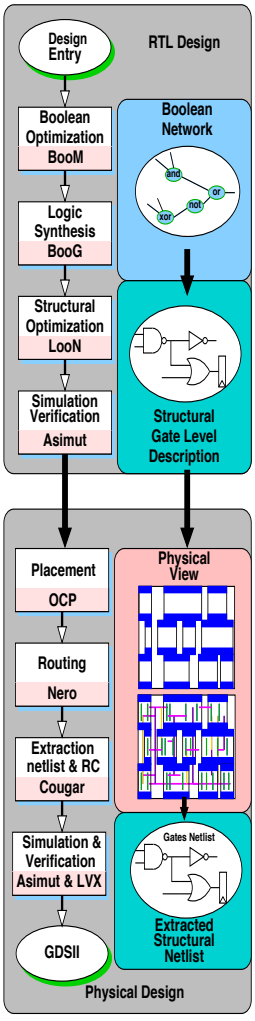


Figure 2.1: Alliance CAD Flow

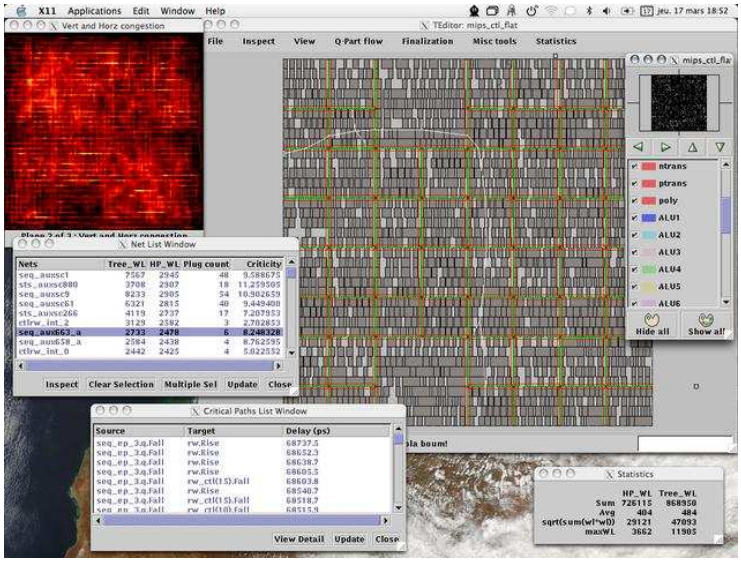


Figure 2.2: Screenshot of the Coriolis platform in action

2.3 Circuit Design: Architecture generator

2.3.1 Architecture Modelisation

In our approach, an Island Style FPGA is defined by 3 main elements: Logic Block(LB), Switch Block(SB) and Input/Output Block(IOB). We use an architecture file to describe the connections topology of every block as follow:

Logic Block: This block can be considered as a black box with a fixed number of inputs and outputs distributed on the 4 sides of the box and connected to the adjacent routing channels as shown in figure 2.3. Every input or output pin has an associated connection vector showing the associated fraction of tracks where it can be connected. Pins connected to global network like the clock are distinguished by the key word *global*.

An example of LB interface and its corresponding description file is presented in figure 2.3. The structural netlist of the LB interface is generated automatically and corresponds to programmable multiplexers for the inputs and a set of tristates for the outputs that drives the routing channels. Figure 2.4 shows an example of input and output vectors.
The internal HDL logic description (netlist) of the LB is written by the designer. Generally it contains a number of different or similar subblocks interconnected together by the internal network. Subblock is generally K-inputs Basic Logic Element (BLE) that contains K-LUT, flip-flop and a bypass multiplexer. Figure 2.5 shows an example of 4-BLE instantiated later 4 times in the clustered LB with a cross-bar for local interconnections.

```
# CLB specification
.BLOCK CLB clb_ref
.in_per_block  4
.out_per_block 4
.ck_per_block  1
.channel_width 7 #adjacent channel width

.port pin0 input top      0  #plug[0]
con_switch: 0110011
.port pin1 input bottom   1  #plug[1]
con_switch: 0110011
.port pin2 input left     2  #plug[2]
con_switch: 1011001
.port pin3 input right    3  #plug[3]
con_switch: 1011001

# The same output exists in 2 sides TOP and RIGHT
.port pin4 output top     4  #plug[4]
con_switch: 011001
.port pin5 output right   4  #plug[5]=plug[4]
con_switch: 1100110
.port CK input top        5
con_direct: global
```

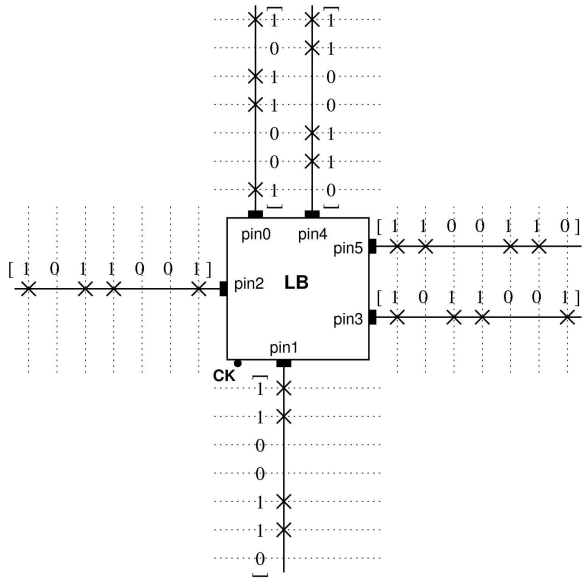


Figure 2.3: Example of Logic Block Model with 4 inputs and 2 outputs

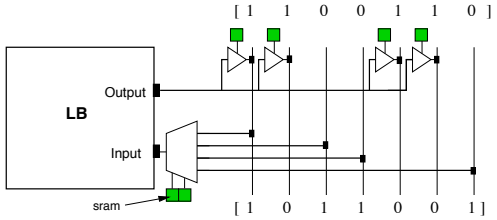


Figure 2.4: Logic block input and output

Switch Block Each Switch Block (SB) has 4 different sides: Top, Bottom, left, right. Connections between the different sides are described in 16 tables shown in figure 2.6. Tracks entering through one side can reach 3 opposite sides or return to the same side. Therefore 4 tables are associated to each source side: Side1->Side1, Side1->Side2, Side1->Side3 and Side1->Side4. Each table has $W1$ rows, $W2$ columns (with $W1$= channel width in source side, $W2$= channel width in sink side) and displays the connections from a track in the source side to tracks in the sink side.
Figure 2.6 shows the example of the *Disjoint* switch block with $W_{Top} = W_{Bottom} = W_{Left} = W_{Right} = 4$. Note: '1' means there is a configurable connection, and '0' means there is no connection. This technique of model presentation allows to define any switch block topology.
Programmable SRAM-based switches within the SB allow connection between two wires. they can be implemented using either pass-transistor, transmission gate or tristate buffer as illustrated in figure 2.6. Generally tristates will be used since standard cell librairies provided by foundries do not contain pass-transistor or transmission gates. Introducing these gates needs to develop specific cells in a standard cell manner including lay-

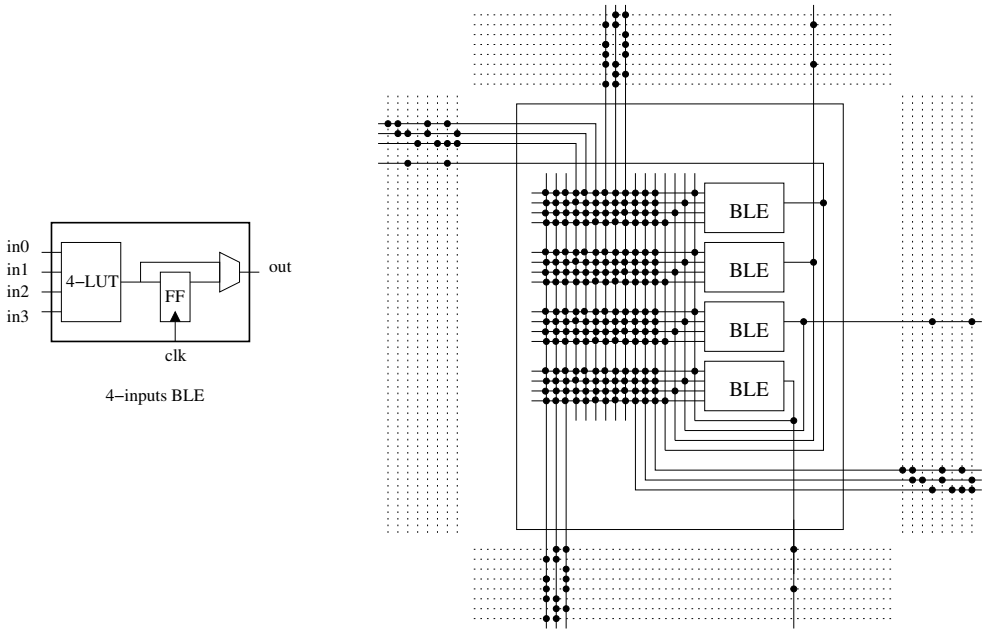


Figure 2.5: An example of clustered Logic block and its internal structure

out, schematic and electric models that must be compatible with standard CAD flow (place&route, timing analysis etc.). Even if we use pass-transistors, the interconnect requires buffers and to avoid conflict each signal direction has its pass-transistor. The SRAM can be shared between 2 pass-transistors connecting 2 wires. For the genericity and the simplicity of automatic netlist generation, every switch is controlled by its independent SRAM cell.

Input/Output Block: Like the LB, the IOB is considered as a black box with 1 output pin and 1 output pin. Each pin is connected to the adjacent routing channel respecting the associated connection vector as shown in figure 2.7.

2.3.2 Generic mesh FPGA model

The first approach to generate a mesh-based FPGA netlist is to break the problem in two levels of hierarchy. At the bottom level, the main blocks are defined in the architecture file as shown previously and the black boxes corresponding to the IOB and LB are written by the designer in high level design language such as VHDL. The top netlist is composed only of IOB, SB and LB and determined automatically by duplicating these building blocks as shown in figure 2.8.
To simplify the problem for the physical layout, we adopt a second approach which consists to compose the FPGA in the form of regular array as shown in figure 2.8. Each tile of the array has a square area and refers to a logic block core, a switch block, and one set of adjacent horizontal and vertical routing tracks.

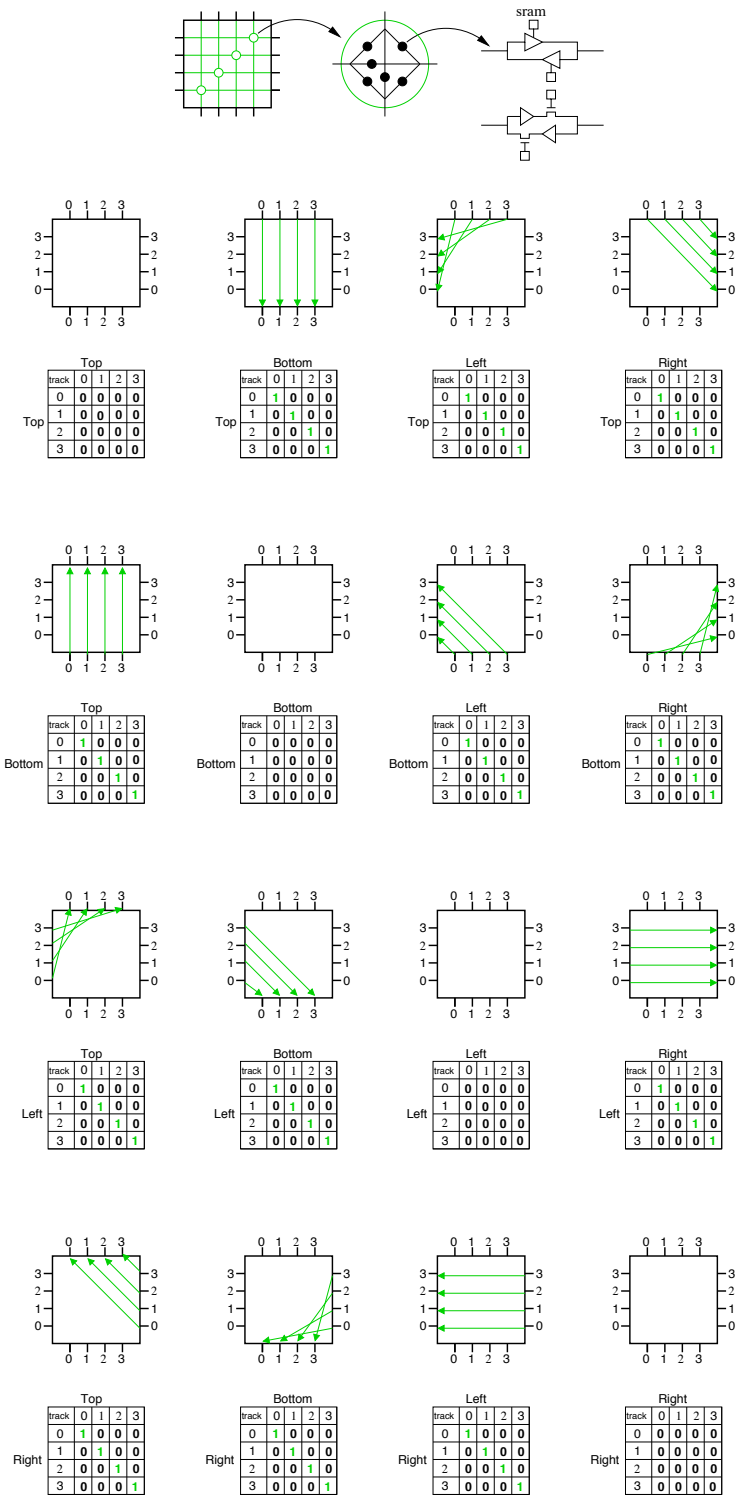


Figure 2.6: Switch Block Description: Disjoint

```
# CLB specification
.BLOCK IOB iob_pad
.channel_width 7 #adjacent channel width

.port pad_out output none      0
con_switch: 1011001
.port pad_in  input  none      1
con_switch: 1100110
```

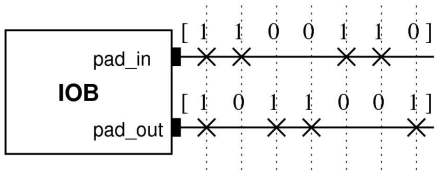


Figure 2.7: Input/Output Block Description

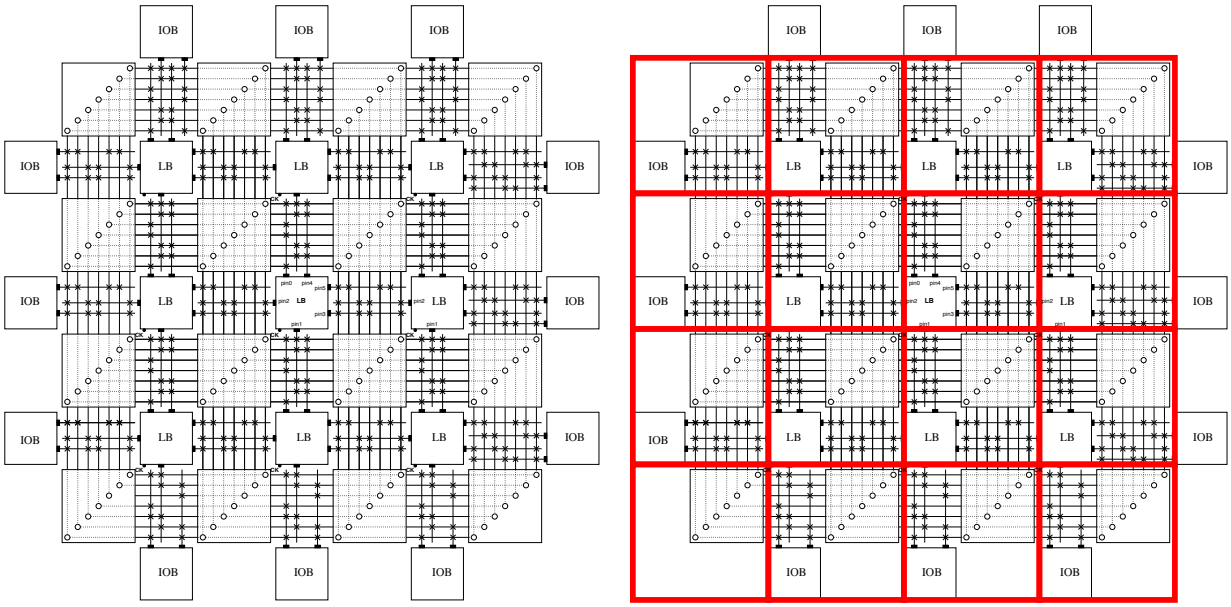


Figure 2.8: FPGA array and tile

2.3.3 FPGA Tiles

A set of 16 different tiles which are abutted repeatedly to compose the whole FPGA array are identified (figure 2.8). The main tile is the **basic** tile, whereas the other tiles are their derivations. The basic tile includes a complete switch block and part of the logic block to form a regular square box as shown in figure 2.9. Input/Output vectors of logic block which exceed the outline of the tile are associated to the adjacent tiles.

The tiles on the leftmost column and the bottom row do not contain any logic block. They contain only a channel which connects adjacent IO pads to the adjacent logic block input. The rest of the tiles contain a top horizontal channel, a right vertical channel, a switch box, and a logic block. Any array size can be generated by replicating these tiles. An important aspect in the tile based design is that adjacent sides of 2 abutted tiles must have equal length. While deciding sizes of the tiles, priority is given to the tile which is used most; in this case it is the **basic** tile. The sizes of the other tiles are adjusted accordingly.

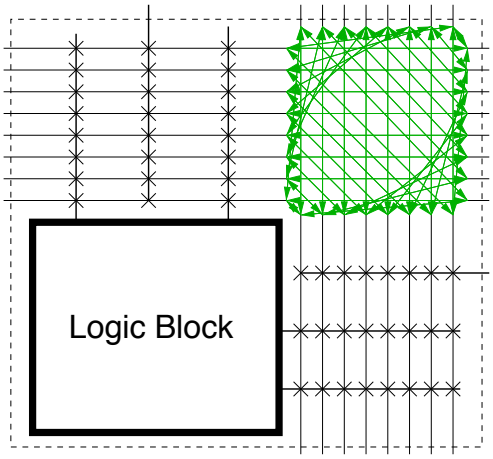


Figure 2.9: Basic Tile Topology

Each tile generator is written in the STRATUS language. The tile generator receives a set of architectural parameters as inputs (from the architecture file). Then it generates the tile netlist in accordance with the given parameters. Loops and conditional statements are used to generate a tile for any parameters. The netlist of each tile is generated directly using the standard cell library. Python routines are also merged in the tile generator for generating VHDL model of specific components. These components are synthesized by the Alliance synthesizer named BOOG. The netlist of each tile is mapped directly onto the standard cell library. The generated netlists of all tiles are passed to the architecture generator which links them together to construct the top level netlist of a complete FPGA. This generated netlist may be integrated in any larger application and design flow.

2.3.4 Programming access

As shown in figure 2.10, we adopt the random access method to write the configuration. Configuration bits are grouped in words and can be programmed selectively. Adresses and data buses are routed through the entire array and additional circuitry is used for decoding. For each configuration cycle, the row and column decoders identify the word being programmed, the data bus distributes the configuration memory content. This programming technique allows direct and independent access to configuration memory to change only the resources in use. Moreover, this method allows the use of simple cell such as latch or SRAM for storing configuration information.

An NX by NY FPGA contains (NX+1) by (NY+1) tiles; NX+1 is the total number of columns and NY+1 is the total number of rows. Each FPGA tile comprises a set of SRAM bits arranged in multiple rows. The SRAM bits in a row are called a **word**. In order to write a tile word data, a row number, a column number and a word number must be

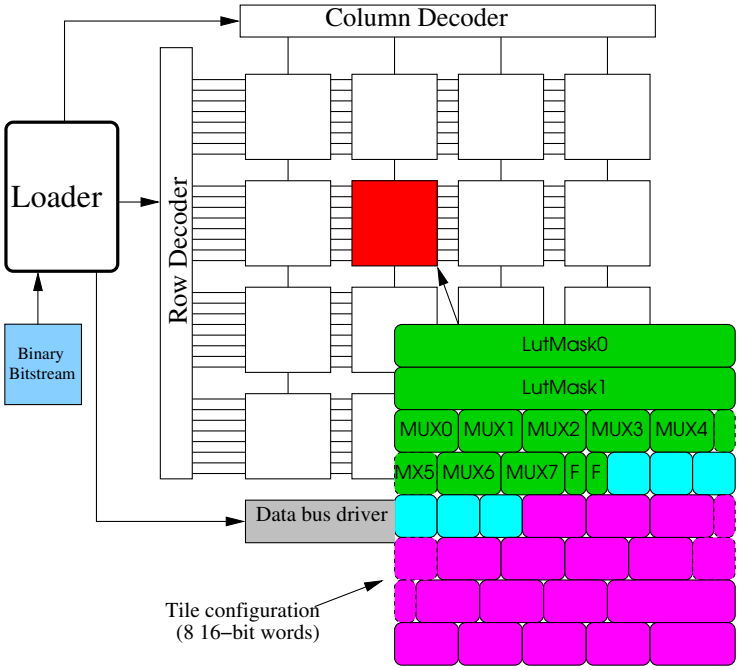


Figure 2.10: FPGA Configuration Technique: Random Access Memory

specified. The row and column numbers give the location of the tile in the FPGA array, whereas the word number gives the location of the word in a tile. All adress and data bits are passed to the loader. Thanks to the row, column and word decoders, the exact strobe is turned on. Thus when word's strobe turns high, the data is written into the specific word of the requested tile. This process is repeated for all words of all tiles. The loader and decoders are generated automatically by the FPGA generator.

2.4 VLSI Layout generator

The use of standard cells requires an efficient optimized action on the FPGA architecture. This necessity arises because the circuits are designed by a general method, and thus do not inherit any design tradeoffs that are implemented when the full custom template is used. On the other hand, standard cell method offers an opportunity to optimize FPGA performances. For example, if a designer requires a low power design, then it is possible to use a low power library instead of a library that is optimized for area or performance. In addition, use of the standard cell method allows the designer to integrate easily the structures that he created into the typical ASIC design flow, since standard cell flows are widely used throughout industry.
The typical standard cell method cannot take advantage of FPGAs regularity if generic steps are used. In this section, we explore the possibility of creating regular FPGA struc-

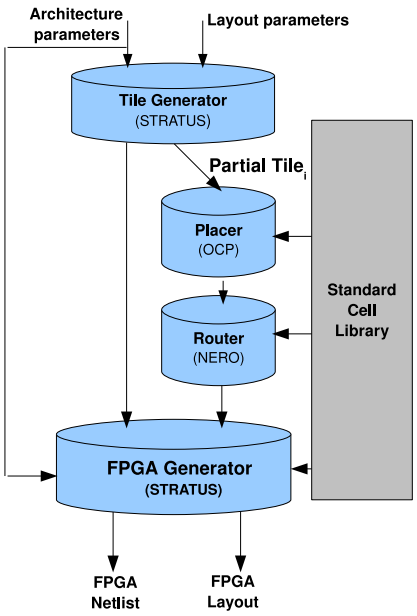


Figure 2.11: FPGA layout core generation flow

tures by extending the standard method.
Our generator adopts a Bottom-Up methodology to build up the FPGA core from the lowest level where it generates basic tiles. The scalability of this tiles can be modulated to obtain the desired FPGA fit. Figure 2.11 gives an overview of the different generation phases.

The VLSI Layout Generator automates the creation of mask-ready layouts from the circuits provided by the *Architecture Generator*. It generates a layout for every tile, then the complete FPGA core and the adequate loader mask are assembled.
The layout generator must be able to create layouts for any conceivable circuit that the high-level architecture generator can produce. In this work we investigate standard cell methods for layout process automating.

This generator works in 3 steps. In the first step, it generates hierarchical netlist of the FPGA core including the top level netlist, the basic tiles and the loader netlists. Second step, it generates a partial layout using generic parameterized algorithms. The partial layout is generated to obtain a fast bitstream configuration mechanism, an efficient power routing and a balanced clock distribution network. Final step completes the remaining layout using automatic placer and router. This two-step scheme allows better layout handling, according to initial constraints.

2.4.1 Tile Layout

The tile generator generates both the netlist and the partial layout of a tile. The partial layout is generated with the help of parametrized algorithms which take a set of layout parameters as inputs. In this case a partial layout is performed to :
- adjust abutment box.
- place critical cells (SRAM, Flip-Flop).
- place peripheral pins in order to overlap with adjacent tiles pins
- route tile power grid.
- route critical nets (clk, data, strobe).

Knowing the library cells characteristics, total tile area is estimated and its dimensions

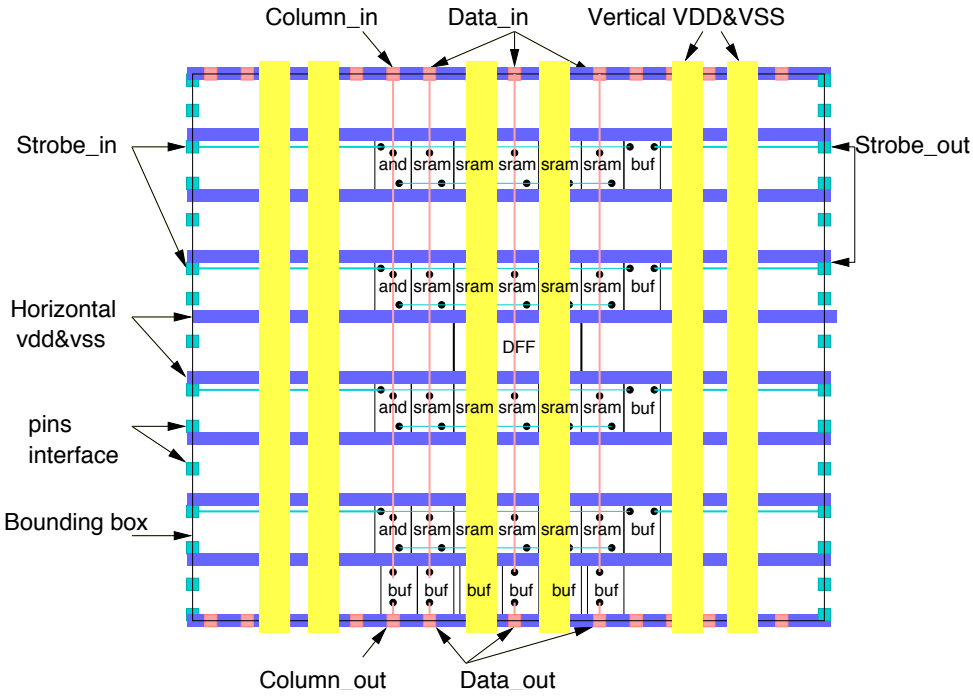


Figure 2.12: Partial Layout

are saved. The X and Y dimensions of the tile are adjusted properly to make sure that a tile does not waste any extra space.
The partial layout generator places all SRAM bits in rows and columns with a fixed distance between 2 rows, as shown in figure 2.12. Each SRAM bit in a row receives a vertical data signal, and an horizontal strobe signal. Data bits are written in all SRAMs of a row only when strobe is *"high"* for that row and when the column is *"high"* for the complete tile. The column and strobe signals come from a bitstream configurator (loader), which is discussed later in section 4. The column and data signals from the top are buffered before they exit on the bottom side of the tile. Similarly a strobe signal from the left is buffered before it exits on the right side of the tile.
The algorithm starts placing the configuration cells from a layout parameter named

'Start Position'. Similarly the height and width of a tile and the total SRAM bits are also adjustable parameters for each different channel width. These layout parameters change with the number of SRAM bits. For this purpose a small database is created which specifies all these variables for different channel widths. The layout algorithm and the database specification are generic enough to handle other architectural parameters that are not yet generic.

Power routing:

The layout generation algorithm generates horizontal and vertical power segments as shown in figure 2.12. The alternating VDD and GND segments in the horizontal direction are fixed, whereas the placement of vertical power segments is supported by few layout parameters. The total number of vertical segments for power and ground in a tile, their positions and their widths are defined in the layout database. These values can be changed for tiles of different sizes. Horizontal power segments use the 1st and 2nd routing layer, whereas vertical power segments use the 5th routing layer.

Pin generation:

In island-style FPGA, tiles connect together by abutment, and pin locations on the boundaries of adjacent tiles must overlap. The positions of few of these pins are calculated on the basis of the layout parameters found in the database. Since the database is common to all tiles, thus the pin abutment problem does not arise for these pins. There are other pins which do not have fixed positions. Since the final automatic placement of all tiles is done independently, it is difficult for the placer to choose correctly the pin locations of the tiles. Consequently a generic algorithm is written to place all remaining pins. This algorithm places the pins in the 4 directions of the tile and ensures that the pins are not congested in a limited place. It utilizes all available space and tries to distribute the pins with equal spacing.

Clock generation:

In this work, we use a tile based approach to route symmetric clock network. A single tile mask is used to reserves routing ressources for clock tree inside any tile as shown in figure 2.13. These ressources correpond to vertical and horizontal tracks in metal 3, 4 and 5. In addition, there is an allocated area for clock buffers in the 4 corners of the tile. This tile mask is automatically merged with the FPGA tiles during the partial layout phase.
The main advantage of this procedure is that we obtain a generic tile based and a balanced clock distribution network. Once tiles are abutted, allocated area and routing tracks give the necessary ressources to route a symmetric clock tree. Figure 2.15 shows how we use the reserved area and tracks to route a balanced clock tree and to put the

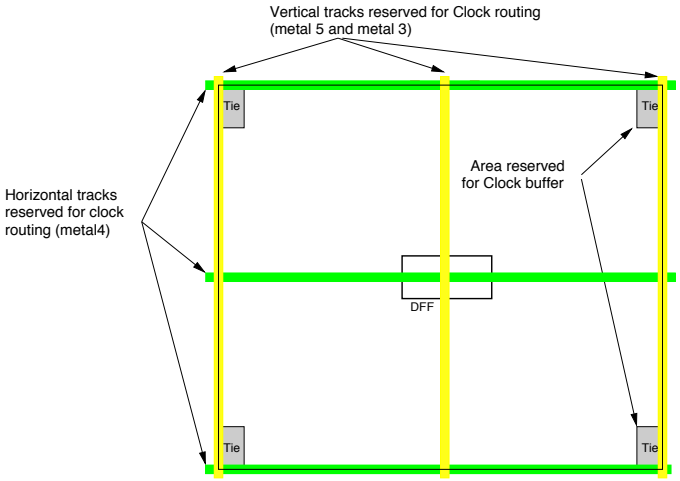


Figure 2.13: Tile mask for clock routing

clock buffers in the right place. Since the clock routing is totally automatic, we can implement new generic algorithm for other clock distribution networks. The main point to consider for a new clock routing algorithm is the placement of clock buffers; the routing wires must overlap with mask tile.

Automatic placement & routing

After the partial layout generation step and ressources reservation for clock routing, automatic tool places the remaining logic of the tile netlist in the free area. The placer automatically adds empty cells to fill up any extra space. Then automatic routing tool is executed to route all unrouted nets inside the tile.
All the tiles are separetly routed using 4 routing layers. Metal layers 5 and 6 are reserved mainly for the clock tree and power segments. ALLIANCE automatic tools OCP and NERO are used for placement and routing respectively. Cells placed and nets routed in the partial layout are respectively respected and not modified by OCP and NERO. The NERO router also avoids using the metal ressources reserved for clock tree. An example of a complete tile layout is shown in figure 2.14.

2.4.2 FPGA layout

Logic array generation

In this work, the logic array is implemented in 5 metal layer and using SXLIB [Alliance, 2006] which is a library of standard cells designed with symbolic rules. The overall process of chip layout generation is shown in fig 2.17. The regularity of the island-style simplifies the automatic structuring of an FPGA Layout. Parameters for the generator are the number of LB per Row and the number of LB per column.

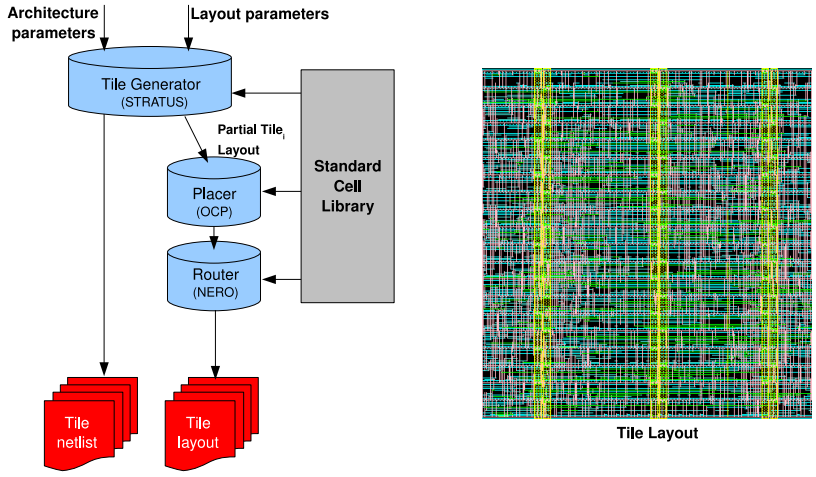


Figure 2.14: Tile Layout Generation

The FPGA generator builds the FPGA array by replicating tiles, then it generates the power ring and the programming infrastructure that integrates the loader and adress decoders. Finally it connects clock signals internal to the array using reserved ressources in internal tiles and consider how these signals are distributed to generate a balanced clock network as shown in figure 2.15. This automatic design system is scalable to large FPGAs and includes realistic features of real FPGA chip. This flow takes into account the periphery of the FPGA core for connecting I/O pads which are placed later using industrial tool.

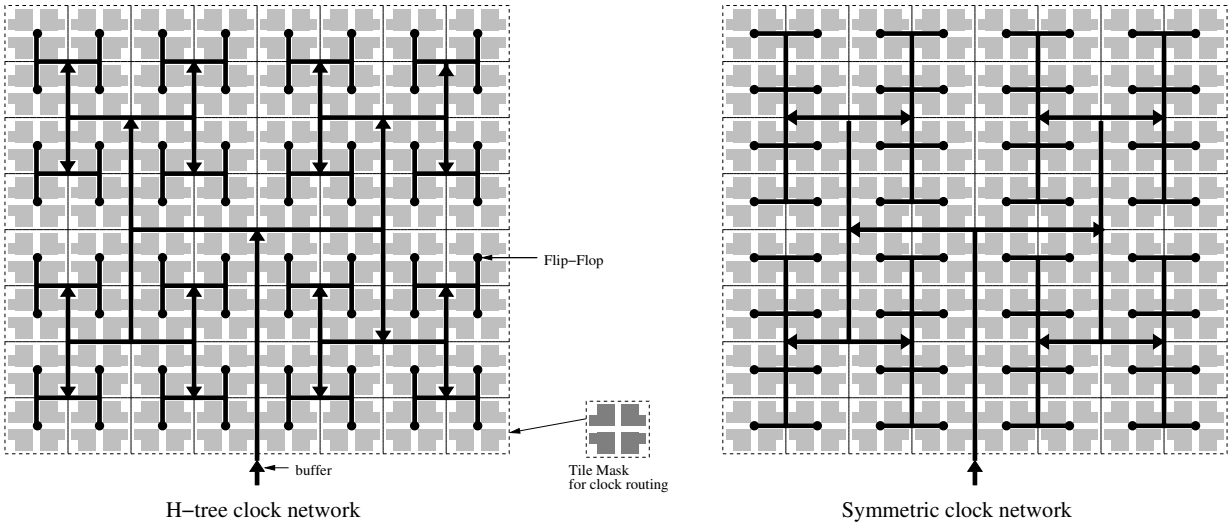


Figure 2.15: Examples of clock network distribution using allocated ressources

To prove that this generator can be used to design FPGA, we export the symbolic

array to real layout. S2R [Alliance, 2006] is an automatic tool intented to export the symbolic layout to the real layout in 0.12 um fabrication process from STmicorelectronics. Access to this technology was provided by the CMP [CMP, 2006]. Our design is generated in GDSII format and is compatible with industrial desgin flow such as Cadence's Virtuoso design platform and Encounter design platform [Cadence, 2006]. As shown in figure 2.18, industrial tools can be used to place I/O pads and also to check the FPGA design rules and circuitry (DRC and LVS).
Figure 2.19 shows a representative FPGA chip that contains 1024 tiles arranged in a 32 by 32 array. The array is surrounded by 73 input pads and 32 output pads and 7 control pads (configurations pads, scan, clk). The routing architecture consists of 8 tracks per channel. All routing tracks have length one and use buffered switches. Every logic block contains one 4-LUT, 4 inputs and 2 outputs. All inputs and outputs pins of the logic block are fully connected to the adjacent channels as shown in figure 2.16.
This chip is fully generated using the design flow presented in this chapter and complies with 0.12μ ST process design rules validated by DRC and LVS check. The 32x32 FPGA takes an area of 3885.6μ by 3882μ.
As mentioned in 2.1, our approach has not any restrictions on the basic cell library or technology choices. It is technology-independent approach since we use standard cell library as provided by the foundry and we don't need any specific cells. Our automatic approach using standard cell SXLIB library generates a tile layout with a total area 4.5 times larger than the equivalent full custom layout designed with optimized basic cells such as transmission gates and 4-LUT presented in chapter 5.
The architecture choices has an important impact on the area overhead. In this work we developed FPGA with bidirectional wires which require the use of three states logic gates, and we are penalized when we use tristates as a standard cell instead pass-transistors (which is not a standard cell). If we target other architectures with unidirectional wires, both full custom and standard cell approches use he same architecture of multiplexers, thus the area overhead will corresponds to the common area overhead between full custom cells and standard cells. We note that the overhead ratio between full-custom and standard cell designs is decreasing with the advance of technology process (because metal wire width decrease slowly compared to transistor width): in 65 nm, a standard cell design can be 1.5 times only larger than the equivalent full custom one. We can also improve the area if we use specific library of cells as done in the GILES work [I.Kuon et al., 2005] where the area overhead represents 36% only.

