THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : Informatique
École Doctorale SICMA

présentée par
Ciprian TEODOROV
Préparée à Lab-STICC UMR 3128

Model-Driven Physical-Design for Future Nanoscale Architectures

Thèse soutenue le 28 novembre 2011
devant le jury composé de :

Jacques-Olivier KLEIN
Professeur, Université Paris Sud 11 / *examinateur (président du jury)*

Dominique LAVENIER
Directeur de Recherches, IRISA Rennes / *rapporteur*

Ian O'CONNOR
Professeur, Ecole Centrale de Lyon / *rapporteur*

Bernard POTTIER
Professeur, Université de Bretagne Occidentale / *directeur de thèse*

Loïc LAGADEC
Habilite a diriger des recherches, Université de Bretagne Occidentale / *co-directeur de thèse*

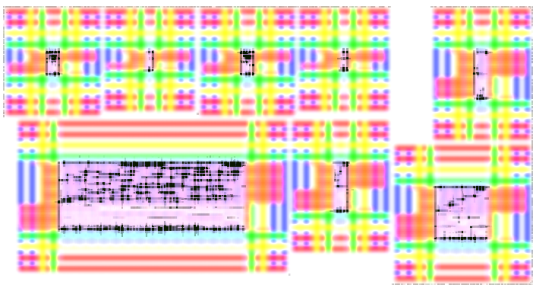
Catherine DEZAN
Maitre de Conférences, Université de Bretagne Occidentale / *co-directeur de thèse*

CIPRIAN TEODOROV

MODEL-DRIVEN PHYSICAL-DESIGN FOR FUTURE NANOSCALE ARCHITECTURES

MODEL-DRIVEN PHYSICAL-DESIGN FOR FUTURE NANOSCALE ARCHITECTURES

CIPRIAN TEODOROV



Integrated Circuit Design in Nanoscale Era

2011

SUPERVISORS:
Bernard Pottier
Loïc Lagadec
Catherine Dezan

Discovery consists in seeing what everyone else has seen
and thinking what no one else has thought.

— Albert Szent-Gyorgyi

Dedicated to the memory of Dan Ioan Marcean
1985 – 2004

Abstract

In the context where the traditional CMOS technology approaches its limits, some nanowire-based fabric proposals emerged, which all exhibit some common key characteristics. Among these, their bottom-up fabrication process leads to a regularity of assembly, which means the end of custom-made computational fabrics in favor of regular structures. Hence, research activities in this area, focus on structures conceptually similar to today's reconfigurable PLA and/or FPGA architectures[165, 160]. A number of different fabrics and architectures are currently under investigation, e. g.CMOL[165], FPNI[160], NASIC[115]. These proof-of-concept architectures take into account some fabrication constraints and support fault-tolerance techniques. What is still missing is the ability to capitalize on these experiments while offering a one-stop shopping point for further research, especially at the physical-design level of the circuit design tool-flow. Sharing metrics, tools, and exploration capabilities is the next challenge to the nano-computing community.

We address this problem by proposing a model-driven physical-design toolkit based on the factorization of common domain-specific concepts and the reification of the tool-flow. We used this tool-flow to drive the design-space exploration in the context of a novel nanoscale architecture, and we showed that such an approach assures design convergence based on frequent quantitative evaluations, moreover it enables incremental evolution of the architecture and the automation flow.

Résumé

Actuellement, comme la technologie CMOS arrive à ses limites, plusieurs alternatives architecturales nanométriques sont étudiées. Ces architectures partagent des caractéristiques communes, comme par exemple la régularité d'assemblage, qui contraint le placement de dispositifs physiques à des motifs réguliers. Par conséquence, les activités de recherche dans ce domaine sont focalisées autour des structures régulières similaires, d'un point de vue conceptuel, aux architectures réconfigurables de type PLA et FPGA[165, 160]. Parmi ces différents travaux, ont peut citer CMOL[165], FPNI[160], NASIC[115]. Ces prototypes architecturaux sont conçus pour répondre à des contraintes de fabrication et incluent des politiques de tolérance aux défauts. Par contre, il manque la possibilité d'exploiter ces expériences et d'offrir une solution qui, en capitalisant les résultats obtenus, puisse offrir une infrastructure unique pour les futurs recherches dans ce domaine. Ceci est vrai surtout au niveau du flot de conception physique ciblant l'automatisation du processus de création de circuit. Le partage de métriques, outils et supports d'exploration est le futur défi de la communauté nano-électronique.

On répond à ce problème en proposant un flot de conception physique, reposant sur une méthodologie de développement dirigé par les modèles, qui factorise les concepts métiers et réifie les éléments du flot de conception. Nous avons utilisé ce flot pour explorer l'espace de conception d'une nouvelle architecture nano-métrique et on a montré qu'une telle démarche permet la convergence du processus de conception à l'aide de fréquentes évaluations quantitatives. De plus, cette méthodologie permet l'évolution incrémentielle de l'architecture et du flot de conception.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

—Donald E. Knuth

Acknowledgments

First of all, I want to express my gratitude to the members of my dissertation committee, starting with the foreman of the jury, prof. Jacques-Olivier Klein, along with prof. Dominique Lavenier, prof. Ian O'Connor, prof. Bernard Pottier, prof. Loic Lagadec, and prof. Catherine Dezan which all have contributed significantly to this work through the preparation, the revision and the evaluation of this manuscript.

I want to thank prof. Lavenier, and prof. O'Connor for accepting the role of external reviewers and for blessing my work with their attention and appreciation during the review and evaluation process.

I am deeply grateful to my supervisors prof. Bernard Pottier, prof. Loic Lagadec, and prof. Catherine Dezan. Bernard courageously assumed the role of principal advisor and never hesitated to steer my energy into the right direction through his encouragements, advices and fatherly mentoring. It has been an honor to work with Catherine who dedicated a lot of her time to my work, and with whom we inquisitively evaluated numerous ideas, and research directions. Her patience and understanding were limitless during the tough times in the PhD pursuit, and her joy and enthusiasm were contagious. I am profoundly indebted to Loic that has not only been a supervisor but also a friend, and a direct supporter of many of my strange ideas.

Part of this work was supported by European University of Britanny (UEB) through a student exchange fund, which enabled me to work 5 months at the University of Massachusetts in Amherst under the supervision of prof. Csaba Andras Moritz. I want to thank the UEB committee for giving me the opportunity to work directly with one of the best nano-computing research teams in the world. Moreover, I want to thank prof. Moritz and the members of Nanoscale Fabrics Lab for integrating me into their team and supporting my research. Besides prof. Moritz, I want to great here Pritish Narayanan, Rahman Mostafizur, Pavan Panchapakeshan, and Prasad Shabadhi for making me feel at home in Amherst, and for the many hours that we have spent discussing everything from research to lifestyle.

I was very fortunate to work with a group of very dynamic people, members of the Lab-STICC's MOCS team. I want to thank all the former and present members of this team who have helped through collaborations and invaluable discussions, especially Damien Picard, Samar Yazdani, Jalil Boukhoubza, Erwan Fabiani, Ahcene Bounceur, and Hritam Dutta.

Besides everybody else, my close friends starting with Sebastien Tripodi, Jean Paul Soleil, and Amara Touil, played one of the most important roles during these last three years by keeping me a socially engaged human being. They were all there whenever I would need them, for whatever reason, and without asking too many questions.

My heart goes also to my family, back in Romania, especially to my parents and my grandma, for their support and patience during the last 28 years of the bull ride that brought me here.

Lastly, but not the least, I want to thank Anamaria, my sweetheart, for her understanding and tolerance during my sleepless nights, for her encouragements and support during the harsh days, and for her heart and love throughout all the way.

Thank you all, thanks for being with me!

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

This thesis explores the tight connection between integrated circuits and their associated design tools in the context where the traditional CMOS technology approaches its limits, and a number of alternatives have been investigated. Amongst these alternatives, the designs using silicon nanowire crossbars are very promising, mainly due to their high integration densities and their bottom-up fabrication process - which can drastically reduce the fabrication costs. A number of different fabrics and architectures are currently under investigation, for example NanoPLA[31], CMOL[165], FPNI[1][160], NASIC[2][115]. They are based on a variety of devices such as field effect transistors (FET)[121], spin-based devices[152], diodes, and molecular switches[161]. All these fabrics include support in CMOS: some, like FPNI, would move the entire logic into CMOS, others, like NASIC, would only provide the control circuitry in CMOS. The rationale for this varies but includes targeted application areas as well as manufacturability issues[122]. However, there is a gap between these technological developments and the tools used to design and exploit them. This gap comes mainly from the lack of flexible and evolutive electronic computer-aided design tools, which limits the reuse of algorithms and sound CAD solutions, forcing the fabric and architecture designers to create proprietary tool-flows. This hinders shared improvement over fabric design, and slows the evolution of the field, mainly due to the poor support for multi-fabric design space exploration. In this work, we rethink the circuit physical design tool-flow. Borrowing ideas from computer science research and software engineering, we propose a flexible and re-targetable tool-flow. This tool-flow improves on the state of the art and creates the necessary environment for future nanoscale CAD research and development.

1.2 Research questions

Over the last half century, computer architects have followed the trend imposed by Moore's law, addressing the need for ever-increasing performances. CMOS technology has scaled along with Moore's law for many years, allowing the architects to build performant systems. Unfortunately, CMOS technology suffers from a problem that deepens everyday. As the technology progresses, CMOS devices become smaller and smaller, nowadays reaching deep sub-micron range(less than 50nm); at this order of magnitude, sooner or later, CMOS devices will cease to scale because

[1] Field Programmable Nanowire Interconnect
[2] Nanoscale Application Specific Integrated Circuits

of their physical properties. But this is not the only problem: as the devices shrink, the cost of fabrication plants increases exponentially, while the ability to handle fabrication process variations decreases. To address these problems, during the last years research groups and industry focused on finding the technology that will permit the evolution of integrated circuits past CMOS limits. A number of alternative technologies are currently under investigation, amongst which the designs based on silicon nanowire (SiNW) crossbar are very promising.

During the last years tremendous progress on the physical, technological, manufacturing, and fabric design greatly improved our understanding on the advantages and the limitations of the SiNW crossbar technology. However, little or no significance was given to design and exploitation tools which in fact are the limiting factors hindering shared improvement over multiple fabrics. The creation of proprietary automation solutions for each new fabric design is very costly in terms of development effort, the resulting tools are not reusable even for closely related projects, and such targeted effort closes the tool exploration axis of the whole design-space exploration problem by directly providing *supposedly* optimized solutions.

In the electronic CAD community the design-space exploration(DSE) problem is typically studied from the perspective of the adequacy application/architecture. In this context, there has been a lot of research to automatically or semi-automatically tune a specific application to match the underlying architectural constraints. The tool optimization is viewed as an independent problem, and many improvements in terms of algorithm complexity, scalability and flexibility were achieved for each step of the circuit design automation tool-flow. Moreover, since the technological and architectural framework was stable, the small number of new challenges addressed at each new CMOS technology node did not disrupt the design-automation flow. However, in the current technological context, with the traditional technology approaching its fundamental limits and the apparition of a large number of emerging technologies competing for adoption, we argue that there is a stringent need for adding a third dimension to the design-space exploration focused on tool design and optimization. This new exploration axis adds to new perspectives to the DSE problem, namely the adequacy tool-flow/architecture and the adequacy tool-flow/application. In the context of this thesis the tool-flow/application adequacy is treated as a secondary issue, while the focus is on the tool-flow/architecture adequacy. The importance of this new perspective comes mainly from the need of tool reuse (to reduce the development costs) and from the need of unbiased evaluation of different technological frameworks at the architectural level (to objectively compare different computing supports).

Amongst all emerging architectures competing for adoption as a CMOS replacement, the fabric-designs based on silicon nanowire crossbar imposed themselves as a viable solution, principally due to their bottom-up fabrication process which offers the opportunity of achieving unprecedented integration density. However, besides their advantages, these designs come with new challenges and constraints. Among these, their bottom-up fabrication process leads to a *regularity of assembly*, which means the end of custom-made computational fabrics in favor of regular structures designed with respect to the application needs. Hence research activities in this area mainly focus on structures conceptually similar to today's reconfigurable PLA[3] and/or FPGA[4] architectures[165, 160]. Based on this observation, it has been assumed that the tools traditionally used in the context of reconfigurable architecture design can be easily retargeted for these new computing supports. During the last ten years, the reconfigurable architecture tool-flow improved and matured relying mainly on a common tool infrastructure implementing generic algorithms and heuristics tuned by externally defined optimization metrics. However, in the context of crossbar-based nanoscale architectures the circuit design optimization tool-flow is still developed in an ad-hoc, architecture-specific manner. This aspect is even more questionable since most of these architecture-specific tools rely extensively on generic FPGA frameworks with design specific proprietary extensions, such as VPR[9] in the context of CMOL[165] and NanoPLA[31], and Madeo[90] in the context of NASIC[115, 96]. In this context, the question is to what extent tools from reconfigurable field can be reused for automating circuit design on nanoscale crossbar fabrics, what are the limitations of

[3]Programmable Logic Array
[4]Field-Programmable Gate Array

such tools, and most importantly, can we provide a generic multi-fabric infrastructure for nanoscale circuit design, similar with today's generic FPGA toolkits?

In the context of nanoscale electronics, one of the principal challenges to overcome is the high-rate of defects, which needs innovative defect-tolerance strategies to enable the creation of reliable computing fabrics. A large number of research work was dedicated to this topic, and different solutions were proposed, ranging from self-healing fabric architectures[115] to defect-tolerance through reconfiguration[165]. However, little or no attention was given to the integration of these defect-tolerance techniques into the circuit automation tool-flow, nor to the infrastructure needed to seamlessly integrate defect-awareness into the design-flow.

In this thesis, we address some of the questions raised in this section from the tool-flow perspective at the physical-design level. The problems raised and the solutions provided are studied in the context crossbar-based nanoscale architecture, but most of the issues are more general, transcending this architectural approach, and the solutions can sometimes be reframed in the larger context of emerging architectures.

1.3 Contribution of this thesis

The results presented in this thesis rely most notably on the exploration and the analysis of the interdependence between the crossbar-based nanoarchitectures and the physical design tools, with the purpose of providing answers that would help reduce the design and exploitation costs for new technologies. The principal contributions presented in this manuscript are:

- The introduction of a *common vocabulary* for nanoscale architecture modeling at different abstraction levels. This vocabulary is based on an abstract meta-model relying on a hierarchical port-graph structure. This meta-model is used for architecture and application modeling, for the specification of different simulation models, and can be extended to address defect and fault modeling and injection.

- The design of a *model-driven physical synthesis tool-flow* which *decouples the architectural model from the physical synthesis tools*. This flow enables the parallel evolution of the architectures and tools, improves the algorithm reutilization, eases the agile development of the design-flow, and creates the necessary conditions for incremental design space exploration. Moreover, the use of the Model-Driven Development in the context of the physical design opens the toolbox offering an unprecedented flexibility and support for the evolution of the tool-flow.

- The design of a *new nanoscale architecture* based on the NASIC fabric concepts along with its performance models and optimization policies. This architecture, named R2D NASIC, is compatible with the NASIC technological framework and fabrication process, and can easily be adapted according to the technological and application constraints. Through its regularity, this architecture enables arbitrary logic placement and routing. Moreover it offers the possibility of implementing max-rate pipeline designs with an average 35X higher-frequency than non-pipelined versions, paving the way to high-performance nanoscale circuits.

- The *bootstrap of the design space exploration* relying on tools used today in the context of reconfigurable architectures. An approach through which early baseline evaluations were performed on the R2D NASIC design. These evaluations were then used to guide the architectural and tool-flow design process.

- The creation of a *new routing algorithm* specific for R2D NASIC, for achieving max-rate pipeline designs. This algorithm balances the pipeline stages over the routing paths. Using this algorithms the performances were improved up to 77X, with 3X better performances per unit area compared to the non-optimized designs.

1.4 Outline of the thesis

This section overviews the content presented in this thesis, briefly announcing the core content of each chapter.

Chapter 2 - Future of Integrated Circuits

The main intent of this chapter is to familiarize the reader with the main terms, concepts and challenges of the electronics industry while introducing the technological framework used in the subsequent parts of the thesis. The context is described with details which emphasize the limitations of the current technology and motivate the research for breakthrough solutions. A number of emerging technologies are briefly presented before the focus is drawn to crossbar-based design, the technological framework that represents the basis of the work presented in this thesis.

Chapter 3 - Bridging the Gap Between Applications and Technology

As the complexity of integrated circuits increases, their design puts more and more pressure on the automated tool-flow used. This chapter gives an overview of the main steps of circuit design automation in the context of current technology. It then presents the physical design of nanoscale circuits from the perspective of this traditional design flow, pointing out the challenges that need addressing in this new technological context.

Chapter 4 - Model-Driven Physical Design Flow

The model-driven software development methodology is used to implement the physical design step of the circuit design tool-flow. The targeted fabric design is described using an abstract model of a hierarchical port-graph. The tool-flow is reified and modeled using a specific object-oriented abstract model, which enables a high-degree of algorithm reuse and drives design-space exploration. Moreover, this tool-flow is backward-compatible and favors high-degree of flexibility and reuse.

Chapter 5 - Nanoscale Architecture Template and Associated Tools

A regular 2D nanoscale architecture template based on NASIC fabric building blocks is presented along with its evaluation metrics and optimization tools. Besides being compatible with NASIC technological and manufacturing guidelines, this architecture enables the creation of highly pipelined circuits while easing the delay estimation at the tool-flow level.

Chapter 6 - Conclusions & Perspectives

The main points evoked and developed in this thesis are overviewed. The principal contributions are summarized and discussed emphasizing the strengths of the presented CAD methodology. Moreover, some insights on future developments and main open research questions for the ECAD targeting nano-electronics are addressed in the perspectives section.

2

The Future of Integrated Circuits

This chapter presents the technological context, which motivates the work presented in this manuscript. After briefly presenting the integrated circuits field and some of the terminology that will be reused in the following chapters, we explain the limits of the current CMOS technology and we present the state of the research on emerging technologies with a focus on the crossbar-based designs.

2.1 Introduction

Since the invention of the first integrated circuit[136], in 1958, the electronics field had an unprecedented evolution getting from a few transistor integrated on a silica plate to billions in the current circuit designs. This evolution was possible mainly due to technological and fabrication breakthroughs that enabled the exponential size reduction of devices integrated on a chip. In 1965 Gordon Moore stated an empirical law which drove the industry for the last four decades. The Moore's law predicted that the number of transistors integrated in a chip will double every two years. This exponential increase in integration density enabled the reduction of the fabrication cost per chip, the design of higher frequency chips with lower power consumption per device, and the creation of smaller and smarter[1] end-products.

Today the feature size of the devices integrated in a chip reached the nano-metric scale. The cost of foundries continues to increase and the ability to handle fabrication process variations decreases. Moreover, at device level, the parasitic resistance and capacitance are starting to dominate the intrinsic resistance and capacitance of the devices, which imposes hard physical limits for performance improvements by reducing the feature size. As a result the experts are announcing the end of the CMOS technology evolution, or at least the end of the Moore's law as we know it - based on the integration density as a metric for evolution. In 2009 ITRS[73] reports the possible replacement of this "geometric scaling" by the "equivalent scaling". Equivalent scaling will continue evolution based on innovative devices (e.g. memristor[167]), revolutionary computing models (e.g. DNA computing[146]), creative architectures, and software evolutions.

A large number of technologies and innovative designs are nowadays under investigation and are considered as possible alternatives for the future of integrated circuits evolution. Some examples are Quantum cellular automata[100], Spin-based electronics[168, 151], Carbon nanotube[131] and Silicon Nanowire designs[17], Markov random networks[124], Neuromorphic designs[196], etc. Each

[1] integrating more and more functional blocks

of these technologies have their advantages and challenges, and for the time being none of them is considered mature enough to replace traditional CMOS circuits.

The SiNW-based crossbars[182] are the most promising building blocks for the future integrated circuit design due to their fabrication methods, their electrical properties, their huge integration capacity and their compatibility with today's technology.

This chapter starts (Sec. 2.2) by overviewing the different integrated circuit designs and their evolution over the years. In Sec. 2.3 the limits of the current CMOS technology are presented. Sec. 2.4 presents the alternatives that are currently under investigation for replacing CMOS and continuing Moore's law. The different crossbar based fabric designs are presented in Sec. 2.5. Sec. 2.6, reviews the most important aspects described in this chapter.

2.2 Integrated Circuits - Overview

Integrated circuits are without doubt one of the most amazing success stories of the last 60 years. They can be found virtually in all electronic equipment today, and have revolutionized society. Today's computers, cell phones, GPS devices, IPods, etc., fiction for our grand-parents, have become embedded in our daily lives. They are made possible by the tight integration of advanced electric circuits enabled by the technological advancements of the last century.

These circuits are made up of basic electrical components such as:

- resistors - passive devices that limit the flow of current passing through and allow the control of the amount of current allowed to pass.
- capacitors - passive devices allowing to store electric charge and to release it.
- diodes - passive devices that allow electric current to pass in one direction while blocking it in the opposite direction. But they can be tuned towards more complex behavior.
- transistors - active devices that behave like a switch and thus are used to allow and disallow the flow of current. Moreover, they can amplify current.

The transistor is probably the most important device in today's electronics. Before its invention, at Bell Labs in 1947, the vacuum tubes were used for the same functions, but in comparison they were big, slow, consumed more power, moreover, they would burn out easily. The transistor didn't have these problems, however solutions to design complexity and imperfections of the manual assembly techniques had to be found, before starting the exponential progress that we have witnessed. In 1958 Jack Kilby and Robert Noyce solved the complexity problem by proposing what we call today integrated circuits. J. Kilby's proposition[83, 136] enabled building many transistors out of a monolithic block of semiconductor, while R. Noyce[129] made this approach practical by adding a metal layer on top, that will be patterned to create wires, thus solving the connection problems.

Based on these fundamental breakthroughs, during the last 50 years, the integrated circuit industry moved fast forward, nowadays integrating billions of transistors in a single chip. This history of evolution, driven mainly by the ability to scale down devices and build denser and denser chips, can be broken down into several generations, according to the integration level (the number of transistors on a single chip):

- Small-Scale Integration, consisting in small circuits assembly of a few (tens of) transistors.
- Medium-Scale Integration, circuits having hundreds of transistors on the same die.
- Large-Scale Integration, having thousands of devices.
- Very-Large-Scale Integration, ranging from 100 000 to, nowadays, several billion transistors.

- Ultra-Large-Scale Integration, stands for further improvements and scaling, but there are technological factors that limit the scaling of the current technology (See. Section 2.3 for details). Today several alternative scaling strategies (based on other metrics than integration density) are investigated, like system-on-chip (SoC), or three-dimensional integrated circuits (3D-IC).

The integrated circuits can be classified into three broad categories, digital, analog, and mixed-signal, according to the way the electrical signals are interpreted either as discrete signals (0,1), continuos signals, or a mix of discrete and continuous signals.

∘Digital circuits are the dominating class of integrated circuits, replacing wherever possible the analog counter-parts, due to their noise tolerance, and automated design process.

Managing Complexity through Design Styles

Due to the huge integration densities the physical design of an integrated circuit is an extremely complex process. In consequence the entire process was decomposed into several easier steps, thus isolating the concerns and rendering the design complexity manageable. However under the market pressure, which demand quick time-to-market and high yield, the gap between the ever increasing requirements and the reality of circuit design complexity pushed designer towards restricted IC models and design styles that reduce the complexity of physical design. The design style can be classified into two distinct categories full-custom and semi-custom. In a full-custom design style the functional blocks of the circuit can be placed arbitrarily on the wafer without overlapping them. On the other hand, in semi-custom designs, different parts of the circuit are predesigned and placed at predefined positions on the layout. Figure 2.1 presents a classification of IC design styles showing further derivations of the semi-custom style. In the following paragraphs we will briefly present the most important design styles used for IC design. For the interested reader more details can be found in [156].