2.5 Embedded FPGA

Another motivation for designing FPGA networks of any size is provided by the today's trend of implementing system-on-chip (SoC) and platform-based designs. These designs could benefit from embedding programmable logic cores on-chip and combining multiple implementations into one piece of silicon. A number of embedded reconfigurable core products are been developed by companies such eASIC [eASIC,

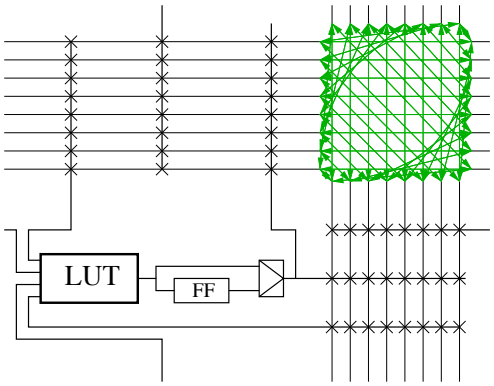


Figure 2.16: Logic block topology

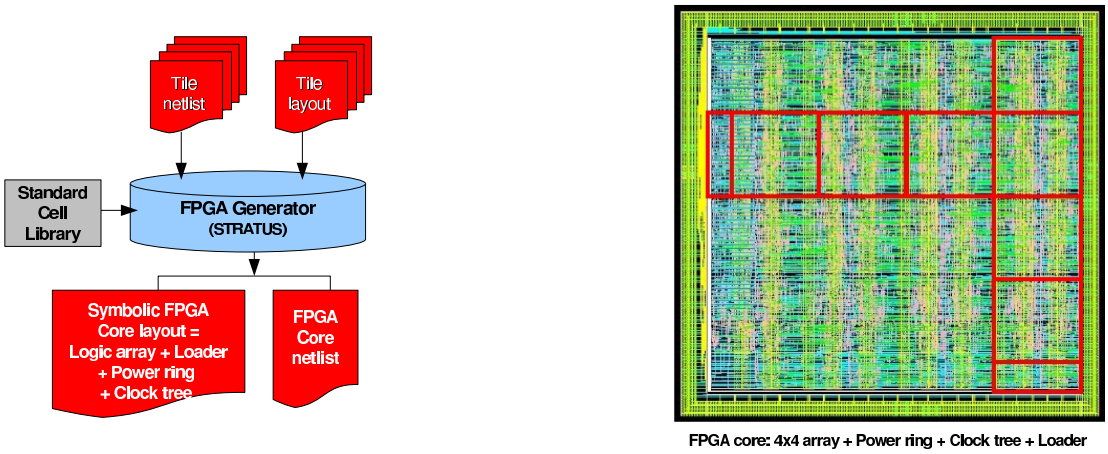


Figure 2.17: FPGA core generation

2008], m2000 [M2000,] and Actel [Varicore, 2001]. Several integrated circuits containing programmable logic cores are described [P.S.Zuchowski et al., 2002] [T.Vaida, 2001] [M.Borgatti et al., 2002]. Other efforts to synthesize FPGA cores are reported in [A.Yan and J.E.Wilton, 2006].
There are numerous other reasons to prefer embedded FPGA cores to regular custom logic in SoC or platform applications. First, the use of FPGA cores retains all traditional advantages of FPGAs, such as making changes to the circuit late in the manufacturing cycle to correct design errors or to comply with emerging standards (post-fabrication flexibility). It broadens market prospects for the same device, since it can be more easily adapted to work in different customer environments or to include entirely new features. Finally, it gives the possibility of tolerating some types of manfacturing defects.
Currently, the design of a custom FPGA core requires specialized tools and the expertise of an FPGA architect. This makes it impractical to apply this methodology for all embedded applications. Tools presented in the previous sections can be used by an embedded system engineer to design a fully specific FPGA core.

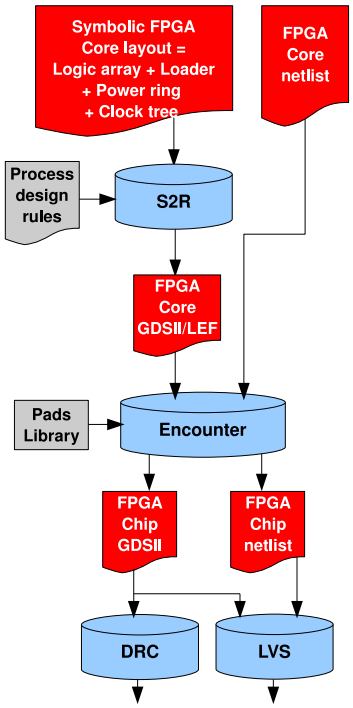


Figure 2.18: FPGA Chip Generation, verification and validation

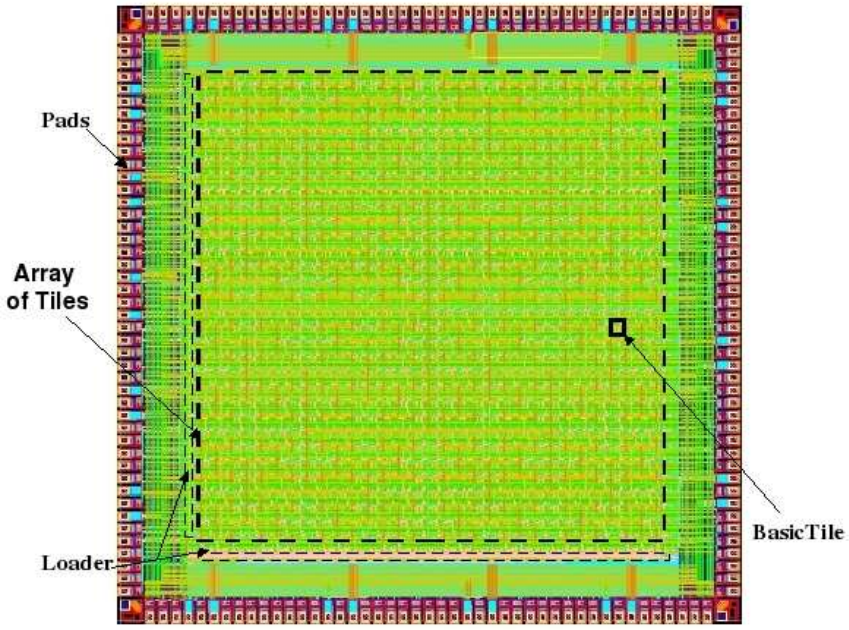


Figure 2.19: FPGA Chip 32x32 in 0.12um ST process

2.6 conclusion

In this work we aimed at presenting an automatic method to generate an FPGA using an open-source VLSI tool-kit and targeting a standard cell library. We can produce FPGAs with different architectural parameters. In the future, we intend to increase the number of adjustable architecture parameters. We also intend to add support for other networks topologies including coarse grain modules such as memory and DSP blocks.
The global method can be reported easily to be used with Commercial VLSI tools and any Founder Library : custom, semi-custom, standard cell libraries and macro cells.
Using this technique we can produce a large spectrum of different architectures and different array sizes. As we noted, our method is not automated completely. To achieve better layout quality some manual specific tasks are suggested, which are related to the target FPGA. The proposed method is validated by generating a set of island-style FPGA layouts which complies with VLSI design rules: DRC, Power, Clock network, Timing etc.

3 Redundant FPGA Core

Due to their general performance, high integration density SRAM based FPGA are very attractive for space, avionic and military applications. But this technology uses SRAM cells to control its logic and interconnect configuration. Since the storage elements are the most Single Event Upset (SEU) sensitive elements, SRAM-based FPGA are penalized for applications with high safety and robustness. This chapter introduces techniques and methods to guarantee safe and secure SRAM-based FPGA operation. Then using the layout generator developed in chapter 2, we create an SRAM based FPGA named REDFPGA, which includes hardware support for the mitigation of SEU. The design was successfully migrated and taped out in 0.12 um 6-metal layer CMOS process from ST.

3.1 Context

Recent industry interest in neutron-induced soft errors focused primarily on data corruption in memory devices. However, neutron-susceptible memory elements are used for configuration storage in SRAM-based FPGAs. There is a significant and growing risk of functional failure in such FPGAs due to the corruption of configuration data.This limits their widespread use in mission critical applications (aerospace, medical, military applications).

CMOS ICs operating in space and radiation environments are subject to 3 main transient radiation effects: single event latch up, performance degradation due to cumulated dose and single event upset (SEU).
SEU is a change of state caused by a high-energy particle strike to a sensitive node in a micro-electronic device, particularly on transistors in semiconductor memory. Figure 3.1

shows SEU phenomena induced by energetic particles hitting silicon device. SEU is not considered damaging permanently :the transistor's, or circuits' functionality. A particle hit with sufficient energy changes the logic state of the memory elements, thus producing a *soft error*. Due to technology scaling and reduced supply voltage, soft errors represent a serious problem in logic circuits because critical charge has now reached a lower level than the charges generated by energetic particle incidence. SEU-induced soft errors contribute significantly to the overall system FIT (Failure In Time) rate for ground-based and airborne equipment

Terrestrial SEU is due to cosmic particles colliding with atoms in the atmosphere, cre-

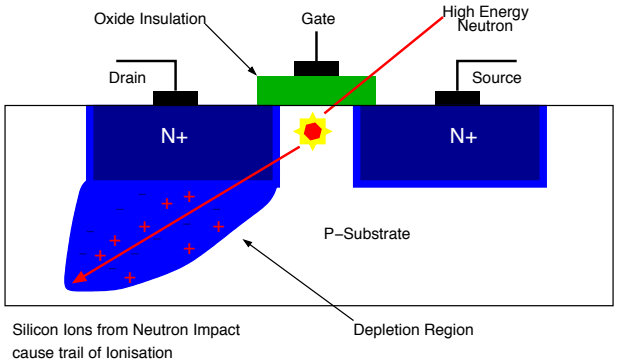


Figure 3.1: Neutrons Interaction with Integrated Circuits

ating cascades or showers of neutrons and protons, which in turn may interact with electronics. In deep sub-micrometer geometries, this affects semiconductor devices in the atmosphere. In space, high energy, ionizing particles exist as part of the natural background, referred to as galactic cosmic rays (GCR). Solar activity and trapping of charged particles in the earth's magnetosphere worsen the problem. Similar energies are possible on a terrestrial flight over the poles or at high altitude. Traces of radioactive elements in chip packages also lead to SEUs.

There are two memory resources in FPGAs, $a)$ user bits, and $b)$ configuration bits. An SEU on user bits cause a *transient* error. An SEU on configuration bits leads to a *permanent* and *hard* error which is the major error types in FPGAs because the number of SRAM configuration cells dominates user-defined memory elements (registers inside LB). Typically, the number of SRAM configuration cells are more than 98% of all memory elements inside an FPGA.

Conventional fault-tolerant schemes [B.W.Johnson, 1998] can only protect user-bits. The most common applicable fault-tolerant mechanism to protect configuration bits in commercial SRAM-based FPGA is to use Triple Modular Redundancy (TMR) scheme in all used logic and routing resources [C.Carmichael et al., 1999]. However, this approach causes area and power overhead in excess of 200% and more often close to 300%. While TMR schemes can mask single error, they will fail if errors accumulate in the circuit. To prevent accumulated errors, *scrubbing* can be used. Scrubbing includes reading back the configuration bits, comparing those with the original configuration bits [V.Maingot

et al., 2007], and writing the correct bits when there is an error. The combination of TMR and scrubbing gives a highly reliable framework at the cost of 100% area overhead and enforces high performances and power penalities. Detection and correction of configuration upsets in SRAM-based FPGA may need several thousands or millions of clock cycles before functional failure is detected.

In this work, we study methods and technologies that can be used to guarantee safe and secure SRAM-based FPGA operation based on:

- Memory cells Hardening
- Error detection and correction techniques

3.2 Robustness of the FPGAs Configuration Memory

If a soft error occurs in a memory element, it is difficult to recover the original data. SRAM hardening can be achieved through redundancy, resistor decoupling, shielding. Hardening with technologies such as: CMOS substrate epitaxy, CMOS on insulator substrate [S.Hareland et al., 2001] and resistive or capacitive hardening [W.Wang, 2004] induce a degradation of performance.
A modified storage cell called Dual Interlocked Cell (DICE) [T.Calin et al., 1996] avoids those drawbacks and errors, thus achieving upset immunity.

3.2.1 Basic SRAM Cell

Each configuration bit in an SRAM-based FPGA is stored on four transistors forming two cross-coupled inverters. Two additional access transistors are used to control the access to a storage cell during read and write operations. This is the typical SRAM cell with six MOSFETs to stores one memory bit, shown in figures 3.2 and 3.3

First, standard SRAM memory is examined in order to evaluate its immunity and its operating limits under SEU impact. To evaluate the maximum current that SRAM can admit, the error duration is fixed and its current amplitude limit is measured. In this work the Upset induced charge is simulated by a time-dependent current source with a triangular shape. The current wave form depends on two factors: fault duration and current amplitude.

By SPICE simulation, we have found that critical current (i.e., the minimal pulse amplitude that provokes a positive upset) for the standard SRAM cell presented in figure 3.3 is depending on environment caracteristics and may be:

- 180uA for a duration of 200ps or,
- 220uA for a duration of 100ps or,

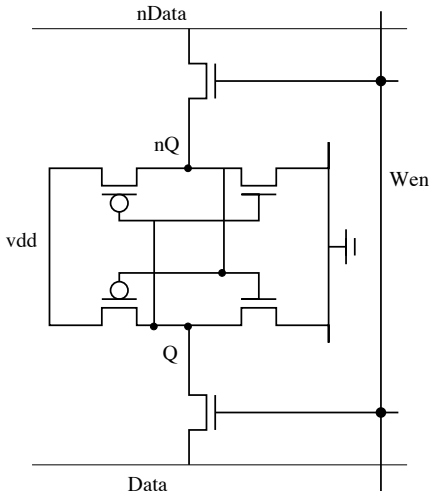


Figure 3.2: Typical SRAM cell

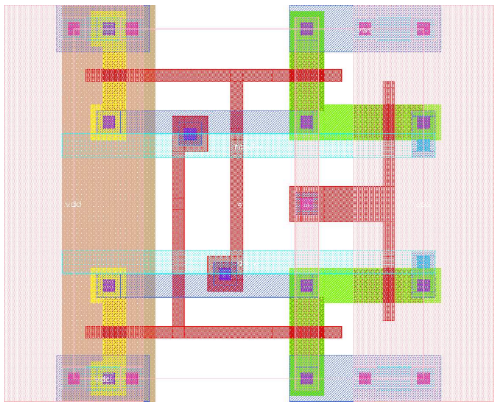


Figure 3.3: SRAM layout: 30λx30λ

- 270uA for a duration of 50ps.

Higher amplitude or period induce a permanent soft error. Figure 3.4 shows how the logic state of Q node changes from 0 to 1 if the induced current ("I(HIT)") of the SEU has an amplitude of 275uA.

3.2.2 The Dual Interlocked CEll (DICE) structure

A DICE memory cell is designed to provide upset immunity, avoiding and correcting SEU errors. The proposed cell does not impose particular constraints on transistor sizes or on technology process.
The DICE structure (figure 3.5) uses a 4-node redundant structure. It includes two conventional cross-coupled (horizontal) inverters latch structures N0-P1 and N2-P3, con-

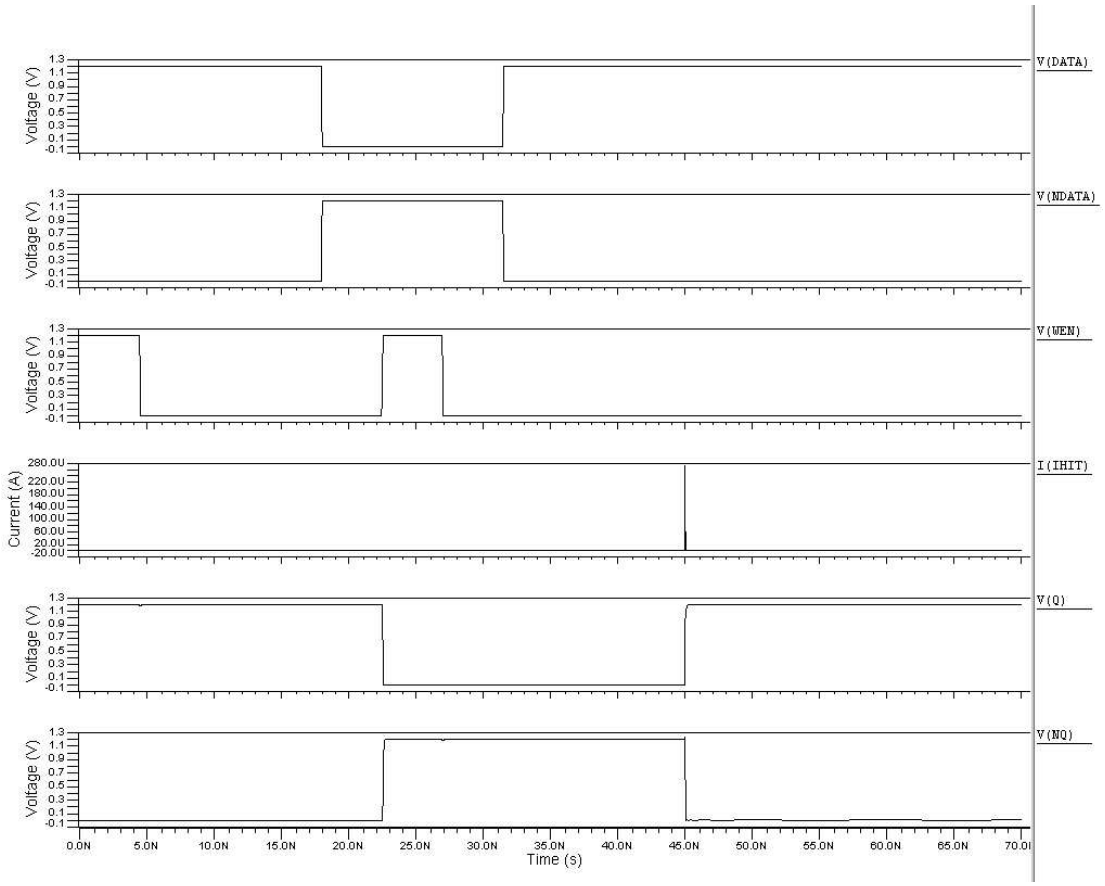


Figure 3.4: Error injection on node Q of the SRAM (275uA, 50ps)

nected by bidirectional feedback (vertical) inverters N1-P2 and N3-P0. The 4 nodes X0, X1, X2 and X3 store the data as pairs of complementary values (1010 or 0101) which are accessed simultaneously using transmission gates for write or read operations.
In figure 3.5 the principle of dual interlocked storage cell is presented. The inverter symbols are in fact either P-type or N-type transistors. They form two opposite feedback loops, a clockwise P-transistor loop P0..P3 and an anti-clockwise N-transistor loop N3..N0. Figure 3.6 presents the flattened view of the transistor level DICE structure.

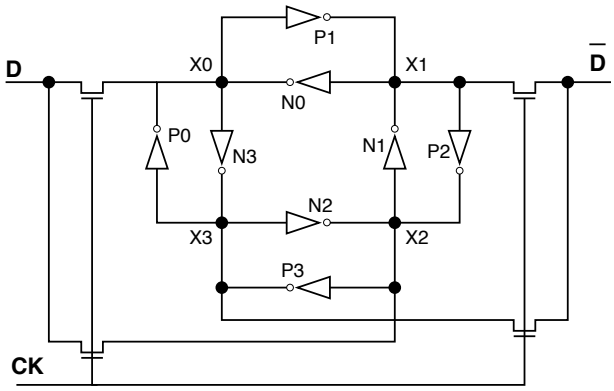


Figure 3.5: principle of the DICE

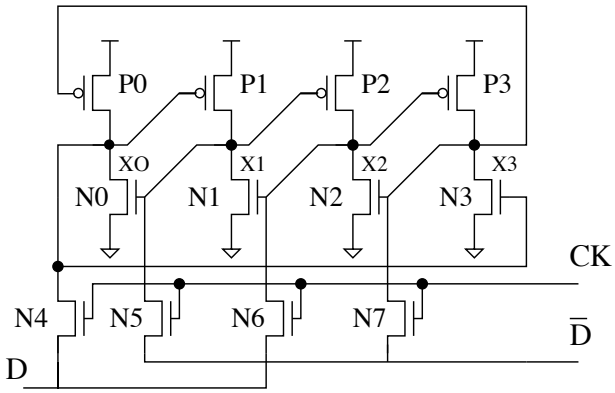


Figure 3.6: Transistor level of the DICE

The logic state of every node of the cell is controlled by two adjacent nodes located on the opposite diagonal. The two nodes on each diagonal don't depend directly on each other, since their state is controlled by the two nodes of the other diagonal. A node Xi controls the two complementary nodes on the opposite diagonal Xi-1 and Xi+1. This is done using a single transistor for each complementary feedback control connection through Ni-1 and Pi+1.
The logic state 1 is considered as "X0.X1.X2.X3"="1010": "N1-P2" and "N3-P0" in conduction form two latches that store the same data. The horizontal transistors pairs N0-P1

and N2-P3 are blocked. They perform a feedback interlock function, thus insulating each vertical latches from the other.
Analysis can be done for a positive transient upset at node Xi. A positive perturbation at node Xi affects node Xi-1 through transistor Ni-1. Nodes Xi+1, Xi+2 keep their state capacitively and restore the correct logic state on the two perturbed node through transistors Ni and Pi+1.

As shown in figure 3.7, the design of two adjacent CMOS SRAM cells can be directly converted into DICE by simply rewiring internal interconnects without changing transistor sizes. Figure 3.8 shows the DICE layout that is more than twice larger than standard SRAM.

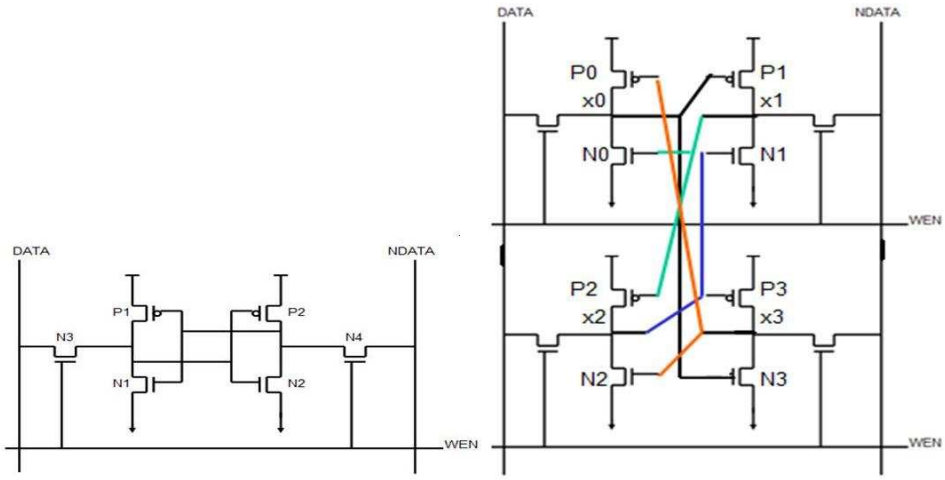


Figure 3.7: Two SRAM cells to design the DICE

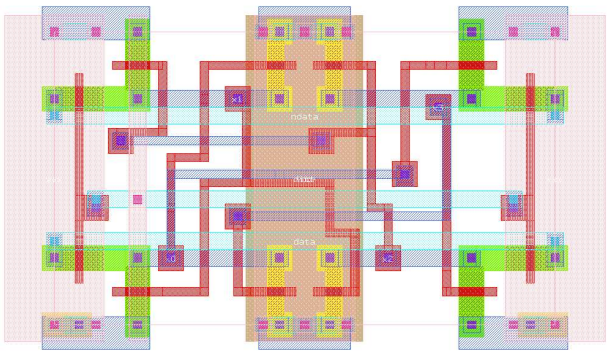


Figure 3.8: DICE layout: 60λx35λ

3.2.3 Testing the DICE: Error Injection

The SPICE model of the DICE cell is extracted from the real layout in a 120 nm technology process. The method of simulation used for the SRAM is applied to the DICE to test its robustness. The particle hit is always represented by a peak of current on one node. The same scenario is applied to all nodes, one node at a time. The DICE cannot recover 2 errors on two different nodes at the same time. Errors are injected independently on the 4 nodes X0, X1, X2 and X3 to study the behavior of the circuit.
In figure 3.9 an error is injected on node X1. It can be noted that on curve X1 the node

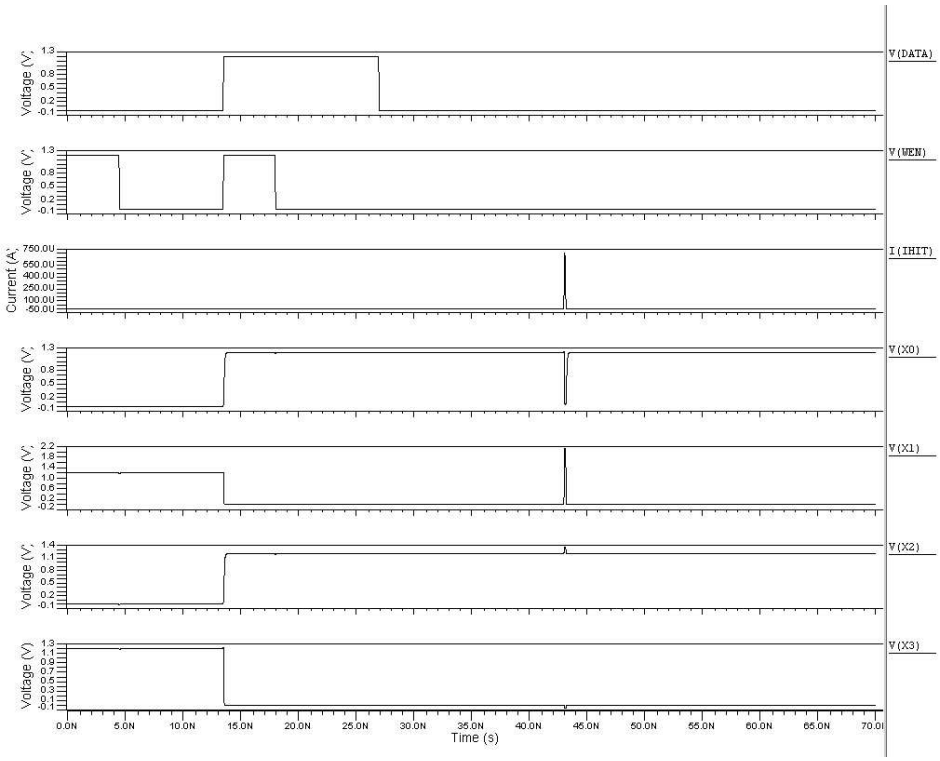


Figure 3.9: Error injection on node x1 of the DICE (800uA, 200ps)

has changed its logic state and after a few seconds it returns to its original state. Also an error on node X1 causes an error on node X0. The two other nodes retain their logic state.
An error must be injected on each node to investigate this circuit and to determine the maximum current it can admit. Recovery time is measured to describe the behavior of the DICE cell.

Figure 3.10 shows an error on node X1. IHIT is the injected error with a 500uA amplitude. The recovery time is defined as the duration of the SEU induced error on a given node. For node X0 the recovery time is larger than in X1, because it is a negative upset corrected via P-transistors. In our layout P-transistors are smaller than N-transistors,

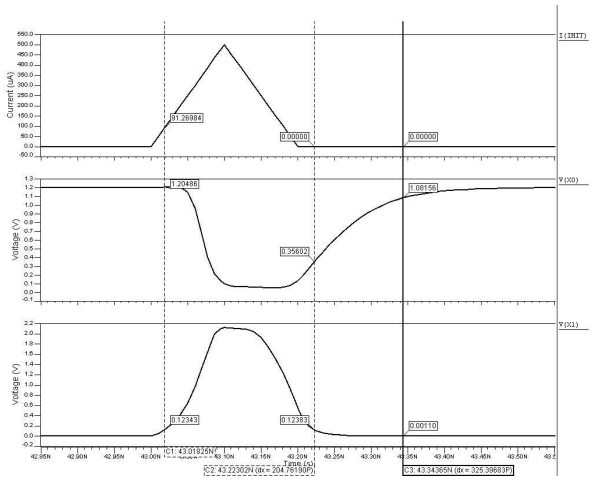


Figure 3.10: Error injection and recovery time

thus P-transistors are slower.

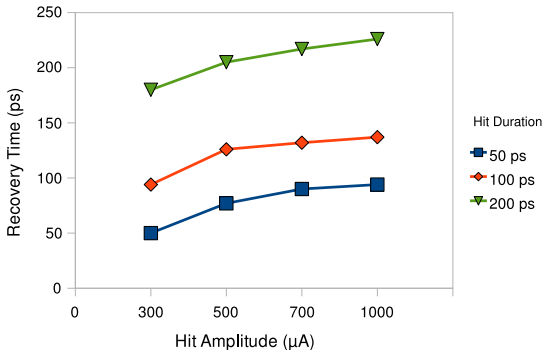


Figure 3.11: Recovery time in affected node

The curves in figure 3.11 show the relation between current amplitude and recovery time, for error durations of 50,100 and 200 picoseconds. We noticed that the recovery time grows with the current and durations.
It should be noted that the DICE circuit is not immune to Multiple Event Upset (MEU). If two simultaneously sensitive nodes of the cell, storing the same logic state, are flipped by a single or multiple particle impacts, then immunity is lost and the cell is upset. The probability of this occurrence can be made very low in ground based and commercial airborne equipment.
To check MEU effects, 2 errors are injected in two different nodes; the system logic state is changed. In figure 3.12, 2 nodes X1 and X3 are hit at the same time. the logic state of

all nodes are changed and they don't return to their original state. This cell is immune to SEU only.

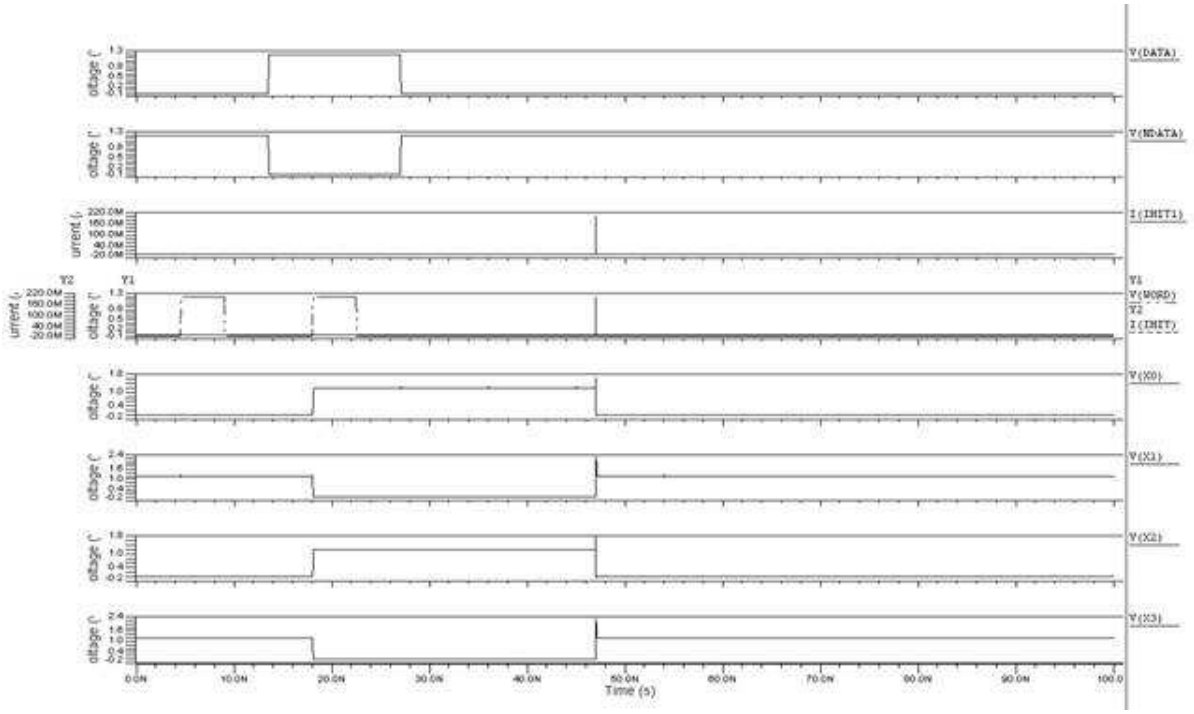


Figure 3.12: Error injection on 2 nodes simulaneously

A DICE detects and corrects errors with a fast recovery time; this is due to the fact that the restoring feedback function is embedded in the memory structure, without requiring any additional feedback system.
Figure 3.26 shows the difference in area between two memory blocks with a same capacity (10 bytes each one). The block in the right using standard SRAM cells (10x8 bits), the second using a DICE (10x8). DICE memory is $2.3\times$ larger than the standard one, and adds storing configuration delay.

3.3 Error Detection and Correction

There are many error detecting techniques, like parity bit control, then specific techniques are used in order to correct the detected errors. Moreover techniques of self checking can detect and also correct errors when occurred.

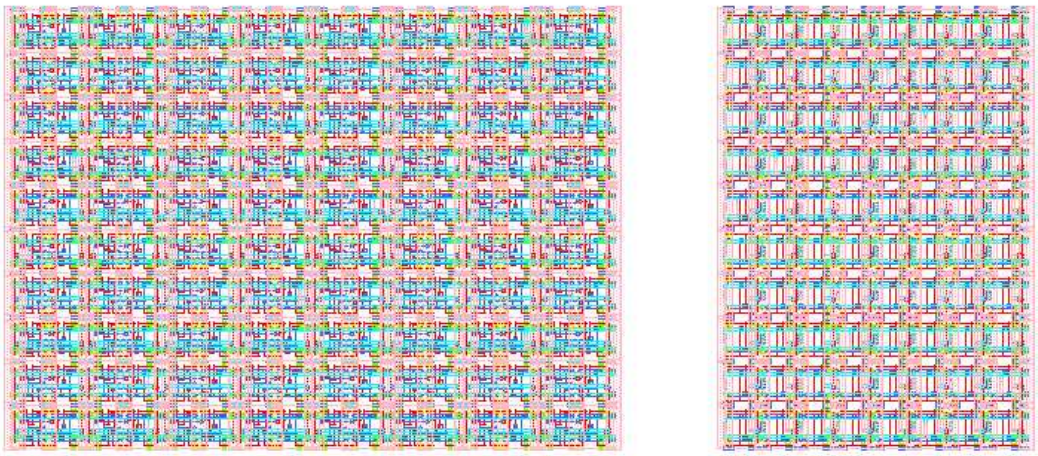


Figure 3.13: 10x8 RAM block using a DICE at the left and 10x8 SRAM block at the right

3.3.1 Parity Check Technique

Parity check (sometimes called VRC, for Vertical Redundancy Check or Vertical Redundancy Checking) is one of the simplest checking mechanisms. Computing parity involves counting the number of ones in a data unit, and adding either a "0" or "1" (called a parity bit) to make the count odd (for odd parity) or even (for even parity). For example, 1001 is a 4-bit data unit containing two 1-bits; since this is an even number, a "0" would be added to maintain even parity, or, if odd parity was maintained, another "1" would be added.

To check even parity, the XOR operator is used as shown in figure 3.14; to calculate odd parity, the XNOR operator is used. Single bit errors are detected when the parity count shows that the number of ones is incorrect, indicating that a data bit has been flipped by noise on the transmission line. Therefore, parity bit is an error detecting code, but is not an error correcting code as there is no way to determine which particular bit is corrupted. The data must be discarded, and retransmitted entirely. Parity does have the advantage, however, that it is about the best possible code that uses only a single bit of space and it requires a reduced number of XOR gates to detect one error.

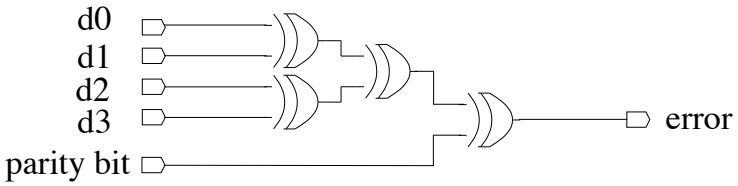


Figure 3.14: Hardware Implementation of 4 bits parity decoder

3.3.2 Hamming Code

This approach of "error detecting and correcting codes" is based on defining a distance between two bit strings by the number of bits that must be changed in the first string to obtain the second string. Extra bits are added to each string; they are set so that a minimum number of bits is changed to obtain the second string from the first one. If the received string isn't valid, it is assumed that the valid string is the corrected one, closest to the received one.
The Hamming code is a set of error-correction codes that can be used to detect and correct bit errors that may occur when computer data are moved or stored. Hamming code is named for R. W. Hamming of Bell Labs.

Like other error-correction code, Hamming code makes use of the concept of parity bits, which are bits added to data so that the validity of the data can be checked when they are read or after they are received from a data transmission link. Systems using more than one parity bit, an error-correction code can identify not only a single bit error in the data unit, but also its location in the data unit.
In data transmission, the ability of a receiving station to correct errors in the received data is called Forward Error Correction (FEC) and can increase throughput on a data link when a lot of noise is present. To implement this, a transmitting station must add extra data (called error correction bits) to the transmission. However, correction may not always be cost saving compared to simple information retransmitting. Hamming codes make FEC less expensive to implement through the use of a block parity mechanism.
Compared to simple parity code, Hamming codes detect two bit errors by using more than one parity bit, each of which is computed on different combinations of bits in the data. The number of parity bits required depends on the number of bits in the data, and is respecting the Hamming rule:

$$d + p <= 2^p - 1 \tag{3.1}$$

The parity bits are placed inside the Hamming word, each one in a position of a power of 2. Thus, if we consider the example of 4-bits data "d1 d2 d3 d4", the 3 parity bits "p1 p2 p3" are placed in the total Hamming code word of size $n = d + p$ (where d is the number of data bits and p is the number of parity bits) as follows: N= "p1 p2 d1 p3 d2 d3 d4".
Parity bits are generated by multiplying the data bits by a code generator matrix. Then reading the entire vector, Hamming's decoder can correct any single-bit error, or detect all single-bit and two-bit errors by computing the data bits and parity bits by the parity-check matrix. Figure 3.15 shows a hardware implementation of a Hamming(4,3) decoder.

Compared to the Parity system, the Hamming system presents the advantage of error correction and double error detection but il also presents a disadvantage in term of area overhead due to the additional parity bits and its decoder system. Figure 3.16

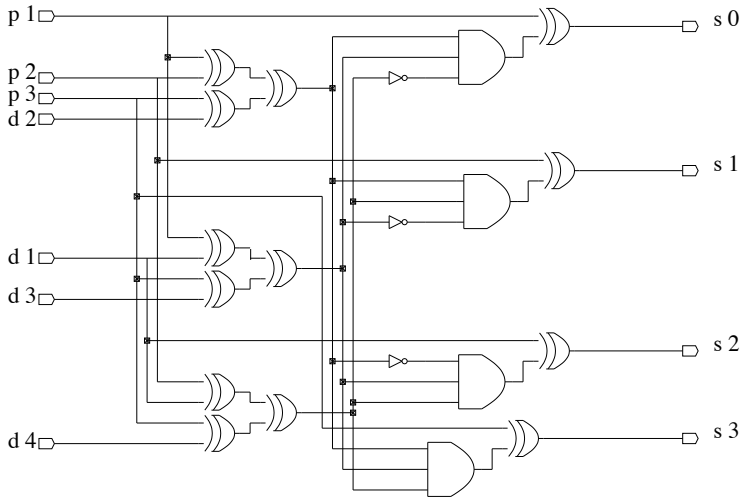


Figure 3.15: Hamming Decoder for 4 data bits + 3 parity bits

shows the area overhead for a RAM word, using the Parity system and using the Hamming system. The system area represents the original SRAMs area (data bits), the additional SRAMs area (parity bits) and the decoder area.
In this work, we look to secure the configuration memory in an SRAM-based FPGA. The original configuration corresponds to the bitstream generated by the FPGA CAD tools. In this case, the encoding phase is done on the bitstream generation with a software encoder module; for this reason we don't take into account an encoder area. We use the SXLIB library and new SRAM cell(figure 3.21) to implement and evaluate the real area of any module.

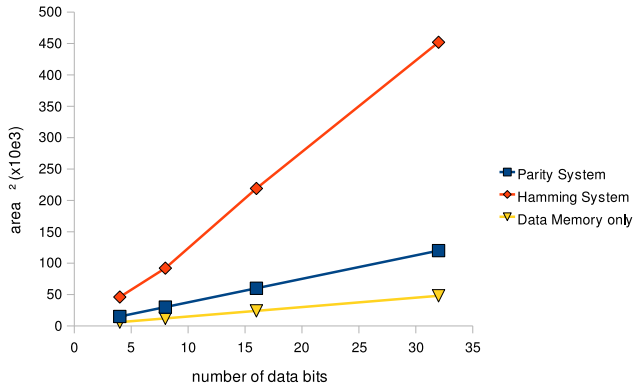


Figure 3.16: Parity and Hamming systems area overheads

Another important factor to take into account is the number of data bits in the word.

For example, to secure a RAM block of 1 KByte by implementing a check system for each word, a word size of 4, 8, 16 or 32 has an important impact on the memory area required for the parity check. Figure 3.17 shows this impact and shows that larger memory words is better to reduce the total memory area overhead.

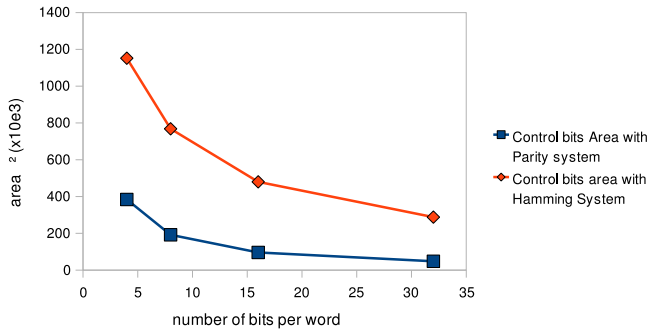


Figure 3.17: Parity system: Memory area overhead

3.4 Architecture Features

3.4.1 Motivations

This work is done in collaboration with the CEA (Commisariat à l'Energie Atomique) in order to develop a first SRAM-based FPGA prototype that integrates simple hardware technique for the configuration memory reliability. In this first step, reliability for internal registers used in the functional mode are not taken into account in this work.
The goal is to build a scalable architecture with a generic technique of error detection and correction that can be extended or adapted if we change the reliability level and the technologie process. We choose the technique that is most compatible with the FPGA generation approach presented in chapter 2 and which can expect benefit from the random access to the configuration memory, the scalability by abutment and the standard cell flow.

3.4.2 REDFPGA architecture overview

An SEU on configuration bits may change the functionality of the look-up tables as well as the interconnect controlled by the SRAM cells, thus can produce a hardware error if there is a driver conflict. The example in figure 3.18 shows how a bitflip changes the signals routing and causes different type of erros. Figure 3.18.(a) shows two different nets A and B, that cross a *subset* switch matrix using respectively programmable switche (W0,S0) and (N0,E0). Figure 3.18.(b) shows an SEU (1 to 0) on switch (W0,S0)

that causes a switch open resulting in a *permanent* error in the gate-level netlist. Figure 3.18.(c) shows an SEU (0 to 1) on switch (N0,S0) that causes a *hard* errors between nets A and B wich can damage the device. In this cases, the configuration bit remains erroneous until the new configuration is downloaded into the FPGA.

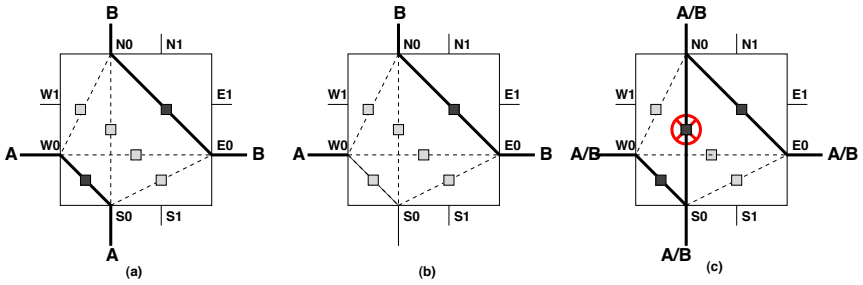


Figure 3.18: The impact of SEU on routing network

These hardware conflicts can be eliminated by using simple decoders to implement a system dependency between switches that drive the same track. As shown in figure 3.19, the ordinary control system of the interconnect (figure 3.19.(a)) can be replaced with an efficient system using simple decoder to control switches (figure 3.19.(b)). This method eliminates the possibility of driver conflicts and reduces the configuration bits number. In the proposed model, the total number of SRAM cells is reduced by 55% and the total area increases by 6% due to the decoders area.

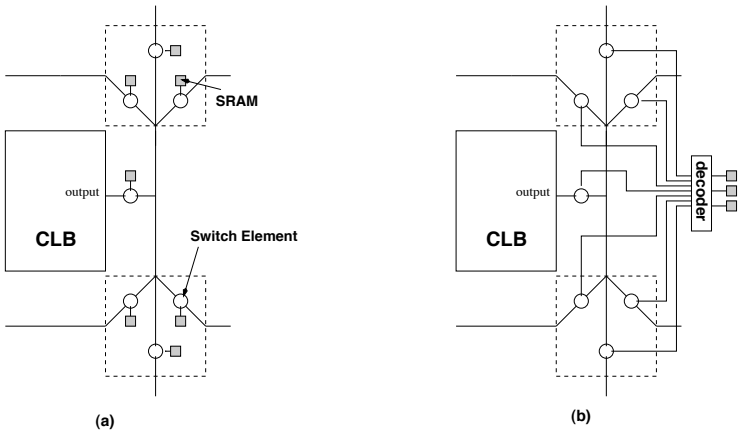


Figure 3.19: System decoder for the interconnect

The REDFPGA model comprises an array of configurable logic blocks (LBs) and figure 3.20 shows the structural basic tile. Each LB contains a 4-LUT followed by a by-pass flip-flop. Each LB has 4 inputs (one on each side) and an output that drives adjacent channels on its top and right sides. The LBs communicate with one another through a disjoint bi-directional routing network. All inputs and outputs of a LB connect with all

wires in a channel. The generated FPGA matrix can have **NX** LBs in the X direction, **NY** LBs in the Y direction, and a channel width **Ch**.
The functionality of the logic block is controlled by programming the multiplexers and the content of the look-up-table. The total number of configuration bits in the tile including all programmable ressources is 77 bits grouped as nine 8-bits words and one 5-bits word that can be programmed selectively. If the implementation of the array is made, we obtain an array of reconfigurable cells grouped as 8-bit words and this array can be programmed similarly to a RAM. Figure 3.21 presents the basic SRAM cell used to store FPGA configuration.

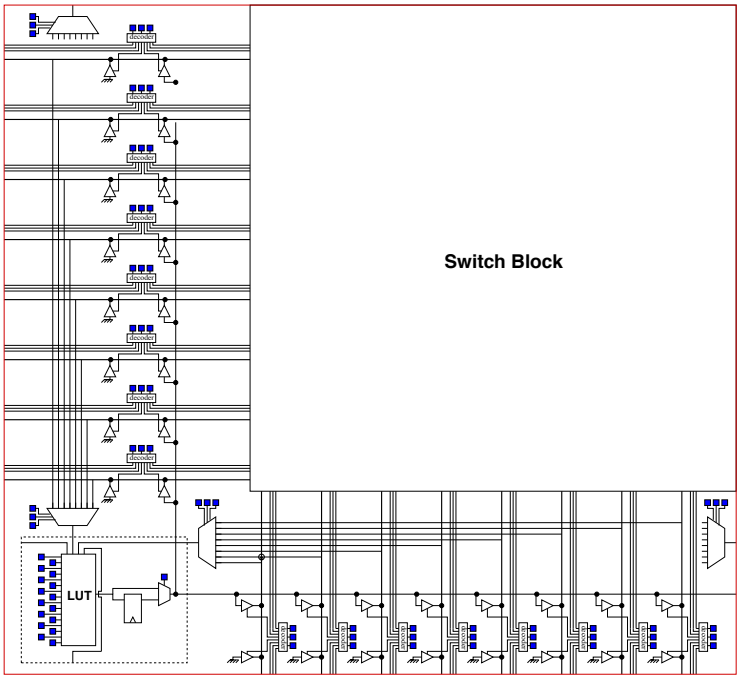


Figure 3.20: REDFPGA basic tile overview

3.4.3 SEU detection and correction in REDFPGA

The FPGA device can contain millions of configuration bits. A continuous readback and verification of the configuration data become an expensive function in terms of cycle number and dynamic power. To perform verification we implement a fine grain verification mechanism. Configurable ressources are organized as an array of tiles. Configuration memory is also organized as an array of data-frames. A single data frame contains configuration data of the tile that lie in that column and row. In each tile and for each data frame, we integrate an SEU detection system. This system enables an error signal if it detects any change in configuration bits. The error signal propagates through row

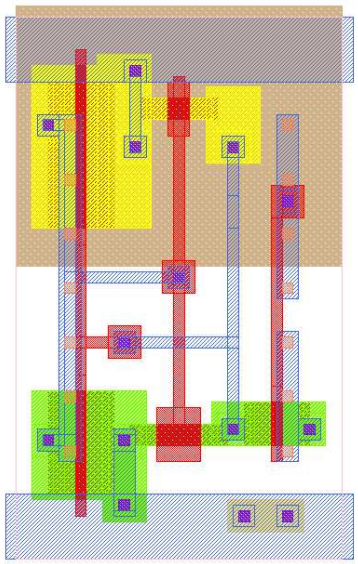


Figure 3.21: SRAM cell in symbolic layout with SXLIB template ($30\lambda x 50\lambda$)

and column. Figure 3.22 shows the content of the basic grain which is a tile with different views: functional logic view, Configuration memory frame view and error-detection module view.

Regularity of the FPGA structure simplifies the structuring of the entire FPGA. The

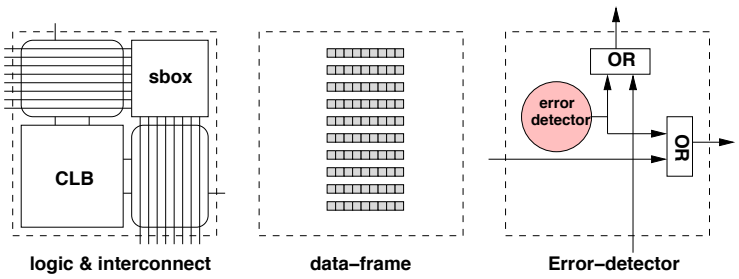


Figure 3.22: The Basic Tile of the FPGA architecture

process of FPGA generation described in chapter 2 is used to generate the REDFPGA. As shown in figure 3.23, the abutment of tiles gives a global error-detection mechanism that can detect any configuration bits error. It also returns the row and column of the defective frame. Moreover the proposed approach is still efficient if errors accumulate in different frames at the same time. Figure 3.24 shows how this technique allows to detect multiple simultaneous errors and indicates the failure frames that must be reconfigured. The highlighted zone correponds to the mapped design area, frames outside this area are known in advance by the configuration module (software module in this work) and false alarms that come from these frames can be ignored. Example in figure 3.24 shows 3 simultaneous errors detection in frames (0,3), (1,3) and (2,2). The global error-detection

and correction mechanism will reconfigure frames (0,3), (1,3), (2,3), (1,2) and (2,2) only.

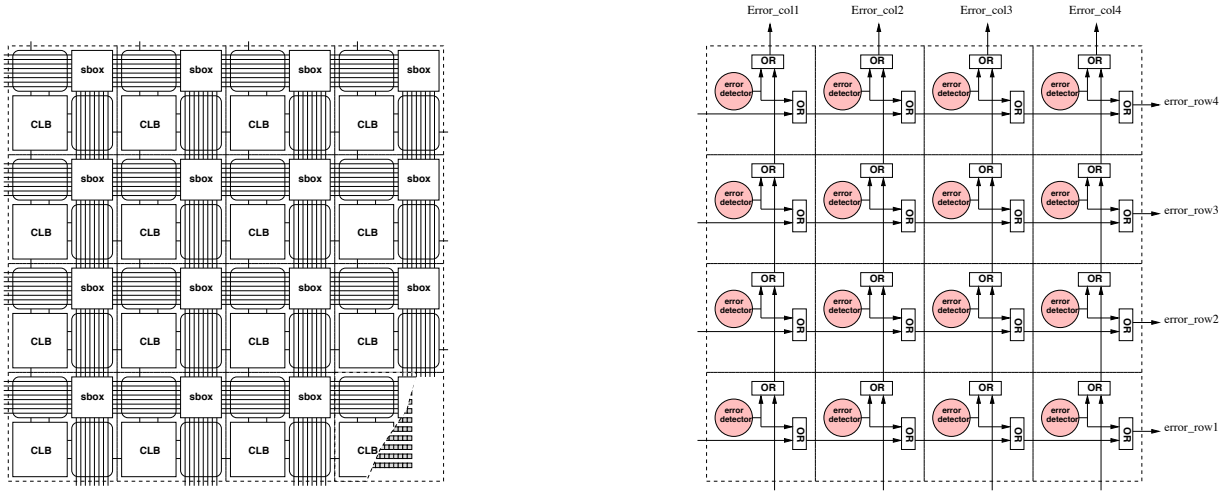


Figure 3.23: Scalable error-detection mechanism

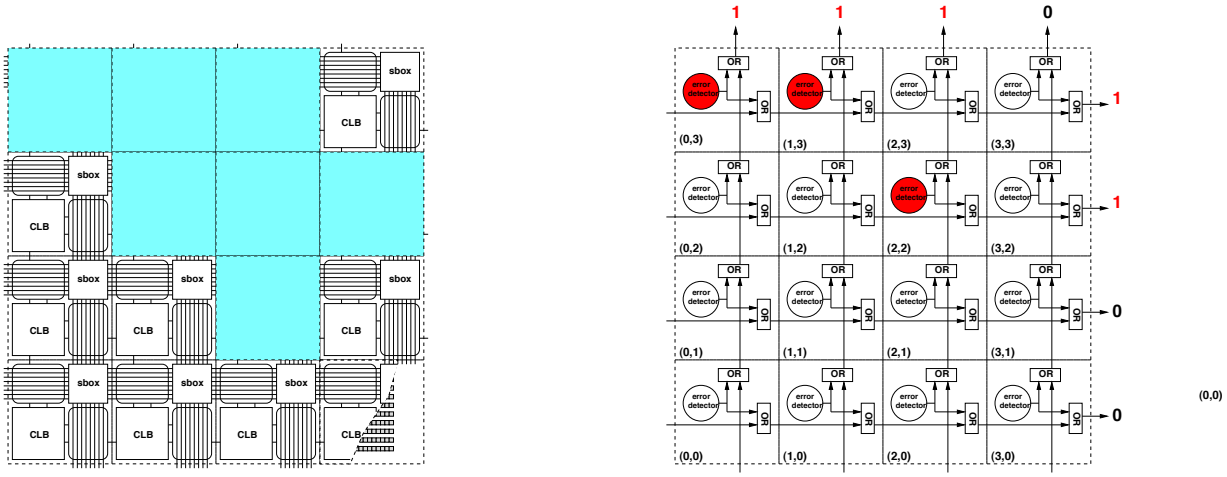


Figure 3.24: Multiple error detection

Thanks to random access technique to the configuration memory, the correction is made rapidly by re-writing only the failure frame. We note that the configuration bit-stream is stored as an array of frames. Each frame has unique address that corresponds to the appropriate couple (row,column).
The REDFPGA prototype integrates a detection-error system based on the *parity* technique. For each byte of the configuration memory we associate one extra SRAM cell and one parity-module that verify in real time the parity of the byte. Without readback, and without comparison to an original bit-stream, this system makes continuous verification. The correction is made on-demand once we detect an error. Only the erroneous

tile frame will be updated. The FPGA configuration system is used to load the intial correct configuration of the tile. This means that the remaining part of the FPGA is still operating if it is functionally independent of the faulty tile, else we wait for a few cycles to reconfigure the faulty tile.
The proposed error-detection mechanism is scalable with any FPGA array size. The frame granularity can be varied to obtain the best balance between area, power and performances. Any accumulated errors are easily detected since they are in differents byte address.
In this work we choose the parity system because it is good enough to validate the complete error-detection and correction mechanism with a minimal additional silicon cost compared to other systems. Other efficient mitigation systems such as methods based on *hamming code* or *CRC checker* can be integrated easily in the basic tile and replace the parity system. Such methods enhance the reliability and allow detection of accumulated errors in the same byte.

3.5 Tape Out

The layout generation is done using a symbolic standard cell library which works on unit λ (lambda). The ALLIANCE tool S2R [Alliance, 2006] (symbolic to real) is used to convert the symbolic design into 120nm technology; the corresponding GDS and LEF files are also obtained. It is noticed FPGA area increases by 19% due to hardware support for SEU mitigation (based on parity technique).
The physical routed layout of the FPGA and the chip micrograph are shown in figures 3.25 and 3.26 respectively. The chip has a logic capacity of 64 4-LUTs arranged in a 8 by 8 array and frozen periphery with 18 input pads and 16 output pads. A 1.2V supply is used for the input/output pads and the FPGA core. The chip area is $2.21mm^2$ in a 0.12μ process routed using 5 layers of metal. The contribution of the array is $1.01mm^2$. The rest of the area is for the power ring, loader and pads. Table 3.1 summarizes the details of the chip.

The generic symbolic design rules help migration to any technology, but with some area penalty. Instead of symbolic library, if the netlist of the generated FPGA is laid out in ENCOUNTER using directly the real 120nm technology ST-library from, a 40% area reduction is noted.
The generated FPGA layout can be used as a black box embedded in any other larger system. For the proof-of-concept, we manufactered an independent standalone FPGA chip. Only the pads are placed and routed using Virtuoso platform. The DRC and LVS verification is performed using CALIBRE [MentorGraphics, 2006].

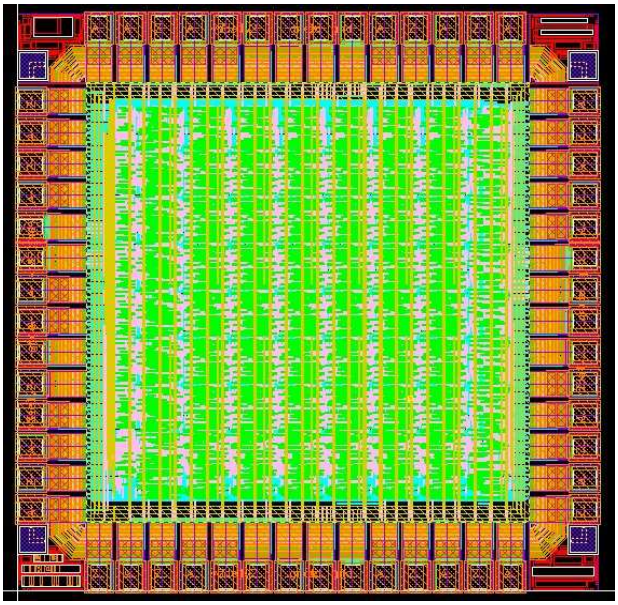


Figure 3.25: The redundant FPGA layout

3.5.1 Simulation:

The generated FPGA netlist is tested on the ALLIANCE simulator called ASIMUT. Several test applications are mapped onto the FPGA with the configuration sofware flow we describe in the following section 3.6. Once the FPGA is programmed, the respective testbench of each test application is applied on its inputs and the outputs are compared. These simulations can also be performed easily on other commercial tools like Synopsys.

3.5.2 Netlist layout comparison:

The netlist and the layout generated must match each other. For this purpose the ALLIANCE extraction tool COUGAR is used; it extracts a netlist from a layout, and the ALLIANCE comparison tool LVX is used to compare the extracted netlist with the generated one. This confirms that the generated layout matches its netlist. This method of layout verification is validated for a set of generated FPGAs, but the flattened 32x32 FPGA matrix is too large to be compared due to the limitations of COUGAR. Therefore instead of LVX, CALIBRE LVS is used to compare the 32x32 FPGA layout with its netlist.

Array size	8x8
Power Supply	1.2V
Chip area	$2.21mm^2$
Core area	$1.26mm^2$
Array area	$1.01mm^2$
Process	0.12μ CMOS

Table 3.1: Chip Specifications

3.5.3 Electric simulation:

The ALLIANCE extraction tool COUGAR can be used to extract the SPICE model of each tile. These models are simulated electrically later using ELDO [MentorGraphics, 2006]. Since COUGAR is unable to support large circuits, it was impossible to simulate electrically a complete 8x8 or 32x32 FPGA. In addition COUGAR does not condider parasitic and antenna constraints, that's why a more efficient extraction using an industrial tool is recommended. However for the proof of concept we simulated successfully the electric model of a smaller 4x4 FPGA matrix with channel width of 8. For the 8x8 prototype, SPICE models were extracted using industrial tools and simulated successfully with ELDO.

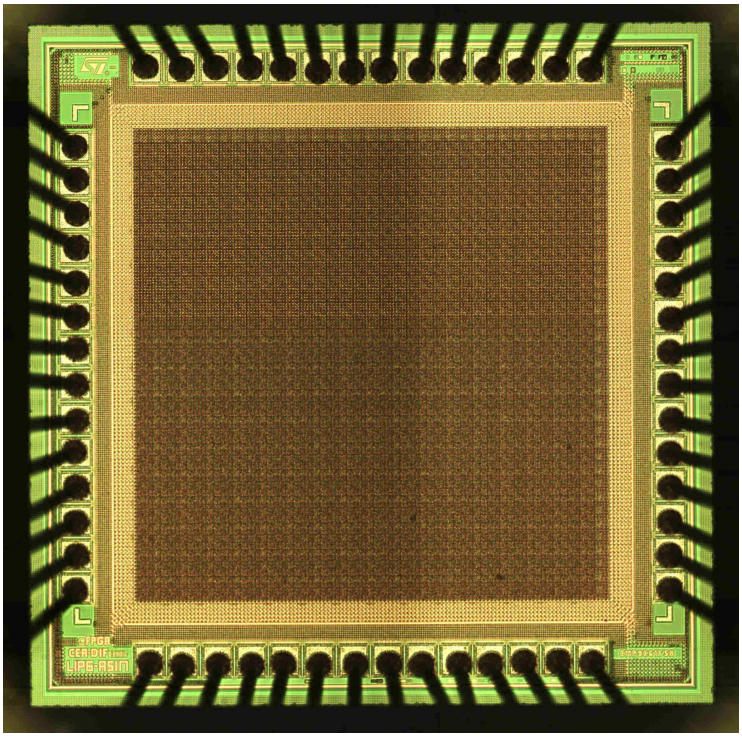


Figure 3.26: The redundant FPGA chip micrograph

3.6 Configration flow

Exploration of the implemented FPGA requires the software flow (see figure 3.28) involving logic synthesis, placement, routing and extraction of the implementation on the target architecture named bitstream. This flow uses only freeware tools like BOOG [Alliance, 2006] for logic synthesis, SIS [E.M.Sentovich et al., 1992] for mapping, T-vpack for clustering, and VPR [V.Betz and J.Rose, 1997] for place and route. We developed a generic extractor of bitstream that analyzes all results of the previous tools to generate the bitstream to load in the configuration memory.

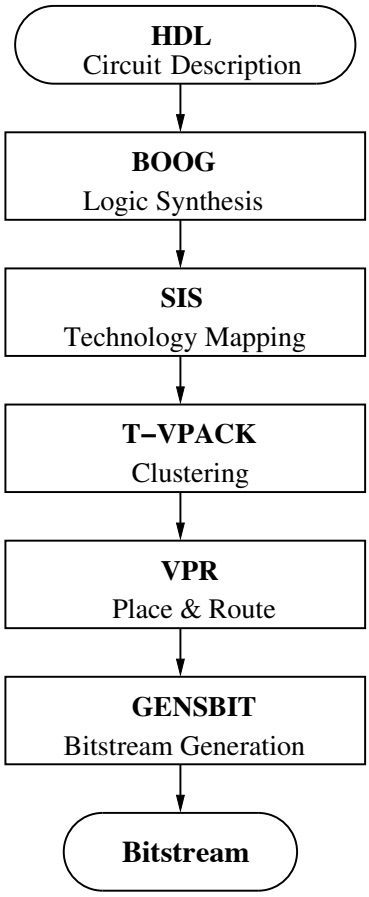


Figure 3.27: Configuration Flow

The software flow is followed to configure the generated FPGA. The sample application (in VHDL format) to be mapped onto the FPGA is the input to the software flow. Initially BOOG synthesizes the VHDL input into a boolean logic gates netlist and flip-flops. Then SIS transforms the boolean network into K-LUT. T-VPACK groups LUTs and flip-flops inside LB then VPR place and route the LBs and I/O blocks using the architecture resources as shown in figure 3.28. Finally the GENSBIT tool we developed

generates a binary stream containing all required information to configure the sample application onto the FPGA.

3.7 Conclusion

Redundant SRAM-based FPGA (REDFPGA) which includes hardware SEU mitigation support was developed by implementing a scalable hardware error-detector for the configuration memory. Compared to ordinary solutions of configuration bits reliability such as TMR, the proposed approach with an embedded parity system achieves higher level of reliability with only 19% increase in FPGA area.
We use a generic technique based on open-source ALLIANCE VLSI CAD tools to develop the physical REDFPGA chip. The REDFPGA layout uses a symbolic standard cell library which allows easy migration to any layout technology. This layout is successfully migrated and taped out in 120nm technology CMOS process from STmicroelectronics.
To configure the REDFPGA chip we use only open-source flow adapted to this architecture, including synthesis, mapping, place&route and bitstream generation tools.
This experimental work gave rise subsequently to new architectures wich are more sophisticated: REDFPGA-2 with 1024 LB taped out in 120nm and the work continued in CEA taking into account those developements.

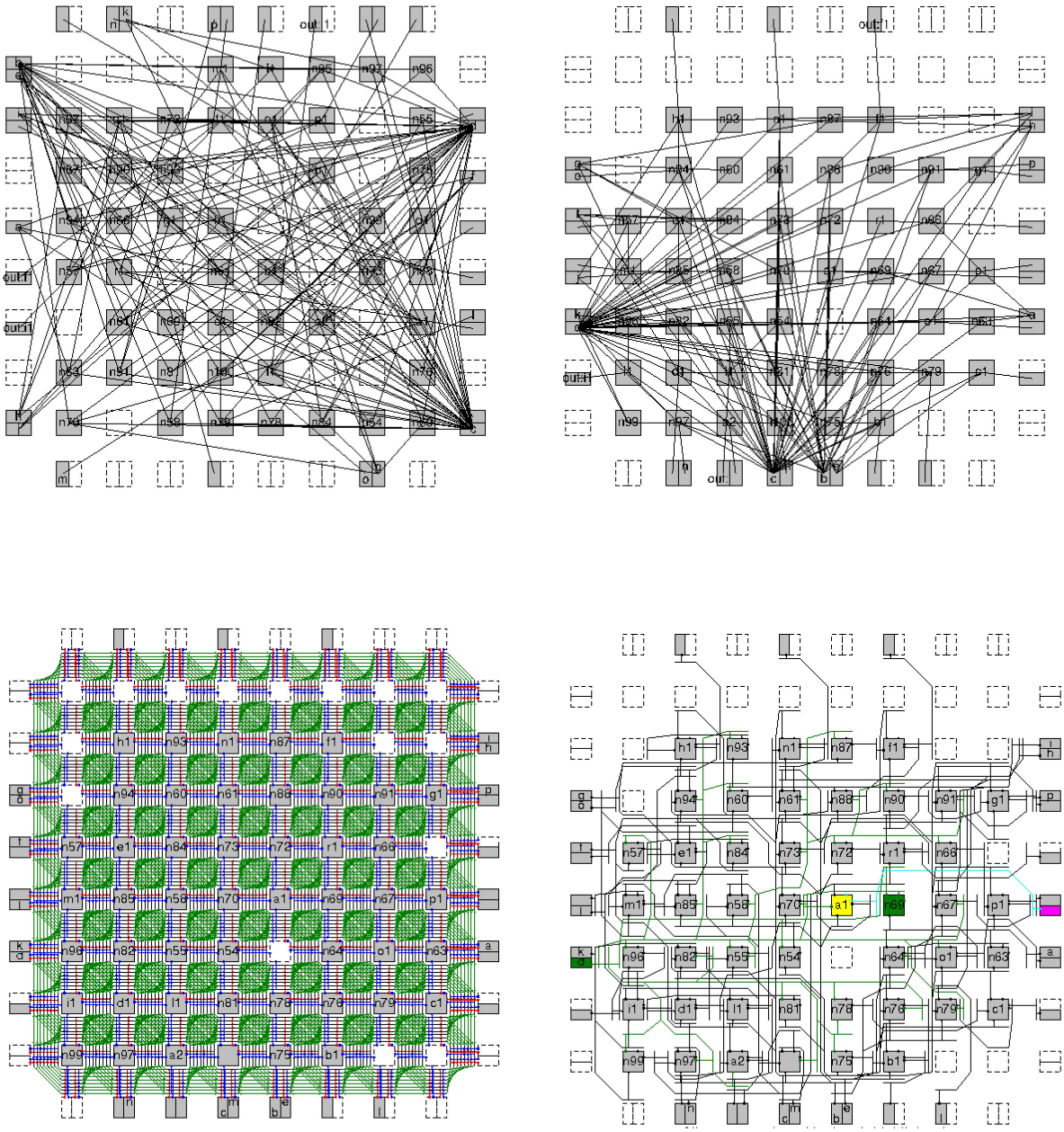


Figure 3.28: Screenshots of automatic VPR place and route

4
MFPGA Architecture

This chapter focuses on optimisation of the dominant part of an FPGA which consists in the programmable interconnect. The general approach used here is to examine the network distribution systematically and to balance the logic and interconnect utilization. We build on the state-of-the-art of interconnection structures by providing new interconnect structure for FPGA. As an alternative, we propose a new multilevel hierarchical FPGA (MFPGA) architecture where logic blocks and routing resources are balanced and sparsely partitioned into a multilevel clustered structure. We prove its validity through an empirical approach. Next we apply analytical and experimental methods to deal with the routing architecture. We give some unexpected results in term of area and density gain obtained with this topology.

4.1 Issues in Reconfigurable Network Design

Driven by Moore's law of semiconductor scaling, larger and larger FPGAs emerge. FPGA area is approximatively 90% programmble interconnect for only 10% logic [G.Lemieux and D.Lewis, 2004]. According to [L.Shang et al., 2002], the power dissipation share of routing, logic and clocking resources are 60%, 16%, and 14%, respectively. Current architectures will not extend directly to the multi-million gate scale because routing requirements grow linearly; they affect negatively area, speed and power consumption of FPGA. In addition, placement and routing computational times are ever increasing nowadays. Excessive FPGA placement and routing runtimes are now often measured in hours.

Design of new devices imposes radical efficient change in architecture to improve

speed, density, power consumption and software mapping time. Relying on industry experience with standard ASICs, we believe that partitioning and hierarchical structuring becomes unavoidable for hardware and software developments.

The ability of an FPGA to support designs with high LUT use is regularly presented as a positive feature. However, high routability across a variety of designs generates significant interconnect costs. Since interconnect is the dominant area component in FPGA designs, simply adding interconnect to achieve high LUT utilization is not always area efficient. Additional interconnect allows to use LUTs more heavily, but resulting often in less efficient interconnect use. Low utilization rate of FPGA interconnect ressources is inherent to such an approach. Li et al show in [F.Li et al., 2004] that this low utilization is equal to 12% of the total routing ressources.

In [A.DeHon, 1999] DeHon asks:

- Is an FPGA with higher LUT use more area efficient than one with lower LUT utilization?
- That is: Is LUT usability directly correlated with area efficiency?

DeHon presents initial evidence from a hierarchical array design showing that high LUT utilization is not directly correlated with efficient silicon usage. Rather, since interconnect resources consume most of the area on these devices, we can achieve more area efficient designs by allowing some LUTs to go unused, allowing us to use the dominant resource, interconnect, more efficiently. In fact DeHon showed that 100% logic use is not necessarily benificial for overall device area minimization.

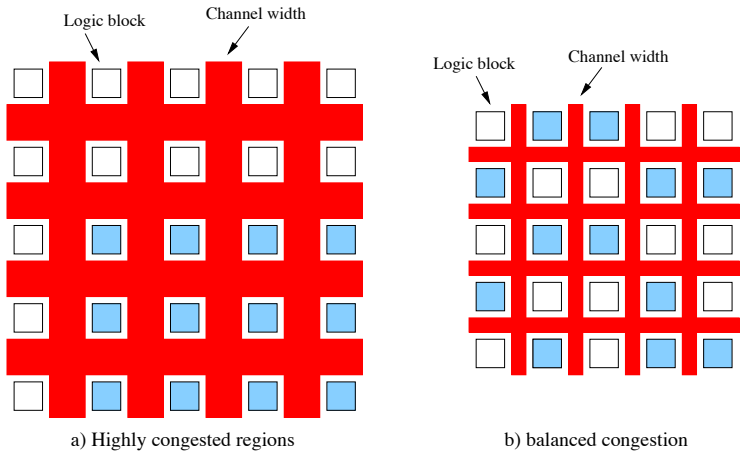


Figure 4.1: Congestion-aware placement

The philosophy behind logic and interconnect balancing is utilization increase through efficient use of the interconnect structure. For a fixed interconnect scheme we must assess the quality of this scheme. To make maximum use of the fixed interconnect in regions of high interconnect requirements where the design is more connected than the

FPGA, we may use sparsely the physical LUTs in the device, resulting in a depopulated LUT placement. In [A.Sharma et al., 2005], authors propose an efficient placement technique, called *independence*, for routing-poor architectures. Routing-poor architectures attempt to increase interconnect utilization at the expense of logic utilization. As presented in figure 4.1, they integrate an approach with the aim of spreading congestion over all the area. In this way the required routing resources (channel width) are reduced considerably. Nevertheless, to create white spaces, instances are moved away and this might increase delays to connect logic blocks and consequently reduce speed performances.