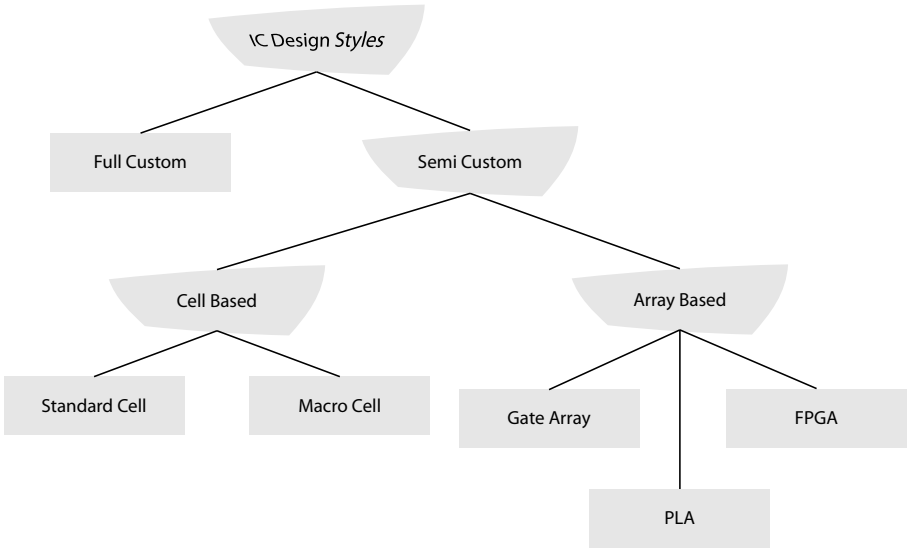
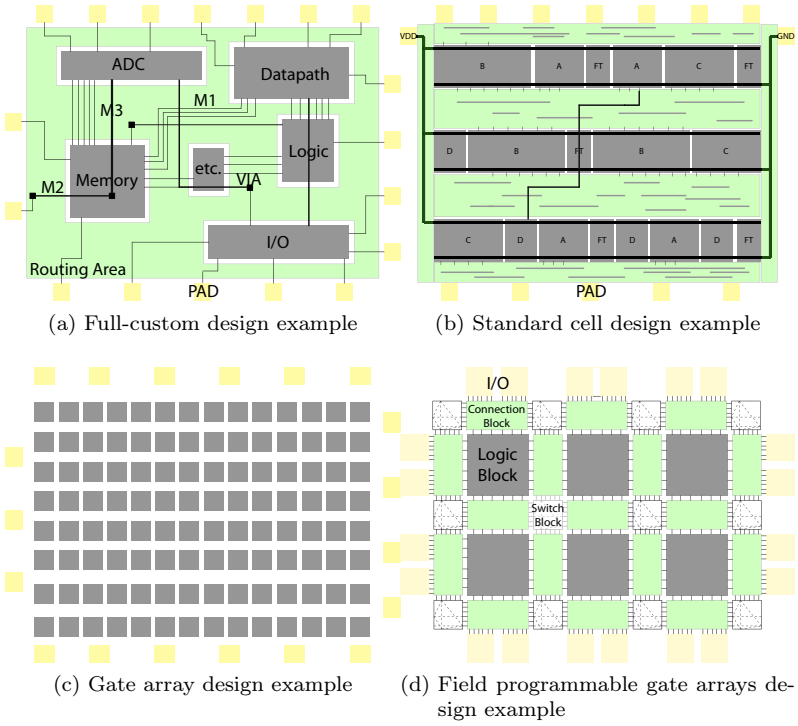


Figure 2.1: A taxonomy of digital IC design styles

Full-Custom Design This is the most general form of circuit design in which the circuit is partitioned into a collection of blocks usually based on the functionality and the density of connection between devices. This process is done hierarchically producing design with several levels of hierarchy. The full-custom design style doesn't constraint the shape nor the size of the functional blocks thus providing an array of heterogenous tiles which are placed on the wafer. Figure 2.2a

shows an example of a full-custom design with a few blocks. For simplicity, only one hierarchical level is shown in this figure. The connections between blocks are realized using several metal layers (three in the figure). The circuits realized using the full-custom design style are very compact. However, this design style is used for mass produced IC, since the high costs of optimizing the layout cannot be justified for low-volume production.



(a) Full-custom design example

(b) Standard cell design example

(c) Gate array design example

(d) Field programmable gate arrays design example

Figure 2.2: Four different IC design styles

Standard Cell Design Compared to the full custom design style, the design process of standard cell is much simples. The blocks (called standard cells in this context) are constrained to have the same height and are placed in parallel rows on the layout. The circuit is partitioned into smaller blocks which are equivalent to some predefined cells. These predefined cells are designed and tested separately prior to the circuit design and they provide a collection of standard functional blocks common to most IC needs. A collection of these cells is called a cell library. According to the circuit needs (identified during the partitioning) a number of cells from the library are instantiated and placed on the layout on a regular manner (in parallel rows). The space between these rows, called a channel, is used for interconnecting the cells to create the desired circuit. If two connected cells are placed on two rows that are not adjacent to each other, the electrical connection between these two is realized using feedthrough cells which are placed in the rows. Figure 2.2b shows a simple example of standard cell design using 4 cells (A, B, C, D). Typically standard cell designs are faster to develop, however an important non-recurring design effort is invested for creating the cell library. This design style is typically preferred for the creation of high-performance application specific integrated circuits.

Gate Array Design Gate arrays appeared to simplify the standard cell design. Unlike standard cell design, the cells in a gate array are identical. Each chip is a regular array of identical gates separated by horizontal and vertical channels. The cells of a gate array can be a simple NAND gate, which is regularly replicated. The gate array wafers are pre-fabricated, and they are

named "uncommitted" as long as the routing connections are not added to the chip. Once the circuit design is known, and placed on the uncommitted array the routing wires can be added to create a customized chip implementing the desired behavior. Figure 2.2c shows schematically an "uncommitted" gate array. The gate array design style has one of the most restrictive forms of layout, which implies simpler design automation algorithms, especially in terms of routing which is conceptually simpler than in the cases of full-custom or standard cell designs.

2.2.0.1 Programmable Logic Devices

A programmable logic device (PLD) is an electronic component with a regular organization, similar to the gate-array design, which enables post-manufacturing configuration to implement a specified logic functionality. The idea behind PLDs is to provide a functionally rich prefabricated chip which can be customized on demand. The user simply configures the already existing interconnect resources to suit its application needs. Two main design directions can be identified, one using a large number of simple logic gates to create the support for implementing logic functions in the sum of products canonical forms, another relying on small memory blocks (LUTs[2]) which can be configured to store the truth table of a logic function. This difference renders the first category more suitable for implementing large combinational logic application, while the second is more suitable for implementing sequential logic applications (i.e. large stage machines, microprocessors).

Programmable Logic Arrays (PLA) is one of the first programmable device relying on a set of programmable AND gate planes linked to a set of programmable OR gate planes to implement logic functions as sum of products. These first PLA devices are programmed during the fabrication process using different masks according to the desired functionality. The main advantage of this approach is the reduced fabrication cost, and the high integration density that can be achieved. However, the need for two different masks (one for the AND plane and one for the OR plane) made these devices less popular.

Programmable Array Logic (PAL) design is conceptually similar to the PLA design, however there are two important differences. The PAL devices implement logic functions using a programmable AND plane followed by a fixed OR plane, whereas the OR plane is programmable in the case of PLA. And probably the most important difference is the way these devices are programmed. If the PLAs are mask-programmed during the manufacturing process, the PAL devices rely on programmable read-only memories (PROM) to achieve one-time field-programming. The field-programming feature of these devices enables foundries to produce large amounts of virtually identical PAL devices, which will be then programmed by the customer to suit its application needs.

Generic array logic (GAL). The next evolution step of PAL devices is the introduction of generic array logic (GAL). Besides the logical properties of PALs these devices are re-programmable, which means that they can be erased and reprogrammed. The re-programmability of GALs is very useful during the prototyping stage of the design, when the eventual logic errors can be corrected by reprogramming. Moreover, since the implemented logic can be replaced after deployment the logic design can be updated or completely changed on the field.

Complex PLDs (CPLD). The main limitation of PLAs, PALs, and GALs is their small size. For bigger circuit design complex PLDs (CPLD) devices were introduced. Conceptually they are composed of a set of PALs (or GALs) integrated in on IC and interconnected by a programmable routing infrastructure that enables the creation of arbitrary connections between the PLDs.

[2]LUT - Look-Up Table

Field Programmable Gate Array (FPGA) designs are an alternative to ASIC design that can dramatically reduce manufacturing costs for low-volume IC circuit production. The idea behind FPGA design is to provide a functionally rich prefabricated chip which can be customized on demand. The user simply configures the already existing interconnect resources to suit its application needs. An FPGA can be seen a regular array of cells - much like a gate array - interconnected by a flexible and customizable routing architecture. The FPGA cells are more complex than standard cells, however most of the cells are identical. Figure 2.2d shows schematically the design of an island style FPGA design. The FPGA cells can be seen as memory blocks which can be configured to remember the truth table of a logic function. The given the input the cell "looks up" for the corresponding output in the stored table and passes it to the output. The big advantage of FPGAs is that there is no need user specifications to fabricate the unprogrammed FPGA. Hence, it is cost effective to produce a large volume of generic (unconfigured) FPGAs. Moreover, from the user perspective an FPGA is an ideal platform for IC prototyping especially in the context of re-configurable FPGAs which can be erased and re-configured to implement another circuit at any time during its lifetime.

2.3 CMOS Technology and Its Limits

The complementary metal-oxide semiconductor (CMOS) technology was, historically, a good choice for circuit design, due to the lower power consumption achieved by leveraging the complementarity of the n and p MOSFET devices. This enabled the creation of logic circuits with virtually no power consumption during the off-state[132]. Moreover the gate-dielectric isolates the logic stages which enabled easier circuit designs (relaxing the loading effect constraint) with large fan-out for a single MOSFET.

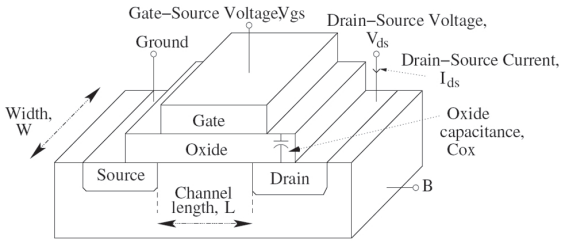


Figure 2.3: MOSFET Structure

The standard structure of a MOSFET device, presented in Fig. 2.3, comports three layers: the metal gate electrode, the gate dielectric, and the semiconductor substrate. Historically, the material used for the gate electrode was poly-silicon, for the gate dielectric silicon-oxide, and silicon for the substrate. Nowadays the metal gate electrode material was replaced by polycrystalline silicon and the silicon oxide dielectric by materials with a high dielectric constant (high-k), like Hf and Zr[17]. The source and drain electrodes are formed by doping the substrate with a material providing (n-type) or accepting electrons (p-type). The source-drain doping has to be opposite than the doping of the channel under the gate. A high voltage on a N-type MOSFET gate creates a bridge between the source and the drain, while for P-type MOSFET the channel between source and drain conducts at a low voltage on the gate.

The main parameters, characteristic of MOSFETs, are:

- oxide capacitance, C_{ox}, the capacitance between the gate and the substrate;
- gate-source voltage, V_{GS}, is the voltage between gate and source that controls the transistor switching;
- drain-source voltage, V_{DS}, the voltage between drain and source;

- threshold voltage, V_T, is the minimum voltage at which the transistors switches;
- drain-source current, I_{DS}, is the current flow when the transistor is switched on.

For the last 40 years MOSFET devices scaled based on a set of simple rules, one of which is constant-field scaling (CFS)[61]. According to CFS if all the dimensions as well as voltages of a MOSFET device are scaled with the same factor, the electric field and the current density remain constant. For logic circuits this results in doubling the integration density, while speed increases by the scale factor and the power density remains unchanged. But as feature size got smaller and smaller, different properties, overlooked by this simple scaling rule, had to be taken into account. In [61] the authors classified the challenges encountered by CMOS scaling today according to 5 categories:

- Physical challenges — refer to side effects of scaling, like parasitics, off-state leakage and doping effects;
- Material challenges — refer to the inability of finding the chemical compounds able to counter-act some of the physical challenges, like better insulators to reduce parasitics and conductors to improve the performances;
- Power-Termal challenges — refer to the increase in power consumption and heat dissipation per chip, which continuously increase due to the exponential increase of the integration density;
- Technological challenges — refer to inability to scale at the same pace the lithography-based fabrication processes; thus wafer diameter increased over time to maintain the productivity but the patterning steps became more and more complicated , and is expected that optic-based fabrication would not be feasible for future technology nodes;
- Economical challenges — refer to the IC production costs. As the complexity of the fabricated chips increase, the testing cost start to dominate the circuit fabrication cost. Moreover, as our capability of handling the fabrication process decrease the testing becomes even more costly and complex. At the same time, the foundry cost increased exponentially. These reasons drive the industry to question the affordability of this technology from a purely economical point of view.

To overcome these challenges, much of the scientific community embraced the "equivalent scaling" idea[73], and during the last 10 years they have tried to pin down the next best technology - after CMOS. The following section (Sec. 2.4) reviews some of the most promising alternatives to CMOS.

2.4 Emerging technologies

The deepening of CMOS problems at the nano-metric scale threatens the future of integrated circuits industry. Today, a large array of possible ways of scaling past Moore's law are under scrutiny. The purpose of this section is to review some of the most promising approaches to better understand the technological context that influenced the work presented in this thesis.

2.4.1 Taxonomy

In [74] ITRS proposes a classification of the emerging research devices according to a taxonomy of the principal technology layers that interact for creating a fully functional system. Figure 2.4 presents these layers starting from the lowest physical level, the state variables, up to architecture level passing through materials, devices, and data representation levels.

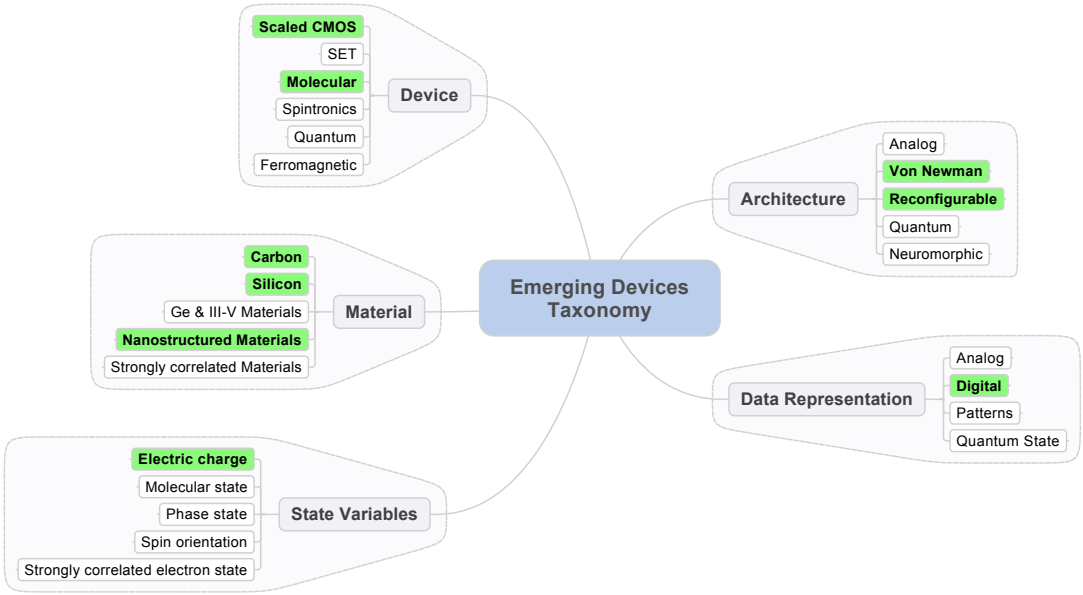


Figure 2.4: Emerging Circuits Taxonomy.

State Variables refers to the physical phenomena that give the intrinsic characteristics of the device and enable computation based on a number of discrete states. The current technology makes use of the electric charge as state variable, relying on charge or voltage state of a node in CMOS logic. A number or research projects focus specifically on finding an alternative set of state variables for creating computational devices. Some examples are:

- Molecular state — relies on particular molecular configurations that can be changed over time. The rotaxane molecule[22], bistable catenanes[163], and molecular quantum dot system[135] are some examples of molecules engineered as building blocks for future molecular electronics[64].
- Spin orientation — exploits the intrinsic electron spin and magnetic moment. In [74], two types of spin transistors are presented: spin-FET, that operates by precession or de-phasing of polarized carriers in the channel; and spin-MOSFET, that use relative magnetization configurations to modify the output current. Moritz et al. proposed a different approach by showing that complex logic devices can be built using spin-waves[151].

Materials refers to the choice of particular chemical substances, molecules, polymers for their physical properties that support different types of devices and functional schemes. During the last years research efforts were made to better understand the advantages and limitations of other materials besides silicon (e.g. carbon, high-k materials, superconductors). Moreover with the advances in nanotechnology and molecular engineering nano-structured materials like silicon nanowires, carbon nanotubes, and graphene became some of the most promising materials for supporting the circuit scaling to few nanometer ranges.

Devices. From the device perspective three principal axes can be identified[74]: one focusing on scaling CMOS to its ultimate limits; the second one using new charge-based devices; and the third one striving to reinvent integrated circuits all over again by using completely new devices and physics (not charge-based). The principal pillars for supporting the ultimate CMOS scaling are devices like finFET[71], unconventional transistors, CNT FETs[29], graphene nanoribbon FETs[56], NW FETs[3], etc. Single-electron transistors (SET)[169], tunnel-effect transistors[133],

and spin transistors[168] are the principal electric charge-based devices that can potentially replace CMOS. As for the third axis, we can cite collective spin devices[18], moving domain wall logic[1], molecular devices like the ones presented in [64], and magnetic quantum cellular automata[38]

Data Representation represents the way information is encoded for computation. Besides the classical analog and digital data representation, there are some new approaches like patterns[196], quantum state[6], probabilistic data encoding[124].

Architecture level is the highest technological level proposed by this taxonomy. It spans geometrical integration (2D, 3D), physical structure (regular and heterogeneous), connectivity (GALS, fan-out, signal distribution), reliability, logic implementation and data storage, application-specific and reconfigurable designs, and computing machinery (vonNeumann, neuromorphic, quantum, etc).

2.4.2 Requirements for Competitiveness

ITRS [73] proposed a number of requirements that should be meet for a new fabric to be competitive. These requirements include:

- inversion and flexibility;
- isolation between the input and the output of each device;
- logic gain, the output may have a fan-out bigger than one;
- logical completeness, ensuring that any logic function can be implemented and thus eliminating the need for additional supporting circuitry;
- self-restoring/stable for ensuring the signal quality;
- low cost manufacturing for increasing the productivity and further decreasing the cost per device ratio;
- reliability ;
- performance.

Based on these requirements and the maturity of today's CMOS technology, it was stated[74] that CMOS designs are difficult to replace by any new technology especially for binary computations based on the von Neumann model. But the new charge-based devices either CMOS-like (e.g. SiNW FETs) or not appear to be the best candidates for continuing the incremental evolution of circuits. Break-through devices based on new state variables, are to be better understood for building novel architectures leveraging their features and eventually breaking the current incremental evolution process.

2.4.3 Some Fabric Propositions

Quantum-dot cellular automata (QCA)[176, 100],Tour's Nanocell[177], Markov Random Network (MRN)[7, 124], and Nanowire Crossbar-based Fabrics (NCF) [60] are a few approaches for building computational structures bridging the gap between the need for powerful, energy efficient, and small circuits and the intrinsic limits of the current integrated circuits. They exploit novel state variables (QCA), random device organization (Nanocell), probabilistic data and circuits (MRN), unidimensional structures (NCF).

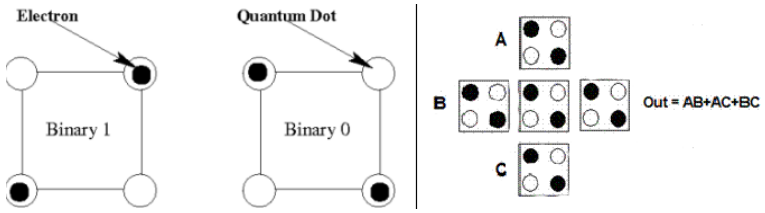


Figure 2.5: Left, QCA cell polarizations; Right, QCA majority gate

2.4.3.1 Quantum-dot cellular automata

Quantum-dot cellular automata[176, 100], introduced in 1993 by Lent et al., and fabricated in 1997 represent a completely new approach for implementing circuits. Based on quantum interactions between electrons, it provides an alternative to current CMOS technology.

A QCA cell can be seen as a set of four charge containers or dots positioned at the corners of a square. Each cell contain two mobile electrons that can quantum-mechanically tunnel between the dots but, by construction, cannot move between cells. In Figure 2.5 (left), we show an abstract view of a QCA cell with the two possible states binary 1 at left and binary 0 at right. The state of each QCA cell is influenced by the state of the neighboring cell. So basically, if we place the cell at left in Figure 2.5 next to the cell at right, the last one will change its state to a binary 1 value. This way a set of interesting computational devices can be obtained (ex. majority gate[127] in Figure2.5 on right).

In [128], the authors presented a FPGA-like architecture using these revolutionary devices. Some of the particularities of this approach compared to traditional FPGA are: *a)* instead of implicit latching of signals QCA FPGA use QCA wires' self-latching capabilities; *b)* the clock for QCA FPGA has four phases instead of just two in traditional circuits; *c)* the use of a QCA wire loop for storing state.

In 2006, a PLA architecture using QCAs was proposed[67]. This PLA fabric is reconfigurable and defect tolerant. The basic building blocks used in this approach are PLA cells made up by one AND gate, one OR gate and one select bit. The AND and OR gates are obtained by constraining a majority gate to perform the desired function. The select bit is used to indicate if the PLA cell acts as a wire or as a logic gate. The PLA logic is build up by using 2 logic planes one AND plane followed by an OR plane. These logic planes are made up of PLA cells which act as AND gates or OR gates according to the position of the two gates inside the PLA cell. The issues of configurability and fault tolerance were also addressed in the same paper. Also the nature of the clocking circuitry and structure were discussed.

However, even though the research in the area of QCA started almost 20 years ago the field is still young, and the published works are just pointing the needs for further research in order to render these technologies a veritable candidate to replace "current flow" based technologies.

2.4.3.2 Nanocell

One of the first proposed computational architectures that make use of nanoelectronic devices is Tour's Nanocell[177]. This fabric is conceived to harness the random nature of nano scale devices by randomly depositing very small conductive particles of gold or platinum on a substrate and then adding molecules having NDR properties to each of these particles. Thus obtaining a random network of switching nanodevices, namely Nanocell shown in Figure 2.6. This network is then trained to perform the desired function using a genetic algorithm. The building block for computational fabrics, using Nanocell approach, are presented in [68].

The main advantage of this approach is that it has inherent fault tolerance by harnessing the random nature of nanotechnology instead of trying to create some order. However, the huge amount of computational resources needed to train a Nanocell renders this approach not scalable for creating large logic arrays.

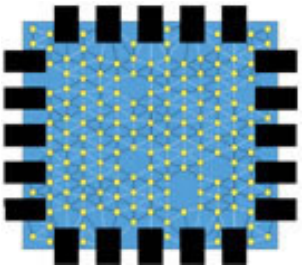


Figure 2.6: Example of a Nanocell Tile

2.4.3.3 Markov Random Network

Researchers at Brown University proposed a probabilistic design method for nanoscale circuits based on Markov Random Fields (MRF)[7, 124]. Independent from the underlaying technology and the presence of defective devices, this computational fabric relies on a graph of stochastic variables, a belief propagation algorithm, and the Gibbs energy distribution[7].

In [124], the authors have shown that CMOS circuits implemented using this methodology can be noise tolerant, feature that can be used to lower the power consumption by lowering the supply voltage.

However, in the case of the CMOS designs the number of devices needed is larger than the one required by classical CMOS logic gates for equivalent functionality. To address this issue the authors propose a carbon nanotube-based implementation, which improves the density but it challenging from a technological point of view. Besides these technological issues, another limitation of this design methodology is the lack of tools for logic and physical synthesis.

2.4.3.4 Hybrid nano/CMOS Nanowire Crossbar Fabrics

Amongst the most promising emerging devices are the Si-nanowire FETs[75] "[...] because of higher on-current conduction due to their quantum nature and also because of their adoptability for high-density integration including that of 3D". A number of fabric structures, relying on these devices, are currently under study, e.g. NanoPLA, NASIC, etc. At the beginning of 2011, Nature reported the first successful fabrication of such a fabric[192]. The experiment showed the design, the manufacturing, and the use of programmable nanowire-based logic arrays as computational primitive for nano-processors. These nanowire arrays, named crossbars in literature, consist of two sets of orthogonally placed nanowires having configurable FET devices at the crosspoints.

Another very promising device is the memristor, which was showed experimentally in[167]. The memristor is a passive device with a resistance that increases or decreases according to the current flow direction, and remains unchanged when there is no current flow. Nanowire crossbars with memristors at the crosspoints are another structure that show very desirable characteristics. In [189] the authors show the experimental results of fabricating a hybrid nano/CMOS reconfigurable structure that use the memristor-crossbar for signal routing and a CMOS layer for logic implementation.

These two success stories, built on the last fifteen years of research, provided insights into the theoretical and practical aspects of semiconductor nanowires and carbon nanotubes. Simple devices and logic gates were realized experimentally in the past and paved the way to this notable success. From the architectural point of view a number of different fabric structures were proposed and are currently under investigation. The following sections of this chapter focus on the principal characteristics of these fabrics, in terms of fabrication process, defect/fault tolerance methodology, the integration with CMOS complementary circuitry, and the different logic implementation strategies. Moreover, four of the most important fabric contribution are reviewed for a comprehensive understanding of the field.

2.5 From crossbars to digital circuits

To continue integrated circuit evolution according to the trend imposed by Moore's law, past the CMOS technology limits, we have seen that a number of alternative technologies are under investigation. Amongst these, the crossbar-based nano-fabrics are some of the most promising candidates. They tackle the impossibility of arbitrary placement and routing of nanoscale structures by using regular arrays of crossed nanowires with functionalized cross-points. These arrays are structured, conceptually, like traditional reconfigurable architectures (FPGA / PLAs). The purpose of this section is to show the principal characteristics of these nano-fabrics for digital circuit implementation. The section starts, Sec. 2.5.1, by presenting different fabrication alternatives for building nanowire crossbars. Sec. 2.5.2 presents one of the most important issue of these structures, the high-rate of defects, along with some techniques proposed for creating reliable circuits. Sec. 2.5.3, shows different ways of implementing digital logic using these supports. Since all these propositions use CMOS circuitry for different purposes, Sec. 2.5.4 reviews the main approaches for interfacing between the nanoscale and CMOS structures. Sec. 2.5.5 briefly details four of these fabric propositions focusing on their respective differences.

2.5.1 Fabrication process

For the fabrication of regular nanowire arrays, the traditional lithographic process cannot be used mainly due to the small feature size and to the need to finely control the fabrication parameters (the number of NWs, the pitch, the NW diameter). To address this problem three new fabrication methods have been proposed:

- NW growth and alignment in-situ
- NW growth with ex-situ alignment
- Patterning and Etching based on Nano-lithography

2.5.1.1 NW growth and alignment in-situ

This method is used for the fabrication of NW arrays by growing aligned nanowires using chemical auto-assembly on the target substrate. Guiding techniques are used to orient the growth process. Substrate or template guiding[62, 153], electric field guiding[48], and gas flow guiding[105] are some examples of guiding methods. The growth, resulting from catalytic processes, depends strongly on the chosen catalyst which has to be compatible with the substrate and the auto-assembly temperature. Moreover, for successful NW alignment, the catalyst has to be placed as aligned dots with the same diameter and inter-dot distances.

2.5.1.2 NW growth with ex-situ alignment

This method uses two independent steps: the nanowire growth, and their placement on the target substrate. Techniques like Vapor-Liquid-Solid[16] can be used for growing NW forests. Then, different techniques can be employed for their alignment and on-substrate placement, like Langmuir-Blodgett[21, 188], liquid or electric field guided alignment[190, 104], organic auto-assembly[78], or contact printing[76]. The principal challenge of this technique is the control over the NW position and orientation on the target surface and, thus the capacity of finely tuning the pitch and the NW lengths. This fabrication procedure is more flexible than the NW growth and alignment in-situ with respect to the materials used for the growth process.

2.5.1.3 Patterning and Etching based on Nano-lithography

Compared to the bottom-up techniques, presented in the previous paragraphs, this fabrication process is a top-down method relying on patterning and etching, similar with the traditional lithography. The patterning step engraves the pattern on the substrate using Nano-imprint lithography

(NIL) [148], or Superlattice nanowire pattern transfer (SNAP) [182], while the etching step is similar to the standard lithographic etching process. For patterning, the NIL technique uses the mechanical deformation by pressing the masks on the substrate. The simplicity is the principal advantage of this approach, since it enables low-cost patterning compared to standard photolithography. The SNAP patterning method is compatible with standard lithographic process and enables the creation of large arrays of NW from a large palette of materials (metals, insulators, semi-conductors). The main challenges of these two techniques are the mask degradation over time and the eventual NW performance degradation due to the etching process.

The principal consequence of using these fabrication techniques is the impossibility of arbitrary placement devices and routing of wires, that permits, nowadays, the creation of high-performance application-specific circuits. To cope with this limitation, the research efforts in the field focus on highly regular fabric and circuit designs based on the replication of virtually identical NW crossbars.

2.5.2 Working Circuits on Unreliable Technology

Besides the regularity of assembly, the NW fabrication policies introduce another challenge for computational fabric designers, the high-rate of defects. The fabrication defects for nanoscale architectures are expected to be orders of magnitude greater than for CMOS technology, thus each fabric proposal should treat this problem seriously if it is to be successful. This section reviews some of the probable causes of defects, introduce the terminology, and compare the approaches taken in the context of some crossbar-based fabrics. The readers interested on circuit test engineering and defect-tolerant nanoscale computing are directed to [58] and [158] for further details.

2.5.2.1 Definitions

A *physical defect* is a physical problem inducing definitive changes in the fabric structure. These changes appear as a result of the fabrication process (manufacturing defect) or due to a permanent device failure (PDF) during the lifetime of the system. In the context of nanoscale architectures, the principal source of physical defects is the bottom-up fabrication process[55], proposed as a cheaper alternative to lithography. Some examples of physical defects are: broken nanowires, stuck open/short FETs.

Process variations are another kind of physical problems to which the systems are prone to, they result due to non-uniform conditions during the fabrication process causing the electrical parameters, such as resistance, threshold voltage, to vary from a device to another. In nanoelectronics, these variations are caused by the bottom-up fabrication techniques and the small features of the devices.

In contrast to a physical defect, a *fault* is an incorrect state of the system due to physical defects, environmental conditions, or improper design. Faults can be:

- **Permanent** — mainly due to manufacturing defects and PDFs.
- **Intermittent** — faults may appear periodically, one cause can be the process variations. They can occur for certain input parameters as a result of large unexpected delay. At nanoscale, the doping variations of the nanowires (NW) used for FET channels or the length variations caused by the metallization process used to separate FETs one from another are some of the main causes of intermittent faults.
- **Transient** — faults occur during the lifetime of the system mainly due to temporary environmental conditions (radiation, crosstalk). The circuit having this kind of faults are not permanently damaged, the faulty behavior disappearing once the noise source disappears.

In this context, *defect tolerance* can be defined as the ability of a system to operate correctly in the presence of physical defects (manufacturing defects or PDFs) while *fault tolerance* can be seen as the ability to operate correctly in the presence of permanent, intermittent or transient faults. Both defect and fault tolerance techniques require redundancy to overcome the problems

within the system. This redundancy may be classified in *space-redundant*, *time-redundant*, or *information-redundant* according to the way it is used in the system. Modular redundancy (like Triple modular redundancy (TMR), n-modular redundancy (NMR)), NAND multiplexing and reconfiguration are space-redundant techniques because they are based on the spatial replication of functionality. Backward error recovery, and re-execution are time redundant techniques. Error correcting codes are information redundant techniques because they exploit the information space to tolerate the faults in the system.

To achieve defect/fault tolerance in a system a number of techniques are used, these can be classified according to the role they play in the system. Thus *Fault detection* can be defined as the set of techniques used to identify the faults present in a system. Once the faults detected *fault isolation* techniques are used to limit their impact on the system. Finally *fault masking or fault avoidance (reconfiguration)* techniques can used to render the system behavior correct.

The level of abstraction at which a defect/fault tolerance technique is applied is very important for the overall effectiveness of the design. The *physical device level* is the lowest level of abstraction and is concerned with the specific properties of the nanoscale device which make it tolerant to different types of defect (for example the Quantum cellular automata (QCA) cells are tolerant to small alignment differences). The *architectural level* is concerned with the way the devices are assembled together to produce usable circuits. The *application level* is the highest level of abstraction, fault tolerance techniques at this level are concerned with the features that make the application execute correctly in the presence of defects and faults on the underlying computing system (one example of such fault tolerant technique is the re-execution used to tolerate transient faults in certain systems).

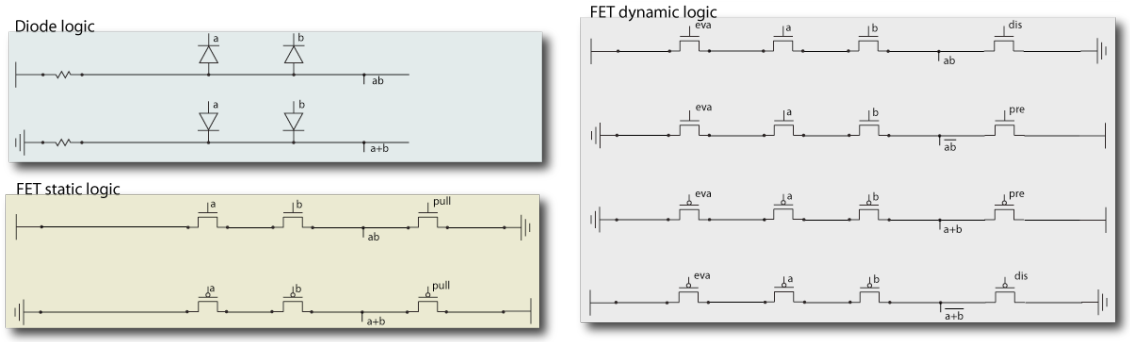
The *yield* of a manufacturing process is a quality metric that represents the fraction of fabricated chips that has no defects[59]. In earlier technology nodes, the yield problems were confined to the manufacturing area and they were solved by improving the fabrication process. As the designs approach the nanometric scale, the principal sources of yield loss are the systematic pattern variations, which affects a set of chips in the wafer, and manufacturing limitations which induce design-specific problems. As the technology further evolved, another source of yield loss appeared, namely, the physical parameter variation associated with the statistical variations in doping, channel length, etc. For the nanoscale designs it is considered that the manufacturing process improvements, and the related traditional techniques, such as Design For Manufacturability (DFM), are not enough to tolerate the predicted high-rate of defects. The yield has become a new tradeoff factor that should be considered by the architecture designer, besides the speed, area and power. Hence, the corrective procedures used to improve the process yield will impact the total cost of the design, and there is a point where achieving 100% yield is impossible due to cost constraints. Moreover, even if the process yield of a specific architecture is high enough to be profitable, this does not guarantees that the circuit will operate as expected during its lifetime, if we consider the existence of intermittent and transient faults. Thus we start worrying about the reliability of the system. The *reliability* of a circuit is associated with its capacity to sustain correct operation considering the occurrence of faults[36].

2.5.2.2 Fault tolerance at nanoscale

Since the defect and fault densities of self-assembled structures are projected to be orders of magnitude greater than in traditional CMOS-based structures, it is very important to investigate the fault models studied in the context of these architectures. In Table 2.1 we can see the fault models used for the architectures presented in Chapter 2.5.5 along with the defect/fault distribution and the fault tolerance techniques proposed by the designers of each architecture. It can be easily seen that 3 from the 4 architectures use reconfiguration for fault tolerance and that just the defects/-faults uniformly distributed are studied. The defect/fault types considered are based on the same abstract fault model from which only a subset is covered. ◦The reason behind this can be found in the limitations of the reconfiguration approach to fault tolerance and the limited expression power of the tools used to model and evaluate these architectures.

Table 2.1: Crossbar-based fabrics and associated fault-models

		FPNI[160]	CMOL[165]	NanoPLA[35]	Nasic[115]
Fault tolerance technique		reconfiguration	reconfiguration	reconfiguration & roll-back recovery	self healing circuits
Defect/fault distribution		uniform	uniform	uniform	uniform & clustered
Permanent	broken NW	•	•	•	•
	bridged NW	-	-	-	•
	stuck open	•	•	•	•
	stuck close	-	-	• Treated as nanowire deffects. Unusable horizontal and vertical nanowires[34]	•
	Nano/CMOS Interface	-	• treated as a defective CMOS cell	• stochastic decoder	•
Intermittent		-	-	-	•
Transient		-	-	• rollback recovery[117]	•



(a) Diode logic and Static FET logic

(b) Dynamic FET logic

Figure 2.7: Different logic styles

2.5.3 Logic Implementation

Another interesting characteristic of crossbar fabrics is the way they implement the logic. NanoPLA[32] uses NOR-NOR arrays build with an diode-based OR stage (like in Figure 2.7 top left) followed by an FET based inversion/buffering stage stage. CMOL[166] uses NOR-NOR logic which is built using latching switches (which is basically a switch with the capacity stay in a certain state (ON/OFF) once configured in that state) for the OR stage followed by an CMOS inversion stage based on an CMOS inverter. FPNI[160] uses the nanoscale crossbar just for routing and thus uses CMOS custom gates for logic implementation. NASIC[184] propose to implement different logic styles, and even a combination of different styles based on FET based logic designs, in Figure 2.7 (left bottom and right) we can see some of the basic constructs used for building static and dynamic NASIC nano-tiles. ◦In conclusion even if each architecture uses different nanoscale devices for implementing the functionality all these nanoscale devices can be abstracted to a set of generic primitives (like diode, switch, etc).

2.5.4 Nano/CMOS Interface

The first architectures using nano scale devices will be hybrid architectures having nano and CMOS parts. Hence, for designing computational fabrics using nanoelectronics, another important aspect is the role played by CMOS technology in the resulting circuit. The CMOS can be used, depending on the architecture, for logic, for interconnect, clocking, power, ground, configuration wires, I/O. To use CMOS devices and nanoelectronic devices on the same circuit an interface between the two has to be made, for addressing the nanoscale wires from the microscale ones. Directly driving the

nanowires with microscale wires is not feasible because the large feature size of the lithographic scale wires will negate the potential benefits of nanoscale devices. To solve this problem, different approaches where proposed. In nanoPLA[31], the use of a stochastic decoder is proposed by DeHon in [33]. In CMOL[166] and FPNI[160] a 3D structure of the fabric is used. In this cases, the nano layer is on top of the CMOS layer and connection pins provide the communication between the two.

2.5.5 Crossbar-based Fabrics

This section summarizes four of the current proposals for building computational fabrics using nano-electronic devices.

2.5.5.1 NanoPLA

In 2004, André DeHon[35] proposed a PLA-like architecture, named nanoPLA, which implements a NOR-NOR logic style based on nanoscale crossbars having programmable diodes at crosspoints. To overcome the limitations of diode logic, the authors propose to insert rectifying field-effect stages between diode stages. To solve the nano/CMOS interface problem DeHon proposes the use of a stochastic decoder[33] for addressing the nanowires.

The reconfiguration is used as the main technique to tolerate permanent defects in the NanoPLA architecture[34]. But in[117] the same authors also propose a fine-grained rollback recovery technique for tolerating transient faults.

2.5.5.2 CMOL

Likharev and Strukov[166] introduced CMOL, a hybrid crossbar-based architecture which uses 3D integration (nano-on-CMOS) for solving the nano/CMOS interconnect problem. Based on this approach the authors proposed a number of architectures like memories[166], reconfigurable architectures similar to cell-based FPGAs[165], neuromorphic networks[166]. For this study, we focus on the reconfigurable architecture proposed in[165], named CMOL FPGA, which implements a NOR-NOR logic style using the nanoscale crossbar for OR logic and interconnect, and CMOS cells for inverters, latches, etc.

The CMOL FPGA architecture uses reconfiguration as a defect tolerance technique, the principal type of defect studied is stuck-open crossbar junction, but since this kind of defect manifests itself in an unusable CMOS cell, it is modeled as a defective CMOS cell, this implies that the nano/CMOS interface defects, and broken NW are somewhat covered too.

2.5.5.3 FPNI

At the beginning of 2007, Snider and Williams, at HP labs, introduced a generalization of CMOL circuits, namely Field-Programmable Nanowire Interconnect (FPNI)[160]. The FPNI architecture trades of some of the advantages of CMOL, such as speed, density, defect tolerance, in exchange for easier fabrication, lower power dissipation and easier routing. FPNI approach is more like traditional cell-based FPGA where CMOS cells implement arbitrary logic. The difference being the signal routing that is done entirely using the nano layer. Thus achieving better densities than traditional CMOS only FPGAs, which dedicate a large part of their area just for interconnect.

Like for CMOL the reconfiguration is used to tolerate the defects of FPNI circuits. The stuck open crossbar junctions and the broken nanowires are the defect types studied.

2.5.5.4 NASIC

Moritz et al. proposed a hierarchical nanofabric architecture that can be tuned towards an application domain. The basic building blocks of this proposal are the nanotiles, built up as a grid of silicon NWs (SiNW) having the junctions acting as FET. NASIC architecture has raised many interesting issues in designing nano/CMOS integrated circuits. Some of these are: *1) Latching on*

the wire[114] to build pipelined circuits without the use of explicit latching (which implies the use of registers); *2*) In one of the latest papers [184] Moritz et al. showed the possibility to *combine AND-OR and NOR-NOR logic styles* to obtain denser logic; *3*) Furthermore in [120] they have shown the possibility to design nanoscale logic circuits using only *one type of FET* at the nanoscale with no degradation of performance, defect-masking, or density, meanwhile reducing the manufacturing requirements. *4*) While in the first stages of the development[185] the fabric was conceived as being reconfigurable, in the later papers[183] the authors *renounced at the reconfigurability* to ease the manufacturing process. But, on the other hand, without a reconfigurable fabric, another approach was needed to render the fabric fault tolerant. *5*) To solve this problem, in [115] Moritz and al. propose structural redundancy based techniques to render NASIC a self-healing circuit architecture.

2.6 Summary

In this chapter we have briefly reviewed the integrated circuits history, the we presented the limitations faced by the CMOS technology. These limitations, acknowledged by the experts in the field, drove the search for new inventions that will enable future evolution of the integrated circuit industry, past the end of Moore's law. In Sec. 2.4 a taxonomy for emerging technologies was presented, some of the requirements imposed on these future technologies were shown, and some novel computational fabrics were described. Sec. 2.5, focuses on the crossbar-based nanoscale fabrics and reviews their fabrication process as well as some of their most important characteristics, such as defect/fault tolerance, logic implementation, nano/CMOS interface. This section ends by presenting four of the most promising crossbar-based fabric propositions.

3

Bridging the Gap Between Applications and Technology

As the complexity of integrated circuits increases their design puts more and more pressure on the automated tool-flow used. This chapter introduces the reader to the field of electronic design automation(EDA). After a brief overview of the EDA field in general, the focus is drawn towards the physical design tool-flow in the context of nanoscale architectures. The design-space exploration problem is presented with a focus on the exploration of the application-architecture adequacy at the physical design level. This chapter ends reviewing the main requirements for a CAD toolkit in the context of nanoscale electronics.

3.1 Introduction

The ability to create and use tools enabled early humans to get to the top of the food chain by being able to accomplish tasks impossible for the human body. We have come a long way since then, and the tools we are using evolved along with us, helping to shape the world around us to suit our needs, broadening our understanding of the laws governing our planet and the universe as a whole. The tools always served as a bridge between our needs, our questions, our ideals and the real world. They are the common denominator between the caveman trying to catch its food with a stick, the scientist trying to understand the laws of physics using mathematics, and NASA trying to communicate with the Mars rovers using powerful computers.

Amongst all tools invented during the history of human kind, computers can be seen as one of the most impressing achievements. These machines supplement human intelligence by offering a number of capabilities well beyond human reach, like fast evaluation of complex mathematical problems and large information storage. They enabled the creation of real-time communication over large distances and the control of large robotic machineries used in our fabrication plants, amongst other things. Today electronic devices, computers, and information technology are integrated deep in our environment, changing our way of life. They have reached minuscule sizes and have become ubiquitous. They have transcended the stage of simple tools and have become part of our daily routine, they enhance our abilities and complement our intelligence helping us gaze over the limits of our bodies and minds.

Since the invention of the first computing machineries at the beginning of the XXth century, the electronics industry evolved at a tremendous pace, based on an orchestration of theoretical, technological, economical and societal advancements. But this evolution comes at the cost of ever

increasing complexity. Today's computers integrate billions of devices on a single component of just a few square millimeters. The manual design of such components is impossible, as is their fabrication. To cope with this complexity, a large number of software tools are used for automating the design, fabrication and exploitation of these components. These software tools are harnessing the power of today's computers to design and create tomorrow's computing technology.

The software tools used for computer-aided design in the case of electronic systems are dubbed electronic design automation (EDA). Their use range from printed-circuit boards to high-performance integrated circuits. As the integration density increases with Moore's law, and the circuit design complexity increases, more and more physical and technological issues need resolving with each new technology generation. Moreover, with the mass production of heterogeneous circuit structures like systems-on-chip that enable the creation of high-performance end-user products with numerous features, the human taste for technology gets more and more refined. The EDA industry challenge is keeping the pace with the growth of the circuit technology and with the public demand for high-end features. With the emergence of new circuit technologies the heterogeneity of the EDA market increases and the gap between the IC capacity and the design productivity widens (Figure 3.1).

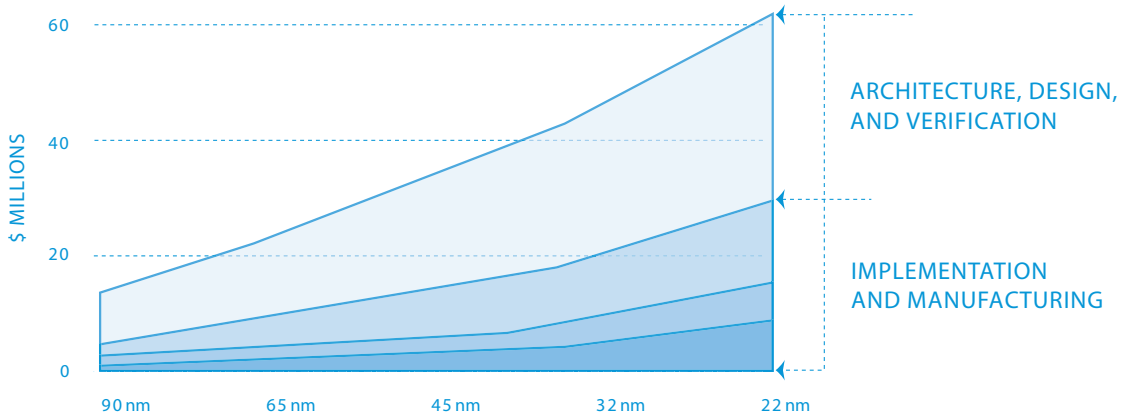


Figure 3.1: SoC development costs in terms of design and manufacturing. (adapted from: Cadence EDA360 Vision)

This thesis focuses on EDA tools for nano-scale digital ICs firstly because the digital ICs are the most prominent in the current IC industry and because their design process is completely automated. Thus the adoption of nanoscale technologies will have the highest impact on the design flow of digital circuits. Which does not tolerate the involution of the computer-based design towards manual intervention at any level of the design flow. In this chapter we will briefly present the main EDA techniques and the requirements imposed by the adoption of nanoscale technologies. For more details on the standard EDA tools the readers are referred to [98, 79]. For analog and mixed-signal circuit design automation readers are directed to [98, 54].

3.2 Electronic Design Automation - Overview

This section reviews the most important EDA tools used to practically transform ideas, expressed as algorithms, to reality, as integrated circuits.

3.2.1 System-Level Design

Today's integrated circuits are heterogeneous, assembling processors, memory, communication buses, and hardware accelerators in a single chip. This applies for either custom systems on chip

or high-end FPGA design. This heterogeneity provides the needed flexibility for creating high-end adaptable systems targeting a wide range of applications. The price payed for this gain in flexibility is the increased complexity of the design. To achieve successful integration of all the embedded components, a complete tool chain from application to the final product is needed. Hardware/software co-design techniques represent the core of the system-level design toolchain, and they try to find the trade-off between the application and hardware constraints. These design tools rely on system-level and behavior-level synthesis, going from high-level, abstract, application specifications (in C or SystemC) towards the target hardware platform. Software and hardware task are extracted from the high-level descriptions, along with communication protocols and interface wrappers that connect the different components. The tools used for achieving these results are generically dubbed electronic system-level (ESL) tools.