In [A.Singh and M.Marek-Sadowska, 2002], authors present a routability-driven clustering technique (iRac) for area and power reduction. The idea is to get a good device utilization by reducing clusters external signals at the cost of using more clusters. As illustrated in figure 4.2, when clusters are sparsely populated, highly congested regions are eliminated and the required channel width is reduced, inducing a reduction in the total area.

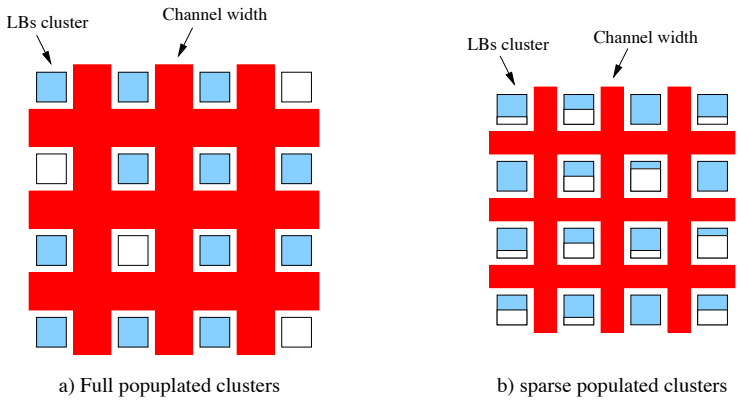


Figure 4.2: Congestion-aware clustering

We proceed as follows:

1. Hierarchical structuring becomes an obligation: we define a hierarchical interconnect model which allows us to vary the density of the interconnect.

2. Balancing logic and interconnect: we introduce a set of tools for "depopulating" the LBs in a hierarchical network to match the limited wiring resources.

3. We map circuits to a range of points in the interconnect space, and assess their total area and utilization. Compare it to a Mesh reference.

4. We examine relationship between LUT utilization and area.

4.2 Previous Works on hierarchical architectures

There is little published research on Hierarchical FPGA (HFPGA) architecture [W.Tsu et al., 1999] [Y.Lay and P.Wang, 1997] [A.A.Agarwal and D.Lewis, 1994]. This is due partly to the fact that industrial leaders use the manhattan mesh architecture. In a hierarchical FPGA, logic blocks and routing resources are organized into levels. At each level there are blocks and routing resources belonging to that level. Every $level_i$ cluster has W_i IO (Input/Output) wire tracks and contains a set of k $level_{i-1}$ clusters connected with $level_i$ switch box. Figure 4.3 shows the typical k-HFPGA as a tree containing N logic blocks, $\log_k(N)$ levels and k is the arity of the tree.

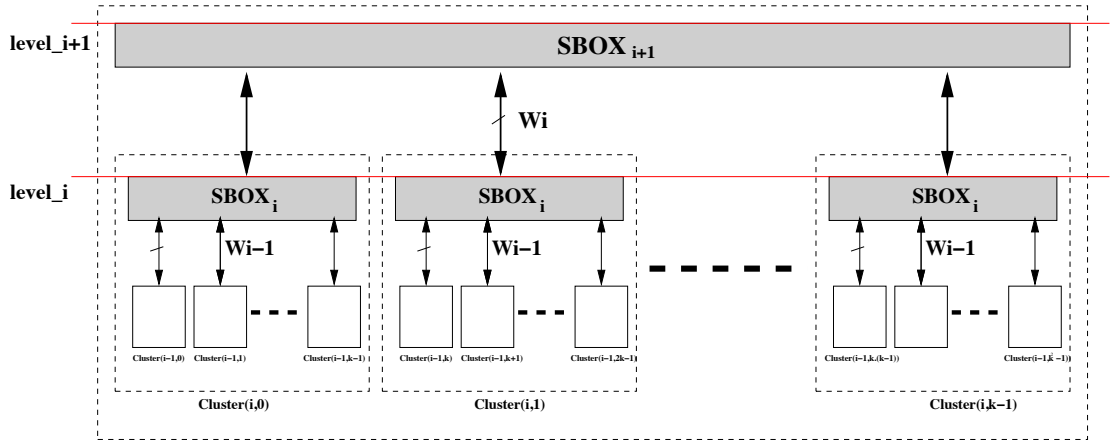


Figure 4.3: k-HFPGA architecture

Tsu et al's HSRA [W.Tsu et al., 1999], Agarwal and Lewis's HFPGA [A.A.Agarwal and D.Lewis, 1994] and Lai and Wang's interconnect [Y.Lay and P.Wang, 1997] are the most known interconnect schemes proposed for hierarchical FPGA interconnect.

4.2.1 Rent's Rule

A common way to compare hierachical architectures with the most studied Mesh-FPGA is the wiring requirement using Rent's Rule which is described by Landman and Russo in 1971 [B.Landman and R.Russo, 1971] to characterize interconnect demand within a circuit. It is an empirical observation which predicts that the amount IO of wiring needed as a circuit increases in size is related to the total number of gates N by:

$$IO = cN^p$$

It states that the number of external IN/OUT (IO) wiring pins of a partition is proportional to a power of the cells number included in this partition. In the FPGA case, N

corresponds to the number of LBs linked together. c is the average number of terminals per LB, and p defines the growth rate of Rent's rule. The value of p measures the complexity of the interconnection topology,

$$(simple)0 <= p <= 1(complex)$$

If most connections are exclusively local and only few of them come from the exterior of a local region, p is small. On the other hand, a region with large p implies that it has relatively more connections with outside cells. It was shown in [J.Pistorius and M.Hutton, 2003], for real logic circuits, that p is typically between 0.5 and 0.6. Figure 4.4 shows the implication of local Rent exponent.

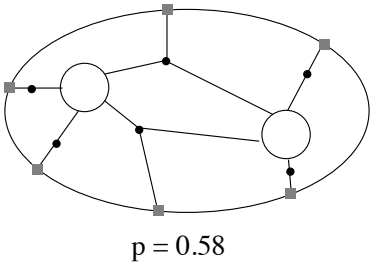


Figure 4.4: Average number of terminals and blocks within circuit model

4.2.2 Analytical comparison: k-HFPGA and Mesh

Rent's rule is adapted to different interconnect topologies. Using this rule and the bisection method presented in [Y.Lay and P.Wang, 1997], Dehon [A.DeHon, 2004] relates the channel width parameter W of a Mesh arranged in a $\sqrt{N} \times \sqrt{N}$ array to Rent's parameters. This method consists in splitting this array vertically, producing 2 entities which contain $\frac{N}{2}$ Logic Blocks as shown in Fig. 4.5. These entities are connected together

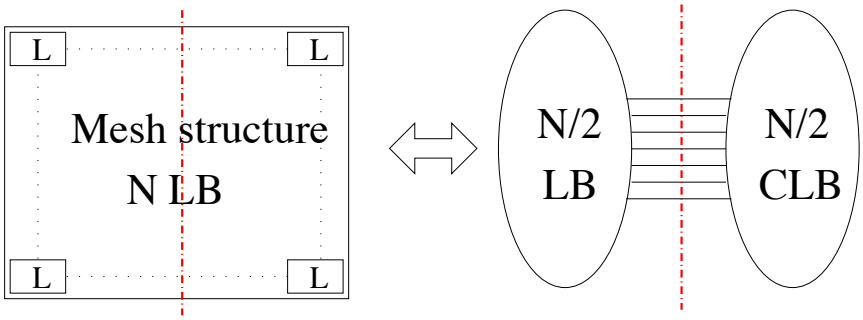


Figure 4.5: Bisection of a $\sqrt{N}x\sqrt{N}$ Mesh FPGA

by $IO = \left(\sqrt{(N)} + 1\right) W$ wires (it corresponds to the total number of wires crossing

horizontally the middle of the FPGA). Therefore he provides a lower bound for W to support a design characterized by Rent's parameters (c, p).

$$IO \geq c\left(\frac{N}{2}\right)^p$$

Thus :

$$\left(\sqrt{N}+1\right)W \geq c\left(\frac{N}{2}\right)^p$$

For large N, we can dismiss the term to obtain :

$$W \geq \left(\frac{c}{2^p}\right)N^{p-0.5} \tag{4.1}$$

The total number of switches per logic block in a Mesh corresponds to the number of switches in a basic tile, including switches in the connection boxes and switches in the switch box. If we consider that each LB has c inputs/outputs, then the number of switches in the tile is

$$N_{switch} = (cF_c + F_s)W$$

where F_c is the connection box flexibility and F_s is the switch box flexibility. Thus

$$N_{switch}/LB = O(W) = O(N^{p-0.5}) \tag{4.2}$$

The number of wires per logic block is

$$N_{wire}/LB = 2W + c$$

Thus:

$$N_{wire}/LB = O(W) = O(N^{p-0.5}) \tag{4.3}$$

Rent's rule can be easily associated to Tree-based topology. In fact a cluster located at level i of the Tree can be considered as a partition, with W_i external signals and k^i LBs (leaves). Hence:

$$W_i = ck^{ip} \tag{4.4}$$

As the k-HFPGA uses full crossbar switch boxes (SBOX), the switching requirement is evaluated as follows. The $SBOX_i$ in level i connects W_i wires from the level above to $k \cdot W_{i-1}$ wires from the levels below. According to (4.4), the $SBOX_i$ contains $kW_{i-1}W_i = c^2k^{(i-1)p+ip}$ switches. Each level i has $\frac{N}{k^i}$ $SBOX_i$. Thus the level i has $Nc^2k^{2ip-i-p+1} = Nc^2k^pk^{(2p-1)(i-1)}$ switches. The total amount of switches in a k-HFPGA with N LBs is the sum of switches in all tree levels and corresponds to:

$$N_{switch} = c^2k^p\sum_{i=1}^{\log_k(N)} Nk^{(2p-1)(i-1)}$$

Dividing by N gives the total of switches per LB:

$$N_{switch}/LB = \begin{cases} c^2 k^p \frac{1-N^{2p-1}}{1-k^{2p-1}} & \text{if } p \neq 0.5 \\ c^2 k^p \log_k(N) & \text{if } p = 0.5 \end{cases}$$

Thus:

$$N_{switch}/LB = \begin{cases} O(N^{2p-1}) & \text{if } p > 0.5 \\ O(log_k(N)) & \text{if } p = 0.5 \\ O(1) & \text{if } p < 0.5 \end{cases} \quad (4.5)$$

For the wire requirement, a $level_i$ adds $kW_{i-1}\frac{N}{k^i}$ wires. Thus the total number of wires per LB is :

$$N_{wire}/LB = c \sum_{i=1}^{\log_k(N)} k^{(p-1)(i-1)}$$

$$N_{wire}/LB = \begin{cases} c \log_k(N) & \text{if } p = 1 \\ c\frac{1-N^{p-1}}{1-k^{p-1}} & \text{if } p \neq 1 \end{cases}$$

Thus:

$$N_{wire}/LB = \begin{cases} O(log_k(N)) & \text{if } p = 1 \\ O(1) & \text{if } p < 1 \end{cases} \quad (4.6)$$

Equations (4.5) and (4.6) compared to equations (4.2) and (4.3) respectively show that k-HFPGA has more efficient switching and wiring area growing as $O(1)$ if $p < 0.5$. When $p > 0.5$ the mesh architecture becomes more interesting. Mesh switching and wiring area grows as $O(N^{p-0.5})$ while the k-HFPGA switching resources diverge and grows as $O(N^{2p-1})$. Since typical designs have $0.5 \leq p \leq 0.75$, mesh architecture is still more efficient than k-HFPGA.

We note that the above k-HFPGA is penalized by the use of a full crossbar switch box that guarantees arbitrary full connectivity, thus overestimating the required number of switches. In spite of its low logic density, the k-HFPGA retains the advantage of lower routing delays; there is a strong reduction of the average number of crossed switches to connect two logic blocks. The worst case is proportional to the number of levels in the tree which is $O\left(\log_k(N)\right)$.

Both Agarwal [A.A.Agarwal and D.Lewis, 1994] and Lai [Y.Lay and P.Wang, 1997] described hierarchical FPGA interconnect architectures with a sparse depopulated switch box. Both topologies tend to depopulate the switch patterns while maintaining 100% routability or at least the same routability as the depopulated linear mesh architecture. Tsu [W.Tsu et al., 1999] describes the HSRA architecture based on the butterfly architecture that needs fewer switches than in the other cases. In general, such binary tree is a limiting interconnect structure that leads to severe routing inefficiencies.

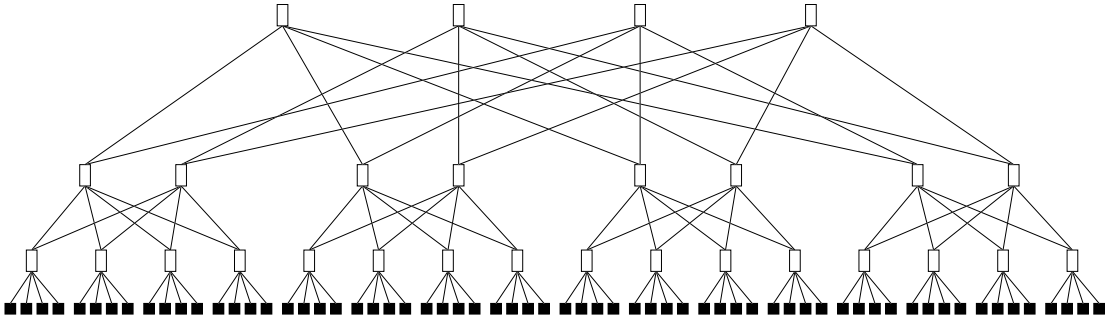


Figure 4.6: Butterfly Fat Tree Topology

4.3 Proposed Architecture

We propose a new Multilevel hierarchical FPGA (MFPGA) architecture, based on butterfly Fat Tree (BFT) style interconnect [C.Leiserson, 1985] shown in figure 4.6. The main motivation for the MFPGA aims at achieving the best area efficiency by balancing logic block and interconnect utilization. Since routing resources consume most of the area (often 80-90%) [G.Lemieux and D.Lewis, 2004], we focus on interconnect check. We try to underuse the first resource in order to use the other fully.

MFPGA is a hierarchical tree whose leaves are logic blocks. It has linear populated switch boxes and unidirectionl wires. This architecture unifies two unidirectional connected networks:

- The downward network based on the Butterfly Fat Tree (BFT) topology, involving a logarithmic population growth of unidirectional switch blocks, which connects these switch blocks to LBs inputs.
- The upward network connects logic blocks outputs to the different levels of the downward network using a BFT-like distribution with feedback connections.

Logic Blocks (LBs) are grouped into k sized clusters and interconnect is organized into levels. Let $nb\ell$ denote the number of levels of a given Tree containing N leaves ($nb\ell = log_k(N)$). In each level ℓ we have $\frac{N}{k^\ell}$ clusters; C is the set of clusters in all levels. A cluster with index c belonging to level ℓ is noted by $cluster(\ell, c)$. Clusters in the same level of the hierarchy are equivalents, and every $cluster(\ell, c)$ whith $\ell \geq 1$ contains a set of inputs $N_{in}(\ell)$, a set of outputs $N_{out}(\ell)$, a switch block and k sub-clusters. A $cluster(\ell, c)$ contains k^l logic blocks with c_{in} inputs and c_{out} outputs, hence:

$$N_{in}(l) = c_{in}k^{lp} and N_{out}(l) = c_{out}k^{lp} \tag{4.7}$$

Sub-clusters of $cluster(\ell, c)$ are $cluster(\ell - 1, k.c + i)$ where $i \in \{0, 1, 2, .., k - 1\}$. k is called $cluster(\ell, c)$ arity. A cluster switch block is divided into separated Mini Switch

Boxes (MSBs). Each MSB corresponds to a full crossbar. The MSBs number in a cluster in level ℓ is $nbMSB(\ell)$. MSB with index m belonging to $cluster(\ell, c)$ is denoted $MSB(\ell, c, m)$.

Each cluster in level 0 is denoted $cluster(0, c)$ or $leafcluster(c)$ and corresponds to the Logic Block (LB) and contains c_{in} inputs, c_{out} outputs, no MSBs and no sub-cluster. Each $cluster(\ell, c)$ where $\ell < nb\ell - 1$ has an owner in level ℓ', where $\ell' > \ell$, denoted $cluster(\ell', c \div k^{(\ell'-\ell)})$. We define for each $cluster(\ell, c)$ a position inside its owner in level $\ell + 1$ (direct owner) by the following function:

$$\begin{array}{rccl} pos: & C & \longrightarrow & \{0, 1, 2, .., k-1\} \\ & cluster(\ell, c) & \longmapsto & c \mod k \end{array}$$

2 clusters belonging to level ℓ and with the same owner at level $\ell + 1$ have 2 different positions. To get the cluster owner in level ℓ' of $cluster(\ell, c)$ $(\ell < \ell' \leq nb\ell - 1)$ we define the function:

$$\begin{array}{rccl} owner: & C \times \mathbb{N} & \longrightarrow & C \\ & (cluster(\ell, c), \ell') & \longmapsto & cluster(\ell', c \div k^{\ell'-\ell}) \end{array}$$

4.3.1 Downward Network

Figure 4.8 shows a sparse downward network based on unidirectional Downward MSBs (DMSBs). Each $DMSB(\ell, c, m)$ contains $I(\ell, c, m)$ inputs and k_ℓ outputs, where k_ℓ is the arity of the $cluster(\ell, c)$. The downward interconnect topology is similar to the butterfly fat tree. Every DMSB in a $cluster(\ell, c)$ where $\ell > 1$ is connected to every subcluster through one and only one input pin. Thus, the DMSBs number in a cluster situated in level ℓ is equal to the input number of a cluster located in level $\ell - 1$: $nbDMSB(\ell) = N_{in}(l - 1)$. In a symetric architecture where all DMSBs have the same number of inputs, every $DMSB(\ell, c, m)$ has $N_{in}(l)\frac{n}{b}DMSB(l)$ inputs and k outputs.
We name $DMSB(\ell', c', m')$ as the successor of a $DMSB(\ell, c, m)$ where $0 < \ell' < \ell$ if there is a downward directed path from $DMSB(\ell, c, m)$ to $DMSB(\ell', c', m')$. The path between a DMSB and its successor is unique. We define the function:

$$\begin{array}{rccl} Mod_\ell: \mathbb{N} & \longrightarrow & \mathbb{N} & \\ m & \longmapsto & m & \mod nbMSB(\ell) \end{array}$$

Thus each $DMSB(\ell, c, m)$ has a successor in each sub-cluster belonging to level ℓ' $DMSB(\ell', c', m')$ where $0 < \ell' < \ell$, with:

$$m' = Mod_{\ell'} \circ \cdots \circ Mod_{\ell-1}(m) \tag{4.8}$$

In this work, we choose a logic block with 4 inputs and 1 output which contains one 4-inputs Look-Up-Table followed by a bypass Flip-Flop. 4-LUTs are shown to be the most efficient K-LUT for SRAM based FPGAs by J.Rose et al in [J.Rose et al., 1990]. We use Rent's parameters to specify the bandwidth growth for every network level.

According to Rent's Rule, if we are limited to 1 logic block, we have $N_{in} = c_{in}(1)^p$ inputs and $N_{out} = c_{out}(1)^p$ outputs. Thus we obtain both values of c:

- $c_{in} = 4$ for the downward network.
- $c_{out} = 1$ for the upward network.

Figure 4.7 shows a $cluster(1, c)$ with the following parameters:$c_{in} = 4$, $c_{out} = 1$, $p = 1$ and $k = 4$ (the arity of the hierarchy). This cluster is composed of k Logic Blocks (LBs) c_{in} Downward MSBs (DMSB) and c_{out} Upward MSB (UMSB). This cluster has 4 LBs so

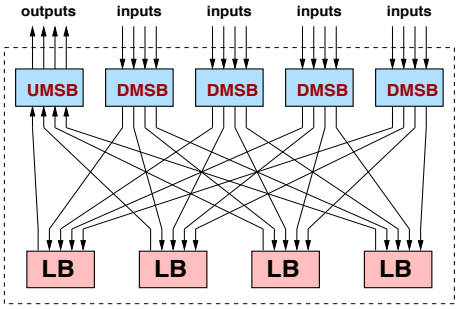


Figure 4.7: Typical cluster of level 1

it must have $N_{in}(1) = c_{in} * (k)^p = 4 * 4^1 = 16$ inputs distributed on the four DMSBs.
Every DMSB is in charge of the interconnection between the upper level and one input of every logic block. Thus the DMSB has 4 outputs and, since we deal with unidirectional wires, they are composed of 4 multiplexers, one for each output.
A configuration of a n level MFPGA can be described using the expression N_0 x N_1 x N_2 x....x N_{n-1}. Let's take the example of a 4x4 MFPGA architecture with 16 logic blocks. Figure 4.8 shows a 2 levels Downward Network (or BFT tree) with the following parameters: $p = 1$ and $k = 4$ (the arity of the BFT). To simplify this figure, we show only the downward network. UMSB and upward network will be detailed in the next section.

In the same way as in the figure 4.7, we create the $level_2$ cluster. We connect 4 $level_1$ clusters to one Switch Block. Since the $level_1$ cluster has 16 inputs, the $level_2$ switch block is divided into 16 DMSBs. Each DMSB has 4 outputs, each output will be connected to one $level_1$ cluster input. The $cluster(2, c)$ contains 16 Logic Blocks, so it has $N_{in}(2 =)c_{in}k^{2p} = 64$ inputs shared out among the 16 DMSBs.

4.3.2 The Upward Network

The Downward Network allows one path from each wire-source in the top level to each leaf (Logic Block) in the lowest level. But what can be the source ? Of course it is a logic block output or an input pad.
Let us begin with the logic block output. How to connect the output of a logic block to

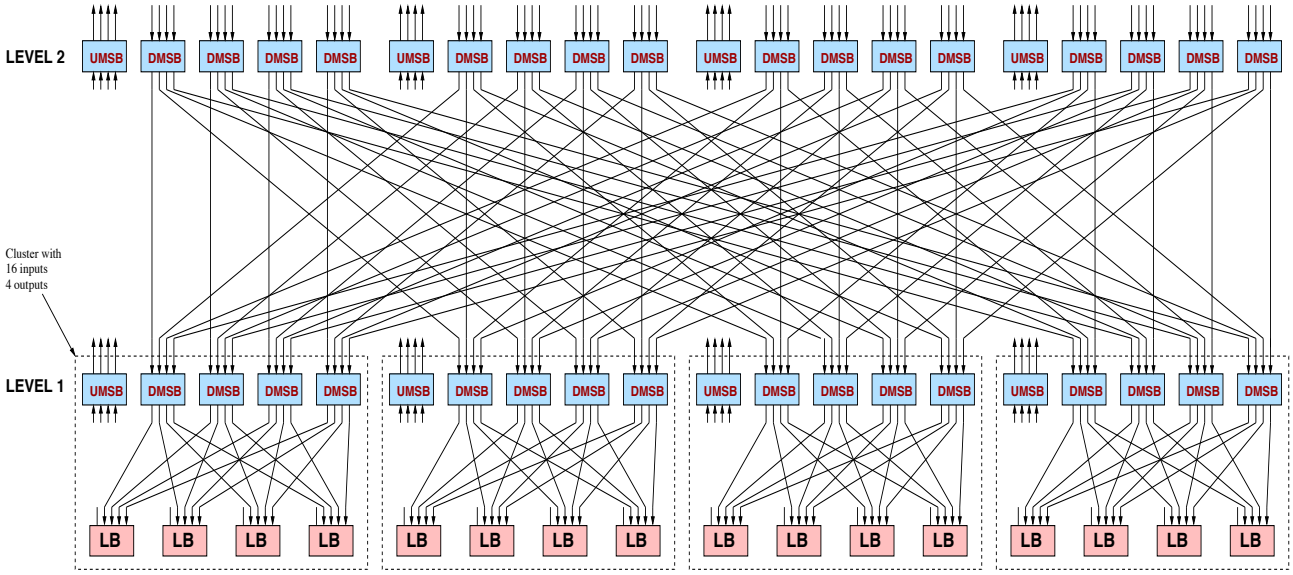


Figure 4.8: MFPGA Interconnect: 2 level Downward Network with $k = 4$ and $p = 1$

the input of another one ? We just have to link the logic block output signal to a specific upper level (the lowest authorized level is the lowest level common to the two logic blocks), then the signal can flow down to the targeted logic block through the BFT. We name these signals $feedbacks$.
The way we distribute the feedbacks among levels has an important impact on the structure routability. Connecting a feedback to n DMSBs with different indexes increases the paths from one source to one destination. We do it as simply as possible and we define the solution presented in figure 4.9. This feedbacks distribution gives several possibilities to reach a destination.

Upward Mini Switch Boxes (UMSB) allow cluster outputs to reach all DMSBs of the owner cluster. The UMSBs are interconnected in a way that allows LBs outputs to reach all DMSBs of the owner cluster. Therefore, following points can be considered as settled:

- clusters or logic blocks positions inside the direct owner cluster are equivalent and re-arranging them is not necessary.

- The interconnect offers many routing paths to connect a net source to a given sink. In fact, one LB instance can negotiate with other instances the use of a large number of DMSBs. This is efficient for mapping netlists since instances may have different fanout sizes. For example in figure 4.9, an LB ouput can reach all 4 DMSBs of its owner cluster at level 1 and all 16 DMSBs of its owner cluster at level 2.

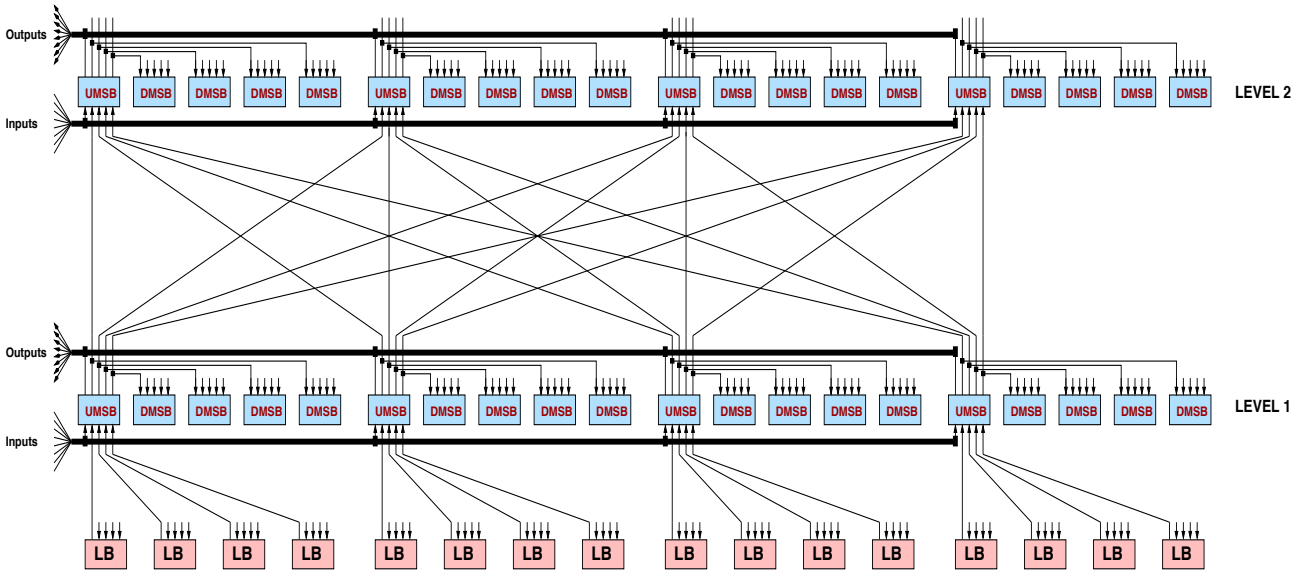


Figure 4.9: MFPGA Interconnect: 2 level Upward network and IO pads connections

4.3.3 Connections with the Outside

As shown in figure 4.9, output and input pads are grouped into specific clusters. The cluster size and the level where it is located can be modified to obtain the best design fit. Every input pad is connected to UMSBs of the different levels. In this way every input pad can reach all LBs inside the reached cluster level through different paths. Input pads which are connected to the upper level can reach all the LBs of the architecture with different paths.
Similarly, output pads are connected to UMSBs in different levels; Output pads connected to UMSBs of the upper level can be driven by any LBs of the architecture.
As one can notice, input/output pads have higher interconnection flexibility than LBs. One pad can be connected to different UMSBs in the same or in different level of the hierarchy to increase its flexibility. In the same way, input/output pads can be connected to the DMSBs but this approach gives more constraints in input/output pads placement unless they are connected to the highest level as shown in figure 4.10.

4.3.4 Interconnect Depopulation

Rent's rule [B.Landman and R.Russo, 1971] is easily adapted to Tree-based structure. Intuitively, p represents the locality in interconnect requirements. If most connections are purely local and only few of them come in from the exterior of a local region, p will be small. In MFPGA architecture, the upward and downward interconnects populations depend on this parameter. We can depopulate the routing interconnect by reducing p architecture parameter in every level. As shown in figure 4.10, $p_{level1} = 0.79$ corresponds to reduce from 16 to 12 the number of inputs in each cluster of level 1 and outputs from

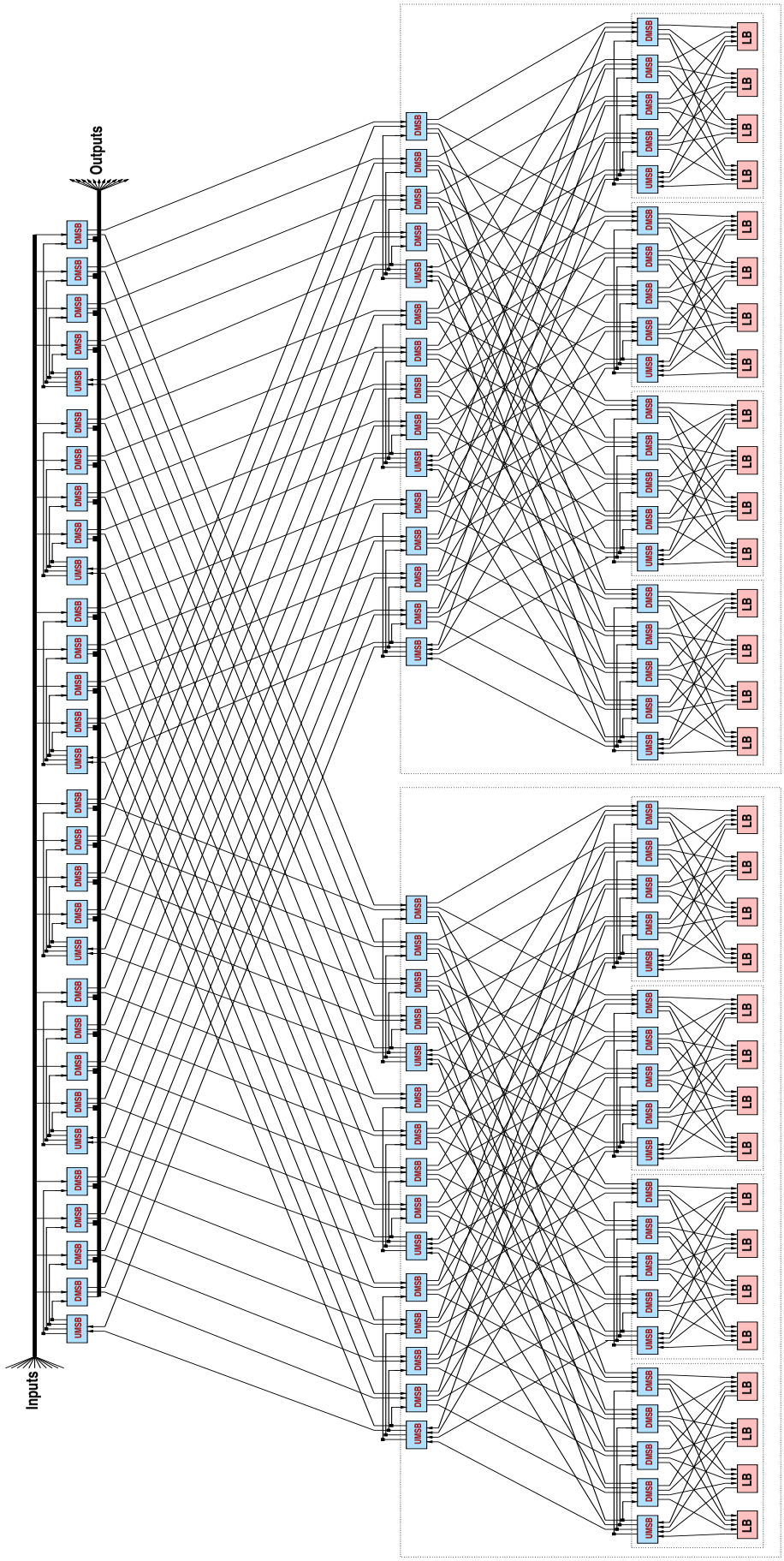


Figure 4.10: 4x4x2 MFPGA interconnect depopulation: $p_{level1} = 0.79$, $p_{level2} = 0.64$

4 to 3. This induces a reduction from 16 to 12 of the number of DMSBs in each cluster of level 2 and the UMSBs number from 4 to 3. In this case, if we consider an architecture with 2 levels of hierarchy, we get a reduction of the interconnect switches number from 521 to 416 (19%). By doing so the architecture routability is reduced too. Thus we must find the best tradeoff between interconnect population and logic blocks occupancy. In MFPGA architecture, the logic occupancy factor is controlled by N, the leaves (LBs) number in the Tree. N is directly related to the number of levels and the clusters arity k. In most cases N is larger than the number of netlists instances. This means that in these cases we have a low logic utilization. This is not really penalizing since it can be compensated by a high interconnect use. In other words, the area overhead due to unused LBs is compensated by congestion spreading and interconnect reduction. This will be demonstrated in the following sections.

4.4 Rent's Rule MFPGA based model

In this section, we evaluate the number of wires and switches in a k-arity MFPGA as depicted above with N LBs. The MFPGA structure is built with inter-level signaling bandwidth growing according to Rent's Rule. The LBs are recursively partitioned into equally sized clusters at each level of the hierarchy.
We note:
- $N_{in}(\ell)$ the number of inputs of a cluster located at level ℓ.
- $N_{out}(\ell)$ the number of outputs of a cluster located at level ℓ.
- c_{out} the number of outputs of an LB.
- c_{in} the number of inputs of an LB.
- k clusters arity (size).

4.4.1 Wires growth in MFPGA Rent model

At any level ℓ of the hierarchy, and every cluster has $ck^{\ell.p}$ input/output wires, where $c = c_{in} + c_{out}$. The total number of wires is similar to the k-HFPGA case:

$$N_{wire\_total} = cN \sum_{i=1}^{\log_k(N)} k^{(p-1)(i-1)}$$

The total number of wires per LB in the MFPGA is

$$N_{wire}/LB = \begin{cases} c\log_k(N) & \text{if } p = 1 \\ c\frac{1-N^{p-1}}{1-k^{p-1}} & \text{if } p \neq 1 \end{cases}$$

Thus:

$$N_{wire}/LB = \begin{cases} O(log_k(N)) & \text{if } p = 1 \\ O(1) & \text{if } p < 1 \end{cases} \tag{4.9}$$

4.4.2 Switch growth in Rent MFPGA model

We can look at the number of switches required by MFPGA knowing that each MSB is a fully populated cross-bar. We modelize upward and downward networks separately.