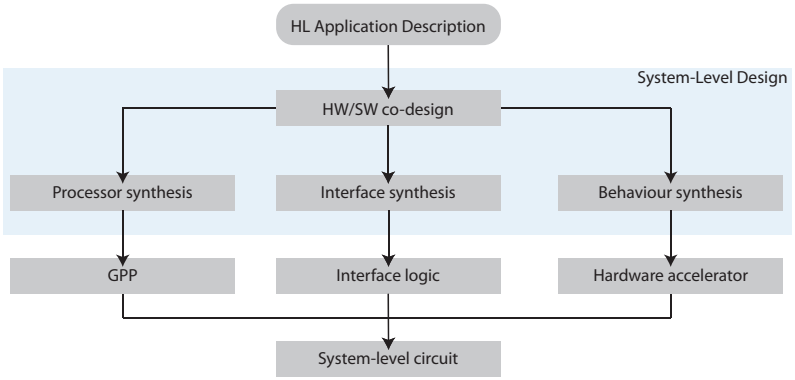


Figure 3.2: Typical system-level design flow

A global view of the system-level design automation flow is presented in Figure 3.2. The main component, HW/SW co-design, assures the partitioning between the HW and the SW task and includes three distinct operations :

- Processor synthesis step which instantiate specific software programmable IP cores based on parameters computed with respect to the target application domain.
- Behavior synthesis, or high-level synthesis (HLS) step is responsible for producing register-transfer-level (RTL) design from functional descriptions. These RTL designs are typically co-processors (accelerators) highly optimized for application-specific operations.
- Interface synthesis is the step responsible for the correct and efficient communication between the principal components of the SoC via wrappers responsible for creating adapting the component specific input/output data to the system communication protocol.

Interested readers can refer to [37] for a taxonomy of ESL design that identifies and classifies the large palette of available tools. A more detailed presentation of the field with an emphasis on the best practices can be found in [108].

3.2.2 EDA Design Flow - Overview

One of the most important parts of the EDA industry is the automated design of single-purpose processors and accelerators using semi-custom or programmable logic device (PLD) technologies. An overview of the typical steps of the design flow are presented in this section, following the elements presented in Figure 3.3. The flow can be divided into two broad categories (synthesis and physical design) with verification and testing as transversal activities present at any step of the flow.

The design flow, presented in Figure 3.3, starts with the application description in terms of the required functionality, and with the platform specification. The platform specification

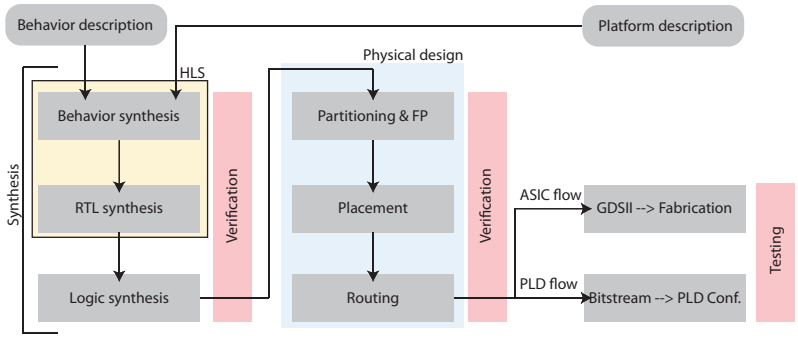


Figure 3.3: Classical EDA design flow

encompasses structural representation, design rules and library gates in the case of an ASIC flow. For a PLD-targeted flow the platform description represent usually an architectural specification in an architecture design language (ADL).

3.2.2.1 Synthesis

The application synthesis step of the design flow transforms the behavior description (usually given in a HL language like C) in an optimized netlist of architecture specific logic gates. This design step is typically decomposed into 2 distinct steps: high-level synthesis (HLS) and logic synthesis.

The HLS step is responsible for converting the functional application specification into cycle-accurate RTL design while separating the datapath from the control unit. The application specification is usually represented internally as a control data flow graph (CDFG) that is partitioned and mapped on datapath structures (functional units - ALU, multipliers; storage units - registers, memories; and interconnection units - busses, multiplexers), and control structures represented as state machines that manage the datapath operations at each clock cycle. The principal task of the HLS are:

- *Scheduling* is responsible for finding an execution order for computational operations. Different scheduling policies can be defined according to the application needs and the platform constraints. As soon as possible (ASAP) and as late as possible (ALAP) are two of the simplest scheduling policies. The first is trying to execute each operation at the earliest opportunity as opposed to the latest opportunity applied for the second one.
- *Allocation* determines the number of physical resources (functional units, registers, etc) needed for the correct operation of the design.
- *Binding* links the operations, variables, etc to the physical resources allocated during the previous step.
- *RTL optimizations* are then performed on the RTL netlist applying different compilation optimizations like constant folding, dead-code removal, code factorization, no-op removal, expression optimization, collapsing small operations, re-encoding state machines, etc. This step, also known as RTL synthesis, can then create a mapping between the design and RTL library gates, or directly map datapath components to PLD logic.

Logic Synthesis is the task of transforming circuit description into a format that can be executed on the target technology. It is typically decomposed in two steps: synthesis and technology mapping. The synthesis step is responsible for different logic optimization of the RTL synthesis results, like combinatorial and sequential optimizations. This step can be performed without knowledge of the target technology with objectives like the minimization of the total amount of

gates, reducing the logic depth of the boolean network. Espresso[147] and Sis[150] are two famous examples of boolean synthesis tools, the former optimizing two-level logic while the later addresses sequential circuit optimization. Technology mapping is the main task of technology-aware netlist optimizations. It transforms the boolean network into a network of logic cells provided as a library. Some examples of technology mapping tools are PLAMAP[19] for CPLDs, Flowmap[23] for FPGA designs. The logic synthesis area is considered to be mature in the context of the current technology, but that might be an overstatement for the emerging computing fabrics, since they introduce new challenges like defect-aware synthesis, variability-aware synthesis, quantum logic synthesis[155].

3.2.2.2 Physical Design

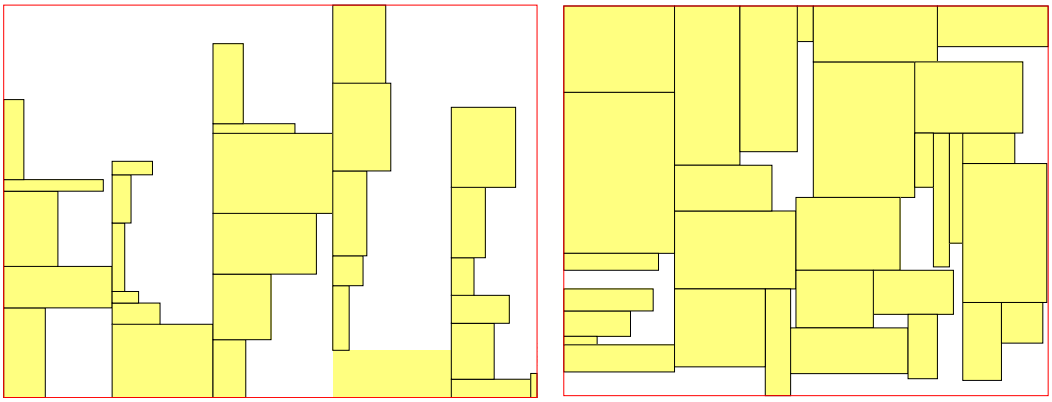
The physical design step is responsible for instantiating (in the case of an ASIC flow) or allocating (for a PLD flow) all design components along with their respective geometrical representation for creating the final IC layout (or configuration bitstream for PLDs). This means that each gate (as well as the other components) of the application netlist will be assigned a spatial location (placement) and then the interconnect signals will be reified using appropriate routing structures (physical wires are routed in the metal layers – for ASICs; PLD routing resources are configured – for PLD flow). Physical design has a direct impact on the circuit characteristics (performance, area, power, etc). The principal step of physical design are: partitioning, floorplanning, placement, and routing. Each of these steps are briefly reviewed in the following paragraphs. For more details the readers are advised to read [79].

Partitioning & Floorplanning are two optional physical design steps that are used mainly as a result of the overwhelming complexity of very large designs (integrating tens/hundreds of million of logic gates). Due to the exponential nature of the physical design, it is not possible to directly compute the layout of the entire IC as one chunk due to memory and computing power limitations. In these cases the circuit is decomposed in a number of blocks of manageable size that are placed and routed separately. This process is called partitioning, and is based on graph heuristics like Fiduccia-Matheysses[49] and hMetis[80].

If after partitioning the block sizes are still too large for the place & route routines the floorplan step can be used to further decompose the layout problem by exploring different layout alternatives for each block and choosing the best amongst them. Figure 3.4 shows the resulting surface covered and the block positions after (Figure 3.4b) applying a simulated annealing optimization over a TCG-S[102] encoding of the instance presented in Figure 3.4a. Some interesting floorplan techniques can be found in[97, 102, 63]. An in-depth survey of floorplanning techniques can be found in [79].

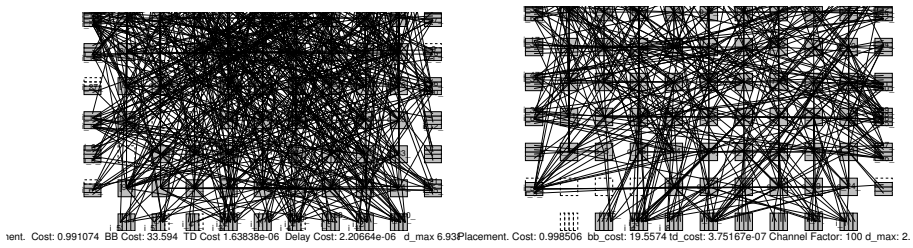
However it should be noted that partitioning the netlist into multiple blocks can have a negative impact on the resulting circuit characteristics (area, speed) notably if, during the partitioning process, the components on the critical path end up distributed across multiple partitions[24].

Placement During placement the physical location of the logic elements of the netlist is computed. In the case of an ASIC design flow, a non-overlaping embedding of the logic components into a 2D surface is computed, and thus it is similar to the floorplanning step. The difference being that, for placement, a large number of blocks with predetermined and fixed shapes are considered for embedding into the regions defined during the floorplan, which render the problem more constrained (than floorplanning). As opposed to ASIC placement, in the context of PLD, the placement is a mapping between the netlist's logic elements and the logic blocks available on the target reconfigurable architecture. The optimization goals during placement are minimizing the total layout area, minimizing the total amount of wiring required for the subsequent routing by placing connected block close together, maximizing the circuit speed by conveniently placing the blocks on the critical path, balancing the wiring requirements across the layout, avoiding cross-talk, etc. These goals may contradict each other, for instance minimizing the layout area may degrade the delay on the critical path. Due to the NP-hard nature of the placement problem,



(a) Initial floorplan (b) Optimized floorplan

Figure 3.4: Floorplanning using TCG-S representation[102]



(a) Initial Placement (b) Optimized placement

Figure 3.5: A FPGA placement instance optimized using VPR placer

heuristic approaches are taken for finding a trade-off between these contradicting optimization goals. According to the type of heuristic used, placement algorithms can be classified into:

- *simulated annealing-based*[30], is a metaheuristic used for finding a global optimum in large optimization spaces. It draws from the metal cooling process that uses a temperature cooling schedule to drive the molecular state of the system into a low energy configuration. This method proved to be very efficient for the placement problem of relatively small systems ($< 50K$ movable objects). One of the most known simulated annealing placers is the one integrated into the VPR FPGA toolkit[9]. TimberWorlf placement is one of the first world-known simulated annealing placement strategies[149].

- *evolution-based*[84], are the algorithms drawing their execution flow from an analogy with Charles Darwin's theory of natural evolution. As opposed to the simulated annealing strategy which iteratively tries to improve the current solution, the evolutionary algorithms start with a large population of different solutions (called chromosomes) from which the most fit chromosomes (based on a problem specific metric) are selected to survive for the next generation. The chromosomes are then transformed through genetic operators, typically crossover (combining two different chromosomes together) and mutation (changing one chromosome according to problem-specific rules).

- *force directed placement* is yet another placement strategy that draws from an analogy with the physical world. This time the analogy comes from electrical charged particles or spring systems. In these systems, each entity exerts a some type of force on the neighboring entities, and after a process of pushing and pulling the system comes to an equilibrium state (much

similar to the low-energy state in simulated annealing). FDP placer [82] is one example of force directed generic placer.

- *partitioning-based* strategies decompose the system into a number of smaller problems which are then partitioned using other methods. CAPO is one example of partitioning-based placer[125].
- *analytical placement*[13] is based on the observation that the placement problem complexity is considerably reduced if the modules can overlap. Based on this relaxation, a typical analytical placer just minimizes a function estimating the wire-length, then the overlap is resolved using force-directed techniques that will iteratively repel overlapping modules until the overlap is solved.
- *multilevel placement* uses a similar idea as the multilevel graph partitioning strategy hMetis[80]. The placement starts with a fine-grain problem that is successively coarsened by clustering and then un-coarsened toward a fine-grain placement solution. One example of multilevel placer is mPL6[15] that relies on a force-directed approach for placing in between the coarsening and un-coarsening phases.

Routing After placement, the routing step finalizes the geometric layout of the IC by interconnecting the logic blocks together according to the netlist. Routing is seen as the last step of a physical design flow before creating the GDSII[1] for fabrication in the case of an ASIC design style or generating the bit-stream to program a PLD. Usually the routing steps has to find a good tradeoff between optimization goals such as: wire-length minimization, critical-path delay minimization, power consumption minimization, improving manufacturing and/or testing, etc. The typical netlist for current IC contains tens to hundreds of millions of gates. In consequence during the routing step millions of nets have to be routed in an enormous search space composed of all possible paths a signal can use, which makes routing very computationally expensive in terms of time and memory. We have seen for other physical design steps, that the problem can be decomposed to make it tractable. This is also the case for routing, which is usually done into two steps: global routing, and detailed routing. The global routing step will generate a coarse-grain view of the routing, by typically coarsening the routing graph into channels(or routing regions) which are used as bins, then the nets are routed through these channels. After global routing, the detailed routing will actually find the real geometrical layout of each net within the assigned channels by either instantiating a physical wire (ASIC-context) or by allocating an existing routing resource (PLD-context).

The routing is typically studied as a graph problem, relying on different graph models like: grid graph, checkerboard graph, channel intersection graph or fine-grain resource graph. The routing problem then can be stated as finding a set of vertex and/or edge-disjoint paths for the nets in the netlist. This problem is approached either in a sequential manner by sorting the nets according to a specific importance metric (usually defined in terms of the net criticality) and then routing each one of them according to that order, or concurrently by solving an integer linear program associated to the routing instance. Due to the high number of nets, in the case of concurrent routing the problem is typically decomposed first into distinct routing regions and an integer linear program is solved for each region.

There is an extensive amount of literature related to the IC routing problem. Some of the most important approaches include: maze routing[77, 14], line-probe[112], shortest path-based[175], negotiation-based[110], Steiner-tree[191].

3.2.2.3 Verification & Testing

The principal focus of a typical EDA flow is the design transformations from abstract behavioral level specifications towards physical hardware implementation. As the synthesis process

[1] GDSII is an industry standard database format for IC layout exchange

progresses, the design is incrementally refined and transformed to finally match the target circuit organization. One important design process, orthogonal with the synthesis progress, is design verification. Verification spans all the implementation steps (see Figure 3.3, and checks if the design transformation altered the desired functionality of the application. Once the physical circuit is created, either ASIC or PLD design, it is then tested. In the case of ASICs, the testing checks mainly for manufacturing defects that alter the desired functionality. For PLDs, the testing phase checks for errors that can appear during the configuration. Verification and testing are essential for the realization of functional ICs, they are transcending the design automation steps and are complementing each other to assure and re-assure the designer during the design realization process. The following paragraphs briefly review some of the verification and testing techniques. Interested readers are directed to [98] for in-depth review of these techniques.

Verification is used after each synthesis step to catch design errors as early on the design flow as possible. In the case where a design error is not identified, it will propagate to the lower design levels, and even to the manufacturing process. As the design flow progresses, the design becomes more and more complex (due to the low-level details that are added), and as a consequence the verification process is more tedious and the designs flaws identified become harder to understand and fix. Moreover, if the errors make their way to the manufacturing process, the costs incurred are very high (mainly due to the high cost of the masks' fabrication, which will have to be replaced). Design verification can be done in different ways :

- *Simulation* of the circuit behavior based on mathematical models that replicate the circuit functionality and execute it with carefully selected input vectors. The simulation results are then compared with the expected results. In the case they are identical, the verification was successful. Once differences between obtained outputs and expected outputs are identified, the problem needs to be localized, isolated and fixed. Simulations can be devised at different abstraction levels (system-level, RTL, or gate level) and can be focused on identifying different types of errors (e. g.functional, timing). Since the number of all possible input vectors is an exponential function of the number of circuit inputs, exhaustive simulation is practically impossible. In consequence, only a subset of the input vectors are used, ideally a subset capable of detecting the maximum number of errors (high fault coverage).

- *Formal Verification.* As opposed to simulation, formal verification tries to prove the correctness of a circuit implementation based on formal inference methods, like model checking and equivalence checking. In the case of model checking the circuit is verified with respect to a known set of properties that it should satisfy. Equivalence checking tries to identify if the circuit under verification is functionally equivalent with a reference design that is known to be correct. Even though these two techniques offer mathematical proofs of the circuit correctness (result not achievable by simulation) these proof-based techniques do not scale for large designs mainly due to the combinatorial explosion of the state space.

- *Emulation* refers to the use of PLD boards for implementation of a prototype that is then verified by executing the design on the PLD and comparing the obtained results to reference results. This technique is much faster than the verification by simulation, to which is conceptually similar.

- *Post-silicon validation* The ultimate verification method, called post-silicon validation, uses real circuits that execute at full-speed, however this method supposes that the design is already functional (in part at least).

Testing While verification validates functional and timing aspects of the implemented design before manufacturing, testing refers to the process of carefully checking each fabricated circuit for manufacturing defects. Due to the high number of devices integrated on a single chip, testing is a tedious process, which can be even impossible for certain circuit structures. In consequence

testing is an integral part of the design methodology (design for test) which has to provide the necessary infrastructure for successful and cheap testing.

3.2.3 Technology CAD

Technology computer-aided design (TCAD) bridges the gap between the physical reality and the circuit design industry. By using simulation models tuned based on experimental results, TCAD tools offers invaluable results that span circuit design, device engineering, process development and integration into manufacturing. TCAD tools broadens our understanding of the material, processes and technology and the impact they potentially have on IC design. In [107], J. Mar identifies four principal application areas for TCAD:

- *Technology selection* refers to the use of TCAD-obtained results to select, reject, or narrow the focus of technological developments in an exploratory manner before investing in costly experiments;
- *Process optimization* is the use of TCAD tools to tune-up process variables, device parameters and manufacturing steps toward different optimization goals (performance, power consumption, manufacturability, etc);
- *Process control* provides theoretical models for diagnosis, optimisation and control of process related aspects during manufacturing.
- *Design optimizations* refers to the use of TCAD for device and circuit optimizations according to different objectives (cost, reliability, etc).

In the context where CMOS technology approaches its limits, the TCAD tools play an important role in understanding and optimizing any new material, technology, manufacturing process that will improve, and eventually replace CMOS.

More details on TCAD tools as well as a historical perspective of the field evolution can be found in [44].

3.3 Physical Design at Nanoscale

Nanofabrics are generally organized into: tiles, hypertiles or nanoblocks that correspond to clusters of PLAs and basic cells or hypercells. The partitioning techniques used to define such blocks are based on clustering heuristics for PLA packing, as PLAmap[19], T-VPACK[9] or the Singh Algorithm[159].

The parameters for clustering are the number of elementary cells or P-terms of the PLA and the number of inputs and outputs associated with the cluster. The placement problem consists of placing each basic cell inside a cluster, once the clusters are defined. This is achieved using generic optimization heuristics like simulated annealing.

Routing procedures for nanofabrics can either use adaptive maze router algorithms like Pathfinder[110], or they can be more specific to the fabric using, for example, custom adaptations of shortest Steiner tree problems[143] or other VLSI algorithms[53]. For reconfigurable fabrics, a defect-map provides extra constraints for placement and routing to configure around the defects previously detected.

Table 3.1: CAD tools used for different fabrics

	NanoPLA[35]	**CMOL[165]**	**FPNI[160]**	**Nasic[96, 173]**
partitioning/ logic mapping	PLAMAP[19]	T-VPack[9]	Singh's greedy algorithm[159] (specific cost)	PLAMAP based heuristic[66]
placement	Simulated Annealing (VPR-like)[9]			
routing	NPR - custom tool (Pathfinder-based)	shortest-path Steiner tree heuristic	maze router (Pathfinder-like) with several interations	Madeo router (Pathfinder-like)

Table 3.1 gives an overview of the different algorithms applied in physical layout tools for the NanoPLA, CMOL, FPNI and Nasic fabrics. Physical-design tools for nanofabrics use two kinds of algorithms or heuristics: adaptive generic algorithms and custom procedures. Adaptive generic algorithms include general purpose optimization heuristics like simulated annealing or genetic algorithms and algorithms for FPGAs like Pathfinder and PLA clustering as the ones implemented in Madeo[90] or VPR toolkit.

The remaining of this section reviews the main particularities of the physical design problem for crossbar-based nanoscale fabrics.

3.3.1 Logic Synthesis

For logic implementation, the crossbar-fabric designs studied in the context of this thesis (presented in Chapter 2.5.5) use different logic families: AND/OR, NAND/NAND, NOR/NOR, etc. The reasons behind the specific choices range from the simplicity of the mapping (the first NASIC proposition[114]) to manufacturing constraints (the choice of using just one type of devices in[120]) and strong technological assumptions (like is the case for CMOL[165]).

The choice of logic family doesn't impact the synthesis step, which usually is technology agnostic, and, as it was said in Chapter 3.2.2, tries to minimize the application netlist using technology agnostic optimizations. In consequence, all the studied architectures use Sis[150] for this step.

For the technology mapping on the other hand, different approaches are presented in the literature.

In the cases that use standard boolean logic as a target implementation for logic designs, like for the fabrics studied in this thesis, they still rely on Sis[150] technology mapping tool but use different logic functions as targets for the mapping[2].

If the logic is implemented through other means than standard boolean logic, specific tools are proposed by the respective research groups. In the following paragraphs we will discuss two of the most interesting approaches.

Stochastic circuits. In[140] the authors present a synthesis method that exploits both the parallelism and the randomness of the underlying crossbar fabric. The approach is based on stochastic computation with parallel wire bundles used for enlarging the state space to assure accurate results in the presence of defects. The synthesis process is based on netlist decomposition using multiplicative binary moment diagrams (*BMD). *BMDs, introduced by Bryant in[142], enable canonical representation for linear functions, similar to the way binary decision diagrams(BDD) are used for representing boolean functions. In this case, after a series of *BMD transformations, the *BMDs are transformed to stochastic function using two components, that the authors call shuffled ANDs and bundleplexers, which add stochasticity to the design.

Logic through percolation This approach, presented in [2], can be seen as a virtualization of digital logic circuits that exploits the nonlinearity produced by percolation[3] to implement digital logic. This method considers a crossbar as a square lattice offering different signal paths. If a crosspoint is defective, certain paths might not be accessible. Based on this observation, the authors define boolean functions as the connectivity inside the lattice, based on the input values and the probability of local connectivity.

This approach harness the intrinsic randomness introduced during the imperfect fabrication process to create usable digital circuit designs. From this point of view it is similar to the Nanocell fabric proposed by Tour[68], which used a genetic algorithm for training random molecular assembly to realize boolean functions. But compared to Nanocell, this approach doesn't have the problem of exceedingly high preprocessing time in order to start computation of a particular function. On the other hand, this approach can fail if the underlying fabric doesn't have enough intrinsic connectivity.

[2]The specification of the logic operators available on the target technology is done using the GenLib format

[3]Percolation is a mathematical theory for understanding and explaining the movement and filtering of fluids through porous materials

As most of the crossbar-based fabric approaches, except NASIC[115], envision to implement a reconfigurable fabric, the preferred method for fault tolerance is the reconfigurability around defects based on a defect map extracted after the fabrication. In these cases the logic synthesis step is not modified. Even in the case of the NASIC fabric, that used structural redundancy for creating a self-healing fabric, the logic synthesis step is not modified, but the redundancies are introduced into the netlist either before the logic synthesis or, after by duplicating certain signals and gates, or by using code-correcting techniques to encode the signals.

3.3.2 Partitioning

When the netlist is logically mapped to the target technology, the physical design process starts, usually by decomposing the netlist into blocks that are directly mappable on the target. This process is typically approached using packing algorithms that rely on a constructive approach to fit as much logic gates on the target tiles (or blocks) as possible. Table 3.1 shows the specific heuristics used in the context of the 4 studied fabrics. In the case of CMOL[165] and FPNI[160] fabrics relying on a CMOS layer for logic implementation, this step uses either T-VPack clustering[9], or Singh's clustering algorithm[159], two standard approaches typically used for FPGA logic packing. NanoPLA[35] and NASIC[115] approach logic implementation using a two-level logic style, similar to traditional PLA architectures. In these cases, the netlist partitioning is implemented either using PLAMap[19] or a PLAMap-based fabric-specific heuristic. In the case of NanoPLA, the choice of using standard PLAMap is motivated by the use of fixed size reconfigurable tiles replicated regularly. Whereas in the case of NASIC, a PLAMap-based specific heuristic is used due to the application-specific nature of this fabric design. This heuristic finds the most adapted PLA size for each application netlist, and based on the result, the fabric layout is computed. More details on this heuristic can be found in Chapter 5.3. Since all these approaches are based on reusing traditional tools from the reconfigurable IC field, in the context of this thesis, we are not detailing their internals. Interested readers can refers to the respective research papers[9, 159, 19] for more details.

3.3.3 Logic Mapping on Crossbars

In the case of architectures that implement CMOS-based reconfigurable logic blocks (e.g. CMOL and FPNI), traditional lithographic infrastructures can be used for the actual logic mapping on the target, provided that a defect map is constructed and the defective blocks are not considered during the placement step.

To better understand the logic mapping on nanoscale crossbar, an abstract structural model of typical two-level logic crossbar tile is presented in Figure 3.6. This model decomposes the nano-crossbar tile into a mosaic of 8 blocks. The lithographic infrastructure (blue rectangles at the periphery) is shown just to place the typical context, and is not included in the rest of the discussion. The 8 blocks, on the other hand, show the possible configurable elements, which are:

- *control A through D*, represent the places where the control signals or the pull-up/down networks are potentially placed. If the tile implements logic though static evaluation, only some of these blocks will be present. e. g. For implementing an AND/OR static logic tile, only control B and control D will be present as a pull-up network for product terms evaluation, and, respectively, pull-down network for output evaluation. If the tile is to implement a NAND/NAND dynamic logic style, all the control blocks will be present, representing, in order, the pre-charge network for the inputs, the evaluation network for the product terms, the pre-charge network for the product terms, and the evaluation network for the outputs.

- the *input plane*, that has the crosspoints populated either with diodes (NanoPLA) or FET devices (NASIC). This plane corresponds to the programmable AND plane of a PLA design.

- the *output plane*, that has the crosspoints configured either as diodes implementing an OR stage, or as hardcoded FET devices. This hardcoded FET stage, similar to the fixed output

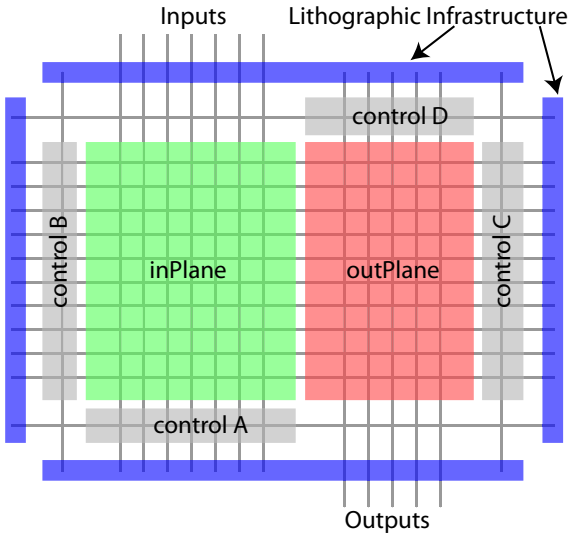


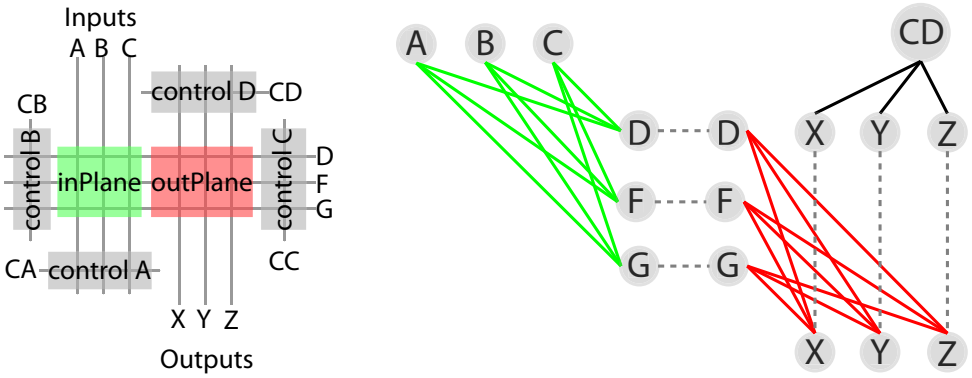
Figure 3.6: Abstract structural model of a tile composed of 6 crossbars. The blue rectangles at the periphery represent the lithographic infrastructure (power, ground, control, and configuration signals).

stage of PAL devices (cf. Chapter 2.2), is used in NanoPLA to implement the inverting/buffering stage.

The entire tile in Figure 3.6 can be seen as a simple crossbar, but the proposed decomposition is useful to better understand the impact each block has on the tile functionality. This is especially important when defective crosspoints are considered. Moreover, this representation gives a degree of generality to our abstraction, which enables the configuration of each block in isolation to match a particular fabric design. The control blocks are only one wire wide(tall) but the resistors (for the pull-up/down) networks, or the FET devices (in the case of dynamic logic implementation) can also be defective in which case the entire row/column will be disabled. This happens because each control device assures the correct functioning of one logic state (see Figure 3.7). Moreover, in some cases, these control blocks can be implemented using lithographic scale wires in one dimension crossed with nanoscale wires, in which case the control structure is more reliable. Such an approach is used in the NanoPLA[35] case as well as in the last versions of the NASIC fabric[121]. Earlier studies in the NASIC context where considering using only nanoscale wires for both the tile control and logic[114].

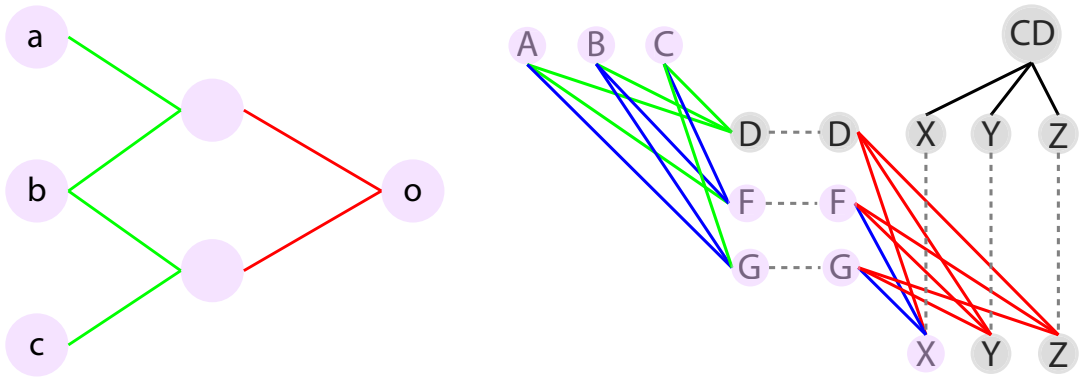
Based on this decomposition, the tile can be represented as graphs, with each node corresponding to a wire and each edge corresponding to the devices present at the crosspoint. As it can be seen from Figure 3.7 the different tile blocks have edges of different colors representing the different devices present in each block. As already mentioned, the malfunctioning of one control device(one black edge), say CD-X, has an immediate effect on the column it drives, all red edges incident to the node X.

Mapping a logic function onto a regular PLA is straightforward, since the PLA structure provides full flexibility of mapping a logic variable to any vertical wire and mapping a product term to any horizontal wire. Moreover since the inPlane graph, and the outPlane graph are fully connected, the mapping problem can be decomposed into two smaller problems. Figure 3.8 show such a mapping between a simple netlist(Figure 3.8a) and a fully connected crossbar graph(Figure 3.8b). For this mapping, any available resources can be used with no constraints, except the choice of product terms, which have to be identical for the inPlane and the outPlane mappings. In the case of NASIC[115], which uses structural redundancy for defect tolerance without reconfiguration, the logic mapping process retains the simplicity of standard PLA mapping with an spatial overhead due to the wire and device duplication. In this case, the logic mapping on the crossbar tiles is



(a) Simple tile (b) Corresponding graph

Figure 3.7: A simple 3x3x3 tile and the corresponding graph representation. Wires correspond to nodes, crosspoint devices correspond to edges. Green edges represent the *inPlane* devices, red edges represent the *outPlane* devices, while the black edges represent the *control* devices. The dotted edges represent the identity relation between nodes in different blocks of the tile. For the clarity of the representation only *control D* network was represented.



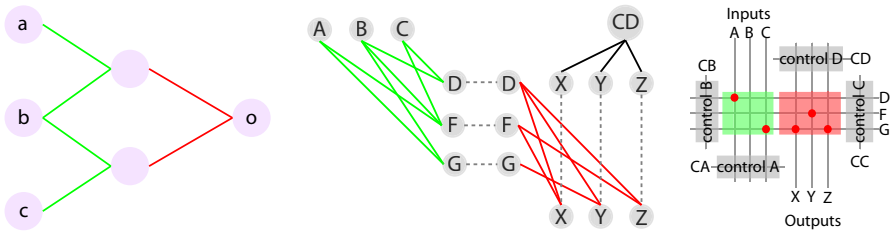
(a) Circuit graph (b) Tile graph with mapped circuit

Figure 3.8: Logic mapping on a defect-free tile graph

done during the fabrication process, through different mask patterns that will define the specific placement of the FET devices during the metallization process. Thus, the logic mapping step corresponds to the mask fabrication step. For the mask fabrication, the tile is considered as a matrix with the PLA logic pattern mapped-on.

However, in cases using PLA based crossbars as reconfigurable logic block, as in the case of NanoPLA fabric, the logic mapping process is more difficult, actually becoming NP-hard[157]. This complexity comes from the arbitrary defect patterns introduced during the fabrication process. This introduces a mandatory post-fabrication defect-discovery step to identify the functioning parts of the fabric. Moreover the logic mapping problem can be approached either by identifying the largest square crossbar with no defects, problem known to be NP-hard, or by embedding the logic graph into a graph constructed by eliminating defective components as well as other devices impacted by the defect. Finding such an embedding is NP-hard as well.

The nature of the possible defects greatly influence the logic mapping problem. As already mentioned, defective control devices completely remove the entire row/column affected from the tile graph, thus reducing the size of the available resources for the mapping. The same goes for broken wires, either inputs, outputs, product terms, or control. With the remark that a broken



(a) Circuit graph (b) Graph of defective tile (c) Defective tile

Figure 3.9: Logic mapping on a defective tile graph

control wire practically disables the tile. The tile graph remains fully connected and as a result, the complexity of application mapping is not affected, provided that the application size is still inferior to the tile size.

But in the case of disconnected wire defects, where the devices at the crosspoints are defective, and the crossbar graphs are not fully connected, the complexity increases drastically. In [157], the authors rely on orthogonal ray graphs to show that logic mapping on crossbar with only disconnected wire defects is NP-Hard, while finding the maximum non-defective square lattice can be solved in polynomial time.

Figure 3.9 show the effects of defects on a tile-graph. The defects are represented as red dots on the tile model in Figure 3.9c. These defects disable certain edges from the tile graph (i.e. edge A-D in Figure 3.9b). From the mapping perspective, in this case, a number of valid mapping on the inPlane are invalidated (i.e. the usage of nodes F & G for product terms) since there are no output wires connected to the required nodes (F & G in our case). This new constraint is at the core of the NP-hard complexity in the case of defect aware mapping. The reason is that it introduces a bidirectional constraint in between the inPlane and outPlane mappings. The outPlane mapping relies on the result of the inPlane mapping, and the inPlane mapping relies on the result of the outPlane mapping.

To solve this problem, in [65] the authors propose using graph monomorphism to find the embedding of the logic graph into the tile graph, by specifying the correspondences between the two graphs. However, this allocation method is based on a complete search of the solution space, which can be prohibitive for large tiles. To address this problem the authors propose to impose a bound during allocation on the number of graph matches attempts. Other approaches to this problem are presented:

- In [170], a heuristic approach approach is presented, which is used for searching the maximum biclique[4] in a bipartite graph that corresponds to finding the maximum $K \times K$ crossbars within the fabricated tile.
- In [144], the authors propose a recursive algorithm for computing the embedding along with heuristics for pruning the impossible mappings.

Similar to any NP-hard problem, a trade-off between the execution time of these heuristics and the probability of finding an acceptable result has to be found. Moreover, for the logic mapping on crossbar tiles, this trade-off can be seen from the productivity perspective as a choice between long testing and mapping time versus decreasing the yield of functional circuits[65]. However, as our understanding of the limits of nanoscale technologies progresses, it is possible that the choice of heuristics usable in this context is conditioned more by the target technology than by the execution time of the mapping. Notably since it appears that for certain technologies the tile size is hardly constrained by the wire fan-in size with respect to logic evaluation time[119].

[4]In graph theory a biclique or complete bipartite graph is a bipartite graph where every vertex of the first set is connected to every vertex of the second set

3.3.4 Placement

For the placement of logic clusters on the tile array, any of the heuristic presented in Chapter 3.2.2.2 can potentially be used. However the simulated annealing-based meta-heuristic is the preferred method presented in the literature[32, 166, 160, 171]. The principal reason behind this is the flexibility of the method, enabling eased target-specific cost optimization. As for the annealing schedule, which drives the solution-space search, the VPR[9] schedule is chosen.

For the cases where existing defects disable certain tiles, these tiles, present in the pre-computed defect map, are simply ignored by removing them from the fabric model before the placement process.

3.3.5 Routing

The routing routine, for the most part, relies on the Pathfinder[110] negotiation-based routing algorithm, similar to the one implemented in VPR[9]. Though, fabric-specific modification are presented for each target. The only exception is the CMOL fabric design that uses a RSA heuristic-based custom routing procedure. The following paragraphs will firstly detail the fabric-specific modifications of the Pathfinder routine, then it will briefly present the approach taken in the case of CMOL.

Vanilla VPR for FPNI. For FPNI[160], the authors propose a timing-driven, maze algorithm relying on the description presented by Betz[9]. For timing analysis, the Elmore-delay is computed after each routing iteration. Based on the route congestion and the delay, the cost of routing resources are modified and algorithm progresses until the exclusivity constraint is satisfied.

NPR router for NanoPLA. The particularity of the NanoPLA routing resides in the use of the tile as unique resource for routing and logic implementation, the tiles. Hence, the NPR router[32] iterates over all logic tiles that are accessible, incrementing the number of required product terms needed for successfully routing the netlist. NPR is a global, directional wire router, negotiating resource usage similarly as Pathfinder[110]. It assigns the routing paths while accounting for the extra NW needed for routing through the tiles.

Nasic is an application specific fabric, thus in the reference papers the routing problem is overlooked, assuming either direct inter-tile connections or a cellular architecture[123]. Chapter 5 introduces a Nasic-based architecture template using a Pathfinder-like router that is detailed in Chapter 5.3.

Shortest-Path Steiner Tree Heuristic for CMOL. The routing problem for the CMOL fabric[166] is shown to be equivalent to finding the shortest-path Steiner tree[70] that is exponentially hard. Thus, in [166] the authors propose a greedy heuristic for tackling this problem. The router proposed uses a two step approach to routing: global routing and detailed routing. During the global routing the nets are routed with the greedy heuristic. The eventual congestions are recursively solved during the detailed routing using an exhaustive search for possible routes.

One common point for NanoPLA, and NASIC, which implement logic and routing at nanoscale, is the need for a directional wire route model due to the directionality of signals imposed by their respective tile evaluation policy.

Another common denominator for all these architectures, except NASIC, is the approach for defect tolerance during the routing step. The solution is based on the defect-map constructed during the post-fabrication testing. Based on the defect map, the affected routing resources are not be integrated into the routing resource graph, and thus they are virtually invisible during the route allocation. For the NASIC fabric, the self-healing approach to defect-tolerance has been proven to work for logic, but we cannot assume that it has the same effectiveness for signal routing. Thus, in the context of NASIC, we consider the routing in the presence of defects an open problem.

3.4 Taming the Complexity - Design Space Exploration

The rapid progress of nanoscale technologies opens many opportunities for IC consumers, which continuously innovate and propose new application pushing the limits between science-fiction and reality. But, since everything in life comes at a cost, the business leaders, the developers, as well as the potential end-users are very sensitive to the cost of these new innovations. Moreover the current CMOS technology proved itself to be very versatile and competitive. Even if along its existence different challenges threatened its future, economic interests has driven the continuous evolution of CMOS past these barriers through huge research efforts[113]. Hence the skepticism towards the different alternative technologies, see Chapter 2.4.

Each of the emerging technologies has its advantages and its problems. The problem of finding amongst them the replacement for current CMOS technology can be simply reformulated as follows:

$$\text{Find technology} X \text{ that maximizes: } \frac{Gain(X)}{Cost(X)} \text{ subject to } \geq \text{ evolution trend} \tag{3.1}$$

The equation 3.1 can be easily recognized as the standard form of a typical optimization problem. The *design variables* are the technologies explored. There is only one *constraint*, to be better or at least equal to an evolution trend like Moore's law, equivalent scaling, etc.. The *optimization objective* is to maximize the overall gain while reducing the cost incurred for any chosen technology. Lets suppose for now that there exists a *model* that could relate the objectives and the constraints with the design variables. We could then use a formally sound technique to objectively find the technology that satisfy our needs. Of course this is an utopian idea mainly because we cannot formulate mathematically the objective of our optimization problem. How would we quantify the different advantages of any given technology to compute the gain? And for the cost? As presented in Chapter 2.4, each emerging technology has its challenges. These challenges are often hard-research topics, and some of them will probably never be solved. But, nevertheless the equation 3.1 is probably the most important optimization problem for the electronic industry.

In the context of current CMOS technology similar decision problems exists: choosing between different design styles i.e. standard cell, gate array, reconfigurable, etc.; selecting the logic blocks to include into a standard cell library; finding the balance between routing resources and logic in a FPGA; choosing the right switch-block architecture; etc. Known as the Design-Space Exploration (DSE) problem, different approaches are presented in the literature for helping the designer analyze the trade-offs. Since the searched design-space is very large, the EDA community proposed a number of automated software solutions for addressing this problem at different levels[81].

3.4.1 Algorithm-Architecture Adequacy

A popular instance of DSE is the "algorithm-architecture adequacy" (AAA). The idea of AAA is to find the most adapted IC architecture for implementing a given application (algorithm) subject to different constraints specified by the designer.

Automated DSE solution for exploring the AAA trade-off relies on three principal components: the applications, the *Architecture Description Language* (ADL) for problem formulation, and the *Metrics* for objective evaluation.

3.4.1.1 Architecture Description Languages.

ADLs are used principally to capture the particularities of different architectures, and to expose the design variables to the users. According to the particular needs of the targeted exploration problems different ADLs are proposed in the literature. Typically the target architecture is described using hardware description languages (HDL) such as VHDL or Verilog. Based on these HDL descriptions architecture-specific tools are developed, for simulation or application synthesis. This approach is typically done manually, hence it incurs high-development costs and low exploration facilities. The hardware/software co-design methodology, described in Chapter 3.2.1, tries

to automate this process by introducing automated design-space exploration, based on which an application specific hardware design is produced along with the associated tools.

Besides generic HDLs, such as VHDL, there are a number of more specific ADLs used for architecture specification in more restricted context. These languages are very high-level, enabling targeted domain specification. VPR ADL is on such example, targeted at describing FPGA architectures[106].

3.4.1.2 Metrics

To evaluate the quality of an architectural solution the impact of different design choices must be quantitatively measured and the results compared. The automated design-space exploration methods rely on evaluation metrics for searching the solution space. In digital circuit design the circuit area, and critical path delay are historically some of the most important metrics used to evaluate the quality of an architectural proposition.

Circuit area refers to the space occupied by a circuit design. Minimizing circuit area, or in other work increasing the transistor density per unit area, is one of the principal concerns of a IC designer. Reduced area means, the possibility to embed more functionality on the same IC chip, and drove the IC industry in the past to creating ever more complex chips with more functionalities.

Critical path delay is the measure of choice for evaluating the performance of a circuit design. It represents the longest time a signal takes for traversing the circuit. Based on this metric the operating frequency of a digital circuit is computed.

The evolution of the IC industry, and the exponential increase in the integration density, increased the complexity of circuit design exponentially while the control over the fabrication process decreased. With the adoption of deep-submicron technology issues like process variability, interference, fault tolerance started to affect the quality of produced circuits. Other metrics have been defined and have been integrated into design-space exploration flow to evaluate the potential impact of these problems on the quality of the produced circuit.

3.4.1.3 AAA Exploration.

To evaluate an architectural solution, and then explore possible design choices two main approaches are present in the literature: analytic evaluation, and dynamic evaluation.

Analytic evaluation. The analytic evaluation methods are based on estimation models that abstractly describe the design-space and estimate the performance of a particular design configuration. The estimation methods are generally fast and enable the exploration of large design spaces. But, since they are based on approximations of the design, the solutions found are not very precise providing only a relative indication of the solution quality. They are based on mathematical and statistical models of the design, and are used to rapidly prune large solution-spaces.

Dynamic design-space exploration methods are based on simulation or compilation processes. The simulation-based methods use simulation frameworks to simulate the different designs. The simulation results are used to compute the evaluation metrics and then to compare the quality of each explored solution. The produced results, using this technique, are very high-quality providing a very detailed characterization of the solution obtained. But, this exploration method is typically used for exploring small design-spaces since the simulation time is usually prohibitive. The exploration by compilation uses the synthesis tools to effectively produce virtual prototypes of the design, which are the used to extract the performance metrics. This technique can be seen as a trade-off between the high-quality results produced by time-consuming simulations, and the low-quality ones obtained by estimators. The principal drawback of this technique is the need for a complete synthesis toolset, which might not be available at the beginning of the design cycle.

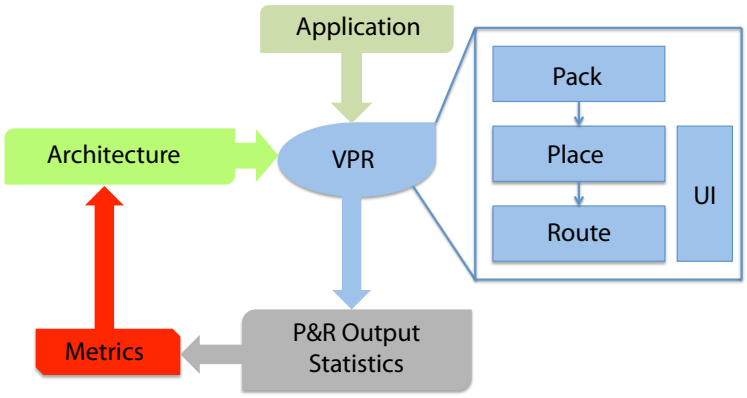


Figure 3.10: VPR tool-flow

The main transversal characteristic that should be considered for choosing any of these exploration methods is the abstraction-level at which the exploration is performed. The abstraction level chosen should reflect clearly the trade-off between the expected quality of the results and the time allocated for the exploration. Moreover, according to the choice of abstraction-level each of these techniques might perform differently. For example, a high-detail estimation model might be slower than a high-level simulation. Furthermore, these techniques can be used complementarily to approach the DSE. High-level fast estimators can for example be used at the beginning of the exploration cycle to rapidly prune the solution-space, then while progressively approaching the final solution, more and more details can be considered and simulations can be started to better characterize the designs.

3.4.2 Some Tools for AAA

In this section we will present two different open DSE toolkits used in the context of reconfigurable architectures, most specifically FPGAs. We believe that these two examples are representative in the context of this thesis since both of them have been retargeted to address nanoscale architectures. The principal difference between these two toolkits resides on the domain coverage targeted. The first one targets specifically FPGA families and propose highly optimized physical design tools. The second one is more generic and flexible enabling a larger domain-coverage.

3.4.2.1 VPR

Versatile Place and Route [9] is a generic FPGA physical design toolkit, including support for a number of specific domain explorations.