Downward network:

Clusters located at level ℓ contain $N_{in}(\ell-1)$ DMSB with k outputs and $\frac{N_{in}(\ell)+kN_{out}(\ell-1)}{N_{in}(\ell-1)}$ inputs. Since DMSB are full crossbar devices, we get $k(N_{in}(\ell) + kN_{out}(\ell-1))$ switches in the switch box of a level ℓ cluster. As we have $\frac{N}{k^\ell}$ clusters in level ℓ, we get a total number of switches, related to the downward network, given by:

$$\sum_{\ell=1}^{\log_k(N)} k \times N \times \frac{N_{in}(\ell) + kN_{out}(\ell-1)}{k^\ell}$$

$N_{out}(0) = c_{out}$ is the number of outputs of a Basic Logic Block. Following equation (4.4), we get $N_{in}(\ell) = c_{in}.k^{\ell.p}$ and $N_{out}(\ell-1) = c_{out}.k^{(\ell-1)p}$. The total number of switches used in the downward network is:

$$N_{switch\_down} = N \times (k^p c_{in} + kc_{out}) \times \sum_{\ell=1}^{\log_k(N)} k^{(p-1)(\ell-1)}$$

Upward network:

Clusters located at level ℓ contain $N_{out}(\ell-1)$ UMSB with k inputs and k outputs. As we assume that UMSB are full crossbar devices, we get $k^2 \times N_{out}(\ell-1)$ switches in the switch box of a level ℓ cluster. As we have $\frac{N}{k^\ell}$ clusters at level ℓ we get the total number of switches, related to the upward network:

$$\sum_{\ell=1}^{\log_k(N)} \frac{k^2 \times N}{k^\ell} \times N_{out}(\ell-1)$$

$N_{out}(0) = c_{out}$ is the number of outputs of a Basic Logic Block. Following (4.4), we get $N_{out}(\ell-1) = c_{out}.k^{(\ell-1)p}$.
The total number of switches used in the upward interconnect is:

$$N_{switch\_up} = N \times k \times c_{out} \times \sum_{\ell=1}^{\log_k(N)} k^{(p-1)(\ell-1)}$$

The total number of MFPGA interconnect switches is:

$$N_{switch}/MFPGA = N_{switch\_down} + N_{switch\_up}$$

$$N_{switch}/MFPGA = N \times (k^p c_{in} + 2kc_{out}) \times \sum_{\ell=1}^{\log_k(N)} k^{(p-1)(\ell-1)}$$

The number of switches per Logic Block is:

$$N_{switch}/LB = (k^p c_{in} + 2kc_{out}) \times \sum_{\ell=1}^{\log_k(N)} k^{(p-1)(\ell-1)}$$

$$N_{switch}/LB = \begin{cases} (k^p c_{in} + 2kc_{out}) \times \frac{1-N^{p-1}}{1-k^{p-1}} & \text{if } p \neq 1 \\ (kc_{in} + 2kc_{out}) \times \log_k(N) & \text{if } p = 1 \end{cases} \quad (4.10)$$

$$N_{switch}/LB = \begin{cases} O(1) & \text{if } p < 1 \\ O(\log_k(N)) & \text{if } p = 1 \end{cases} \quad (4.11)$$

4.4.3 Analysis and comparison with Mesh Model

We notice that the number of the upward network switches is smaller than the switches number in the downward network, this is due to the LB topology where the number of inputs linked to the downward network is higher than the number of outputs linked to the upward network. The ratio is given by the following relationship:

$$\frac{N_{switch\_down}}{N_{switch\_up}} = k^{p-1}\frac{c_{in}}{c_{out}} + 1$$

With $p = 1$, $k = 4$, $c_{in} = 4$ and $c_{out} = 1$ this ratio is equal to 5. In figure 4.11, we show the distribution of interconnect resources between the upward and the downward networks for different MFPGA sizes and with the same Rent parmeter for both networks. This distribution has an important impact on MFPGA routability. Netlist benchmarks with high LB fanouts may need a rich upward network to go up in the hierarchy, thus we can use different Rent parameters for the upward and downward networks with:

$$p_{upward} > p_{downward}$$

Like the Rent parameter p, the arity of the tree has an impact on the total number of switchs in MFPGA. Equation 4.10 gives the relationship between the number of switches per LB, N, p and k. For $k = 4$, figure 4.12 shows the effect of the Rent parameter p and the size of the MFPGA N on switch growth per LB. We see that the number of switch per LB grows slowly with N growth especially if $p < 0,9$. We see also that p has an important impact in the switches number per LB which grows slowly for $p <= 0.6$ and strongly for $p >= 0,9$.

Compared to common Mesh architecture, equations (4.2) and (4.11) show that in the MFPGA architecture, switches requirement grows more slowly. These results are

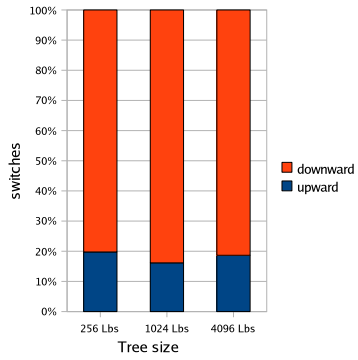


Figure 4.11: Interconnect switches distribution

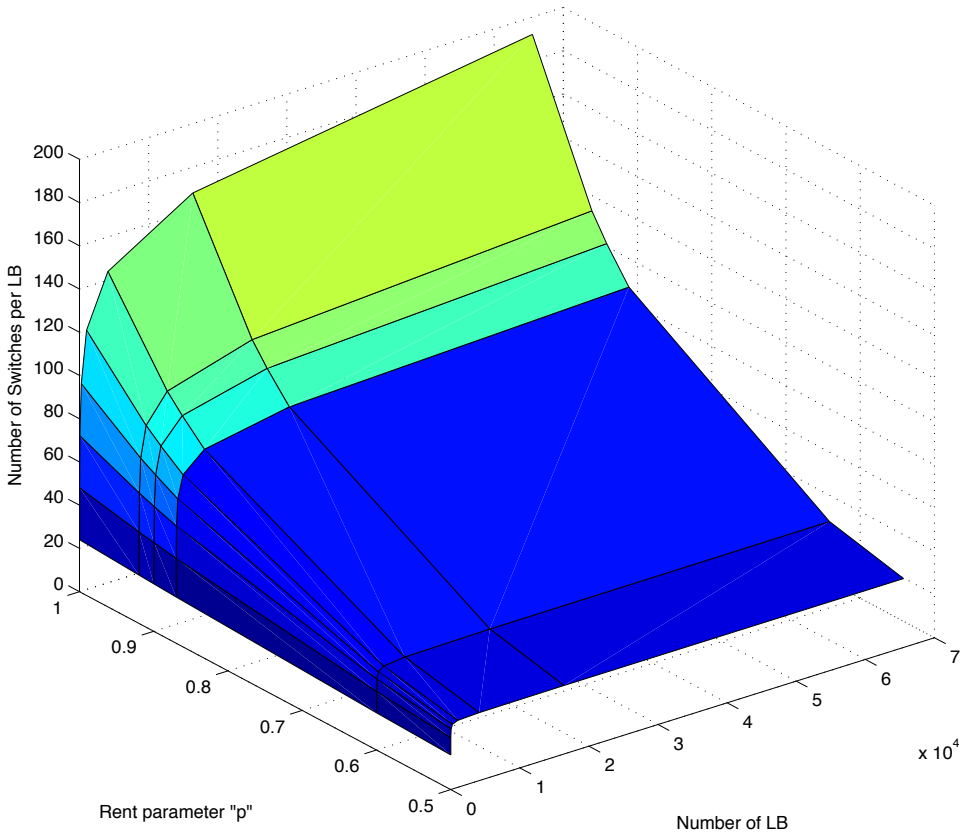


Figure 4.12: LB switches number variation versus N and p

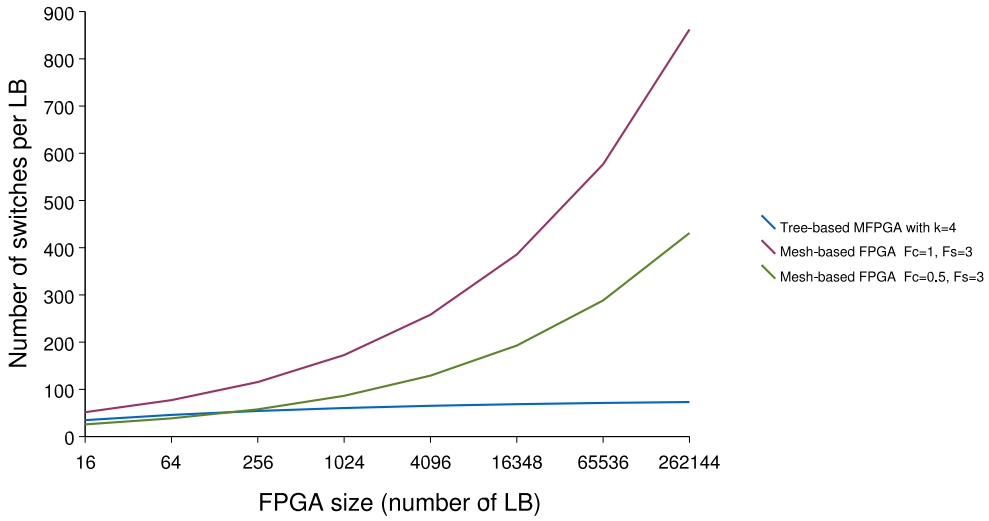


Figure 4.13: Switches number variation in Mesh and MFPGA both with $p = 0.75$

promising for constructing very large structures, especially when p is less than 1. Figure 4.13 shows the variation in number of switches per LB in MFPGA and mesh architectures with a typical $p = 0.75$. MFPGA topology is more efficient in term of switches number for small and large structures.

This comparison is still empiric, and there is no garantee that the 2 architectures are equivalent in term of routability when they use the same Rent parameter p. But in any case even if our architecture uses a higher p, there is enough margin to be better, especially for large structures.
The best way to check this point is through experimental work, based on benchmark circuits implementation; we compare the resulting areas in the case of MFPGA and the VPR clustered Mesh FPGA.

4.5 Architecture exploration methodologies

To compare different architectures we must ensure first that they have the same flexibility and the capability to implement netlists with equivalent congestion. Thus, architecture evaluation must be based on benchmark circuits implementation. We propose an experimentation platform enclosing a set of architecture adaptive tools. Then, we present different metrics and models to evaluate interconnect structures efficiency in terms of area and speed.

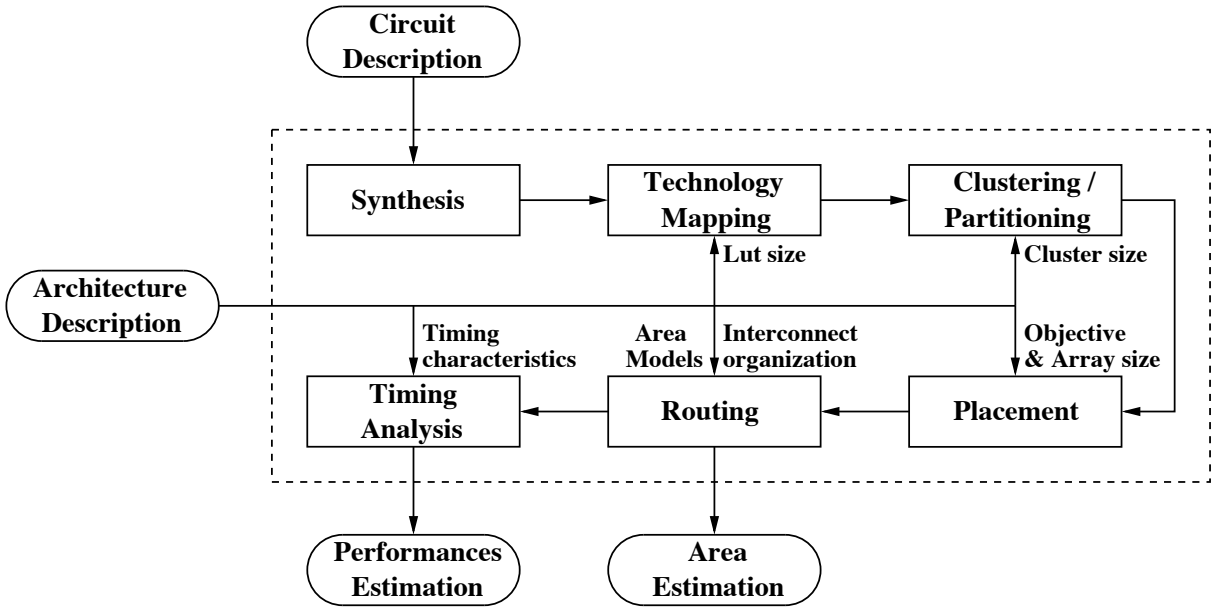


Figure 4.14: Architectures exploration platform

4.5.1 Experimental platform for MFPGA

Rent's rule provides an empirical estimation of switching and wiring requirements, which is not sufficient since it does not give accurate information about interconnect routability. FPGA interconnect flexibility is a very important feature since it reflects the architecture potential to route different highly congested benchmark netlists. The best way to check this point is to implement different benchmark circuits and to evaluate the required area and minimal clock frequency.

Since we are exploring different architectures parameters, we need as generic as possible CAD tools which can deal with different types of architectures. We propose a set of generic tools requiring a minimum of effort to be adapted to a specific architecture topology. In figure 4.14 we present the dependency between each phase in the CAD flow and the target architecture.

Synthesis and Mapping

Synthesis consists in translating a circuit description into a gate-level representation. As illustrated on figure 4.14 this operation is architecture independent. For a given circuit, SIS tool is used [E.M.Sentovich et al., 1992] to perform a technology independent logic optimization and to conduct the technology mapping with Flowmap [J.Cong and Y.Ding, 1994]. It can be replaced by any other commercial synthesis tool.
As explained in chapter 2, mapping consists in translating the description based on boolean logic gates into a description with k-input LUTs and flip-flops. The only required architecture parameter is k, the LUT inputs number. In our flow we use FlowMap

algorithm [J.Cong and Y.Ding, 1994], which is included in SIS package. As presented in figure 4.14, this tool depends only on LUTs size and can target any interconnect topology. It can be driven by different objectives like timing (depth optimization) and area (LUTs number).
It can be noted that today, industrial synthesis tools can target specific architectures interconnects. Thus, in this early stage, they can alleviate routing congestion and improve performance.

Partitionning

Our routing resources are depopulated and we have few different ways to connect a source to a destination. We apply a multilevel partitionning to distribute evenly the nets to route among clusters. We use a top-down recursive partitionning approach which includes a multilevel clustering and a multilevel refinement phases. Since logic blocks positions inside the owner cluster are equivalent, there is no need for detailed placement, phase and clusters positions are allocated randomly inside their owners.

The way how logic LBs are distributed between MFPGA clusters has an important impact on congestion. It is worthwhile to reduce external communications, since local connections are cheaper in terms of delay and routability. Another way to decrease congestion consists in balancing clusters occupation, distributing sparsely mapped circuits and leaving some many logic blocks unused. Methods and algorithms used for partitionning are described in [Z.Marrakchi, 2008].

Routing and Timing Analysis

FPGA routing consists in assigning netlist signals to routing resources such that no routing resource is shared by more than one net. Thus routing is interconnect dependent. Fortunately, $Pathfinder$ [L.McMurchie and C.Ebeling, 1995] is a truly architecture-adaptive routing algorithm, since it can deal with any graph representing the interconnect routing resources. Consequently the only element depending on architecture interconnect is the routing graph. It describes the way netlist instances are routed using architecture resources. It also allows to evaluate routing delays between netlist instances connections. A path connecting two instances crosses several wires and switches. The connection delay is equal to the sum of resources delays.
To evaluate performances of a circuit implemented on MFPGA in terms of clock frequency, we use a timing graph for critical path extraction; it is a direct acyclic graph generated from the netlist hypergraph. Nodes correspond to instances pins and edges to connections between them. Based on the resulting routed graph, each edge is labeled with the corresponding routed connection delay.

4.5.2 Area Model

This section describes the area model used to compute performance metrics for FPGA architectures under investigation. Based on this model, we can compare architectures area efficiencies and achieve different tradeoffs.

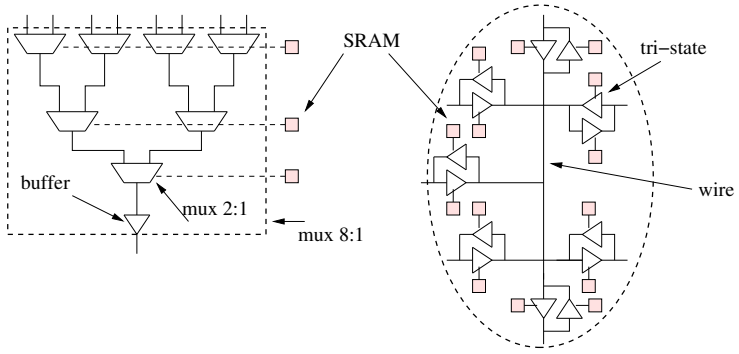


Figure 4.15: Switch elements MFPGA vs. Mesh

As was explained by DeHon in [A.DeHon, 2001], the large area of switches compared to wires area is one of the key reasons why we must care about the number of switches required by a network. As mentioned in [V.Betz et al., 1999], discussions with FPGA vendors have revealed that transistor area, and not wiring density, is the area limiting factor.

Results from DeHon [A.DeHon, 2000] show that a N-node fat-tree can be laid out in an $O(N)$ active area (area dedicated to nodes and switches), using $O(log(N))$ wiring layers. Since MFPGA is similar to BFT, we assume that MFPGA area is switch dominant, and the total area is the sum of MSBs and LBs areas. The large area ratio means that we definitely need to take much care about switch count in the interconnect. In this work we consider 2 models for comparison:

- Switches count: the required interconnect switches number.

- Area evaluation: We estimate the FPGA area as the sum of areas required for all logic and switching cells. We use symbolic standard cells library [Alliance, 2006] to estimate the FPGA required area. Basic cells areas are presented in table 4.1. Figure 4.15 shows switching cells for Mesh and MFPGA architectures based on tristates and multiplexers respectively.

4.5.3 Mesh-based candidate architecture

To evaluate the proposed MFPGA interconnect topology, we focus on a comparaison with a typical Mesh based architecture. The Candidate mesh FPGA (CFPGA) is the well known VPR clustered architecture [V.Betz et al., 1999] that uses an uniform routing with single-length segments and a disjoint switch block. Each Clustered Logic Block (CLB) in

Cell	Area λ^2
sram	30×50
tri-state	35×50
buffer	20×50
flip-flop	90×50
mux 2:1	45×50

Table 4.1: Standard cells characteristics

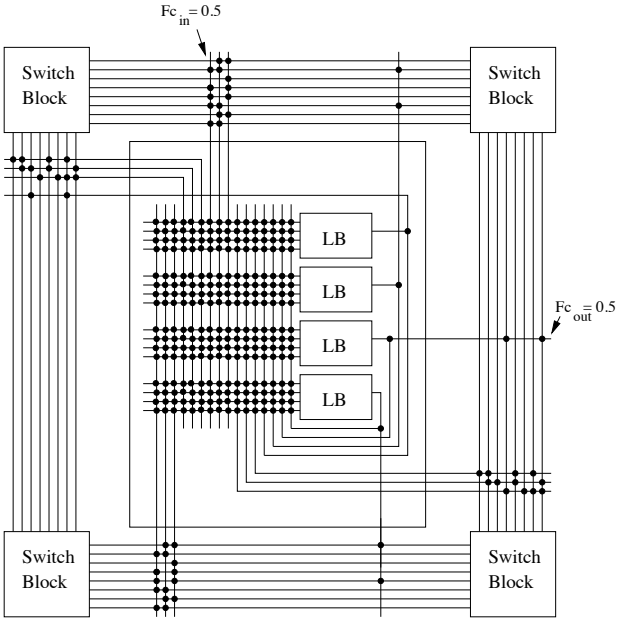


Figure 4.16: CFPGA cluster containing 4 LBs, 10 inputs and 4 outputs

CFPGA contains 4 LBs, 10 inputs and 4 outputs which are distributed over the cluster sides. Each LB contains one 4-LUT and a bypass Flip-Flop. LB pins are connected to cluster pins using a full local crossbar. Connection block population is defined by $F_{c_{in}}$ and $F_{c_{out}}$ parameters, where $F_{c_{in}}$ is the flexibility factor for cluster input connection to adjacent routing channel and $F_{c_{out}}$ is the cluster output flexibility factor to routing channel. Work in [E.Ahmed and J.Rose, 2000] shows that $F_{c_{in}} = 0.5$ and $F_{c_{out}} = 0.25$ are the most efficent flexibilities. In figure 4.16, we show an CFPGA cluster and its surrounding interconnect.

We chose the CFPGA because that approach dominates by far the literature, and it was necessary to limit the scope of comparison in order to make this work tractable. Today's architectures consist in improved versions of CFPGA and it is considered as a reference to evaluate improved architectures performance. For example in [V.Betz et al., 1999], authors state that, increasing wire segment length from 1 to 2 logic blocks manhattan distance, increases the speed of long connections by 61% and reduces area by about 20%.

Comparing MFPGA architecture to CFPGA gives us an idea about MFPGA efficiency compared to different recent FPGAs.

SIS [E.M.Sentovich et al., 1992] is used to perform a technology independent logic optimization; Flowmap [J.Cong and Y.Ding, 1994] in SIS is used to conduct the technology mapping. The physical design in VPR [V.Betz and J.Rose, 1997] is then carried out, including timing-driven packing, placement, and routing. VPR generates an FPGA array whose size just fits the given benchmark circuit. Further, VPR selects the routing channel width W as W = 1.2Wmin, and Wmin is the minimum channel width required to route the given benchmark successfully. This means that VPR is customizing the FPGA for a given benchmark so that it reflects the "low-stress" routing situation which usually occurs in industrial FPGAs for "average" circuits.

4.5.4 Benchmark circuits

In order to experiment and quantify the assets of diverse architectures, we use Microelectronics Center of North Carolina (MCNC) designs. As presented in table 4.2, these circuits cover various application types with several sizes (<10K 4-Luts), In/Out Pads number and congestion levels. We used also some itc and $opencores$ circuits [itc, 1999] [opencores, 2009] which are large circuits containing only lookup-tables with 4 inputs (4-LUT) and flip-flops. When we evaluate a specific interconnect topology, we tailor different architectures to each benchmark. For the same interconnect topology we select an architecture with an appropriate level of routability based on the benchmark congestion level (estimated by Rent parameter). For example in the case of Mesh architecture, VPR tool executes a binary search to determine the smallest architecture with the minimal channel width that can route a specific benchmark circuit. Architectures tailoring is interesting to explore different topologies and to check if they can deal with different applications families. In the case of commercial FPGAs, the architecture is set independently of the targeted applications. Nevertheless, we can find different FPGA families proposed by one vendor to fit better specific applications and constraints. In our experimentatal approach, we consider each circuit benchmark as representing a typical design family.

4.6 Experimental Results

4.6.1 Architecture optimization

As explained in section 4.5.3 we use the channel minimizing VPR 4.3 router [V.Betz and J.Rose, 1997] for the mesh. VPR finds the optimal size as well as the optimal channel width needed to place and route each benchmark. We also vary the IO_ratio to achieve the optimal array size. The number of switches needed by each benchmark corresponds to the total number of switches used by the overall optimal target architecture.

Design Name	4-LUTs	In Pads	Out Pads	Function
MCNC				
alu4	584	14	8	ALU
apex2	1878	39	3	
apex4	1262	9	19	
b9	61	41	21	Logic
bigkey	1707	263	179	Key Encryption
c2678	363	233	140	ALU and Control
c5315	725	178	123	ALU and Selector
c7552	881	207	108	ALU and Control
cc	33	21	20	Logic
clma	8383	61	82	Bus Interface
count	37	35	16	Counter
decod	32	5	16	Decoder
des	3235	256	245	Data Encryption
diffeq	1497	64	39	
dsip	1370	229	197	Encryption Circuit
elliptic	3604	131	114	
ex1010	4589	10	10	
ex5p	1064	8	63	
frisc	3556	20	116	
i4	110	192	6	Logic
i9	471	63	522	Logic
misex3	1397	14	14	
pcle	29	19	9	Logic
pcler8	40	25	19	Logic
pdc	4575	16	40	
s298	1931	4	6	PLD
s38417	6406	29	106	Logic
s38584	6447	39	304	Logic
seq	1750	41	35	
spla	3690	16	46	
tseng	1047	52	122	
ava	14964	9	74	AVA Decoder
ITC99				
b_14	2217	33	54	Viper processor (subset)
b_21	4683	33	22	Two copies of b14
b_22	6648	33	22	A copy of b14 and Two modified versions of b14
b_15	3298	37	70	80386 processor (subset)
b_17	10325	38	97	Three copies of b15
Opencores				
usb_funct	5284	128	121	USB function core
aes_core	8583	259	129	AES Chipter
wb_conmax	17746	1130	1416	WISHBONE Conmax IP Core
vga_lcd	31253	89	109	VGA/LCD controller

Table 4.2: Benchmarks characteristics used for experiment

To find the best tradeoff between device routability and switches (area) requirements, we explore MFPGA architectures by varing the number of level, arity of every cluster level and network Rent's parameter p_{net}. The purpose is to find for all benchmark circuits, the architecture with the smallest necessary area. With our tools (section 4.5) we can consider, in the same architecture, different p values for different levels. Clusters located at the same level have the same Rent's growth parameter. We adjust Rent's parameters at every level in order to obtain the smallest architecture fitting every benchmark circuit. Like VPR which applies a binary search to find the smallest architecture channel width, we apply to each level a binary search to determine the smallest Rent's parameter. We apply a dichotomic approach to optimize an MFPGA. We choose a level randomly and decrease its input/output signals number, depending on the previous result obtained in this level; then we move to an other level. In this way we move randomly from a level to another until all levels are optimized.

4.6.2 Area Efficiency

Table 4.3 summarizes the basic results for Mesh and for MFPGA architectures.
Given a benchmark of some fixed size, and performing an experiment on MFPGA with specified parameters, we get results which are summarized in the right part of table 4.3. Column 9 shows the occupation average of each circuit in the target MFPGA. There is a low occupation average in the majority of the benchmarks, due to the depopulation of the interconnect. As mentioned previously, we underuse the logic resources in this type of structure. In addition, the size of smallest MFPGA containing the circuit under investigation is penalized due to the coarse granularity of this architecture. In spite of these constraints we achieve a gain in area efficiency, compared to the mesh architecture. Columns entitled "Switches number" and "Area" in table 4.4 exhibit the decrease of the switches number and total area for the MFPGA structure compared to the Mesh one.

In table 4.4, we observe that the MFPGA architecture has a better density and can implement circuits with lower switches number than for circuits using the Mesh-based architecture. An average 59% reduction of the switches number is achieved. We achieve a 42% switches reduction with $alu4$ (smallest circuit), and 60% with ava (large circuit). This confirms that MFPGA interconnect is very attractive for small circuits and even more for large circuits.
Another advantage of the MFPGA shown in table 4.4 is the gain in SRAM number. This is due to the efficeincy of the MFPGA routing which uses less programmable switches than Mesh FPGA, and also thanks to the decoded multiplexers used in MFPGA compared to one-hot coding in Mesh architecture (see figure 4.15).

We compare the areas of both architectures using a refined estimation model of effective circuit area described in section 4.15. We use the same cells library for both architectures and the total area (logic, routing, and configuration memory) is the sum of basic cells areas (SRAM, multiplexers, tristates, buffers and flip-flops).

Benchmark **circuits**				**Clustered Mesh cluster size 4**			**MFPGA architecture**	
Circuits Names	LUTs Number	IN Pads	OUT Pads	Mesh NxN	Occupation %	Channel Width	Architecture levels	Occupation in Fsize %
alu4	584	14	8	13x13	86	32	4x4x4x4x4	57
apex2	1878	39	3	23x23	88	40	4x4x4x4x4x2	91
apex4	1262	9	19	19x19	87	42	4x4x4x4x4x2	61
bigkey	1707	263	197	21x21	96	28	4x4x4x4x4x2	83
clma	8383	61	82	47x47	94	51	4x4x4x4x4x4x4	51
des	3235	256	245	29x29	96	29	4x4x4x4x4x4	78
diffeq	1497	64	39	20x20	93	29	4x4x4x4x4x2	73
dsip	1370	229	197	19x19	95	31	4x4x4x4x4x2	67
elliptic	3604	131	114	31x31	94	41	4x4x4x4x4x4	87
ex1010	4589	10	10	35x35	93	43	4x4x4x4x4x4x2	56
ex5p	1064	8	63	17x17	92	44	4x4x4x4x4x2	51
frisc	3556	20	116	30x30	98	45	4x4x4x4x4x4	86
misex3	1397	14	14	20x20	87	36	4x4x4x4x4x2	68
pdc	4575	16	40	35x35	93	61	4x4x4x4x4x4x2	55
s298	1931	4	6	23x23	91	27	4x4x4x4x4x2	94
s38417	6406	29	106	41x41	95	37	4x4x4x4x4x4x2	78
s38584	6447	39	304	41x41	96	36	4x4x4x4x4x4x2	78
seq	1750	41	35	22x22	90	40	4x4x4x4x4x2	85
spla	3690	16	46	31x31	96	53	4x4x4x4x4x4	90
tseng	1047	52	122	17x17	90	27	4x4x4x4x4x2	51
ava	14964	11	74	64x64	91	63	4x4x4x4x4x4x4	91
b_14	2217	33	54	24x24	96	44	4x4x4x4x4x3	72
b_21	4683	33	22	35x35	96	47	4x4x4x4x4x4x2	57
b_22	6648	33	22	42x42	94	50	4x4x4x4x4x4x2	81
b_15	3298	37	70	30x30	92	41	4x4x4x4x4x4	81
b_17	10325	38	97	53x53	92	47	4x4x4x4x4x4x3	84
usb_funct	5284	128	121	37x37	96	51	4x4x4x4x4x4x2	65
aes_core	8583	259	129	47x47	97	39	4x4x4x4x4x4x3	70
wb_conmax	17746	1130	1416	70x70	91	81	4x4x4x4x4x4x4x2	54
vga_lcd	31253	89	109	91x91	94	84	4x4x4x4x4x4x4x2	95
Average	5499	104	129	34x34	93	44		73

Table 4.3: Netlists and architectures characteristics

	Clustered Mesh Cluster size 4			MFPGA architecture			Gain MFPGA		
Circuits	Switches $\times 10^3$	SRAM $\times 10^3$	Total Area $(\lambda^2)\times 10^6$	Switches $\times 10^3$	SRAM $\times 10^3$	Total Area $(\lambda^2)\times 10^6$	Switches %	SRAM %	Total Area %
alu4	138	101	422	47	43	182	66	57	57
apex2	506	375	1541	173	127	565	66	66	63
apex4	359	267	1092	138	103	466	62	61	57
bigkey	349	253	1056	129	101	450	63	60	57
clma	2541	1879	7672	1031	821	3614	59	56	53
des	667	487	2047	326	247	1087	51	49	47
diffeq	307	226	954	121	108	445	61	52	53
dsip	310	224	934	143	107	484	54	52	48
elliptic	944	701	2883	326	247	1087	65	65	62
ex1010	1234	915	3763	515	410	1804	58	55	52
ex5p	305	224	915	134	103	460	56	54	50
frisc	952	811	3287	346	254	1134	64	69	66
misex3	354	263	1085	150	113	502	58	57	54
pdc	1636	1207	4889	714	523	2329	56	57	52
s298	380	280	1192	121	108	445	68	61	63
s38417	1508	1126	4662	493	439	1807	67	61	61
s38584	1501	1113	4590	535	452	1898	64	59	59
seq	463	343	1411	163	123	541	65	64	62
spla	1144	847	3448	428	299	1350	63	65	61
tseng	216	157	665	110	90	370	49	43	44
ava	5618	4121	12397	1428	1047	4661	75	75	62
b_14	593	439	1793	231	179	777	61	59	57
b_21	1331	979	4011	545	448	1906	59	54	52
b_22	2002	1480	6027	681	514	2252	66	65	63
b_15	882	651	2681	351	266	1158	60	59	57
b_17	3042	2241	9185	1364	930	4250	55	59	54
usb_funct	1586	1166	4760	583	474	2010	63	59	58
aes_core	2073	1532	6332	959	751	3223	54	51	49
wb_conmax	8734	6361	21008	5865	3940	17404	33	38	17
vga_lcd	14436	10665	29599	5865	3940	17404	59	63	41
Average	1870	1381	4877	801	577	2536	60	58	54

Table 4.4: Comparison between MFPGA and clustered VPR-style Mesh

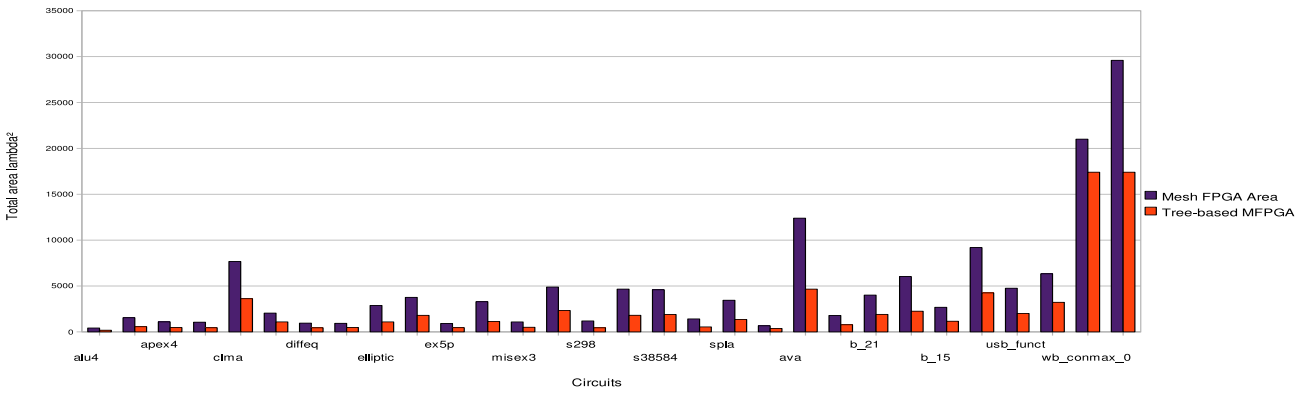


Figure 4.17: MFPGA area Vs. Mesh area (30 benchmark circuits)

Circuits	Level 1	Level 2	Level 3	Level 4	Level 5
apex2	1	0.89	0.86	0.84	0.77
tseng	0.79	0.79	0.79	0.72	0.67

Table 4.5: Levels Rent's parameters for 2 circuits

As presented in figure 4.17, in all cases, the required MFPGA area is smaller than the Mesh one. On the average with the MFPGA architecture we save 54% of the total area.

The MFPGA architecture efficiency is due essentially to its ability to control simultaneously logic blocks occupancy and interconnect population, based on LBs number N and architecture Rent's parameter p respectively. For example in the case of $apex2$ circuit, we used an architecture with a high logic occupancy (91%) and a high Rent's parameters as shown in table 4.5. In the case of $tseng$ circuit, we have a low occupancy (51%) and we achieve routability with a low architecture Rent's parameters as illustrated in table 4.5. This confirms that we can balance interconnect use with logic blocks utilization thanks to logic occupancy decreasing and congestion spreading. In fact we have a 20% lower LBs occupancy than for Mesh case, the logic extra area allows us to exploit interconnect better. The MFPGA high-interconnect/low-logic utilization approach is just opposed to the high logic utilization approach that has been adopted for Mesh-based FPGA. As shown in figure 4.18, unlike Mesh case where interconnect occupies 90% of the overall area, in MFPGA architecture interconnect occupies 73%. In this case logic area is increased by 20% and interconnect area is reduced by 69%.

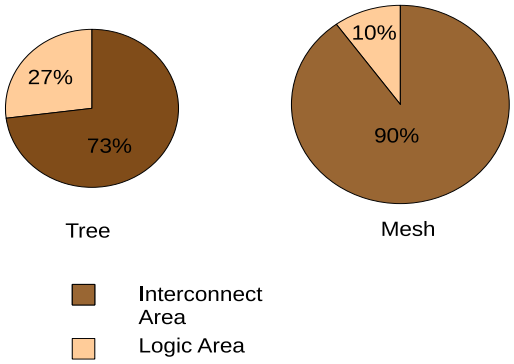


Figure 4.18: Area distribution between interconnect resources, logic blocks

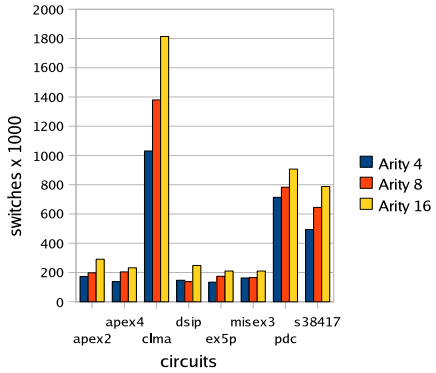


Figure 4.19: Clusters arity effect on switches number

4.6.3 Clusters Arity Effect

As one can notice, we considered in table 4.3 Tree architecture with clusters arity basicaly equal to 4. To get an idea about arity effect on architecture density and speed performances, we vary clusters arity and we evaluate for every benchmark circuit the required switches and wires number and the resulting critical path. To evaluate performances, we used a simple model based on evaluation of the number of switches crossed by the critical path. This estimation consists in determining the longest path in terms of switches (ignoring wires delays).
We notice that when we increase clusters arity, the required switches number increases. When clusters arity increases, the required multiplexers grow larger and consequently the bound on area efficiency goes down. As shown in figure 4.19, switches number is increased by 23% when we increase clusters arity from 4 to 8.
When we increase clusters arity, the architecture levels number decreases. Consequently multiplexers sizes increase and their total number decreases. Thus the total number of

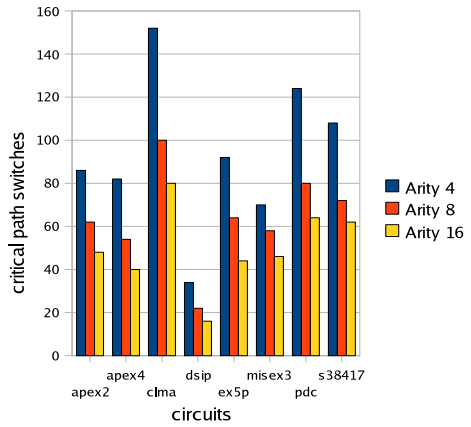


Figure 4.20: Clusters arity effect on critical path crossed switches

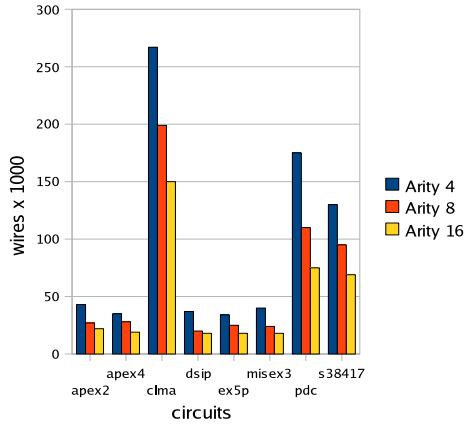


Figure 4.21: Clusters arity effect on wires number (<=> Muxes number)

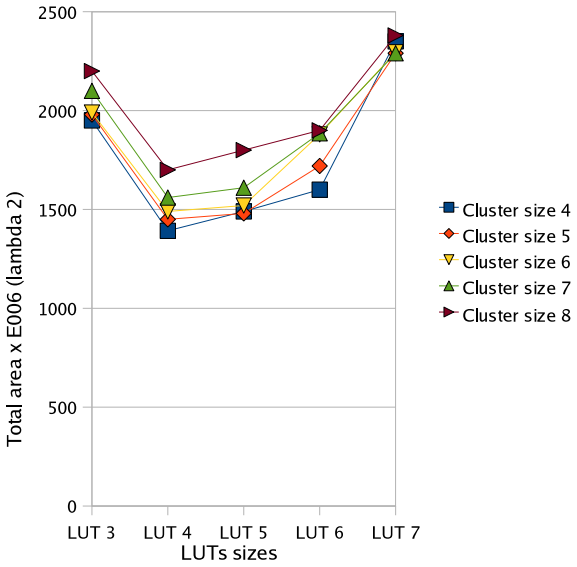


Figure 4.22: Total area for clusters sizes 4-8 (21 benchmark avg.)

wires decreases. For example, as shown in figure 4.21, wires number is reduced by 32% when we increase clusters arity from 4 to 8. In terms of performance we notice, as shown in figure 4.20, that the number of switches crossed by the critical path decreases when we increase arity. With larger clusters arity, we can absorb larger number of nets, and communication becomes local. For example when we increase clusters arity from 4 to 8, the crossed switches number in the critical path is reduced by 27%.
The choice of clusters arity must be consistent with the application specifications and constraints. For applications requiring high speed performance and low power dissipation, it is recommended to use clusters with high arity (8-16). If we need to reduce silicon area, using small clusters arities seems to be more efficient(2-4).