VPR can be mainly seen as:

- an environment for the description and the exploration of different FPGA designs;
- a platform for FPGA CAD algorithm development;
- a toolkit for packing, placement and routing routines for FPGAs.

Figure 3.10 show the global VPR tool-flow. The FPGA architecture is specified using a structural ADL language. The specified architectural parameters are: the number of logic blocks(LB), the number of LUT per LB, the size of a LUT, speed and area models, the type of switch block, etc. Based on these grammatical specification an internal FPGA model is instantiated. The application to be placed and routed is given as a netlist of logic gates, usually optimized and technology-mapped to LUTs using Sis[150]. The netlist is packed to the target CLB structure, then placed and routed on the target architecture. The results consist of a place-route output file specifying the allocated resources, and different allocation statistics such as: the size of the

routing channels, the critical path delay, the estimated area, the occupancy of the logic resources, etc.

To address the complexity of actual FPGA design, VPR toolkit is under constant evolution. New versions targeting a larger architecture family, including not only homogeneous resources. The architecture description language has also evolved to target commercial logic-block structures and routing architectures[106]. Prospective FPGA architecture exploration is another axis addressed by the VPR toolkit, to address new technologies, in [87, 88], the authors propose automated methods for integrating low-level FPGA design aspects like transistor sizing to the exploration flow. Based on the obtained results an FPGA architecture repository was created containing architectural model optimized in terms of area and delay for prospective CMOS technology nodes[197].

The open-source development policy of this toolkit, as well as the generality of the place & route routines integrated made it the preferred choice for targeting different physical design steps of some of the nanoscale architectures (i.e. NanoPLA, CMOL) studied in this thesis, see Chapter 3.3.

3.4.2.2 Madeo

Madeo[95] is a design suite for the exploration of reconfigurable architectures. It includes a modeling environment that supports multi-grained, heterogeneous architectures with irregular topologies. Madeo framework initially allows to model FPGA architectures. The architecture characteristics are represented as a common abstract model. Once the architecture is defined, the Madeo CAD tools can be used to map a target netlist on the architecture. Madeo embeds placement and routing algorithms (the same as VPR[9]), a bitstream generator, a netlist simulator, and a physical layout generator. It supports architectural prospection and very fast FPGA prototyping. Several FPGAs have been modeled, including some commercial architectures (such as Xilinx Virtex family), and prospective ones (such as STMicro LPPGA). Based on Madeo infrastructure further research projects emerged such as DRAGE[137], that virtualizes the hardware platform and produces physical layouts as VHDL descriptions.

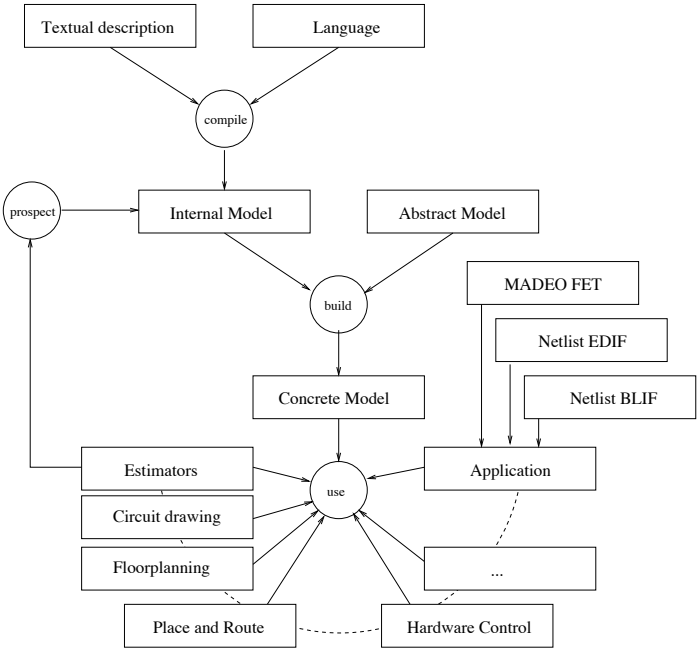


Figure 3.11: Madeo toolkit overview

The Madeo infrastructure has three parts that interact closely (bottom-up):

- *Domain model and its associated generic tools.* The representation of practical architectures on a generic model enables sharing of basic tools such as place and route (P&R), allocation, circuit edition[90]. Figure 3.11 illustrate MADEO on an island style FPGA. Specific atomic resources, such as operators or emerging technologies, can be merged with logic, since the framework is extensible.

- *High-level logic compiler (HLLC).* This compiler produces circuit netlists associated to high-level functionalities mapped to specific technology models. Leveraging object-oriented programming flexibility in terms of operators and types, the HLLC produces primitives for arbitrary arithmetic or symbolic computing.

- *System and architecture modeling.*The framework enables the description of static and dynamic aspects specific to different computing architectures, like: logic primitives, memory, processes, hardware-platform management, and system activity.

The compiler uses logic generation to produce configurations, binds them to registers or memories, and produces a configured application. The control over the P&R tools enables building complex networks of fine or medium grain elements.

During its 15 years of existence Madeo toolkit provided a flexible infrastructure for reconfigurable architecture research. Research results using Madeo showed that dynamically typed languages can serve for creating high-density logic designs[139, 39], proposed the first virtual FPGA prototyping environment[91], and proved that, harnessing the power of object-oriented software design, one can create a flexible yet competitive EDA toolkit that even today enables breakthrough research in the field, such as software-like debugging[92], or multi-grain DSE targeting embedded reconfigurable units for SoC[137].

In [96, 40] the extensibility of the MADEO framework was put to a test with the advent of emerging technologies. The core concepts of the NASIC fabric[115] were introduced into the framework, and a reconfigurable nanoscale architecture, called NFPGA, was designed. This required to extend both the reconfigurable architecture model and its associated tools in such a way that NASIC can be modeled and programmed.

3.4.3 DSE at nanoscale

For the nano-architectures studied the DSE problem is mainly approached through a dynamic, compilation based approach. Each research group proposed a set of physical design tools targeting their respective fabric proposition, cf. Chapter 3.3. The results obtained using these tools are then integrated into typical metrics such as area, critical path delay, power consumption, etc. The design propositions evaluated are then compared with traditional CMOS architectures using these metrics as mean for comparison. This approach is very effective for obtaining relative measures of the gains obtained by each evaluated architecture. But, at the same time this approach has a number problems coming from the complexity of the technologies used, and from the lack of tools for design exploration and automation. Besides, these emerging architectures are very different from their CMOS counterparts, and most of the time suffer from issues no one would encounter in classical CMOS designs, like high-defect density, regularity as technological constraint, etc. Moreover, compared to the mature CMOS technology none of these alternative is well understood, much less manufacturable. In consequence, the use of this fabric-specific DSE approach doesn't provide any measure of the potential costs required for practically developing such a fabric.

In this context, we believe that a principal improvement to this methodology must come from the EDA side by providing an integrated platform for DSE targeting this new technologies, thus enabling an effective comparison between similar approaches as well as gain with respect to CMOS. Design toolkits like VPR and Madeo showed that automating the DSE for an architectural family (FPGAs in their case) improves the understanding of particular design choices by enabling the users to focus on domain-specific problems, instead of reinventing the tools. Besides such a common environment will create the ideal infrastructure for innovation from an algorithmic perspective, effectively adding an algorithm-exploration axis to the AAA DSE problem.

The next section reviews the main reasons a comparison between the proposed crossbar-based fabrics is impossible, even though as we could see in Chapter 2.5 they are much similar.

3.4.4 Comparing Nanoscale Architectures

For the architectures presented in section 2.5.5 the nanoscale devices play different roles, in NASIC they are used for logic and routing, in CMOL for OR logic and routing, in NanoPLA for logic, and in FPNI just for routing. Each approach has its advantages and disadvantages but it is almost impossible to compare the results presented for each one. The main reasons for this lie in:

- architectural differences,
- different physical parameters used for evaluation,
- different evaluation strategies,
- different hypotheses during evaluation for the aspects not being investigated.

The architectural differences refer to the different design choices that were made for each architecture to optimize a specific aspect of the system. For example, FPNI trades off some of the speed, density and defect-tolerance of CMOL in exchange for easier fabrication, lower power dissipation, and greater freedom in nanodevice selection. NASIC trades off the advantages of fault tolerance by reconfiguration for easier fabrication process.

The different physical parameters used for evaluation, like nanowire pitch (NWP), nanowire width (NWW), nanowire resistance, CMOS wire width, etc., differ from one architecture to another. For example FPNI uses a 9nm NWP and a 5nm NWW while NASIC architecture bases its results on 10nm NWP and 3∼4nm NWW.

Different evaluation strategies refer to the fact that each architecture was evaluated using a specific tool according to a specific evaluation policy. For example, while FPNI and NASIC were evaluated using a yield simulator, CMOL and NanoPLA architectures used a place and route approach based on configuration around defects.

Even if there is one aspect being evaluated for all architectures, the results cannot be compared because the evaluation is based on different hypotheses for the other aspects not investigated.

The high rate of defects and the need for novel fault tolerance techniques render the comparison even more difficult since for each architecture the faults studied are different as well as the techniques used to create a robust architecture.

∘In conclusion, except the architectural differences all the other reasons clearly point to the lack of a common vocabulary along with a set of integrated tools targeted for these specific architectures, which imposes the development of specific tools for analyzing each architecture alone using different metrics and under different sets of assumptions.

3.5 Requirements for an Emerging-Fabric CAD Toolkit

This section briefly reviews the most important requirements for building an automated CAD toolkit targeting crossbar-based nanoscale fabrics. These requirements are classified across 3 main categories: 1. Transversal requirements; 2. Application specific requirements; 3. Domain specific requirements. In the following sections we will detail these requirements and we will present the way in which they impact the development of CAD tools.

3.5.1 Transversal Requirements

Generality In object-oriented software development generalization is used to create a *genus* (hypernym) from *species* (hyponyms), i.e. to create super-types from subtypes, thus abstracting away from a number of subtypes and their respective differences[86]. In our case, the generality represents the capacity to abstract away low level details related to the technology (i.e. CMOS,

nano, hybrid) or to the architectural type (i.e. reconfigurable or not) in order to build a set of tools which can be selectively specialized by the user to provide specific functionality.

Expressiveness represent the complexity to describe a certain concept. The expressiveness eases the burden of the user giving him the exact vocabulary he needs for describing the specific issues he is interested on. But at the same time a tradeoff has to be made between the generality of a software system and its expressiveness. In our case for example one of the most generic ways of describing nanoscale architectures might be to use a generic programming language (like c, java, smalltalk) but these languages lack the expressiveness needed for this job due to their generality. Thus the user is forced to use a generic vocabulary to describe his domain of interest. Architecture description languages like VHDL, SystemC, etc try to bridge the gap between the generality and the expressiveness in order to ease the task of architecture description.

In our case we feel that in order to better understand nanoscale architectures the designers need an even more expressive vocabulary, which includes the main concepts used to describe these type of architectures. This vocabulary should reify concepts specific to this domain like: 1. crossbar; 2. nano/CMOS interface; etc. at the cost of loosing some of the generality.

Evolution in our case represents the ability to change over time due to different factors like: technology changes, the addition of new algorithms. In the context where the field of nanotechnology is not mature enough, and the nanodevice-based architectures are just emerging, the software tools targeting these technologies must be able to evolve along with the technology in order to ensure their perenniality. But this perenniality comes at the cost of loosing expressiveness. As the technology evolves over the years the new constraints and requirements emerge which should be expressed, thus the domain language has to have the capacity to adapt to these new concepts without changing the semantic of the older concepts.

The adaptative object-model [194] is a software architectural style which emphasize this much needed flexibility by providing a tradeoff between generality, expressiveness and evolution. However, the software systems based on this style tend to be hard to understand in order to use, extend or maintain them.

Scalability is the ability to handle growing amount of work in a graceful manner or to be readily enlarged. In the context where the nanoscale architectures promise to deliver chips having orders of magnitude larger integration densities the scalability of the CAD tools used to design, analyse and exploit them is a critical issue. We identified three principal axes along these software system need to scale: 1. spatial complexity, 2. temporal complexity, and 3. domain complexity .

Spatial complexity refers to the capacity to integrate millions of devices on an architecture. From an object-oriented system point of view this means the capacity to scale from systems having to work with tens of objects to systems working with thousands, millions, or even billions of objects. Hierarchical composition of entities[51] is one way to manage this complexity at the description level. Flyweight design pattern is a way to deal with this complexity in a running application by minimizes memory use through sharing as much data as possible with other similar objects[51].

Temporal complexity (algorithm complexity) refers to execution time taken by a certain software tool to compute a certain result. The scaling in this direction happens through the use of better and better algorithms.

Domain complexity represents the growth of the number of concepts used in the domain, it relates to evolution. Through domain evolution some new concepts get identified and need to be included in the working vocabulary of the software tool.

Constraints are the specific conditions that a system must satisfy in order to be considered valid. In the case of our domain we identified two main classes of constraints which are transversal to the domain: geometric constraints (i.e. component x is at 10nm from component y), and spatial constraints (component x is at left of component y and inside component z).

Metrics provide the ability to objectively compare elements from different solution spaces. In our case the metrics can be classified as: transversal metrics (i.e. execution time), application specific metrics (i.e. circuit area, circuit delay) and domain specific metrics (i.e. architecture robustness). One aspect worth mentioning that the integration of metrics into the tools should not impact any other requirement.

3.5.2 Application Specific Requirements

This type of requirements represents a set of needs specific to this type of CAD tool.

Multi-level simulation represents the capacity of simulating the system at different abstraction levels. Multilevel simulations open the way for scalability giving the possibility to check for certain properties at a high abstraction level in the case of big designs, and to further refine a design from an abstract architectural level to the physical level. In our case we identify 3 important abstraction levels: 1. logic gate level, 2. switch level, and 3. device level.

Standard tool integration enables the use of external tools for different specific tasks. Spice or Modellica can be used for small scale low level simulations to validate various parts of an architecture, enabling the use of powerfull tools already validated on the backend. Abc and/or Sis[150] can be used for circuit synthesis, optimization and technology mapping.

Physical design integration refers mainly to partitioning, place & route routines that should provide a generic fabric agnostic back-end much as in the case of VPR, or Madeo. At the same time the domain-models should be isolated as much as possible from the physical-design routines to improve the modularity of the system and enable plug-and-play algorithm exploration.

3.5.3 Domain Specific Requirements

The requirements directly corresponding to the studied domain are presented in this section.

Defect and fault aware Represents the capacity to model, inject defect/faults and to integrate fault tolerance techniques. In the context were the nanoscale architectures are expected to have high rate of defects due the bottom-up assembly process the defect awareness is a strong domain specific requirement.

Nano/CMOS interfaces should be modeled as a first-class concept, to enable a rapid modeling and eventual interchange in between fabric architectures. As an example, when developing a reconfigurable nanoscale architecture, the stochastic decoder proposed by A. Dehon in[33] should be available as a design component.

Domain coverage represents the ability to model all the elements of the studied domain. The proof of concept tools should be extensible enough to cover an architectural family, crossbar-based nanofabric in our case: NASIC[115], NanoPLA[34] in the context of 2D architectures and CMOL[165] and FPNI[160] for the 3D architecture case.

Connectivity the ability to express the connectivity between the components. The connectivity between the components is an important aspect of an architecture as it practically represents the logical view of an architecture. In our case we can identify two types of connectivity: the structural connectivity, and the logical connectivity. The structural connectivity relates to the structure of the architecture, while the logical connectivity relates to the functionality of the architecture. The connectivity serves as a basis for routing graph extraction, for different cost oriented optimizations (i.e. critical path, etc).

3.6 Summary

This chapter introduced the EDA field, presenting, Section 3.2 some of the most important tools for automating the IC design. Section 3.3 overviews the physical design tools targeting crossbar-based nanoscale fabrics. In Section 3.3 the design-space exploration problem is introduced along with two automated approaches from the reconfigurable domain. Then nanoscale DSE challenges are presented and the principal factors limiting unbiased nano-fabric comparison are discussed. Finally, Section 3.5 lists some of the most important requirements needed for creating an automated DSE CAD flow targeting nanoscale crossbar-based architectures.

4

The Way to Model-Driven Physical Design

This chapter presents the model-driven physical design tool-flow proposed. The presentation is structured around three important aspects: domain modeling, tool design, and tool-flow modeling. The targeted fabric design is describes using an abstract model of a hierarchical port-graph. The tool design is based on the transformation metaphor. The tool-flow is reified and modeled using a specific object-oriented abstract model, which enables a high-degree of algorithm reuse and drives design-space exploration. Moreover, this tool-flow is backward-compatible and favors high-degree of flexibility and reuse.

4.1 Introduction

For years the principal challenge of the EDA industry was keeping the pace with the rapid progress of the CMOS technology by providing automated solutions to bridge the gap between applications and ICs. But nowadays, the integrated circuit industry progress trends are threatened by the intrinsic limits of the current CMOS technology. To address this problem a number of emerging technologies are under investigation. Each of these technologies has its advantages and presents unique issues, never encountered before, like the bottom-up fabrication techniques, the high-defect ratios, or the regularity of assembly. Moreover, a large number of challenging requirements are imposed on these technologies to be competitive against the current standards, see Chapter 2.4.2. From the EDA perspective, the gap between technological breakthroughs and automated design is getting deeper, and more challenging to address. This gap is even more problematic at the physical-design level, the last automation step between applications and technologies. Meeting the physical, and functional constraints for different technological targets needs innovation at both the algorithmic level, and the methodological level.

From a methodological perspective, today, after 50 years of EDA innovation, it is unacceptable to develop new tools from scratch for each new technology. Especially when these tools rely extensively on existing heuristics. This is, for example, the case of NW crossbar-based fabric designs, presented in Chapter 2.5.5. These designs rely heavily on tools from the reconfigurable field for physical-design automation, see Chapter 3.3. As discussed in Chapter 3.4.4, the main reason for this is the lack of an open toolkit proposing a methodological approach for design-automation on new fabric architectures.

In this Chapter, we present MoNaDe, a model-driven physical design toolkit to address this problem. This toolkit leverages new innovations from the software engineering field to provide an integrated development environment targeting circuit design automation and domain-space

exploration(DSE) in the context of NW-based fabric designs. To the best of our knowledge, except our preliminary studies in[40, 174], this proposition is the first MDE-based approach addressing the physical-design automation problem. The principal contributions in this chapter are:

- *A novel framework*, based on the MDE methodology, for the physical design aspects of the EDA flow. This framework stresses on the need of decoupled software artifacts, and isolated concern-oriented software modules. Moreover, from the practical point of view, this methodology doesn't impose any constraint on the implementation language or formalism for designing the algorithms and heuristics needed, which is very important given the computational- and memory-intensive solutions required in this context.
- *An abstract vocabulary* for structurally describing the applicative and architectural artifacts involved in the flow. The use of this abstraction level enables the creation of a common API that can be used for transversal features, such as domain agnostic manipulations, querying, model serialization, etc.
- *A novel way to conceptually describe, characterize and implement physical design algorithms* (e.g. placement, routing) using the "transformation metaphor". These algorithms are viewed as model-to-model composite transformations organized as hierarchical directed-acyclic graphs. This view helps the algorithm designer to focus on the specific problem to be solved while enabling fine-grain reuse of software artifacts.
- *A model-driven tool-flow model*, which reifies the tool-flow and its elements. This approach creates the ideal conditions for the independent evolution of architecture, algorithms and tool-flow. This flow improves the algorithm reutilization, eases the agile development of the design-flow, and creates the necessary conditions for incremental design space exploration. Moreover the use of the Model-Driven Development in the context of the physical design opens the toolbox offering an unprecedented flexibility and support for high-performance physical-design in the context of dynamic technological targets.

However, creating a full physical-design automation flow from scratch is a very challenging job, due to the numerous issues that should be taken into account. Besides, it will defeat the purpose of our argument against creating new tools for each new technology (or even technology family for that mater). To address this issue, besides the model-driven-toolkit presented, the last section of this chapter presents an incremental evolution strategy used to refactor an existing object-oriented CAD framework[90] towards our model-based methodology, and to extend it for targeting crossbar-based designs.

The rest of this section introduces the reader to the model-driven development methodology, presents some state of the art approaches using this methodology in the context of high-level synthesis (HLS), and reviews the main software engineering innovations that enable this approach.

4.1.1 Model-driven development

Twenty years ago object-oriented software for EDA could not be imagined. The executing environments provided by the virtual machines was considered too slow for solving the hard combinatorial optimization problems of circuit design. But industrial successes such as the ControlWorks wafer fabrication platform developed by Texas Instruments using the Smalltalk environment[8] coupled with research efforts such as Programmable active memories (PAM)[181], which relying on C++ reifies the application netlist enabling circuit generation from a high-level object-oriented language, as well as the MADMACS layout editor[52] proposing a functional language (similar to Lisp) for creating VLSI layouts inspired the community and provided the incentives to using interpreted languages for circuit designs. Amongst the numerous efforts in this direction, we would like to mention here the Madeo project[90], which, overcoming the performance challenges of object-oriented languages in the context of FPGA design automation, showed that dynamically typed languages can serve for creating high density logic designs[139, 39], proposed the first virtual FPGA prototyping environment[91] and proved that, harnessing the power of OO software

design, one could create a flexible yet competitive EDA toolkit that even today enables breakthrough research in the field[92, 137]. Meanwhile, OO design became widely accepted by the EDA community through languages like C++(i.e. systemC[134]) and java(i.e. JHDL[69]). However, OO design suffers from a number of problems, common in software engineering, especially from a software evolution perspective. As the target domain evolves, the software tools used modeling, simulation and reasoning about the domain have to evolve accordingly. Hence, without a methodological approach for evolution, systems become unmaintainable and, even more, functionality is lost through numerous integration cycles. Model-driven engineering promises a number of solutions for this problem by abstracting and decoupling the different software artifacts, most notably through the use of aspect-oriented programming[1] and component-based software design[2]. In consequence, today we assist at yet another step in the evolution of EDA industry, the move towards model-driven and component-based design, especially in the context of embedded system development and high-level synthesis.

Model-driven paradigm provides a methodology to tackle the complexity of software development using abstraction, problem decomposition, and separation of concerns. This methodology places models and transformations at the center of the software system development. A model can be seen as an abstract and simplified way to describe and understand complex systems. According to [111] a **transformation** is "the automatic generation of one(or more) target model(s) from one(or more) source model(s), according to a transformation definition. A **transformation definition** is a set of rules that together describe how a model in the source language can be transformed into a model in the target language."

The massive adoption of the model-driven paradigm was favored principally by the standardization of Unified Modeling Language (UML) and Model-Driven Architecture(MDA) by the Object Management Group (OMG)[27]. Modeling languages such MOF, EMOF[130] or Ecore[162] as well as the development of environments like Eclipse Metamodeling Framework [162] have been largely adopted as the *de-facto* standard for model-driven software engineering. Efforts, like Kermeta[116], proposed ways to "breathe life into (your) metamodels", by adding an executable meta-language, for operational semantics specification, to the contemplative (structural) side of model-driven development.

But the OMG is not the only actor improving on the state of the art of this field. The orthogonal classification architecture, proposed by Atkinson et al. [5, 4], provides a new insight into some possible future metamodeling environments, able to solve some of the current limitations. The conceptual separation between the ontological and linguistic dimensions of modeling, addressing the dual classification problem, the use of clabjects for addressing the class/object duality and for deep instantiation, along with the infrastructure presented, gives us some insight into a possible future of meta-modeling.

The works from the Software Composition Group, proposing the use of Smalltalk as a "reflexive executable meta-language"[41] for "meta-modeling at runtime"[85], address some more pragmatic issues around model-driven engineering, e.g. the use of a single language – smalltalk in this case – for all the modeling aspects; the possibility to dynamically change the meta-model at runtime; the need to have executable meta-descriptions that alleviates the need for code generation.

Other approaches, like Platypus[138], showed the importance of abstracting the mechanisms for accessing domain-specific data described formally using the STEP standard[72]. Using this abstraction enables the automatic generation of data access interfaces independently of any particular system of implementation language.

Table 4.1 compares three model engineering approaches with respect to a number of empirical criteria. The OCA[4] approach is not presented due to the lack of usable engineering tools. All three environments support code generation, propose an interchange format, and have multi-language support. EMF and Platypus environments are based on industry accepted standards. Compared to EMF and Platypus the FAME approach enable the meta-model creation based on

[1] Aspect-oriented programming (AOP) is a programming paradigm which aims to increase modularity by allowing the separation of cross-cutting concerns.

[2] Component-based software design is a branch of software engineering that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system.

criteria	FAME	Platypus	Eclipse Modeling Framework
code generation	✓	✓	✓
visitor generation	possible	✓	possible
builder generation	possible	possible	possible
interchange format	MSE or XML	STEP	XMI
model - metamodel link	bidirectional	unidirectional	none
opposite attributes	✓	✓	✓
computed attributes	✓	✓	✓
metamodel inference	✓	no	no
model description language	ST80	EXPRESS	UML
behavior description	ST80	EXPRESS or ST80	OCL, StateCharts
model-generate-use	✓+ model-use-generate-use	✓	✓
standardization	no	✓	✓
multi-language support	✓(ST80, java, C#, Python, Ruby)	✓(ST80, C, Java)	✓(Java, C, C++, etc)

Table 4.1: Fame vs. Platypus vs. EMF – comparison

an existing object-oriented implementation, using annotations, moreover the generation step can be skipped during prototype implementation and validation since the modeling is done using an executable language (smalltalk). This feature is particularly useful since it permits direct model execution at the modeling language level.

In the context of this study, we have adopted a hybrid approach based on the FAME metamodel for domain modeling, and an ad-hoc smalltalk-based behavior and algorithm specification approach, relying on the "transformation" metaphor, implemented using the visitor design pattern and isolated software components.

4.1.2 Model-driven HLS

UML extensions (profiles) such as SysML[179], Embedded UML[109], and MARTE[180] have been proposed for embedded system design. These profiles allow the specification of systems using high-level models. Low-level (i.e. VHDL) code is generated by mapping UML concepts to the target language syntax . In [141, 50], for example, the authors propose a HLS and design space exploration flow based on the MDE methodology, which based on successive refinements of a high-level model generates VHDL hardware accelerators.

Moreover, concerns such as interchange or debug are, mainly, considered during HLS. As an example, RedPill[93] supports probe-based validation of running applications. Probes are inserted in the high-level code, but also appear in the generated code, and bring controllability to the hardware under execution. RedPill makes use of domain modeling (in that case, the application) through Platypus[138], a STEP/EXPRESS framework that offers both a standard way of modeling the domain, and interchange facilities. In particular, Platypus has been used in the scope of the Morpheus FP6 Project[94] to support multi-target and cross-environment synthesis/compilation. The output was a netlist to be further processed by low-level tools (physical design).

4.1.3 Enabling Technologies

During the last years Smalltalk world has undergone tremendous evolutions, through truly open environment like Pharo[10], language innovations such as the adoption traits for fine-grain reuse[43], language boxes for embedded DSLs[145] and runtime meta-modeling frameworks like FAME[85]. Moose project experience report[42], as well as our experience with Platypus[94] led us to creating an agile MDE-based prototype, targeting FPGA physical design automation, using the smalltalk environment. Our goal is to create an evolution-aware platform relying on our legacy code-base that can withstand the test of time

The most important aspects that need to be addressed by such a toolkit are: domain modeling, domain-specific languages, code reuse, legacy and fast prototyping, and external tools integration. In the following paragraphs we will try to review each of this aspects and briefly present some smalltalk technologies that can address each issue.

Domain modeling is at the core of any EDA toolkit. It enables the expression of domain specific aspects by creating a common vocabulary that will then be exploited to model different systems. In the context of rapidly changing IC technology, the domain model has to evolve rapidly, eventually changing its internal structure. For addressing this problem the FAME meta-modeling library[85] proposes a solution by offering run-time access to the meta-information, appearing in a meta-model. By creating a bidirectional causal connection between the meta-level and the implementation-level the model updates can be reflected in-between the two levels at runtime. In Section 4.2.1 we will present in details a FAME-based abstract meta-model that is used throughout our framework as a common base for creating the different domain models needed.

Domain-specific languages are used extensively in the context of circuit design automation. Their principal roles are: describing the IC architecture (Hardware Description Languages (HDL)), describing the different physical, technological, and design constraints, describing the functional requirements, specifying test cases and objectives, describing the application netlists, etc. In the context of our framework, currently, we have implemented a dozen different parsers for interfacing with other tools and we have a proprietary HDL used for FPGA architecture description and instantiation. The SmaCC-based[12] parser description are difficult to modify according to the evolving needs of the EDA field. The parser-combinator libraries such as PetitParser along with the class-box concept implemented in the Helvetia DSL development environment[145] provide a new solution to this problem. Helvetia enables the seamless creation of embedded DSLs, while PetitParser brings inheritance to grammar specifications, thus offering the possibility to isolate the grammar specifications from the abstract syntax tree creation. These developments provide a smart way for defining a concrete text-based syntax for the instantiation of our domain models. It should be noted that due to the high number of hardware modules included into a typical architecture, visual languages are not well suited for the architecture description. Moreover IC designers are very proficient using textual description languages.

Code reuse is the most important concern from a software evolution point of view. OO methodology provides inheritance and polymorphism as *de-facto* solutions for enabling large-scale code reuse. With the adoption of traits[43], in Smalltalk dialects such as Squeak and Pharo, the OO toolbox for reuse gained a new tool. Traits are a mechanism for fine-grain method reuse that decouples functional method implementation from the object state.

Legacy and Fast prototyping As stated in the introduction, a big issue lies in the reuse of our legacy code-base. Moreover, any evolution methodology has to enable incremental development, such as to be able to go on using our environment during the whole evolution process. We bootstrapped the prototype by reusing the MADEO project infrastructure, and incrementally moved towards our new MDE-based framework. In Section 4.5 some of the evolution steps are presented as to illustrate our approach.

External tools integration is a high-level reuse mechanism by which tools developed by third-parties can be integrated into the toolkit. Historically we have been using inter-process communication (via UnixProcess class) and FFI (such as DLLCC) for executing external programs and interfacing with our toolkit. The Alien FFI, proposed for Newspeak language[11] and adopted by Pharo, provides a new OO-friendly way of interfacing smalltalk environment with the external world. Moreover, the transformation metaphor, proposed for algorithm design, opens the toolkit even more by enabling a fine-grain external tool reuse, via the *External* atomic transformation.

The remaining of this Chapter presents our practical approach for abstract domain modeling and a novel conceptual framework for designing and implementing algorithms based on the transformation metaphor.

4.2 Domain Modeling

Domain modeling is one of the most important aspects of EDA. Through domain modeling the main characteristics of the modeled problem are captured and explicitly described using entities, associations and constraints. A domain model offers a conceptual high-level description of the reality providing a common vocabulary for describing the scope and the meaning of the concepts specific to the domain. Besides these conceptual advantages a domain model can be used as a basis for software tool implementation, representing the link between the real problem and specific algorithms used to solve it.

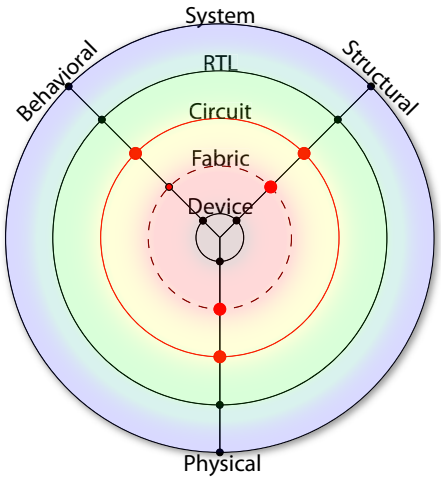


Figure 4.1: Adaptation of the Gajski-Kuhn Y Chart for nano-electronics

Gajski-Kuhn Y-chart propose a high-level domain decomposition capturing the design considerations of the EDA industry. The domain-specific concerns are separated around three complementary axes: behavioral, structural, and physical. The behavioral axis is concerned with the functional aspects of the system, the structural axis is related to the connectivity and the interconnection between different blocks, while the physical axis relates to geometry and spatial aspects. Each of these axes being then divided into different abstraction levels. Figure 4.1 presents an adaptation of the standard Gajki-Kuhn chart for nanoelectronics. It has 5 abstraction levels starting from the device to the system and passing through fabric-design, circuit design, and RTL abstractions.

As described in Chapter 2, for nanoelectronics, most of the research efforts are focused at the device level and fabric level. These research efforts are geared towards understanding the physics of these nanoscale building blocks, with some prospective fabric propositions, which try to harness their advantages while minimizing the challenges (especially in terms of fabrication and fault tolerance).

In the context of the MoNaDe toolkit our focus, in terms of domain modeling, is at the fabric and circuit level, as we are trying to bridge the gap between applications and these novel fabric propositions.

In Chapter 3 we identified the lack of a common vocabulary along with a set of common tools as the main reason behind the impossibility to compare and objectively evaluate the crossbar-based fabric propositions. To tackle this problem this chapter proposes a fabric model for crossbar-based fabrics, which defines a common vocabulary for describing the main concepts of these fabrics. This model is describing principally the structural and physical axes, for the behavioral specification relying on external simulation tools, such as HSpice, using simulation models specific to each fabric proposition. The defect-aware mapping of logic on crossbars is the principal automation routine acting at this level.

Moreover, since one of the most important constraints at nanoscale is the regularity of assembly,

at the circuit level, most of the architectural proposition propose a regular composition of fabric blocks similar to today reconfigurable architecture. Thus, for circuit-level modeling and physical-design tools (place & route) we propose a high-level model based on the Madeo framework model, with domain-specific extensions targeting nanoscale circuit design.

From a domain modeling perspective, even though these two models are placed at two different abstraction levels, they share a number of common characteristics. These characteristics are transversal and appear also in the application and simulation models. Hence, MoNaDe toolkit abstracts them away and propose a core model used as a common vocabulary for describing all these concerns. This model, presented in Chapter 4.2.1, is then specialized to target the fabric model, the circuit model, the application model, and the simulation model, see Figure 4.2.

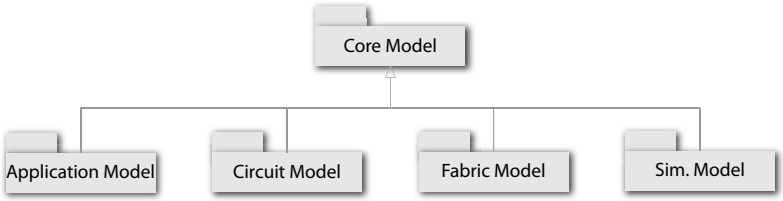


Figure 4.2: Global view of the domain models and their relation with the core model

This approach has the advantage of offering a homogeneous high-level modeling language for addressing problem-specific aspects of the physical-design automation. Moreover a number common tools can be created to work at the core-model level. Examples of such tools are: generic visualizations, common concrete DSL for building instances, interchange infrastructure.

The remaining of this section presents the different models used in the context of MoNaDe toolkit.

4.2.1 Fame-based Abstract Model

From a modeling point of view most of the EDA tools are structured around hierarchical models appearing at different levels of abstraction. These models are used to represent the two principal axes of EDA:

- *Application* The applications are typically seen as composition of operators, that can be further broke-down into simpler constructs towards elementary systems primitives. The behavior specifications can be treated at different abstraction levels: i.e. control data flow graphs, with hierarchies representing processes and/or different functions used to implement the application, combinatorial or sequential logic, used to describe the system behavior as composition of simple Boolean functions, etc.

- *Hardware* The hardware designers rely heavily on hierarchical descriptions to simplify the integrated circuit development. As for the applications, the hardware is described hierarchically at different abstraction levels. At the system level, for example, an IC can be seen as an array of blocks, each of which implements a specific functionality (i.e. processors, GPUs[3], network modules, etc.). Each of these blocks can then be decomposed into its own building blocks (memory, logic, interconnection, etc.). At the logic level, a digital circuit can be seen as a composition of logic blocks, implementing different functions according to the Boolean equations characterizing the blocks. At the circuit level the system is again specified hierarchically, using transistors, diodes, etc. as primitives.

Based on these observations, we created an abstract meta-model that is used to structurally describe our domain as a hierarchical composition of primitive elements interconnected together.

[3]graphical processing unit

This abstraction describes mainly a hierarchical annotated port-graph. Using the FAME meta-modeling framework was straightforward creating a smalltalk implementation of this high-level domain-specific modeling language.

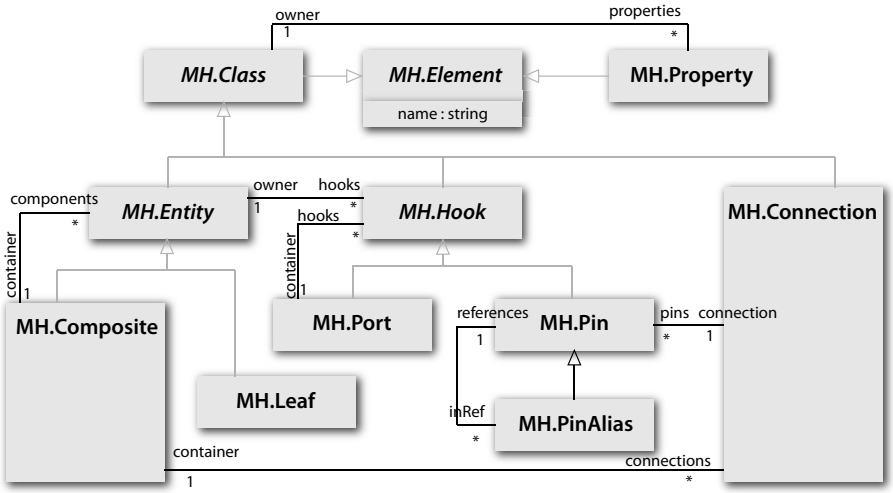


Figure 4.3: A view of the core structure of the proposed meta-model

In the proposed meta-model, see Figure 4.3, the domain structure is captured as a *Composite* that contains one or more *Entity* instances (called "entities" from now on). *Entity* is an abstract construct specialized as *Leaf* and *Composite*. The *Leaf* instances are domain primitives which are viewed as the indivisible building blocks of more sophisticated structures. The entities have hooks (*Hook*) which provide an external interface through which they can be interconnected together. The abstract *Hook* concept is specialized as *Pin* and *Port*. The *Pin* element allows connection between entities. It can be a physical contact, or a logical interface depending on the specification level or on the refinement degree. The role of *Port* instances is to group together and structure the *Pin* instances, they can be seen as namespaces structuring the access to particular pins. The *Connection* purpose is to glue together the entities of a particular system by linking pins together. These connections can be interpreted as wires, communication channels, or simply as relations between entities.

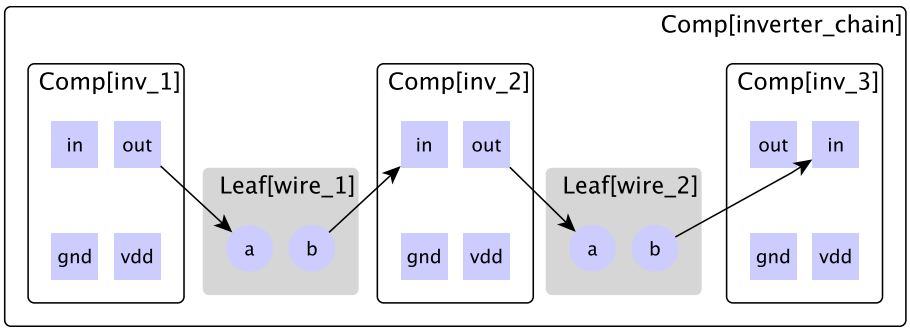


Figure 4.4: An example visualization of an inverter chain modeled using the fabric-level specialization, and viewed using the meta-viewer. For clarity only two level of the hierarchy are shown.

All these concepts are refinements of the *Class* element which owns *Property* instances. The *Property* represent different attributes particular to the modeled domain. Some examples of properties are: the position of an entity on a layout, the capacity or resistance of a wire modeled as a Connection instance, the logical value of a circuit pin represented as a *Pin* instance.

The meta-model, presented in Figure 4.3, represents a common abstraction which can be used to specify any kind of interconnected system having any number of hierarchical components that present the system at different abstraction levels.

Figure 4.4 shows a meta-visualization (visualization at the abstract level presented in this section) of a composite block modeled using the fabric-level specialization, presented in Chapter 4.2.3. The use of this kind of meta-visualization enables quick visual verification of the modeled component without having the need to create a specific viewer. The principal drawback is that while generic, this abstract viewer will generate a large amount of visual artifacts, which in some cases will obscure important details. Nonetheless, having this meta-visualizations proved very useful, especially during the first prototyping phases of the refined models.

4.2.2 Transversal concerns

A number of transversal concerns emerge when considering the physical-design problem. One of them is the possibility to extract a number of common abstract modeling elements, presented in the last section. Other transversal aspects, emerging especially in the context of nanoelectronics, are the management of configuration/reconfiguration cycles, and the need for defect-aware modeling with defect/fault injection.

4.2.2.1 (Re-)configuration

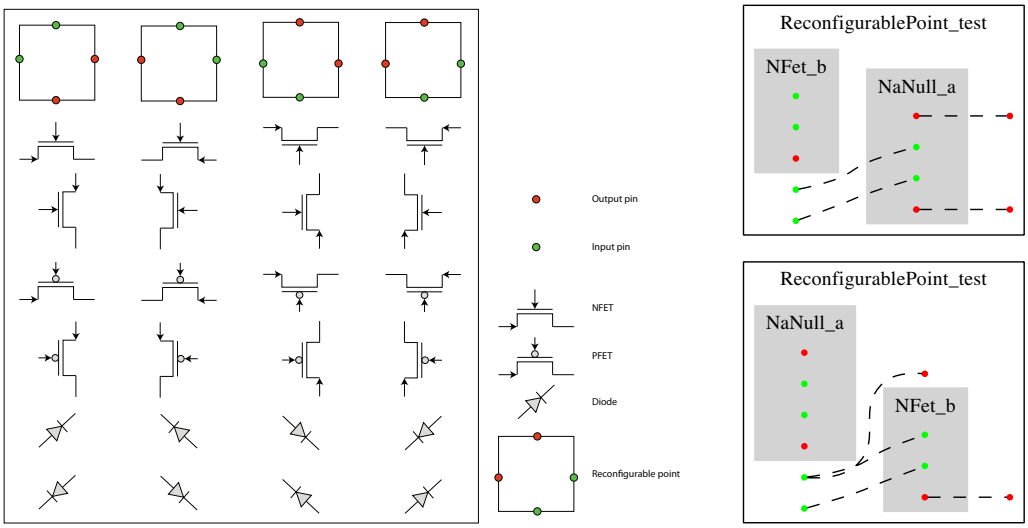
As already discussed, the fabrication processes proposed for nano-electronics (especially crossbars) impose a regularity constraint in terms of wire and device placement. Turning this constraint into an opportunity led the different research groups to propose highly regular fabric design with large amounts of identical logic primitives that can be specialized according to the application needs, approach which is conceptually similar with PLA and FPGA structures. The exception to this rule is the NASIC fabric architecture, which gave up reconfigurability to enable high-density self-healing application-specific circuits. But, in[172], we showed that even though NASIC fabric doesn't support reconfigurability, still we can rely on this technique at the tool-flow level for automating the physical-design. In this case the NASIC blocks are viewed as an one-time configurable, the configuration representing the actual crossbar functionalization during the metallization step.

In consequence, the fabric and circuit configuration (and reconfiguration) emerge as a common denominator for all crossbar based nano-architectures. Thus, support for configurability must be integrated in any tool-flow targeting these proposition.

In the MoNaDe framework, the (re-)configuration process can be seen from three different perspectives:

- *Structural configuration*, which represents the physical and structural process of configuration at the crossbar level. During this process an otherwise simple wire crossing turn into an active device, either a FET or a diode. At the physical level this process is enabled by different doping profiles at the crosspoint or by an external metallization process. At the crossbar-modeling level this process is seen as morphing one device into another while preserving the structural integrity, especially in terms of connectivity, of the assembly. Figure 4.5 shows the possible configurations of a 2 wire crosspoint as a FET or as a diode, and give an example (Figure 4.5b) of an initially simple crossing (top) morphed to a NFet device (bottom). To enable this the configurable crosspoints are viewed as composite devices with one device connected and one not connected to the embedding crossbar. The morphing process is managed using a configuration controller, which is presented in the following paragraphs.

- *Fine-grain functional configuration*, which represents the detailed configuration of all available resources at the circuit level. The target configuration bitstream is generated based on these information. Instead of a configuration bitstream, for the NASIC fabric the transistor layout is computed during this stage. At this level all details of the target architecture are captured, and set in the final configuration state.

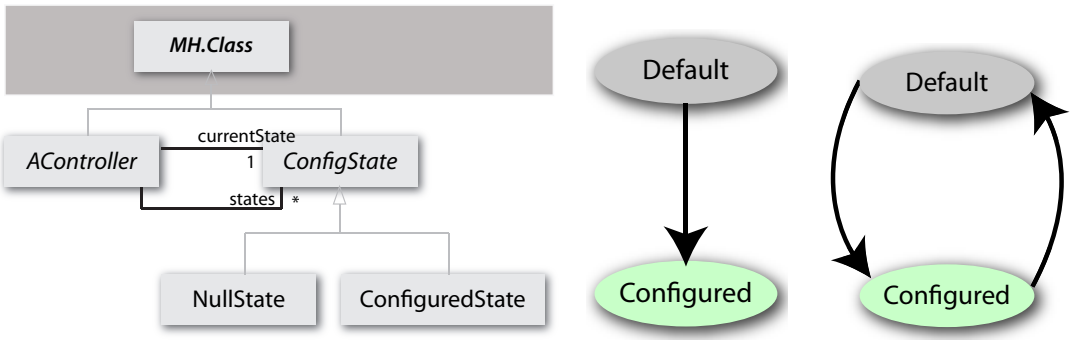


(a) Possible structural configurations of a crosspoint.

(b) Non configured crosspoint (top). Configured crosspoint (bottom)

Figure 4.5: Crossbar-level reconfigurable point. Possible configurations and example.

- *Coarse-grain functional configuration*, which represents an abstract, functional view of the fine-grain configurations. This view is used to speed up some of the physical design algorithms by providing them with a condensed representation of the target architecture. i.e. at this level the wires in a routing channel are seen as a virtual connection having a certain capacity and an occupancy.



(a) Abstract configuration model

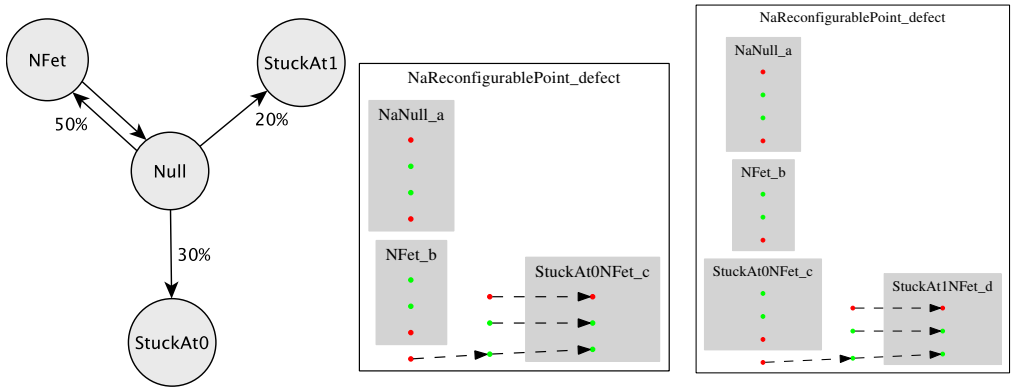
(b) One-time configuration state machine

(c) Reconfiguration state machine

Figure 4.6: Configuration model and two different configuration policies

For representing the configuration state of any circuit element (at the crossbar, and circuit level) we rely on a configuration controller described based on the state-machine design pattern, presented inFigure 4.6. These fine-grain controllers represent the real configurability of each physical component and are managed as a whole based on a global configuration controller. The global configuration controller corresponds to the physical configuration controller present on the circuit.

Figure 4.6b, and 4.6c show two different configuration policies. One-time configuration (Figure 4.6b) is characteristic for anti-fuse-based FPGAs[89], and NASIC in our case. In these cases,



(a) Fault injecting configuration controller (b) StuckAt-0 fault model (c) StuckAt fault model. The reconfigurable point is stuckAt-1.

Figure 4.7: Fault injection using fault-configuration and examples of defective reconfigurable point

the architecture, even though programmable, can only be programmed once by applying a high voltage that will turn the anti-fuse into a conductive device, or during the NASIC metallization phase. Reconfiguration (Figure 4.6c) is a reversible configuration process that enables the use of the same physical circuit for executing different applications according to the configuration chosen.