4.6.4 LUT Size Effect

In this section we evaluate the effect of LUTs size k (number of LUT inputs) on MFPGA performances. Mapping is the phase where logic gates are transformed into k-bounded cells. When k increases the size of LUTs increases and their number decreases. Thus, as shown in [E.Ahmed and J.Rose, 2000], the effect of k increasing is not predictable and can only be determined by experimentation.
Our experimentation is based on generating circuits with LUTs sizes ranging from 3 to 7 and implementing them on MFGA with these LUTs sizes and clusters arities ranging from 4 to 8. First, as shown in figure 4.22, we evaluate the effect of LUTs size changing on MFPGA area. Results correspond to the average area of 21 circuits. We notice that initially there is a reduction in area between 3-LUT and 4-LUT and afterwards there is

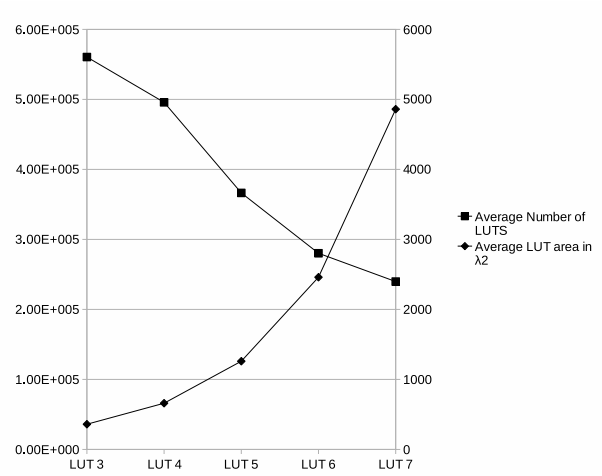


Figure 4.23: LUTs area and LUTs number versus LUT size (for cluster arity = 4)

an increase in area with the rise in LUT and cluster size. It can be noted that architecture with 4-LUT 4 and cluster size 4 gives overall most efficient average area for benchmark circuits.

The total area can be broken into two parts, the logic block area and interconnect area. The logic area is the product of the total number of LUTs times the area per LUT. A plot of these 2 components for clusters arity equal to 4, is given in figure 4.23 (the left vertical axis presents area per LUT in (λ^2) and the right vertical axis presents LUTs number). The logic block area grows exponentially with LUT size as there are 2^k bits in a k-inputs LUT. As k increases, though, the number of LUTs decreases (because each LUT can implement more logic functions) as shown by the downward curve in figure 4.23. However, the rate of increase in area is steeper than the rate of decrease in LUTs number. Concerning the interconnect area we notice that it decreases with LUT size increase. Since logic area increase is steeper than interconnect area decrease, we obtain the upward trend in figure 4.22.

The second key metric is critical path delay. We only evaluate the number of switches crossed by the critical path. Figure 4.24 shows the average critical path switches number across all circuits as a function of clusters arities and LUTs sizes. Observing the figure, it is clear that increasing clusters arity and LUTs size decreases the number of switches crossed by the critical path. These decreases are significant: an architecture with cluster arity 4 and 3-LUT has an average critical path switches number of 180 while cluster arity 8 and 7-LUT has an average critical path switches number of just 76. This behavior is explained by the decrease of the number of LUTs and clusters in series on the critical path. Nevertheless, to get an idea about the accurate delay we have to consider the increase of intrinsic LUT delay when its size increases.

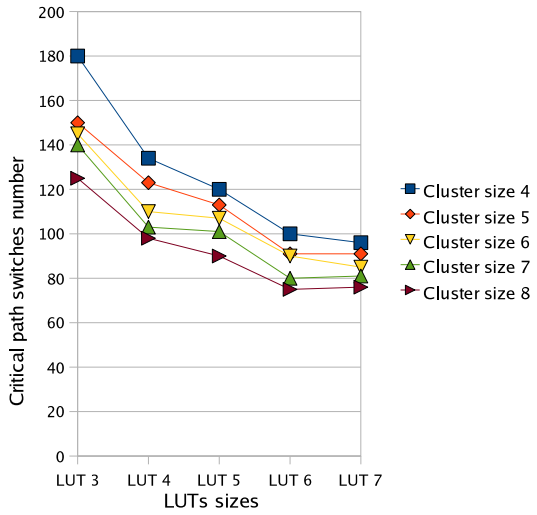


Figure 4.24: Critical path switches number clusters sizes 4-8 (21 benchmark avg.)

4.7 Conclusion

We described a new hierarchical multilevel MFPGA architecture and its suitable configuration tools. We show that good balancing of LUT utilization and interconnect utilization implies lower area than the traditional Mesh topology.
The downward and upward networks are predictable interconnects, due to unicity of the downward or upward path leading to destination. This particularity yields a very interesting advantage for the routing phase and routability of the implemented design. The new topology based on two hierarchical unidirectional networks seems more robust and can achieve better speed than symmetrical Mesh-based FPGA architectures. The average path length between two LBs grows as $O(\log_k(N))$ for an MFPGA whereas it is $O(N)$ for a Mesh-based FPGA.
Experimental results based on implementing MCNC, opencores and ITC99 benchs shows that MFPGA architecture has better area efficiency than the common VPR clustered Mesh (CFPGA). We also showed that MFPGA efficiency in terms of area can be controlled by interconnect Rent's parameters and clusters arity. In this way it is a scalable architecture which can be adapted to specific domains to satisfy different tradeoffs. Nevertheless, it is not proved that the physical layout of the MFPGA is switch dominant with a total area equivalent to the logic and switches areas. Planarization of the MFPGA network is not a simple packing process with simple replication of basic tiles. These difficulties for MFPGA to fit planar chip structure and the important number of wires per logic block can induce a wire dominant layout which impacts its total area and may distort some assumptions. For this reasons we are interested to design the MFPGA layout, to test its limitations compared to the standard Mesh and finally to measure the real area.

5 Physical Planning of the Tree-Based MFPGA

Designing the tree interconnect is a major challenge for Hierarchical FPGA such MFPGA. In this chapter, we propose a floorplanning method that can preserve the hierarchy of the MFPGA and place LBs regularly as well as switch fabrics according to user-defined tree parameters such as network topology, physical dimension of the network nodes, along with floorplan specification (number of levels, arity of the different levels of hierarchy, locations of the I/O). This layout generation method is scalable and can create a floorplan that best satisfies different design and architecture constraints.

5.1 Challenge for MFPGA layout design

Although quite different in their topologies, FPGA networks have an important aspect in common: regularity. Preserving regularity and hierarchy assemblies in the silicon floorplan is critical for MFPGA implementation. Furthermore, on-chip interconnect delays and power consumption add additional requirements to FPGA design. In order to reduce wiring delays and interconnect energy dissipation, total network wirelength must be minimized.

Contemporary VLSI processes provide easily 6 layers of metalization for wiring. With the advent of modern VLSI processes, it is feasible for process technology to add metal layers as long as the cost of the extra mask steps and processing are justified by the area gains, resulting in considerable progress over traditional VLSI circuits.

If active devices for some structure take up total area A, it is interesting to ask if the active device can be laid out compactly to fit in O(A) two-dimensional surface area and

can be supported by the multilayer wiring. Otherwise this device becomes wire dominant. Further, we should ask how many wiring layers are required to support the most compact active area layout.
MFPGA structure is a Butterfly Fat Tree like. Results from DeHon [A.DeHon, 2000] show that a N-node fat-tree can be laid out in an $O(N)$ active device area (area dedicated to nodes and switches), using $O(log(N))$ wiring layers. No existing publication of real physical layout prove such theoretical results until now. Those results don't take into account all additional problems of real layout related to configuration memory, power tracks, congestion distribution or I/0 connections. Our work could constitute a silicon proof of this concept.

Unlike traditional floorplanning that deals with circuit macro block placement and wire routing, Tree-based FPGA floorplanning must solve these problems from a different perspective, namely:

- Folding and planarization: Tree network topologies are multi-dimensional. Tree planar layout requires that clusters are tiled and abutted on the floorplan in a planar two-dimensional area. The planarization process is also constrained by the pre-defined aspect ratio and by the numbers of level of the hierarchy level.

- Regularity, symmetry and scalability: planarization of the MFPGA network is not a simple packing process with simple replication of basic tiles. In MFPGA topologies, each level of hierarchy presents a regular replication of lower level with a new switch block that is different in other levels. Different sizes of the switch blocks at each level creates local irregularity and makes scalability more complicated. We adopt a specific technique for the floorplan to preserve regularity, symmetry and scalability, .

- Critical path and total wirelength: Interconnect delays and power consumption are the two critical issues in FPGA network design. On one hand, a large fraction of the timing delay is spent on signal propagation through the interconnect On the other hand, interconnect is a significant amount of energy that is also dissipated on charging and discharging the load capacitance on the wires. Therefore, an optimized interconnect floorplan is very important for FPGA performance and energy consumption.

This work merges the issue of programmable switch circuit design with the design of the routing architecture. We propose a novel approach for laying out tree-based architecture with a generic technique that optimizes the overall design area and preserve tree scalability. In addition, this work consists in developing techniques for designing MFPGA routing networks of aribitrary size. Since specifc applications impose a particular set of requirements on FPGA resources, it is advantageous to develop a different routing network for each application domain; for example, some circuits may have many high-fanout signals that benefit from long wires with buffers and needs many routing

resources in the highest levels of hierarchy, while other circuits may be dominated by many local connections which need more routing resources in the lowest levels of hierarchy.

5.2 MFPGA Wiring requirement

To estimate the required wiring area in a two-dimensional layout, we refer to Thompson's argument about bisection width [C.Thompson, 1979]. He defines the minimal bisection width of a graph as the number of cuts needed to divide it in half. In figure 5.1 we show the smallest number of edges whose removal disconnects one half of the vertices from the other. Let the Minimum Bisection Width (MBW) of a network be $\propto (W)$

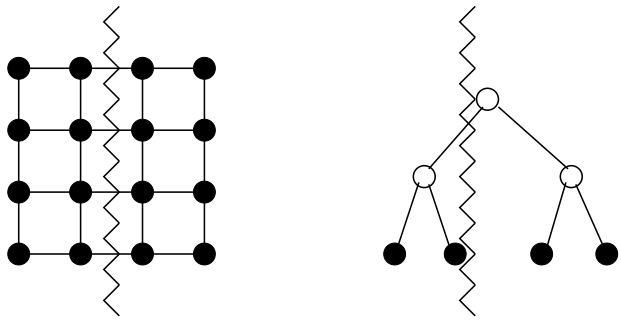


Figure 5.1: The minimal bisection width of a Mesh and a binary Tree

(proportional to W), meaning that in any layout, $MBW \propto (W)$ wires are necessarily crossing between the two halves of the layout. When we are limited to 2D-VLSI, this means that MBW wires must cross the 1D-line which bisects the chip. This fact gives a lower bound of MBW on the width of the chip, if we assume a fixed number of wire layers. Since this property holds recursively, we can establish that both the width and height of the layout must be $\propto (W)$, making the entire chip area $\propto (W^2)$.
This property (relation between bisection width and wiring area) is an interesting feature to estimate whether FPGA area is dominated by wires or switches.

In the case of our MFPGA, at each level ℓ of the hierarchy, every switching node has $n_{in}(\ell)$ inputs and $n_{out}(\ell)$ outputs. This makes the bisection width equal to $(c_{in} + c_{out})k^{\ell.p}$. Since

$$\forall \ell \in \{1, \ldots, \log_k(N)\} \quad k^{\ell.p} \leq N$$

, the bisection width is $O(N^p)$. For a 2-dimensional network layout this bisection width must cross the perimeter out of the subarray. Thus the perimeter of each subarray is $O(N^p)$. The areas of the subarray will be proportional to the square of its perimeter, making: $A_{subarray} \propto N^{2p}$. The required area per logic block (LB) based on wiring constraints, is therefore evaluated by:

$$A_{LB} \propto N^{2p-1}$$

In MFPGA architecture we can control bisection bandwidth in each level based on Rent's parameter ($p < 1$). Consequently, physical layout generation may be much optimized since wiring is no more dominant and the total area is the active area of LB and switches.

5.3 Problem Formulation

In a Mesh FPGA floorplan, each couple of logic block and switch block is placed as a dedicated hard block tile. In direct Network as in the case of the Mesh interconnect, the FPGA can be tiled into a two-dimensional array.
In indirect networks, as in the case of the Butterfly-like MFPGA network, the tiling of the switch fabrics is constrained by the locations of the target logic block.
Different network topologies and application requirements impose different constraints on floorplanning problems. To be more specific, we summarize the constraints that are relevant for FPGA floorplanning:

1. Regularity constraints: physical FPGA placement should preserve the regularity of the original network topology.

2. Hierarchy constraints: FPGA networks have hierarchical topologies (clusters). The placement should also preserve this hierarchical aspect.

3. I/O constraints: An FPGA is implemented on a single chip. Some Logic Nodes (or node switches, in the case of switch fabrics) are used as I/O nodes; therefore, they need to be placed at the periphery of the floorplan. An FPGA floorplan must accommodate those nodes at their own locations.

4. Aspect-ratio constraints: Chip die size is limited by the silicon area and aspect ratio. Therefore, node logic blocks and node switch blocks need to be packed into a two-dimensional array with predefined numbers of rows and columns.

5. Critical-path constraints: The links between some nodes of logic blocks can be the critical paths. Therefore, the nodes connected by the critical paths must be placed closer to each other.

6. Total net-length constraints: Reducing the total net length achieves shorter interconnect delays with lower power consumption.

The floorplanning problem is to determine a mapping from 3 to 2 dimensions, such that the constraints are met and the overall wiring length is minimal. Such a problem is computationally intractable, and is the object of extensive investigation in the ASIC domain. We propose an approach that takes into account the special properties of MFPGA topologies.

5.4 Network Floorplan

On-chip implementation of MFPGA requires planarisation of the interconnect network onto the slicon floorplan. We propose a physical planning method that allows floorplans for different hierarchical structure under different architecures parmeters and design constraints.
For clarity, we provide a construction of a small 4x4 tree-based MFPGA architecture floorplan. The wires numbers in this candidate architecture grows with Rent parameter p=1. To make the difference in switches and wires levels, upward and downward networks, we use different colors for all switches and wires classes. As shown in figure 5.2, every $UMSB_{level1}$ and its outputs wires are Red, every $DMSB_{level1}$ and its outputs wires are Blue. Green and Brown colors are used for $UMSB_{level2}$ and $DMSB_{level2}$ respectively.

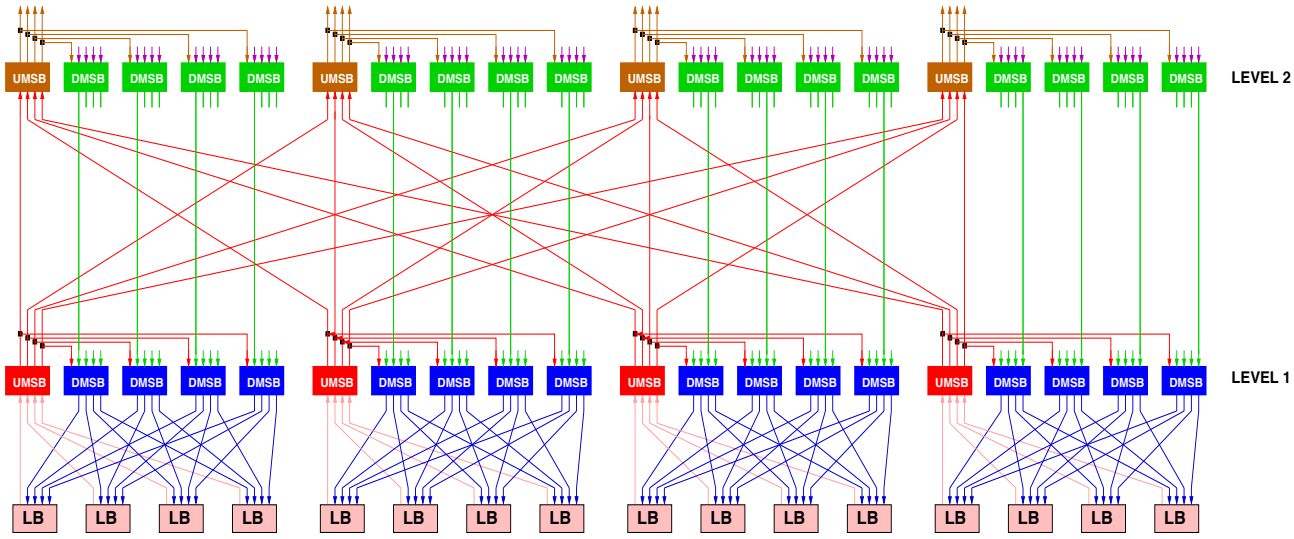


Figure 5.2: light representation of 4x4 MFPGA Architecture with Rent parameter p=1

The flattened MFPGA presented in figure 5.4 is topologically equivalent to the original MFPGA view presented in figure 5.2. All MSBs are broken down in switch level and every switch is placed in front of its successor horizontally or vertically. As shown in figure 5.5, every DMSB has 4 outputs correponding to 4 multiplexers outputs that drives 4 inputs of 4 different subclusters (or logic block in the lowest level). Every multiplexer correponds to a set of switches grouped together and placed in the same row or the same colum as their successor. Figure 5.3 shows new type of interconnect organizaton which brings together every cluster and its corresponding interconnect only, in order to form what we called $Part_{level1}$ and $Part_{level2}$. The $Part_{level1}$ is replicated on column to form level1 then $Part_{level2}$ is replicated on rows to form level2 of the hierarchy.

By replicating $Part_{level1}$ and $Part_{level2}$, we can increase the arity of the tree in level1

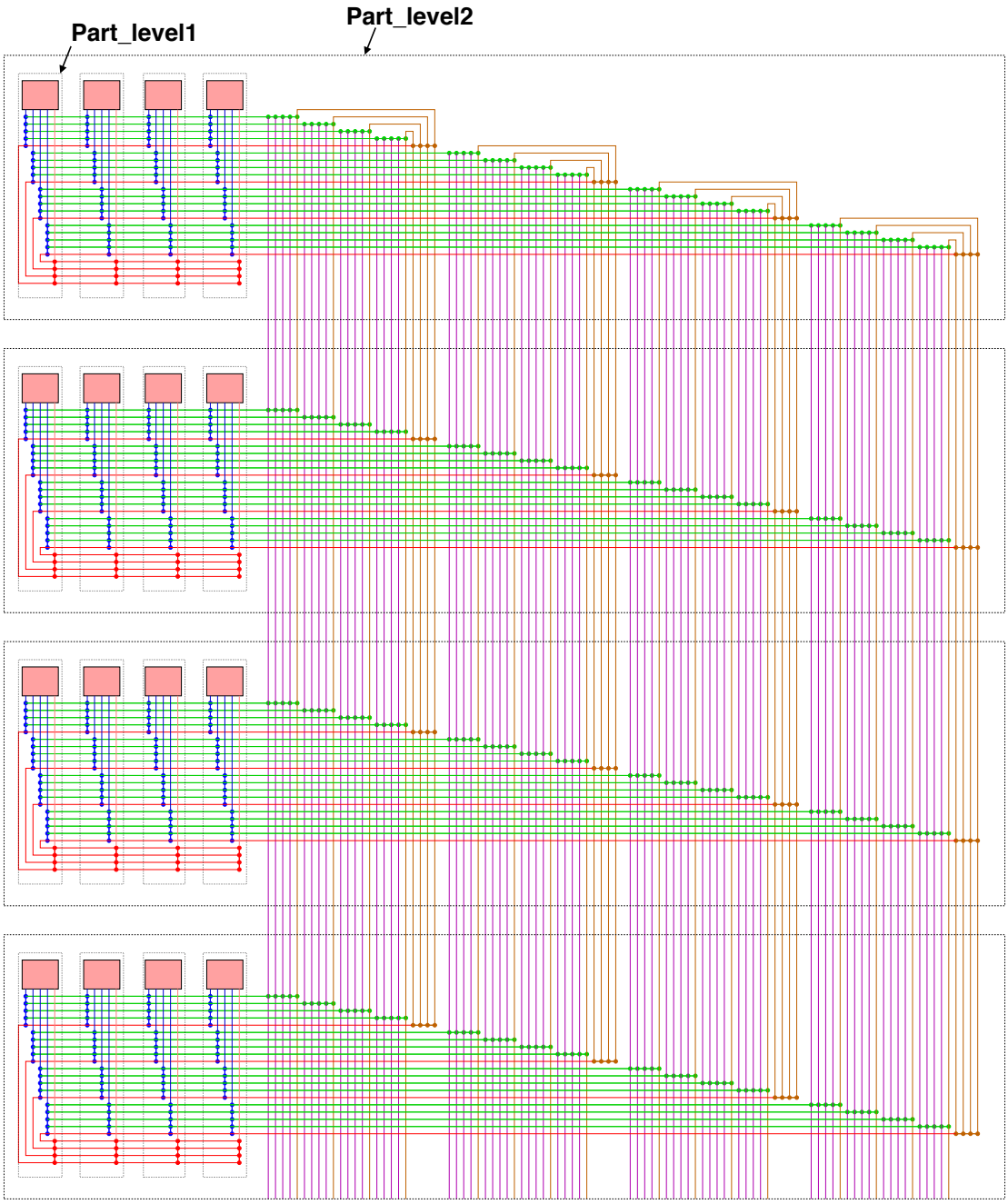


Figure 5.3: Flatten 4x4 Tree-based MFPGA

and level2 respectively. This is the way that gives the scalability of the hierarchy in the terms of the arity named k. In the same way we can build the next levels of hierarchy.

In order to spread the congestion and the wires density over the MFPGA surface, we interweave different interconnection levels to build the rearranged MFPGA presented

in figure 5.4, which is also topologically equivalent to the original MFPGA view presented in figure 5.2. This figure pops up a regular structure based on tiles, each one including: Logic Block, a set of level1 switches and a set of level2 switches. In order to vary the number of inputs or outputs in a cluster level, we do it simply by varying the number of switches in the tile. This corresponds to change the multiplexers sizes in level1 or level2. Thus we include a scalability in terms of clusters inputs number and ouputs number.

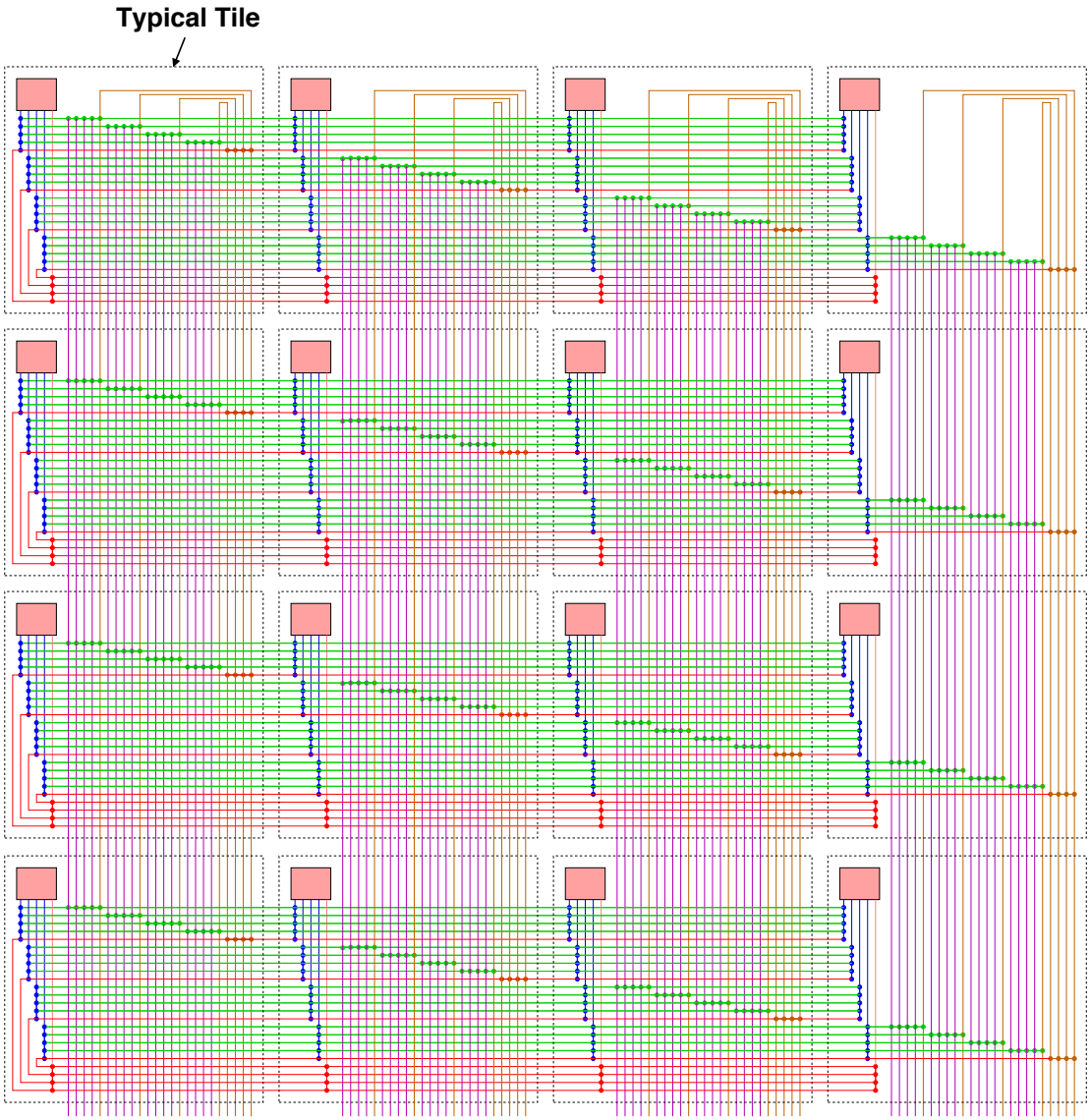


Figure 5.4: The rearranged 4x4 Tree-based MFPGA

Figure 5.6 shows the 3D rearranged 4x4 MFPGA. For clarity we show the downward

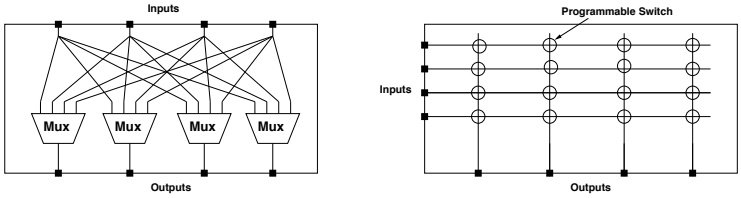


Figure 5.5: Mini Switch Box topology

inter-level wires (green wires) only in the first plan, the rest is simply a replication of this plan. Figure 5.7 shows the projection in the 2D plan of the rearranged MFPGA and its interconnect organization that refers to figure 5.4. All tiles are equivalent in terms of density, logic and switches distribution. All tiles of the same column are equivalent; through tiles of the same row have some differences in routing topology, they are still equivalent in terms of number of switches.

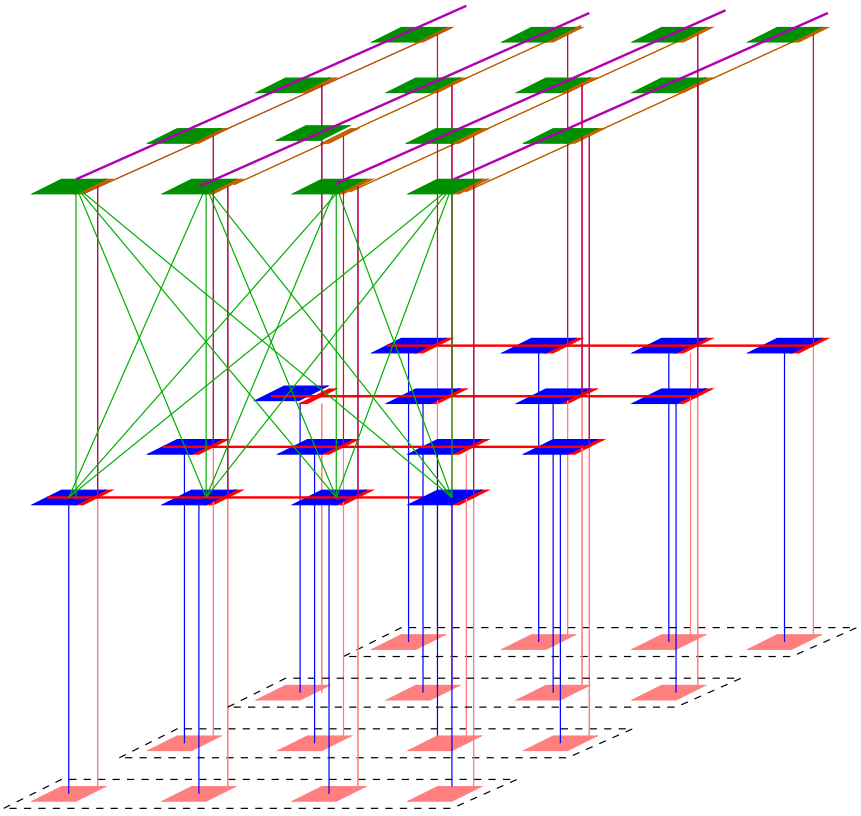


Figure 5.6: 3D view of the rearranged 4x4 Tree-based MFPGA

To build larger tree-based MFPGA, we apply the same technique of interweaving the different interconnect levels of the hierarchy . This technique allows to vary the ar-

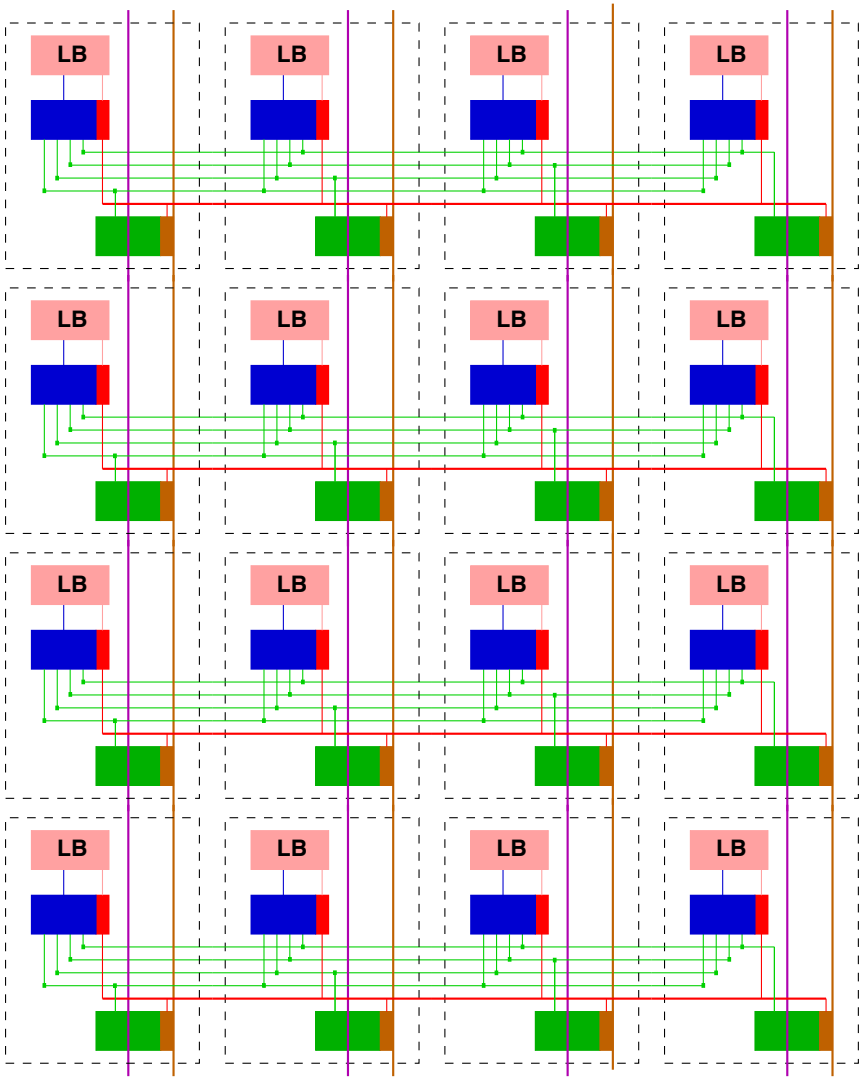


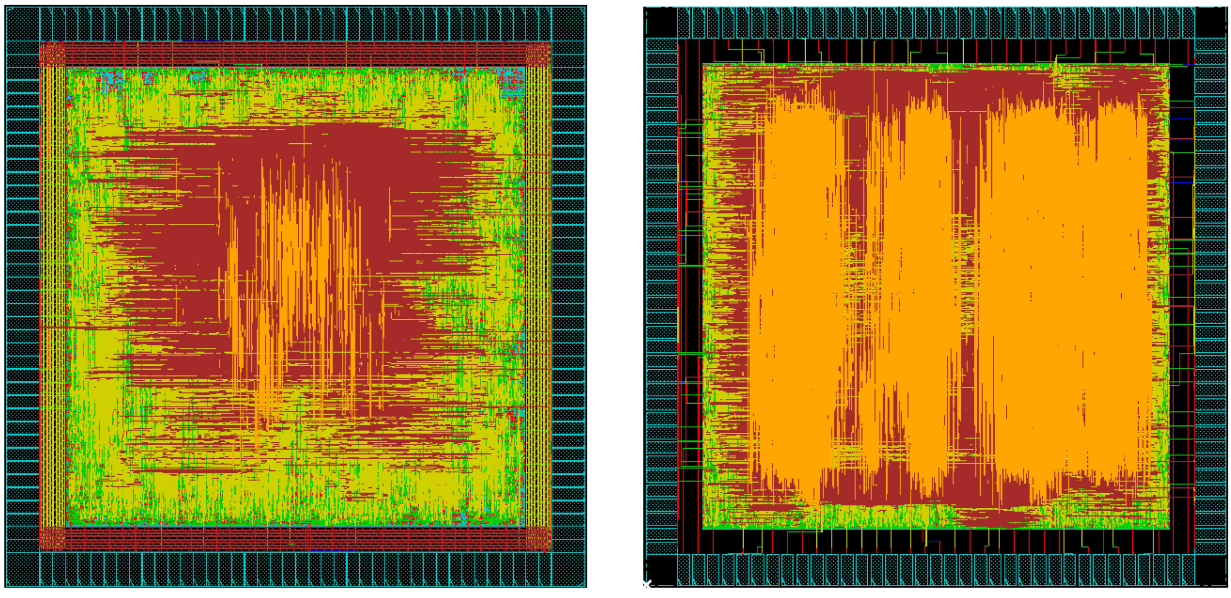
Figure 5.7: Rearranged 4x4 Tree-based MFPGA mapped in 2D

ity at each level and also vary the number of inputs/outputs of each cluster-level while maintaining the regularity of the rearranged structure. In order to decrease the number of tiles, we can enlarge the logic grain by placing an entire cluster of level1 inside every tile. Figure 5.9 shows the rearranged compact layout of the 8x8x8x4 tree based MFPGA architecture with the following parameters:
- level1 cluster: 8 LB, 24 inputs and 6 outputs ($p_{level1} = 0.86$).
- level2 cluster: 64 LB, 144 inputs and 36 outputs ($p_{level2} = 0.86$).
- level3 cluster: 512 LB, 864 inputs and 216 outputs ($p_{level3} = 0.86$).
- level4 : 2048 LB, 70 inputs and 54 outputs connected to the Core I/O pins (blue pads in he figure 5.9).

For clarity, we merge the downward and upward networks routing switches and channels in Figure 5.9. This figure shows also the basic tile which is replicated to form the overall rearranged MFPGA. The basic tile contains 8 LB, a set of switches of level2, level3 and level4. All tiles are by-products of the basic one. The I/O pads are connected to tiles on the board, i.e all tiles on the 4 sides of the MFPGA core. As shown in this figure, I/O pads can be connected to the upper level of the hierarchy (blue pads) or to lower level such Magenta pads that are connected directly to the 3rd level of the hierarchy.

Compared to an automatic placement of such architecture, the rearranged compact layout allows a more balanced routing congestion which is distributed across the overall chip area. Figure 5.8(a) shows the routing congestion using a full automatic placement done by Encounter [Cadence, 2006] and concentrated in the center of the chip. Figure 5.8(b) shows the routing congestion using the rearranged MFPGA pre-placement that uses more wires in the upper layers (Metal 6 and 7) due to the outbreak of MSB, being still more regular than the automatic placement approach. The proposed rearrangement uses more wires per tile, but this number of wires is constant, independently of the structure size since we replicate the same tiles. However other approaches don't allow such scalability, and give irregular structure with very congestioned regions that cannot be routed when we reach large sizes (4096 LBs and more).

We note that we use STmicroelectonics 0.12 standard cells library and we apply the Encounter automatic router for both approaches.



(a) (b)

Figure 5.8: Rearrangement effect on routing congestion and structure regularity

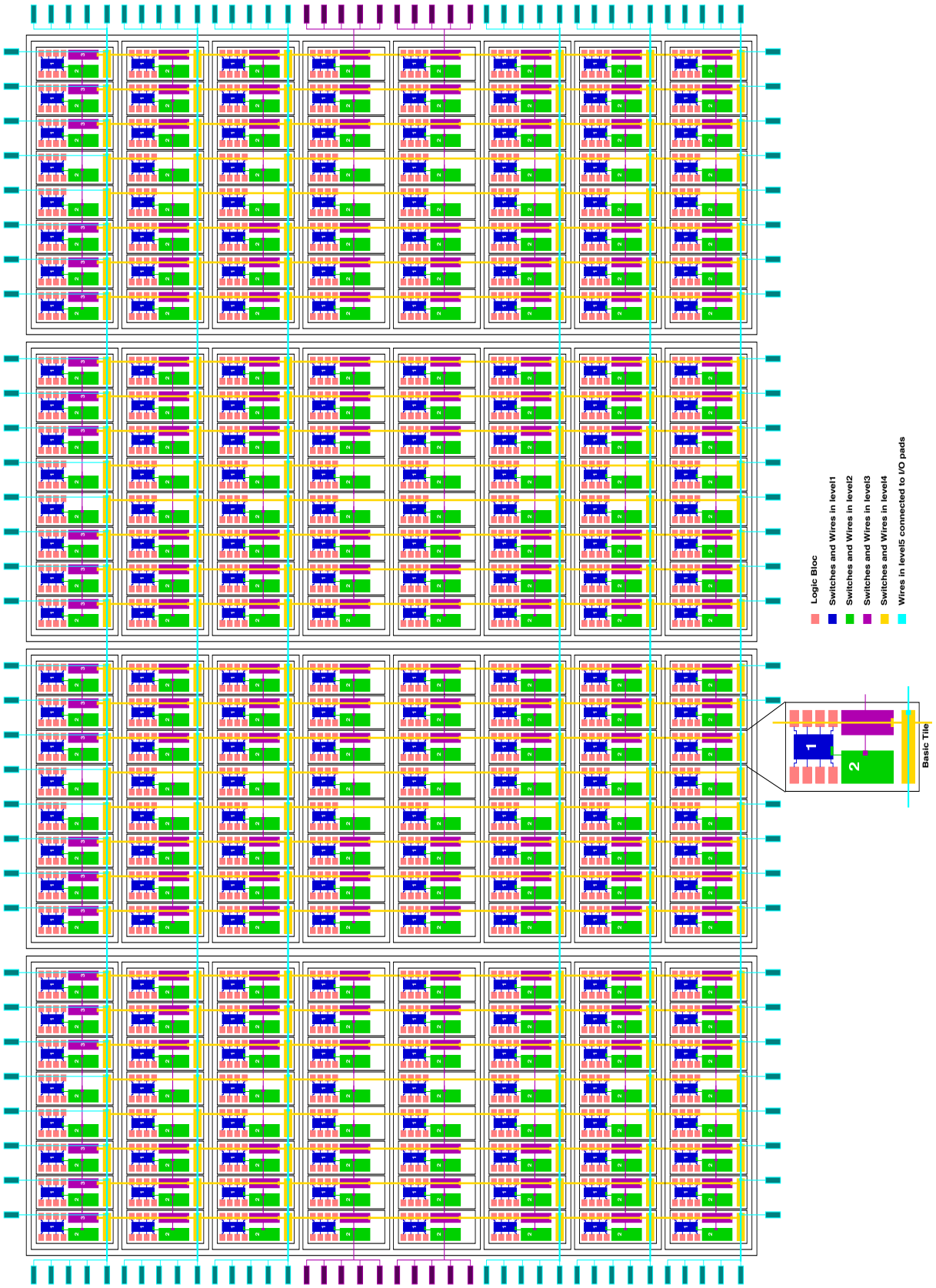


Figure 5.9: Rearranged 2048 nodes MFPGA layout: 8x8x8x4 architecture

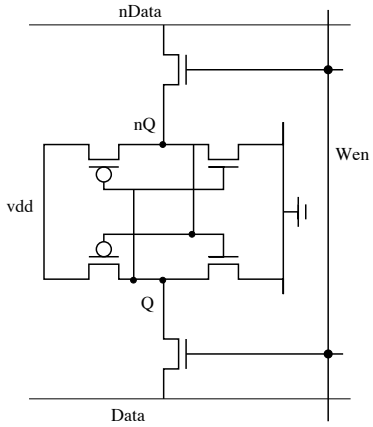


Figure 5.10: SRAM cell

5.5 MFPGA Full Custom Layout

5.5.1 4-LUT based logic block

The logic block (LB) structure in this MFPGA model is similar to the LB of the REDFPGA presented in chapter 3, but the transistor netlist and the layout is different since we target full custom design in this section. The LB consists basically in a 4-LUT, Flip-Flop and programmable bypass multiplexer that enables the sequential or the combinatorial mode within the LB. Each LUT uses 16 configuration bits and one multiplexer with 16 inputs as shown in figure 5.11. The output of the logic block is driven by a bypass 2-inputs multiplexer controled by an SRAM cell that uses either the Flip-flop output or the LUT output. This multiplexer has an impact on the functionnal delays.
Each logic block contains a Synchrone Flip-Flop used for sequentiel user-design. Figure 5.12 shows the layout mask of the complete logic block including: LUT4, sff1_x1, SRAM cells and the bypass multiplexer.

5.5.2 Progarmmable interconnect

All the interconnect in MFPGA is based on the scalable Mini Switch Box (MSB) which is basicly a set of multiplexers that share the same inputs as shown in figure 5.5. By changing MFPGA architecture parameters, we observe that for an arity included between 4 and 16, typical inputs number of the multiplexers varies between 3 and 12. Compared to decoded multiplexer for this range of inputs, "one-hot" coding multiplexer is smaller in terms of area.

As explained before we split all MSBs to build the rearranged MFPGA. An MSB is implemented using transmission gates with binary-encode controlled by SRAM cells. The basic element of the programmable interconnect is the programmable switch named Switch Point (SP). As shown in figure 5.13 the transmission gate of the SP is driven by

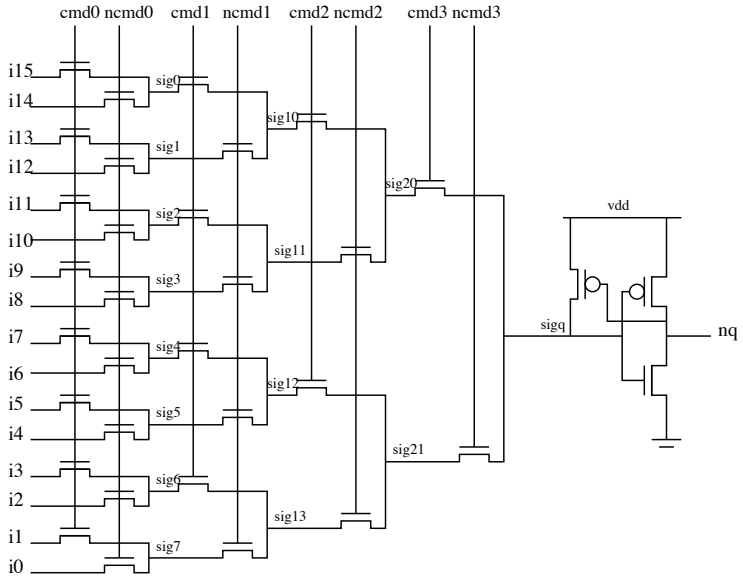


Figure 5.11: Look Up Table Multiplexer

an SRAM cell that is also inside SP. Using SP, we obtain a scalable multiplexer where the number of inputs can be increased simply by duplicating SP.

Figure 5.13 shows the basic SP module that puts together a transmission gate and its control SRAM cell. An SP cell is designed to be replicated easily to form a complete multiplexer or cross-bar with any chosen size. Figure 5.14 shows the layout of a 4-inputs multiplexer with different geometries and flattened layout views of vertical and horizontal multiplexers built by using the same basic switch element. During the configuration step, the delay in the critical path is equal to the time required to latch the data in the SRAM cell. During the functional mode, the input signal goes through only one SP. When compared to a standard multiplexer realization (tree of pass transistors), this situation improves overall FPGA performance.

5.5.3 Physical placement and routing

As shown in figure 5.15, the basic tile of the MFPGA contains LBs and SP from every interconnect level. Each level is represented by a local Sparse Cross Bar (SCB) composed of SP and outputs buffers. SCBs inside the tile are connected using short local routing wires. Connections with other clusters use unidirectional horizontal and vertical global routing wires. As shown in figure 5.9, wires of level3, level4, level5 driven by SCBs in level3, level4 and I/O pads respectively, span (cross) 8, 8 and 32 tiles respectively which correpond to $k2$, $k3$ and $k4 * k2$ respectively (k2,k3 and k4 are arities of level2, level3 and level4 respectively).

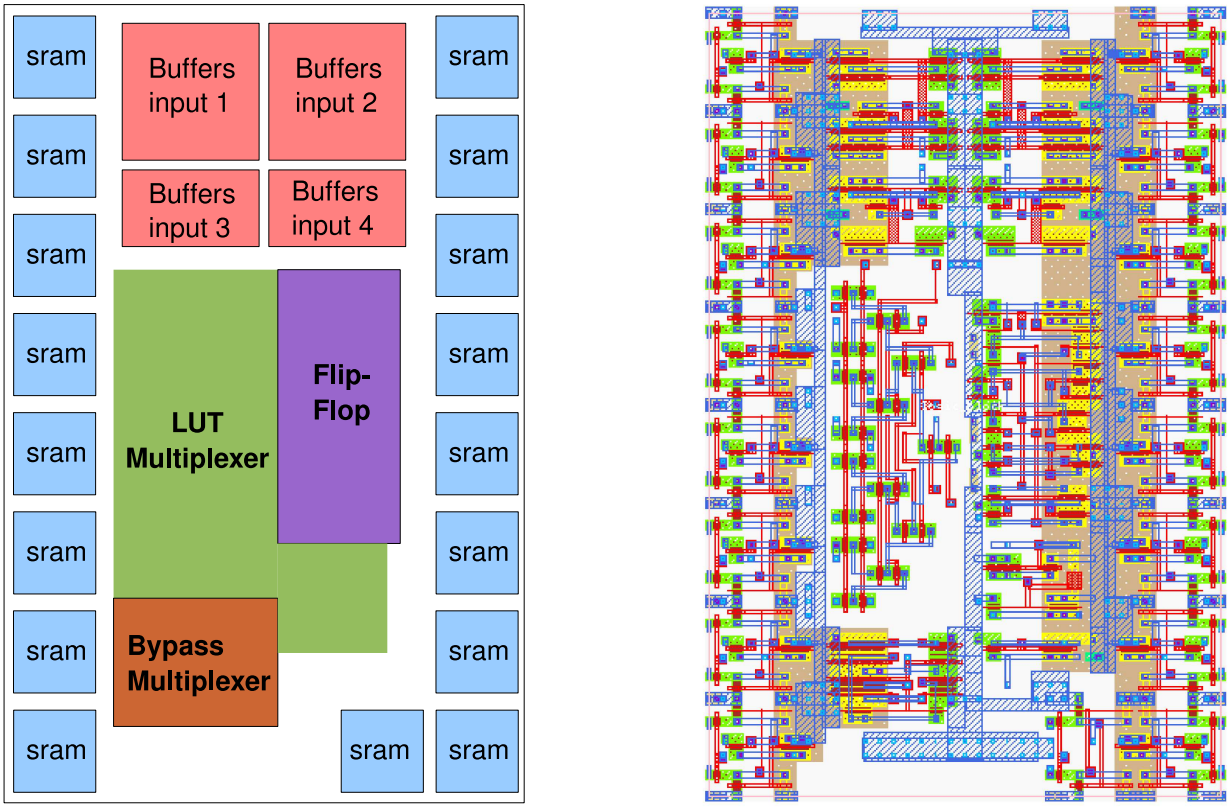


Figure 5.12: Real Logic Block: 4-LUT + Flip-Flop + bypass Multiplexer + 17 SRAM cells

The switches and their corresponding SRAM bits determine the area required by the SCB since the input and output wires can simply be brought in on metal layers above these devices. In order to achieve a compact layout, therefore, it is desirable to pack the switches as closely as possible. Since we don't use a full crossbar, there are few switches per input line, and this causes the input lines to be more closely packed. Jogs are necessary to connect input lines to some of their proper switches as shown in figure 5.16. This figure shows a real example of the sparse cross bar layout that drives 1 LB in the MFPGA and where switches are placed in a very regular manner in order to obtain good routability.

We have chosen a layout topology that restricts us to 6 or fewer wire pitches per layer per SP in both vertical and horizontal direction. This mean that any interconnect requiring more than 6 jogs or wire pitches per SP forces us to use additional metal layer or to spread out our switches in the east-west direction, wasting silicon area in our final layout.

Figure 5.17 shows the abutment of 4 basic tiles to build the rearranged $Part_{level2}$. This figure shows also vertical and horizontal wires that connect all tiles. Figures 5.18

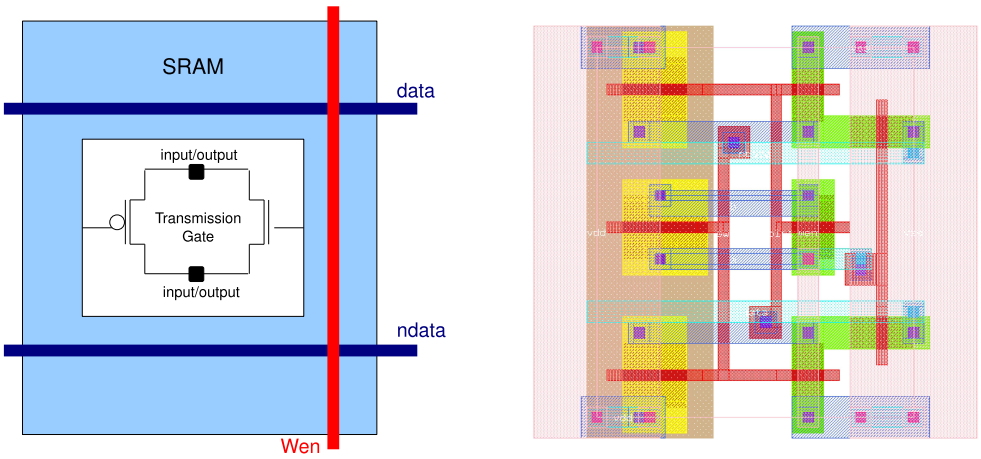


Figure 5.13: Switch Point Layout

and 5.19 shows a 2048 LBs MFPGA congestion map and layout view respectively. This MFPGA corresponds to the 4-levels architecture(8x8x8x4) described in section 5.4 and figure 5.9. The layout generation is running in CORIOLIS's layout environment with a symbolic grid, and uses Python routine that was written for placement and routing. The layout is designed with cells library presented above, and uses 6 metal layers.

5.5.4 Configuration Storage and Distribution

The abutment of LB and SP cells generates a regular array of memory cells. The configuration bits are grouped as words that can be programmed selectively. This allows programming only the ressources that are needed. The size and the number of word per array can be varied to obtain the needed SRAM storage block. The selective adressing technique requires circuitry for decoding, and imposes the routing of configuration data and address buses through the entire array. Configuration data and adresses buses are routed by abutment of SP and LB cells in a manner similar to RAM structure.

5.6 Timing analysis

5.6.1 Delay Model

The delay through the routing network may easily be dominant in a programmable technology. The 2 following factors are significant in this respect:

- Wires delay:
  Delay on a wire is proportional to its length and capacitive loading (fanout).

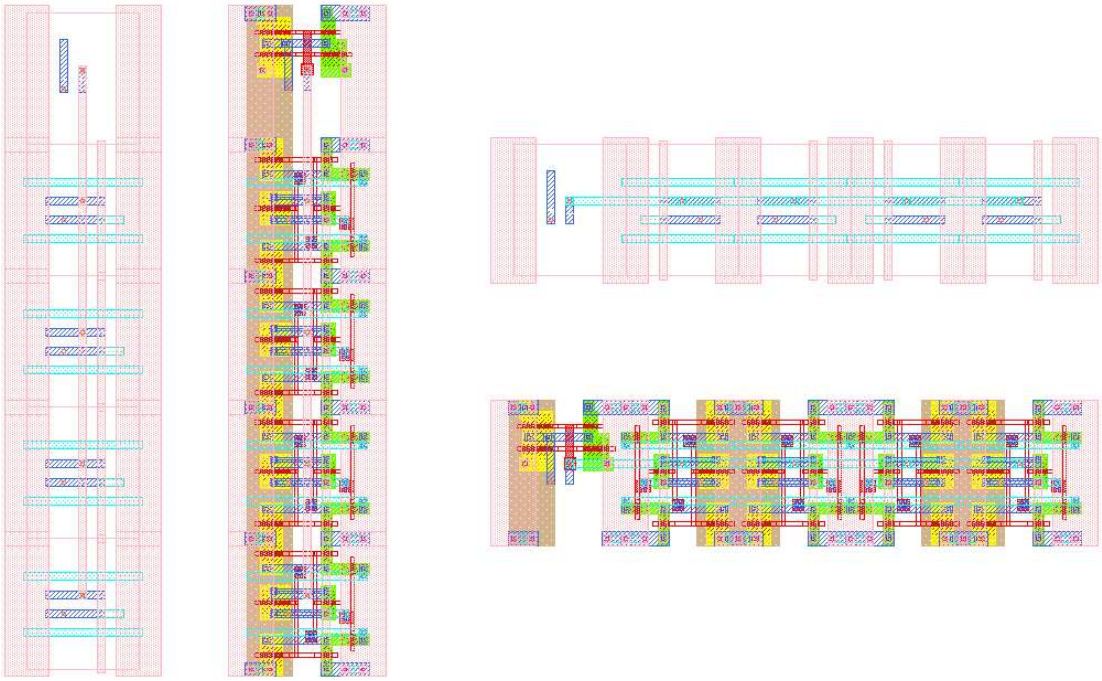


Figure 5.14: Vertical and Horizontal 4:1 Multiplexers

- Switches delay:
  Each programmable switches (multiplexer) in a path adds delay. This delay is generally larger than wires delay.

A path connecting a source to a sink in MFPGA consists in going from a source up to a particular level and then down to the sink. We propose to divide a path into several sub-paths. Each sub-path connects two sucessive Multiplexers as shown in figure 5.20. The number of crossed sub-paths depends on the number of levels.

Targeting the basic cells library in 0.12μ STmicroelectronics process, we obtain highly accurate delay estimation for SP-based multiplexers and LB using the SPICE circuit simulator. We note that every multiplexer output drives one input of the succesor multiplexer. Due to the MFPGA topology, every downward multiplexer (Dmux) in a level i has a fanout of $ki - 1$ and every upward multiplexer (Umux) in a level i has a fanout of $k_i + k_{i+1}$ (k_i is the tree arity in a level i).
Thus sub-path delay variations in all LB-Umux couples or Umux-Dmux or Dmux-Dmux couples depend on wire length between the 2 concerned modules. The delay model was created by running SPICE simulations on a variety of MFPGA branches including basic cells and wires.

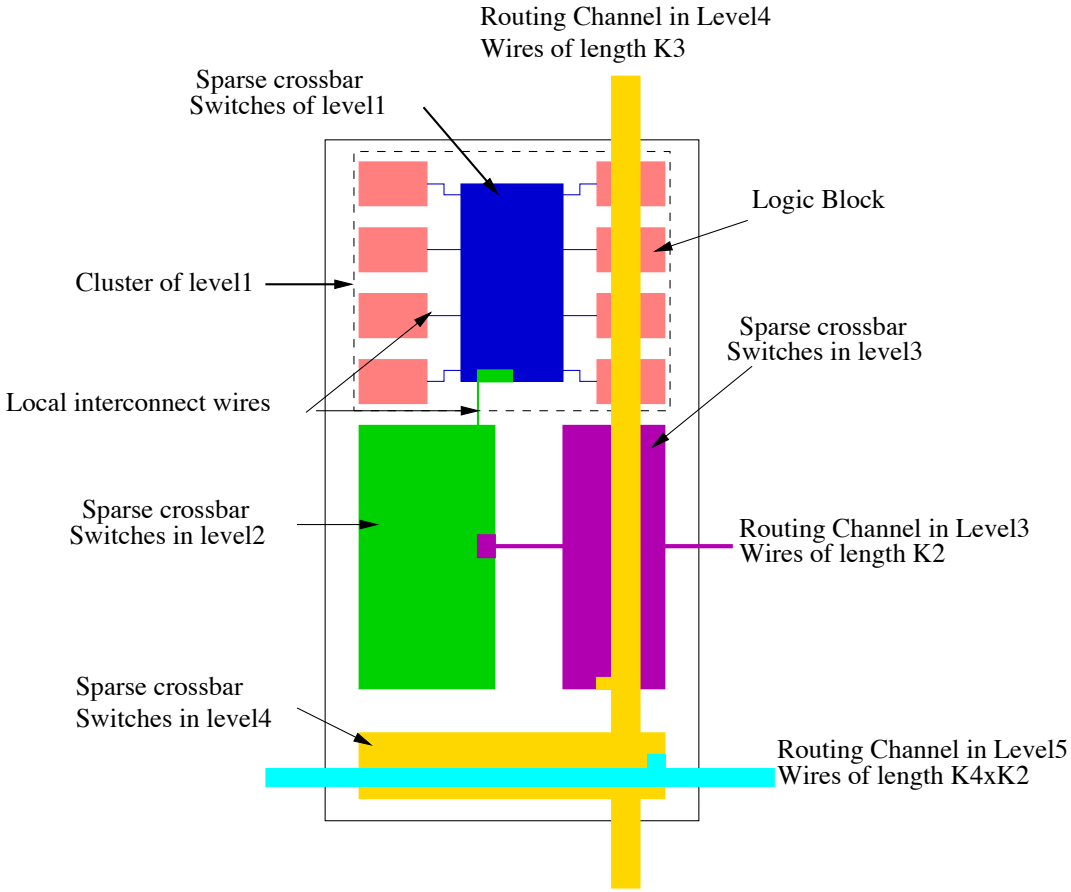


Figure 5.15: MFPGA Basic tile topology

5.6.2 Critical path extraction and speed performances

Once the circuit has been placed and routed we obtain a direct graph called "routing graph". This graph describes wires that are used to connect LB pins as described in the netlist. Wires and LB pins represent the "routing graph" nodes. Programmable passing switches become directed edges. Edges are also added between LB inputs and outputs, this corresponds for example to the combinational paths through LUT inside LB. Figure 5.21 shows a simple circuit implemented via 2-input LUTs and registers, and the corresponding "routing graph". On this graph we can isolate easily different sub-paths through a depth-first traversal. We replace each sub-path by only one edge labeled with the sub-path delay. We obtain a new direct acyclic graph called "timing graph". In this graph nodes represent the input and output pins of basic circuit elements, such as registers and LUTs. Register input and output are not linked, there are no edges incident to output pin and no edges leaving the input pin (acyclic graph). Similarly, primary inputs (input pads) have no incident edges and primary outputs (out pads) have no exit edges.

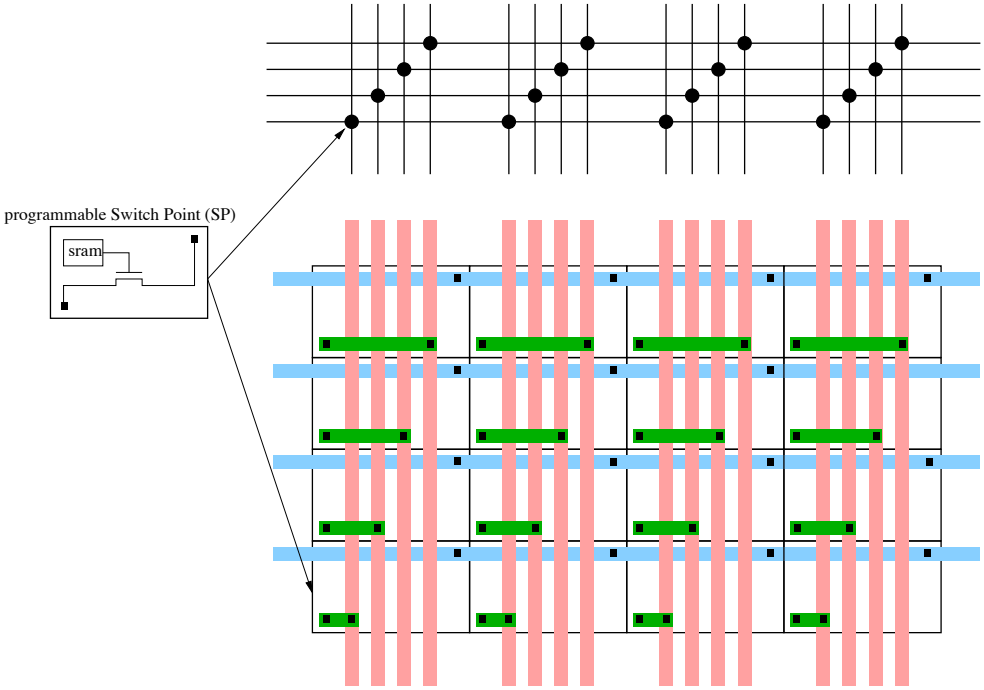


Figure 5.16: Compact Layout of a sparce cross bar: buffers are not presented

Every edge is labeled with the corresponding sub-path delay required to pass through circuit element or routing. Figure 5.21 shows also the generated "timing graph" of the routed circuit.
An automatic timing analyzer tool can determine the minimum required clock period with $O(n)$ computation for a "timing graph" with n nodes via a breadth-first traversal. This traversal begins at nodes with no incident edges (primary inputs and register outputs) and labels each one with a signal arrival time, $T_{arrival}$, of 0. Each node which has incident edges from previously labeled nodes is then labeled with its arrival time according to:

$$T_{arrival}(i) = \max_{j \in fanin(i)} \{T_{arrival}(j) + delay(j,i)\}$$

where node i is the node being labeled, and $delay(j,i)$ is the delay value marked on the edge joining node j to node i. This procedure continues until labeling all nodes in the graph. Primary output or register input node with the largest arrival time defines the maximum delay D_{max} (= minimum clock period), through the circuit. "Timing graph" in figure 5.21, shows an arrival time at node Reg equal to 5.5 ns, this represents the largest arrival time, and hence the maximum circuit delay.

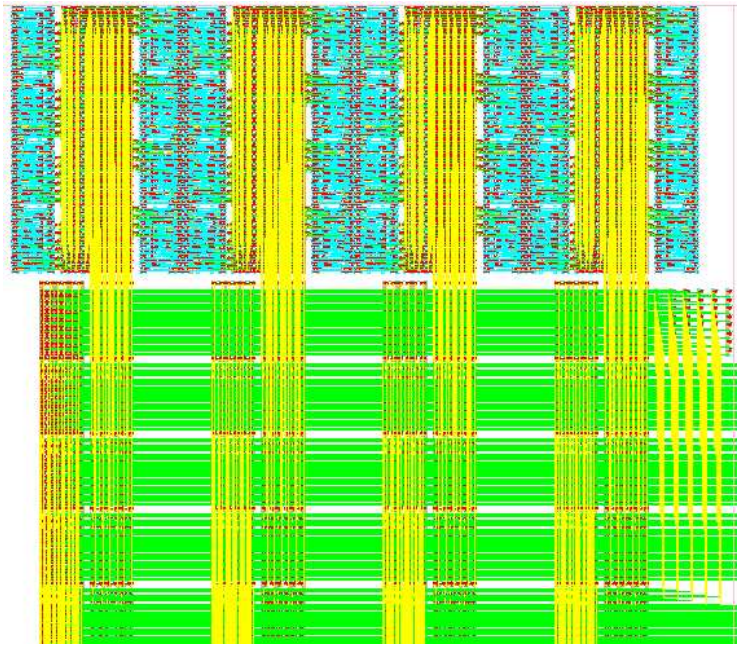


Figure 5.17: The created rearranged $Part_{level2}$ of the MFPGA layout

5.6.3 Speed performances

Previous comparisons shows that the MFPGA architecture is more efficient than Mesh in terms of area and this could be a positive effect on circuit speed: the smaller the area, the shorter the connecting wires. In addition the speed of a net is determined by the number of routing switches it must cross. In a Mesh structure, the number of wires segments and switches in series between LBs increases linearly with the Manhattan distance. However in a Tree topology the number of wires segments and switches in series between LBs grows as a logarithmic function of the Manhattan distance.
We compared the speed of the MFPGA architecture to the Mesh. We implemented the same circuits and we used our timing analyzing tool for the MFPGA and the one proposed in VPR for the Mesh (note: we applied a VPR timing-driven placement and routing). Timing results are presented in table 5.1. In this comparison we only used small benches (< 1024 BLEs).

We notice that MFPGA largely outperforms clustered Mesh architecture (40%) in terms of speed despite we did not integrate timing driven techniques yet. Nevertheless, we expect that the gain ratio will decrease in the case of larger benchmark circuits.

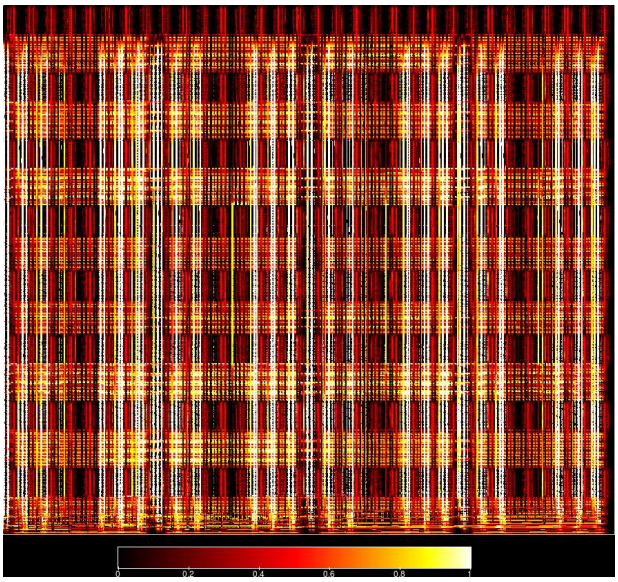


Figure 5.18: Layout congestion map of 2048 LBs MFPGA

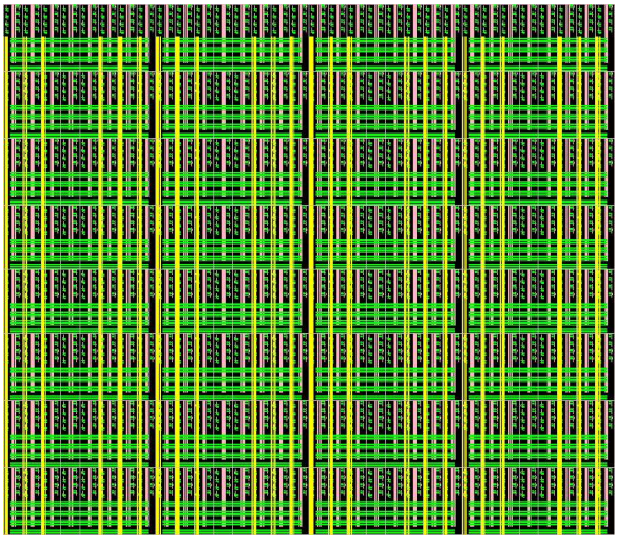


Figure 5.19: 6 metal layers 2048 LBs MFPGA full custom layout

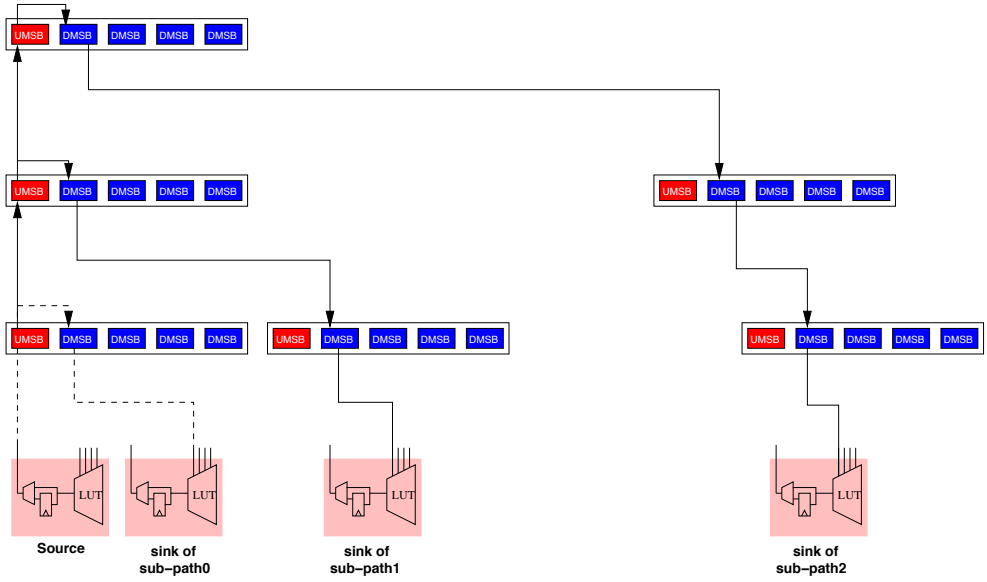


Figure 5.20: Sub-paths timing caracterisation

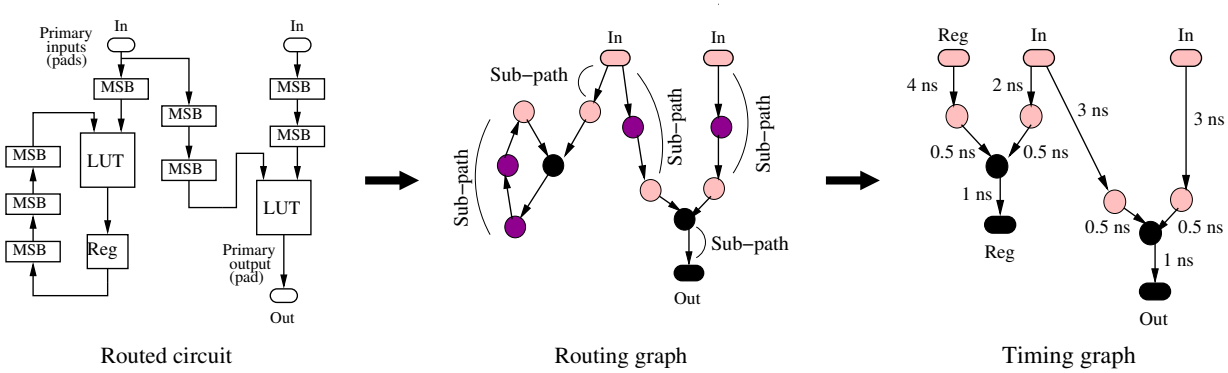


Figure 5.21: Timing graph modeling of a simple circuit

5.7 The area gap between MFPGA and ASIC

In this section we measure the area gap between MFPGA and standard cell application-specific integrated circuits (ASICs). We chose the standard cells because that approach is currently the dominant choice in ASIC implementations. This comparaison is performed by implementing a range of benchmark circuits using the same IC fabrication process geometry which is STMicroelectronic's CMOS 120 nm. For the standard cell we use STmicroelectronics cell libraries optimized for density provided by CMP (http://cmp.imag.fr/). For the MFPGA we use the cells presented above and converted to the same process (ST 120 nm).

Circuit	LUTs	Mesh T(ns)	MFPGA T(ns)
pcle	29	6.1	4.34
decod	32	5.36	3.56
cc	33	6.5	4.25
count	37	8.55	7.78
b9	61	10.77	6.98
i4	110	11.14	6.03
c2670	363	19.95	9.97
i9	471	15.67	10.90
alu4	584	20.9	11.26
C5315	725	24.63	15.17
Average	245	12.95	8.02

Table 5.1: Speed Comparison ($0.12\mu m$ CMOS, $1.2V$)

We select a variety of benchmarks (Verilog or VHDL) from the list descibed in section 4.5.4 and table 4.2. The only factor that is considered in benchmarks selection was ensuring that the design doesn't contain multiplication operations (example "*diffeq*"). Our MFPGA doesn't integrate hard structures like memories or multipliers blocks, it would be agressively penalized if we used LUTs and Flip-flops to map such blocks.
To make sure that RTL is synthesized similarly for both targets we use the same synthesis tool Synopsis design Compiler [Synopsys, 2006] to get gate level netlist, then we use SIS for FPGA mapping. MFPGA softwares presented in figure 4.14 are used for partitionning, place and route. Cadence SOC Encounter [Cadence, 2006] is used for standard cell place&route and area evaluation.
The area of the standard cells implementation is simply the final core area of the placed and routed design. For the MFPGA, the area is calculated using the actual silicon area of MFPGA that can implement the benchmark design, this includes all resources of the MFPGA.

To avoid disclosing any proprietary information, no absolute areas will be reported in this work. The area gap between MFPGAs and ASICs is reported as the ratio of the MFPGA area to ASIC area. The measurement methodology described above was applied to each of the benchmarks and results are shown in table 5.2. The example of *b22* circuit shows that area of tree-based MFPGA able to implement it, is 39 times larger than its optimized standard cell implementation. We observe that the ratio of slicon area required to implement a circuit in tree-based MFPGA designed on symbolic layout and optimized standard cell ASIC is on average 51.

The approach used in [I.Kuon and J.Rose, 2007] is more optimistic for the MFPGA and consits in calculating only the silicon area of resources used by the design. This means that they take only the area of blocks covered by the implemented circuit de-

ITC benchmarks	Gap ratio MFPGA/ASIC
b14	46
b15	57
b17	55
b21	59
b22	39
Average	51

Table 5.2: Area Ratio MFPGA/ASIC

sign. This includes the area of routing ressource surrounding the used LBs where the entire area of the cluster is used regardless of whether only a portion of the cluster is used. The candidate FPGA used in the work done by Kuon et al. is similar to the Xilinx Virtex-E, a relatively modern commercial architecture. Their results show that circuits implemented entirely using LUT and Flip-Flops, an FPGA is on average 40 times larger than a standard cell implementation. This approach ignores the fact that FPGAs, unlike ASICs, are not available in arbitrary sizes. A designer must select one particular discrete size even if it is larger than required for the design. While this is an important factor, we can focus on the cost of programmable fabric itself; therefore, we ignore any area wasted due to the discrete nature of FPGA devices. In addition we can increase the MFPGA density by a factor of 30% if we target a real layout grid instead of the actual symbolic one which penalizes the density. Using optimistic approaches, tree-based MFPGA architectures would give better results than the Virtex-E like architecture. We believe we can reduce to 25 the gap ratio between tree-based MFPGA and ASIC.

5.8 Conclusion

This chapter describes floorplanning methods used to create the layout of tree based MFPGA architecture. We introduced a switch placement technique to rearrange the MFPGA interconnect resources and balance the routing congestion over the entire surface of the MFPGA chip. We confirmed that the 2D rearranged tree-based MFPGA is a regular and scalable structure that uses a hierarchical interconnect with short-local and long-global routing wires.

We develop the layout of the basic cells library to build the MFPGA and we introduce the incorporation of sparse crossbar into our MFPGA architecture. We used a switches soothing organisation to compact fully the crossbar layout, hence the MFPGA layout.

This chapter also presented measurements quantifying the gap between MFPGAs and ASICs. We observed that MFPGA is on average 51 times larger than a standard cell

implementation. We note that the MFPGA layout presented in this work uses a symbolic grid in order to allow portabiliy for different processes. The use of real processes rules enables a substantial reduction of the layout area, thus we can achieve higher density for the MFPGA.

Conclusion

1 Summary of contributions

This dissertation has presented methodologies to improve FPGA design position in the IC industry. We investigated layout techniques to automate Mesh based FPGA in the first part, then in a second part we explored a topology and physical design of a new interconnect architecture that may prove to be important in FPGA performances enhancements. The following remarks were retained along various explorations.

1.1 Automating layout generation of specific Mesh-based FPGA

Technology scaling has brought IC industry to the point where several distinct components can be integrated into a single chip. Many IC designs can benefit from inclusion of programmable logic on the silicon die, as it can add general computing ability, provide run-time reconfigurability, or can be used for post-manufacturing upgrades. Moreover, by tailoring the reconfigurable fabric to specific domains, additional area/delay/power gains can be achieved over current, more general fabrics. We deal with the automatic design of domain-specific and embedded reconfigurable fabrics based on Mesh topology.

In chapter 2 we show how we can reduce significantly the time-to-market of a specific FPGA fabric by automating his transistor level layout design, using standard cells while maintaining the structure regularity. We presented a total flow to automate process with the adequate CAD flow, allowing designers to create their own FPGA fabric. We extend academia LIP6 CAD tools such as ALLIANCE and CORIOLIS platforms [Alliance, 2006, C.Alexandre et al., 2005] to take into account physical layout constraints in the context of FPGA-architecture, as well as the automated FPGA layout procedure. We developed a technology-independent layout generator which is easily adapted to any standard cell library geometry and to any process rules. Using the ALLIANCE symbolic standard cell library, we show how we can produce automatically a large spectrum of SRAM Mesh-based FPGA cores and chips with different architectural parameters such as routing channel width and array size. We show how alleviating the FPGA core design costs by automating the physical design process.
Chapter 3 presents a proof of the concept presented in chapter 2 by generating a specific

SRAM Mesh-based Redundant FPGA chip (REDFPGA) which uses a robust routing interconnect and a fully random access to the configuration memory. This memory integrates a hardware error-detector system for SEU mitigation. Compared to the ordinary solutions of configuration bits reliability such as TMR, the proposed approach achieves higher level of reliability with only 19% increase in the FPGA total area. The design was successfully migrated and taped out in 0.12 um 6-metal layer CMOS process from ST. To configure the REDFPGA chip we use only open-source software platform adapted to this architecture, including synthesis, mapping, place&route and bitstream generation tools.

1.2 Multilevel Hierarchical FPGA architectures

Mesh is the most common architecture in academic and industial fields. Much research effort was deployed to improve it in terms of area, speed and power dissipation. Modern Mesh based architectures have at least two level of hierarchy with clustered logic blocks, different wire lengths and well adapted CAD tools to optimize circuit implementation. Despite its good properties, Tree-based architecture has been overlooked up to now; this is due to its physical design complexities and its scalability constraints.

In chapter 4 we propose a Butterfly Fat Tree (BFT) based FPGA architecture (MFPGA) and optimize it in terms of density and routability. Our approach is based on balancing interconnect and logic utilization in order to use more efficiently the most dominant resource which is interconnect. The major features we used to depopulate interconnect are network topology and clusters signal bandwidths. Then we compared the resulting MFPGA architecture to Mesh-based topology and show how we can achieve more area efficient designs. To evaluate and compare different architectures we used an adaptive configuration tools flow. In order to experiment and quantify the assets of several architecture parameters, we use MCNC, ITC and Opencores circuits that contain only lookup-tables and flip-flops. These benchmark circuits were placed and routed; required area is evaluated by switches count and cells area sum. We showed that with such architecture we can achieve an average gain of 54% in terms of area, compared to Mesh-based architecture. Despite increasing logic block area by 20% we reduced the total area more than twice thanks to a good balancing between logic and interconnect resources. We achieved more area efficient designs by allowing some LUTs to go unused and by efficiently exploiting interconnect resources which accounts only for 75% of the total area in the case of MFPGA. We see that high LUT use does not imply lower area and that LUT usability is not always directly correlated to area efficiency.

Finally in chapter 5, to demonstrate the concept, we presented a physical floorplanning technique to lay out the BFT-based architecture using modern VLSI multilayer processes. We showed that MFPGA is not a wire dominant architecture, and given a sufficient wiring layer, it can be laid out compactly to fit a 2D area corresponding to the active area dedicated to logic blocks and switches.

2 Future work

In this work, we explored many of the parameters associated with designing Tree-based MFPGA networks but many more design parameters deserve additional study. We limited this study to logic blocks holding a simple LUT; it should be interesting to see how MFPGA interacts with heteregeneous leaves including DSP and Memories macro blocks. We expect that larger benchmarks mixing LUT and macro blocks will better demonstrate the efficiency and the scalabiliy of this architecture. A more careful review of timing and power consumption effects would also be benificial.

Stimulated directions have emerged from our exploration. In particular the following points seem promising.

2.1 Tree-based MFPGA architecture improvements

The aim of this work around tree-based MFPGA was to optimize interconnect in order to reduce area and to improve performances. Interconnect is not the only factor that effects FPGA performances, logic modules also influence density, speed and power. Today's all industrial FPGA contain an ever increasing large number of hard macro blocks such as DSP, RAM and sometimes Microprocessors cores. Including this macro blocks and varying the logic block topology in a tree-based MFPGA deserves advanced exploration. Figure 1 shows what may resemble an heterogeneous MFPGA including fine and coarse grain logic modules.

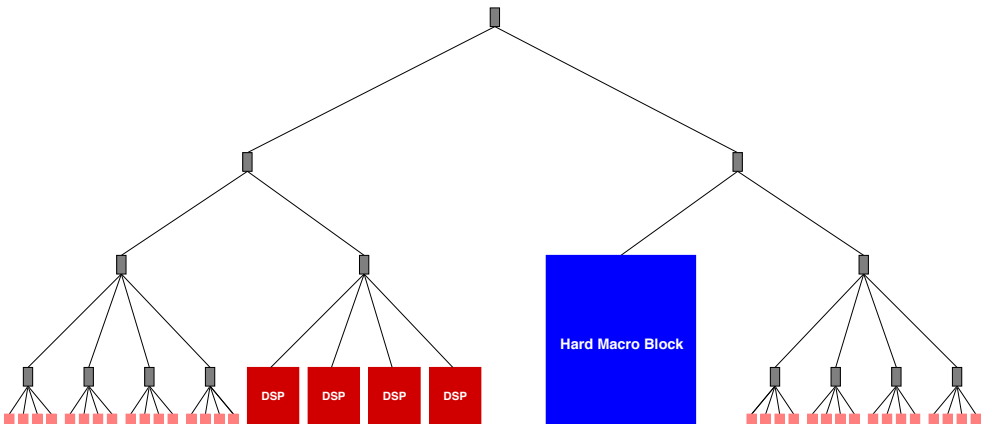


Figure 1: Coarse grained Tree based MFPGA

The upward network has an important impact on MFPGA routability and performances. UMSB offers high flexibility to reach upper levels or go back to the same clusters using local feedbacks. The disadvantage is that to connect 2 leaves of the same cluster, the concerned net must cross 2 switches at least, which is expensive in terms of performance. An intersting alternative is to use feedbacks driven by the leaves and connected directly to the downward network and without crossing UMSB. These feedbacks

can reach the top level wihout using UMSB. Figure 2 shows an example of 4-level tree based MFPGA with direct feedbacks driven by LB outputs and connected to all levels of the hierarchy. In this figure, direct feedbacks look as long wires, but if we look to the rearranged layout, these feedbacks are really short and rapid wires as shown in figure 2. Thus finally to go for example from LB to the top level we use a short wire and we save 4 switches compared to the original solution. In the same way other alternatives of direct connections between neighbours tiles and LBs that are physically adjacents should be explored. We mention for example the carry chain for arithmetic circuits.

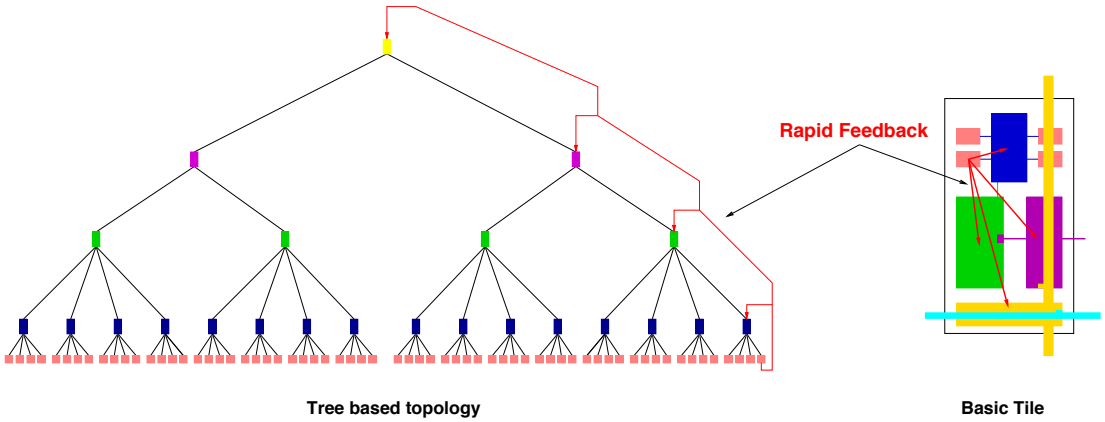


Figure 2: Rapid connection for the upward network

2.2 Delay and power models

Layout generation is very important to get accurate timing and power consumption characteristics. Number of switches in a critical path or the activity of a path is easy to determine and can be extracted from routing and simulation tools respectively, however the wiring effects are linked to the physical layout. Wire lengh depends on physical layout that changes for every architecture parameter change (Arity 'k', Rent parameters 'p', Architecture size 'N').

In order to study these characteristics with the variation of the architecture parameters we can develop a generic model that allows us to estimate wire length without developping a complete layout. The rearranged layout explored in this work represents a high regularity and scalability which is linked to the architecture parameters, thus we can develop a model to determine wiring constraints. Using cells library and wires electric characteristics we can estimate delay and power consumption.

2.3 Large tree-based FPGA

New deep-submicron semiconductor technologies (65nm, 45 nm etc.) involve smaller transistors and wires. This makes transistors faster and wires get slower [E.Lee et al., 2006]. Long interconnect wires behave like an RC transmission line where delay grows quadratically with length.
Considering Tree-based Layout, we showed that by increasing the tree size, wires becomes longer for every additional level of the hierarchy. This presents a serious limit for large hierarchical FPGA. Evaluation based on MFPGA layout generated on 120 nm technology show that in the case of architectures larger than 8192 LBs, wire delays in the upper level became critical and dominant compared to switches delays. In this case it becomes essential to decrease all these wires to reduce the quadratic delays growth. It would be beneficial to move towards Mesh structure from this break-even point to reduce wires delays. We propose to build a Mesh of Tree architecture where the basic cluster corresponds to some largest effective MFPGA. In this way wires lengths in the highest levels of the architecture depend only on Mesh cluster size. The 3D layout is also an interseting alternativ for large tree-based FPGA where long wires in the upper level of the tree would be converted to TSV (Through Silicon Via).

List of Publications

Some of the following publications can be downloaded from:
http://www-asim.lip6.fr/publications/

Journal papers

- FPGA Interconnect Topologies Exploration
  Zied Marrakchi, Hayder Mrabet, Umer Farooq, and Habib Mehrez.
  Accepted in the International Journal of Reconfigurable Computing (IJRC)
  http://www.hindawi.com/journals/ijrc/aip.html

- Performances Comparison between Multilevel Hierarchical and Mesh FPGA Interconnects
  Marrakchi Zied, Mrabet Hayder, Masson Christian, Mehrez Habib
  International Journal of Electronics, Jan. 2008, vol. 95, num. 3, pp. 275-289, Taylor and Francis

Refereed papers in international conferences

- Automatic Layout Generator of Domain Specific FPGA
  Mrabet Hayder, Zied Marrkachi, André Tissot, Mehrez Habib
  IEEE International Conference on Microelectronics (ICM 2008), Sharjah, UAE, December 2008

- Optimized Local Interconnect for Cluster-based Mesh FPGA Architecture
  Marrakchi Zied, Mrabet Hayder, Mehrez Habib
  IEEE International Conference on Microelectronics(ICM'2008), Sharjah, UAE, December 2008

- The Effect of LUT and Cluster Size on a Tree Based FPGA Architecture
  Umer Farooq , Zied Marrakchi , Hayder Mrabet and Habib Mehrez
  International Conference on Reconfigurable Computing and FPGAs(ReConfig'08), Cancum, Mexico, December 2008, pp. 115-120

- Efficient Tree Topology for FPGA Interconnect Network
  Marrakchi Zied, Mrabet Hayder, Amouri Emna, Mehrez Habib
  ACM Great Lakes Symposium on VLSI (GLSVLSI'2008), Orlando, Florida, USA, May 2008, pp. 321-326

- Efficient Mesh of Tree Interconnect for FPGA Architecture
  Marrakchi Zied, Mrabet Hayder, Masson Christian, Mehrez Habib
  International Conference on Field-Programmable Technology (ICFPT'2007), Kitakyushu, JAPAN, December 2007, pp. 269-272

- Mesh of Tree: Unifying Mesh and MFPGA for Better Device Performances
  Marrakchi Zied, Mrabet Hayder, Masson Christian, Mehrez Habib
  1st ACM/IEEE International Symposium on Networks-on-Chip (NoC'2007), Princeton, USA, May 2007, pp. 243-252

- Implementation of Scalable Embedded FPGA for SOC
  Mrabet Hayder, Marrakchi Zied, Mehrez Habib, Tissot Andre
  IEEE International Conference on Design & Test of Integrated Systems in Nanoscale Technology (DTIS'2006), Tunis, Tunisia, September 2006, pp. 74-77

- Performances Improvement of FPGA using Novel Multilevel Hierarchical Interconnection Structure
  Mrabet Hayder, Marrakchi Zied, Souillot Pierre, Mehrez Habib
  IEEE/ACM International Conference on Computer-Aided Design (ICCAD'2006), San Jose, California, USA, November 2006, pp. 675-679

- Evaluation of Hierarchical FPGA partitioning methodologies based on architecture Rent Parameter
  Marrakchi Zied, Mrabet Hayder, Mehrez Habib
  2nd IEEE Conference on Ph.D. Research in MicroElectronics and Electronics (PRIME'2006), Otranto, Italy, June 2006, pp. 85-88

- A new Multilevel Hierarchical MFPGA and its suitable configuration tools
  Marrakchi Zied, Mrabet Hayder, Mehrez Habib
  IEEE Computer Society Annual Symposium on Emerging VLSI (ISVLSI'2006), Technologies and Architectures, Karlsruhe, Germany, March 2006, pp. 263-268

- Hierarchical FPGA clustering based on multilevel partitioning approach to improve routability and reduce power dissipation
  Marrakchi Zied, Mrabet Hayder, Mehrez Habib
  International Conference on Reconfigurable Computing and FPGAs (ReConFig'2005), Puebla city, Mexico, September 2005

- Hierarchical FPGA clustering to improve routability
  Marrakchi Zied, Mrabet Hayder, Mehrez Habib

IEEE Conference on Ph.D. Research in MicroElectronics and Electronics (PRIME'2005), Lausanne, Switzerland, July 2005, pp. 139-142

- Automatic Layout of Scalable Embedded Field Programmable Gate Array
  Mrabet Hayder, Marrakchi Zied, Mehrez Habib, Tissot André
  International Conference on Electrical Electronic and Computer Engineering (ICEEC'2004), Cairo, Egypt, September 2004, pp. 469-472

Refereed Presentations at Workshops

- FPGA Interconnect Topologies Exploration
  Zied Marrakchi, Hayder Mrabet, Umer Farooq, and Habib Mehrez.
  Reconfigurable Communication-centric SoCs (ReCoSoC'2008), Barcelona, Spain, July 2008

- Performance improvement of FPGA using novel multilevel hierarchical interconnection structure I
  Mrabet Hayder, Marrakchi Zied, Souillot Pierre, Mehrez Habib
  Reconfigurable Communication-centric SoCs (ReCoSoC'2006), Montpellier, France, July 2006

Posters

- A Routability Driven Partitioning and Detailed Placement Approach for Multilevel Hierarchical FPGA
  Marrakchi Zied, Mrabet Hayder, Souffleteau Gregory, Masson Christian, Mehrez Habib
  ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'2007), Monterey, Californie, USA, February 2007, pp. 225-225

- A multilevel hierarchical interconnection structure for FPGA
  Mrabet Hayder, Marrakchi Zied, Souillot Pierre, Mehrez Habib
  14th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'2006), Monterey, California, USA, February 2006, pp. 225

- Configuration tools for a new multilevel hierarchical FPGA
  Marrakchi Zied, Mrabet Hayder, Mehrez Habib
  14th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'2006), Monterey, California, USA, February 2006, pp. 229

- Implementation of Scalable Embedded FPGA for SOC
  Mrabet Hayder, Marrakchi Zied, Mehrez Habib, Tissot Andre

Reconfigurable Communication-centric SoCs (ReCoSoC'2005), Montpellier, France, June 2005. ISBN 2-9517-4611-3.

Bibliography

[A.A.Agarwal and D.Lewis, 1994] A.A.Agarwal and D.Lewis (1994). Routing architectures for hierarchical field programmable gate arrays. *In Proceedings 1994 IEEE International Conference on Computer Design*, pages 475–478.

[Actel, 2008] Actel (2008). Actel corp. website. *http://www.actel.com*.

[A.DeHon, 1999] A.DeHon (1999). Balancing Interconnect and Computation in a Reconfigurable Computing Array (or, why you don't really want 100% LUT utilization). *Proceedings of ACM/SIGDA International Symposium on Fiel-Programmable Gate Arrays, Montery, CA*.

[A.DeHon, 2000] A.DeHon (2000). Compact Multilayer Layout for Butterfly Fat-Tree. *ACM Symposium on Parallel Algorithms and Architectures, Bar Harbor, Maine, USA*, pages 206–215.

[A.DeHon, 2001] A.DeHon (2001). Rent's Rule Based Switching Requirements. *System Level Interconnect Prediction Workshop*.

[A.DeHon, 2004] A.DeHon (2004). Unifing Mesh and Tree-Based Programmable Interconnect. *IEEE Transactions on VLSI Systems*, (12):10.

[A.J.Alexander et al., 1995] A.J.Alexander, J.P.Cohoon, J.L.Colflesh, J.Karro, and Robins, G. (1995). Three-Dimensional Field-Programmable Gate Arrays. *Proc. Intl. ASIC Conf.*, pages 253–256.

[Alliance, 2006] Alliance (2006). http://www-asim.lip6.fr/recherche/alliance/.

[Altera, 2008] Altera (2008). Altera corp. website. *http://www.altera.com*.

[A.Mishchenko, 2005] A.Mishchenko (2005). ABC: a system for sequential circuit synthesis. *Berkeley Logic Synthesis and Verification Group: http://www.eecs.berkeley.edu/ alanmi/abc/*.

[A.Sharma et al., 2005] A.Sharma, C.Ebeling, and S.Hauck (2005). Architecture Adaptive Routability-Driven Placement for FPGAs. *International Conference on Field Programmable Logic and Applications FPL*, pages 427–432.

[A.Singh and M.Marek-Sadowska, 2002] A.Singh and M.Marek-Sadowska (2002). Efficient circuit clustering for area and power reduction in FPGAs. *International Symposium on Fiels Programmable Gate Arrays*, pages 59–66.

[A.Yan and J.E.Wilton, 2006] A.Yan and J.E.Wilton, S. (2006). Product-term synthesizable embedded programmable logic cores. *IEEE transactions on very large scale integration (VLSI) systems*, 14:474–488.

[B.Landman and R.Russo, 1971] B.Landman and R.Russo (1971). On Pin Versus Block Relationship for Partition of Logic Circuits. *IEEE Transactions on Computers*, 20(1469-1479).

[B.Riess and G.Ettelt, 1995] B.Riess and G.Ettelt (1995). Speed: Fast and Efficient Timing Driven Placement. *IEEE Symposium on Circuits and Systems* , pages 377–380.

[B.W.Johnson, 1998] B.W.Johnson (1998). Design & analysis of fault tolerant digital systems. *Addison-Wesley Longman Publishing, ISBN:0-201-07570-9, Boston, MA*.

[C.Ababei et al., 2005] C.Ababei, H.Mogal, and K.Bazargan (2005). Three-dimensional Place and Route for FPGAs. *Proc. IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*.

[Cadence, 2006] Cadence (2006). Cadence design systems.inc http://www.cadence.com.

[C.Alexandre et al., 2005] C.Alexandre, H.Clement, S.Marek, C.Masson, and E.Remy (2005). Tsunami: An integrated timing-driven place and route research platform. *Design Automation and Test in Europe Conference, Date 2005, Mucchen Genrmany*, pages 920–921.

[C.Alexandre et al., 2006] C.Alexandre, M.Sroka, H.Clément, and C.Masson (2006). Zephyr: A static timing analyzer integrated in a trans-hierarchical refinement design flow. *Power and Timing Modeling Optimization and Simulation, PATMOS'2006), Montpellier, France*, pages 319–328.

[C.Carmichael et al., 1999] C.Carmichael, E.Fuller, P.Blain, and M.Caffrey (1999). SEU Mitigation Techniques for Virtex FPGAs in Space Applications. *Proceedings of the Military and Aerospace Applications of Programmable Logic Devices(MAPLD), Washington D.C.*

[C.Leiserson, 1985] C.Leiserson (1985). Fat-trees: Universal networks for hardware efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901.

[CMP, 2006] CMP (2006). Circuit Multi Project.

[C.Sung et al., 1998] C.Sung, R.Cliff, J.Huang, B.Wang, K.Nguyen, X.Wang, K.Veenstra, B.Pedersen, and J.Turner (1998). A silicon efficient FLEX6000 programmable logic architecture. *IEEE Custom Integrated Circuits conference*, pages 273–276.

[C.Thompson, 1979] C.Thompson (1979). Area-time complexity for VLSI. *ACM Annual symposium on theory of computing*, pages 81–88.

[D.Huang and A.Kahng, 1997] D.Huang and A.Kahng (1997). Partioning-Based Standard-Cell Global Placement with an Exact Objective. *ACM Symposium on Physical Design*, pages 18–25.

[D.Tavana et al., 1996] D.Tavana, W.Yee, and V.A.Holen (1996). FPGA Architecture with Repeatable Tiles Including Routing Matrices and Logic Matrices. , (618,445).

[E.Ahmed and J.Rose, 2000] E.Ahmed and J.Rose (2000). The effect of LUT and cluster size on deep-submicron FPGA performance and density. *Proceedings of the International Symposium on Field Programmable Gate Arrays*, pages 3–12.

[E.A.Kusse and J.Rabaey, 1998] E.A.Kusse and J.Rabaey (1998). Low-energy embedded FPGA structures. *Int. Symp. On Low Power Electronics and DesignLow-Power FPGA*, pages 155–160.

[eASIC, 2008] eASIC (2008). eASIC nextreme 90nm NEW ASIC. *www.easic.com*.

[E.Lee et al., 2006] E.Lee, G.Lemieux, and S.Mirabbasi (2006). Interconnect Driver Design for Long Wires in Field-Programmable Gate Arrays. *IEEE International Conference on Field Programmable Technology*, pages 1–8.

[E.M.Sentovich et al., 1992] E.M.Sentovich, K.J.Singh, L.Lavagno, C.Moon, R.Murgai, A.Saldanha, H.Savoj, Stephan, P., R.K.Brayton, and A.Sangiovanni-Vincentelli (1992). SIS: A System for Sequential Circuit Synthesis. *Technical Report No. UCB/ERL M92/41. University of California, Berkeley*.

[E.Sentovich, 1992] E.Sentovich (1992). SIS: a system for sequential circuit synthesis. *Tech. Report No.UCB/ERL M92/41, University of California at Berkeley*.

[F.Li et al., 2004] F.Li, Y.Lin, L.He, and J.Cong (2004). Low-power fpga using predefined dual-vdd/dual-vt fabrics. *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 42–50.

[F.P.Preparata and J.Vuillemin, 1981] F.P.Preparata and J.Vuillemin (1981). The cube-connected cycles: A versatile network for parallel computation. *Comm. of the ACM*, pages 300–309.

[G.Borriello et al., 1995] G.Borriello, C.Ebeling, S.Hauck, and S.Burns (1995). The Triptych FPGA Architecture. *IEEE Transactions on VLSI Systems*, 3(4):491–501.

[G.Lemieux and D.Lewis, 2004] G.Lemieux and D.Lewis (2004). Design of Interconnection Networks for Programmable Logic. *Kluwer Academic Publishers*.

[H.J.Chao et al., 2001] H.J.Chao, C.H.Lam, and E.Oki (2001). Broadband packet switching technologies: A practical guide to atm switches and ip routers. *Wiley-Interscience*.

[I.Kuon et al., 2005] I.Kuon, A.Egier, and J.Rose (2005). Design, Layout and Verification of an FPGA using Automated Tools. *ACM/SIGDA Symposium on Field Programmable Gate Arrays*.

[I.Kuon and J.Rose, 2007] I.Kuon and J.Rose (2007). Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on CAD*, 26(2):203–215.

[itc, 1999] itc (1999). http://www.cad.polito.it/tools/itc99.html. .

[J.Cong and Y.Ding, 1994] J.Cong and Y.Ding (1994). FlowMap: An optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table based FPGA Designs. *IEEE Transactions on Computer-Aided Design*, pages 1–12.

[J.Cong and Y.Hwang, 1995] J.Cong and Y.Hwang (1995). Simultaneous Depth and Area Minimization in LUT-Based FPGA Mapping. *ACM/SIGDA International Symposium on Field Programmable Gate Array*, pages 68–74.

[J.Duato et al., 1997] J.Duato, Yalamanchili, S., and L.Ni (1997). Interconnection networks, an engineering approach. *IEEE Computer Society Press*.

[J.Pistorius and M.Hutton, 2003] J.Pistorius and M.Hutton (2003). Placement Rent Exponent Calculation Methods, Temporal Behaviour and FPGA Architecture Evaluation. *ACM/IEEE 5th Internationale Workshop on System Level Interconnect Prediction*.

[J.Rose et al., 1990] J.Rose, R.Francis, D.Lewis, and P.Chow (1990). Architecture of Field-Programmable Gate Arrays: The Effect of Logic Functionality on Area Efficiency. *IEEE Journal of Solid State Circuits*.

[J.Rose and S.Brown, 1991] J.Rose and S.Brown (1991). Flexibility of interconnection structures in field programmable gate arrays. *IEEE Journal of Solid State and Circuits*, pages 277–282.

[K.Padalia et al., 2003] K.Padalia, R.Fung, M.Bourgeault, A.Egier, and J.Rose (2003). Automatic transistor and physical design of FPGA tiles from an architectural specification. *in Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays, Sigda Symposium on Field Programmable Gate Arrays, Montery*, pages 164–172.

[K.Poon and S.J.E.Wilton, 2002] K.Poon and S.J.E.Wilton (2002). A flexible power model for fpgas. *International Conference on Field-Programmable Logic and Applications,britsh colombia*.

[Lattice, 2008] Lattice (2008). Lattice semiconductors corp. website. *http://www.latticesemi.com/products/fpga/index.cfm*.

[L.McMurchie and C.Ebeling, 1995] L.McMurchie and C.Ebeling (1995). Pathfinder: A Negotiation-Based Performance-Driven Router for FPGAs. *Proc.FPGA'95*.

[L.Shang et al., 2002] L.Shang, A.S.Kaviani, and K.Bathala (2002). Dynamic Power Consumption in Virtex-II FPGA Family. *Tenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*.

[M2000,] M2000. FLEXEOS Configurable IP Core. *www.m2000.fr*.

[M.Borgatti et al., 2002] M.Borgatti, F.Lertora, B.Forêt, and L.Cali (2002). A reconfigurable System featuring Dynamically Extensible Embedded Microprocessor, FPGA and Customisable I/O. *in Proceedings of Custom Integrated Circuits Conference*.

[MentorGraphics, 2006] MentorGraphics (2006). CALIBRE. *http://www.mentor.com/products*.

[M.Huang et al., 1986] M.Huang, F.Romeo, and A.Sangiovanni-Vincentelli (1986). An Efficient General Cooling Schedule for Simulated Annealing. *International Conference on Computer Aided Design*, pages 381–384.

[M.Leeser et al., 1998] M.Leeser, W.Meleis, M.Vai, S.Chiricescu, W.Xu, and P.Zavracky (1998). Rothko: A Three-Dimensional FPGA. *IEEE Design Test Computers*, pages 16–23.

[M.Lin and A.Gamal, 2007] M.Lin and A.Gamal (2007). A routing fabric for monolithically stacked 3D-FPGA. *Proceedings of the ACM/SIGDA 15th international symposium on Field programmable gate arrays*, pages 3–12.

[N.Kafafi et al., 2003] N.Kafafi, K.Bozman, and Wilton, S. (2003). Architectures and algorithms for synthesizable embedded programmable logic cores. *ACM/SIGDA Symposium on Field Programmable Gate Arrays*, pages 3–11.

[opencores, 2009] opencores (2009). http://www.opencores.org. .

[P.Guerrier and A.Greiner, 2000] P.Guerrier and A.Greiner (2000). A generic architecture for onchip packet-switched interconnections. *Proceedings of the Design Automation and Test in Europe Conference 2000 (DATE 2000), Paris, France*, page 250 256.

[P.S.Zuchowski et al., 2002] P.S.Zuchowski, C.B.Reynolds, and R.J.Grupp (2002). A Hybrid ASIC and FPGA Architecture. *International Conference on Computer-Aided Design*.

[S.Belloeil et al., 2007] S.Belloeil, D.Dupuis, C.Masson, J.P.Chaput, and H.Mehrez (2007). A procedural circuit description language based upon python. *International Conference on Microelectronics, ICM 2007, Cairo, Egypt*, pages 275–278.

[S.Hareland et al., 2001] S.Hareland, J.Maiz, and M.Alavi (2001). Impact of CMOS process scaling and SOI on the soft error rates of logic processes. *VLSI Technology. Digest of Technical Papers*, pages 73–74.

[S.Phillips et al., 2004] S.Phillips, A.Sharma, and S.Hauck (2004). Automating the layout of reconfigurable subsytems via template reduction. *International Symposium on Field-Programmable Logic and Applications*, pages 857–861.

[S.Phillips and S.Hauck, 2002] S.Phillips and S.Hauck (2002). Automatic layout of domain-specific reconfigurable subsytems for system-on-a-chip. *ACM/SIGDA Symposium on Field Programmable Gate Arrays*, pages 165–173.

[Stratix-III, 2008] Stratix-III (2008). Stratix-III Device Handbook. *http://www.altera.com/literature/lit-stx3.js*.

[Synopsys, 2006] Synopsys (2006). Design Complier Reference Manual. *http://www.synopsys.com*.

[T.Calin et al., 1996] T.Calin, M.Nicolaidis, and R.Velazco (1996). Upset Hardened Memory Design for Submicron CMOS Technology. *IEEE Transaction on nuclear science*, 43(6).

[T.Vaida, 2001] T.Vaida (2001). PLC Advanced Technology Demonstrator TestChip. *Proceedings of the 2001 Custom Integrated Circuits Conference*, pages 67–70.

[Varicore, 2001] Varicore (2001). Varicore Embedded Programmable Gate Array Core(EPGA) 0.18 um Family Data Sheet. *www.actel.com*.

[V.Betz et al., 1999] V.Betz, A.Marquardt, and J.Rose (1999). Architecture and CAD for Deep-Submicron FPGAs. *Kluwer Academic Publishers*.

[V.Betz and J.Rose, 1995] V.Betz and J.Rose (1995). Using architectural families to increase FPGA speed and density. *ACM Sigda Symposium on Field Programmable Gate Arrays, Montery*, pages 10–16.

[V.Betz and J.Rose, 1997] V.Betz and J.Rose (1997). VPR: A New Packing Placement and Routing Tool for FPGA research. *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, pages 213–22.

[V.George and J.Rabaey, 2001] V.George and J.Rabaey (2001). Low-energy FPGA - Architecture and Design. *Kluwer Academic Publishers*.

[V.Maingot et al., 2007] V.Maingot, J.Ferron, R.Leveugle, V.Pouget, and A.Douin (2007). Configuration errors analysis in SRAM-based FPGAs: software tool and practical results. *Microelectronics Reliability*, 47(9-11):1836–1840.

[W.Tsu et al., 1999] W.Tsu, K.Macy, A.Joshi, R.Huang, N.Walker, T.Tung, O.Rowhani, V.George, J.Wawrzynek, and A.DeHon (1999). HSRA: High Speed, Hierarchical Synchronous Reconfigurable Array. *Proceedings of the International Symposium on Field Programmable Gate Arrays*, 20(1469-1479):125–134.

[W.Wang, 2004] W.Wang (2004). RC hardened FPGA configuration SRAM cell design. *Electronics Letters*, 40(9):525–526.

[Xilinx, 2008] Xilinx (2008). Xilinx Inc Website. *http://www.xilinx.com*.

[Y.Lay and P.Wang, 1997] Y.Lay and P.Wang (1997). Hierarchical Interconnection Structures for Field Programmable Gate Arrays. *IEEE Transactions on VLSI Systems*, 5(2):186–196.

[Z.Marrakchi, 2008] Z.Marrakchi (2008). Exploration and Optimization of Tree-Based FPGA Architectures. *Thesis, University Of Pierre et Marie Curie, www-asim.lip6.fr*.