4.2.2.2 Fault-Modeling and Injection

Since the presence of high rate of defaults is one of the major concerns for all nano-electronics industry. Support for defect and fault modeling, injection and fault-tolerance policy evaluation has to be considered in design automation tool-flow targeting these nanoscale computing designs.

In the MoNaDe toolkit the support for defect/fault modeling emerges inherently, and is based mainly on the object-oriented design and the configuration policies (described in the last section). In this section we discuss briefly the methodology used.

Defect and Fault Modeling. From a modeling perspective, the default and fault modeling is simply a matter of specializing the defect-prone domain concepts into faulty entities according to the domain-specific default and fault models. In our case any MH.Entity, MH.Hook, or MH.Connection entity (or their subclasses) can be specialized into a faulty element, that will be treated as such by the defect-aware tools of the toolkit.

If we consider, for example, the stuck-at transistor fault-model its integration into our domain-model is done by specializing the NFet and the PFet entities of the fabric model (see Figure 4.8 for details) into a StuckAt0 and a StuckAt1 entity. The same process can be applied to the Fa.Wire entities, to represent broken wires.

At the higher, circuit abstraction level, only the faults will be modeled through a similar process. Consider, for example, a nanoscale block A connecting two other block C and D. If the block A happens to have a defect pattern which manifests as the impossibility to connect C and D, the we can view it as a faulty block, and use a specialization of the initial block to represent this faulty element.

Defect and Fault Injection. For defect and fault injection two different mechanisms are considered: fault-configuration and object-swapping. *Fault-configuration* is one possible way of injecting faults using the configuration mechanism. Figure 4.7 illustrates this mechanism. A fault-aware configuration controller is created. This controller includes the defect/fault model, and the configuration state machine is transformed to a probabilistic state machine, where the transition probabilities reflect the probability that a certain fault occurs. Figure 4.7a shows an example with

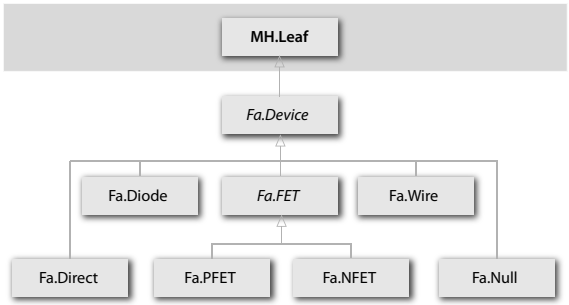


Figure 4.8: Primitives for crossbar-level modeling

the stuck-at fault model for a NFet reconfigurable point (CP). The CP managed by this controller has a 50% fault probability, with a 30% probability of stuck-at 0 and 20% stuck-at 1. These probabilities represent experimentally found technology-specific probabilities for the occurrence of each fault. The fault injection via *object-swapping* is used for non-reconfigurable resources, or in the cases where the certain fault have to appear with a 100% probability, for testing different hypotheses. This mechanism, much simpler than the *fault-configuration*, replace the correct object instance in the domain model with a faulty version (according to an arbitrary fault model) of the same object. This mechanism is used for example for injecting broken wires into the model, and in this case the fault probability is applied in a centralized manner. If the broken-wire probability is for example 0.03%, then all the wires (of a certain type - nano or lithographic scale) are selected and 0.03% of them - arbitrarily chosen - are replaced by broken-wire instances.

From these two mechanisms, the first one has the advantage of being able to model the defect/-fault probabilities in a fine-grain manner - localized at each CP - with the drawback of eventually using large amounts of memory. This method is used to inject clustered faults, or other spatial distributed faults at precise locations. The *object-swapping* method reduces the amount of memory used but considers all similar device types at the same time without discriminating about their physical placement on the surface, ideal approach for uniformly distributed faults.

4.2.3 Crossbar-level modeling

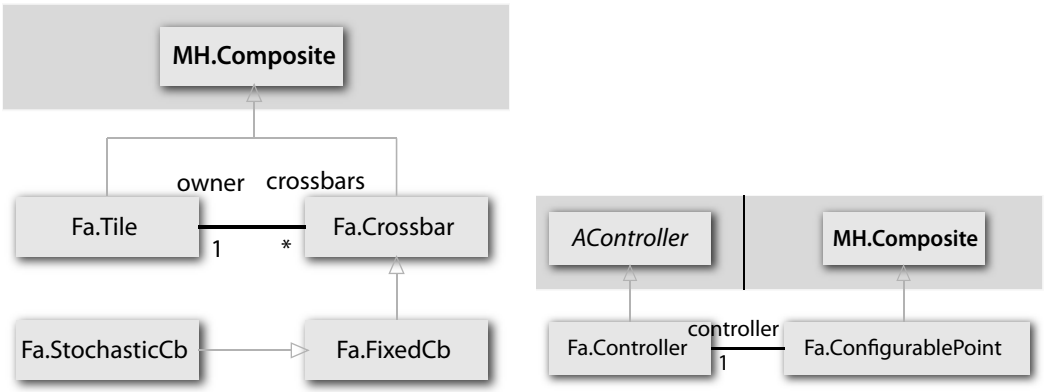
Above the device-level, the fabric abstraction level focuses on the creation of architectural building blocks that are to be integrated into fully functional designs. The structural axis, models the interconnection between the devices trading off the reliability of lithographic-scale components with the advantages of nanoscale devices. From a behavioral point of view, the computational components are simulated and validated using compact models, and spice-like simulators. The physical aspects of interest at this layer are the layout of the nano and lithographic layer such that the density is maximized ensuring physical correctness while meeting the eventual fabrication constraints.

In the context of this thesis, the core fabric-level structure considered is the crossbar. At this level, MoNaDe framework proposes a common vocabulary for describing and reasoning about these architectures.

Figure 4.8 shows the fabric primitives, used for crossbar-level modeling. They represent devices such as FETs and diodes, as well as wires, and direct connections (capacitive connections between wires).

Figure 4.9a shows the core composite components considered at this level. This models the crossbars and their composition in tiles. The Fa.Crossbar structure, representing reconfigurable crossbars, is further specialized as Fa.FixedCb for modeling non-reconfigurable structures like in the case of NASIC, or the inversion/buffering crossbars of NanoPLA. Fa.StochasticCb models decoder-like crossbars similar to the ones used for NanoPLA configuration, or stochastic inversion/buffering.

At this level, the configuration is viewed structurally, and modeled with Fa.ConfigurablePoint



(a) Composites for crossbar-level modeling (b) Configuration modeling at crossbar-level

Figure 4.9: Crossbar-level composites and configuration models

instances controlled by a configuration controller, see Figure 4.9b. Fa.ConfigurablePoint models the different structural configurations presented in Figure 4.5.

This fabric model is compatible with the abstract crossbar model presented in 3.3.3. The Fa.Tile instances are converted towards graph representations for logic mapping using the vf2 algorithm[25]. For simulation a mapping can be created between Fa.Device instances and compact behavioral models for spice simulation.

To illustrate this approach the following sections present the crossbar-modeling support via two case studies: an application-specific architecture, namely NASIC, and a reconfigurable architecture, namely NanoPLA. In the case of the other two architectures considered for this study (CMOL and FPNI) the results are similar and thus they are not presented in detail.

4.2.3.1 Case I: NASIC

To model the NASIC fabric architecture the particularities of the fabric have been identified and classified as nanoscale specific, nano/CMOS interface and CMOS superstructure elements. The main architectural building block is the NASIC tile which is composed from two nanoscale crossbars assembled together to form a PLA-like structure. One particularity of these crossbars is that they are application-specific (non-reconfigurable), the functionalization of the PLA is done during the manufacturing process and cannot be changed afterwards. One important observation is that from a software engineering point of view the functionalization[122] process can be seen as one time configuration of a reconfigurable architecture (a configuration which cannot be changed during the lifetime of the PLA). Based on this observation the NASIC crossbars are represented using the concept of reconfigurable crossbar with the constraint that the crossbar can be configured only once during its lifetime (Fa.FixedCb). Around the PLA-like structure in NASIC we can identify some additional nanoscale control circuitry for pull-up/down or for dynamic logic implementation. These structures are modeled at device level by composition of the wires and the active devices which implement them. The nano/CMOS interface is represented by direct connections between nanoscale wires and micro-scale wires. For a NASIC tile the CMOS superstructure consists of micro-wires, surrounding the NASIC nano-tile, which carry ground, power supply voltage, and control signals for the dynamic evaluation of the output.

Figure 4.10 presents a high-level view of a dynamic-style NASIC tile modeled using CVA. In the middle of the figure the two crossbars creating the PLA-like structure can be identified, having around the additional nanoscale control circuitry (veva, hdis, vpre, and heva), required to implement a fully operating dynamic-style NASIC tile. The nano/CMOS interfaces, presented in the figure, create the connection between the nanoscale components and the CMOS superstructure.

The logic placement on a NASIC tile is implemented using a simple mapping algorithm that maps the logic circuit expressed as a PLA to the tile according to the logic type of the PLA-like

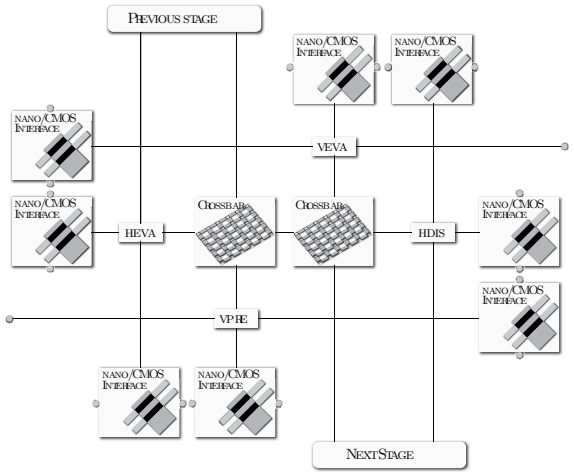


Figure 4.10: Dynamic style NASIC tile modeled using CVA

structure (e.g. AND-OR, NOR-NOR, etc.). Once the mapping is computed the crosspoints of the crossbars are configured in the final state.

The obtained tile can then be simulated using a switch level circuit simulator, or a device level simulator like SPICE. To simulate a specific tile a model-to-model transformation is done between the high-level tile model (presented in Figure 4.10) and the specific simulation model used (switch level model or device level model).

4.2.3.2 Case II: NanoPLA

The NanoPLA architecture uses a reconfigurable PLA-like structure as the main architectural building block. Two types of crossbars have been identified: a reconfigurable crossbar having diodes at the crosspoints in the configured state (see Figure 4.11 the crossbars labeled OR), which is used to implement an OR (or an AND) logic stage, and a crossbar structure used for inversion and buffering which is somehow similar to the crossbars present in the NASIC architecture since they are not reconfigurable (the crosspoint FETs are created during manufacturing step). There have been identified three nano/CMOS interfaces in NanoPLA architecture: one configuration interface (see Figure 4.11, to the left labeled config.), used to configure the reconfigurable crossbars, one control interface (see Figure 4.11, around the crossbars labeled control) realized by the funtionalisation of the crosspoint between a nanowire and a micro-wire used for passing control signals to the crossbars, and one direct nano/CMOS interface, realized via ohmic contacts between nano-wires and micro-wires, used for interfacing with the CMOS superstructure carrying power, ground signals.

4.2.4 Circuit Modeling

At the circuit level the main concern is the optimization of fabric resources used for application implementation. At this level the fabric blocks are composed together into architectures, the fabric details are abstracted away to a functional view. The applications are functionally placed on a mesh of functional blocks, then routed to implement the design. From a simulation point of view, the switch-level and device level behavioral models are replaced by their logic representation.

In our case, due to the regularity of assembly most of the architectural propositions are implemented using regular meshes of fabric blocks (crossbar tiles), approach conceptually similar to today FPGA designs. Thus, the model used in the MoNaDe toolkit for circuit modeling is based on the FPGA model of the Madeo framework[90].

To capture the particularities of these architectures, the meta-model, presented in Section 4.2.1, was refined. Figure 4.12 shows the domain-specific concepts added. Using this meta-model the

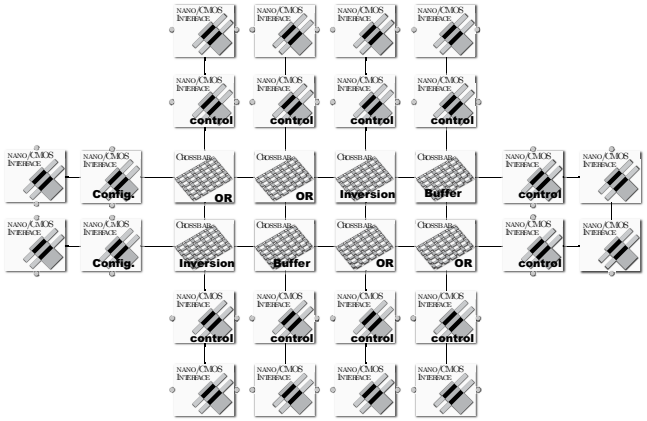


Figure 4.11: NanoPLA tile modeled by CVA

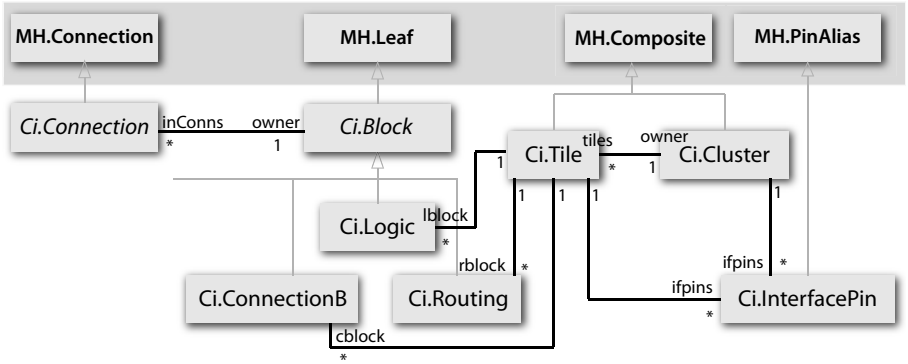


Figure 4.12: Circuit model extension of the core meta-model

circuits are modeled based on 3 primitives (specializations of MHLeaf): Logic Block, Switch, and Connection Block. *LogicBlock* instances represent the logic blocks which provide the basic computation and storage elements. The routing architecture is modeled using *Switch* and *ConnectionBlock* instances. These primitives contain a number of *internalConnections* that specify the way their pins are connected internally (see Figure 4.13). *Tile* instances aggregate these primitives into logical bricks which are replicated regularly to form a *Cluster*. The primitive's pins are aliased at the tile level and cluster level, using *InterfacePin* instances, to expose the possible connection points.

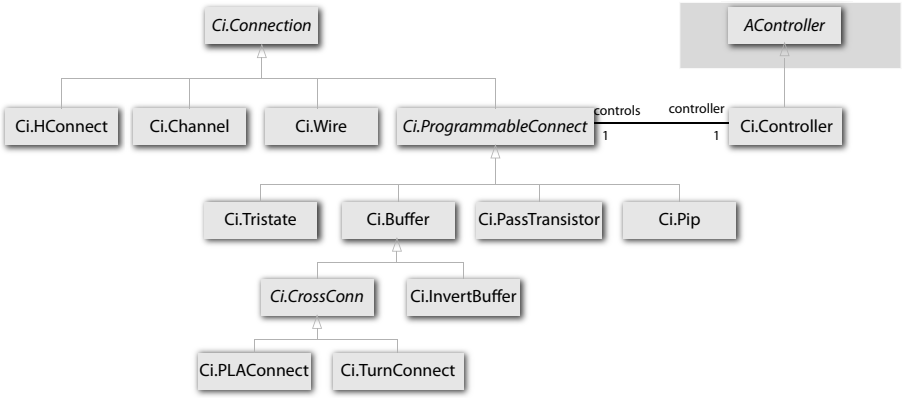


Figure 4.13: Connection hierarchy

Figure 4.13 presents a view of the principal types of internal connections. *HardConnect* can be seen as a direct connection between components without any physical property (it is used for example to represent long wires which logically are composed of different wire segments, the connection between these wire segments is a HardConnect instance). *Wire* represents a physical wire. *ProgrammableConnect* is an abstract representation of electrically configurable components. For crossbar-based routing approaches Ci.CrossConn instances are used to represent through-crossbar connections. These connections can be Ci.TurnConnect, representing a connection using a crosspoint to connect two components, or Ci.PLAConnect, representing a connection passing through two crosspoints. The configuration state for these components is controlled by an instance of *Ci.Controller*, which implements a configuration state machine.

In the following sections the instances of this circuit model are named ArchM.

4.2.5 Application Model

In this section we present another refinement of the meta-model, presented in Section 4.2.1, this time focusing on hierarchically description of the application. This application model is an internal model used as a common representation for application specification, similar to the Intermediate Representation(IR) used for language compilers.

Figure 4.14 show the primitive and composite entities used. The different specialization of the SC.LogicNode are used to represent logic primitives from different providers. They can be used for problem decomposition, and are refined through successive transformations using external logic synthesis tools to match the target architecture primitives. The concrete entities modeled are similar to the internal representation used in the Madeo toolkit, but in our case they use the facilities provided by the abstract model (e.g. pins, ports, hierarchical composition, etc.), instead of implementing a different object hierarchy as is the case in Madeo.

The logic connection between logic blocks are modeled using SC.Signal entities, which simply connect the output pin of a logic block to its fanout. Figure 4.15 presents these concepts. The SC.Netlist, and SC.VectorialPort can be seen as syntactic-sugar[4] representing a collection of sig-

[4] *Syntactic-sugar* is a computer science term that refers to the syntactical structures within a programming

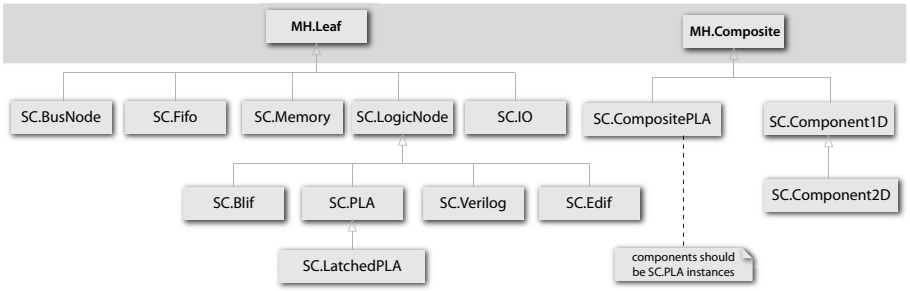


Figure 4.14: Extension of the core meta-model for application modeling (entities)

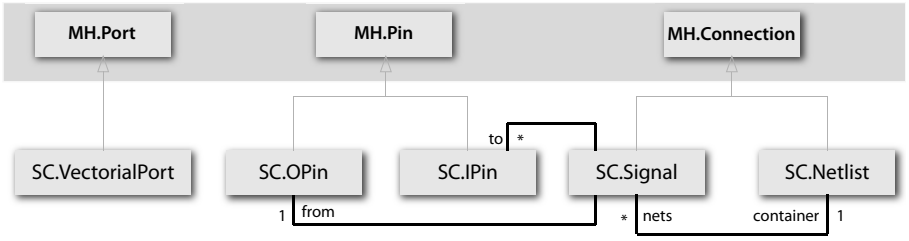


Figure 4.15: Extension of the core meta-model for application modeling (connection)

nals, and respectively a port instance with consecutively numbered pins (e.g a 32bit integer port having pins from 0 to 31 representing each bit).

In the following sections the instances of this application model are named AppM.

4.3 Transformation Metaphor for Tool Design

Besides structural domain modeling, algorithm design is the most important aspect of any EDA toolkit. Since almost all the optimization problems encountered in electronic CAD tools are NP-hard in terms of complexity, most of the practical solutions rely on heuristics. Hence, the main concern of EDA tools users and designers is the heuristic performance in terms of execution time, memory requirements, and solution optimality. It is commonly accepted for an EDA tool to run for days or weeks on high-end machines having huge memory resources. In terms of implementation, these optimization heuristics are very complex. Most of them are implemented in highly optimized C, for performance issues.

Physical-design toolkits, such as Madeo[90] and VPR[9], rely on external tools, mainly for logic synthesis and technology mapping. But it implements generic heuristics for the physical design problem to assure the flexibility of the mapping. The idea is that these generic heuristics can be used in the context of different FPGA architecture with the minimum human intervention for optimization goals parametrization.

To address the algorithm design problem for the MoNaDe toolkit, we propose a technique that we call "Transformation metaphor". This technique appears as a conceptual framework; the algorithm designer looks at the algorithm implementation as it was a model-transformation problem.

This approach mainly reifies the tight implicit dependency between algorithms and structural domain models, through explicit transformations, isolating their respective concerns, thus increasing the flexibility and the expressivity of the toolkit.

With the transformation metaphor each algorithm (or heuristic) is seen as a hierarchical composite transformation. Figure 4.16 presents the different types of transformations proposed by

language that are designed to make things easier to read or to express, while alternative ways of expressing them exist.

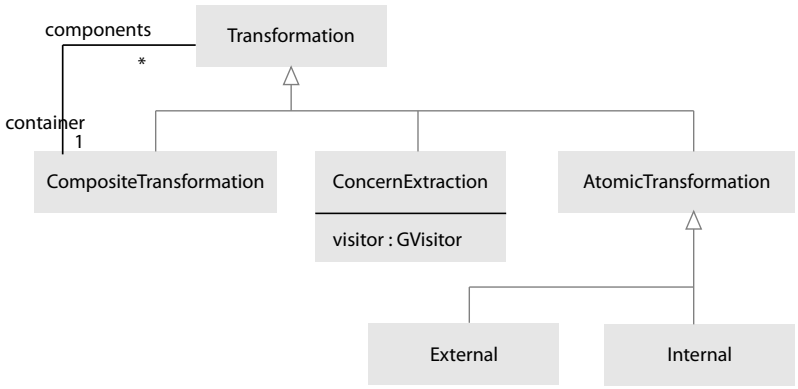


Figure 4.16: Transformation metaphor model

this approach. The primitive transformation types being: *concern extraction*, and *atomic transformations*.

The *concern extraction* represents the mapping from a domain model to simpler models required by the algorithm being implemented. The purpose of the concern extraction is to decouple the domain specific details of the model from the algorithm-specific details.

Assume, for example, that the domain model contains an entity with 4 attributes: id, color, time, and distance. If an algorithm X needs to know the speed in order to execute, then there is no need to pass it the color attribute. Moreover since the speed can be computed as *distance/time*, why not compute it first and then pass directly the speed to the algorithm X. We call *concern extraction* this model pre-processing step, that extracts the time and the distance from the domain instance and then computes the speed that will be passed to the algorithm X.

From an implementation perspective, the *concern extraction* is nothing more than implementing a visitor (according to the visitor design pattern[51]) that will iterate over the domain model extracting information and instantiating an algorithm-specific model.

The *atomic transformations* represent actual algorithms (or heuristic) needed to solve the problem. According to the specific needs it can further decomposed in more fine grain modules and composed using the transformation metaphor or it can be directly implemented in a programming language of choice. The only requirement is to export clear interfaces so that it can be integrated as an *atomic transformation* in the framework.

This approach has the advantage of being able to integrate external tools implemented using any programming language and/or computing model. Future work focuses on formalizing these concepts into a concrete transformation engine, which will be able to provide the users with an environment for algorithm design and integration into domain specific tool-flows.

Case Study – Routing Algorithm

This section presents in detail the decomposition of a standard FPGA routing algorithm using the "transformation metaphor" to better illustrate the approach.

The FPGA routing architecture can be modeled with a routing graph $G_r(V_r, E_r)$[110, 9], which is a directed graph. The vertex set V_r represents the input and output pins of logic modules, and the wire segments. The edge set E_r represents the possible connections between the nodes. The nets to be routed are directed hyper-edges on G_r, $(s_i, t_i^j)_{1 \leq j \leq n}$, where s_i and t_i^j represents the source, and respectively the sinks of the net i.

A net routed corresponds to a subtree of G_r, called the routing tree for the net. The root of the routing tree corresponds to an output pin (in the FPGA architecture) and is the source of the net. The leafs of the tree are input pins of logic blocks, and correspond to the sinks of the net. Since an electrical path cannot be shared by different signals the routing trees for the nets should be vertex disjoint, this is called the exclusivity constraint[20].

The routing problem can be stated as finding vertex disjoint routing trees in G_r for all nets while satisfying performance constraints.

In the context of this section we consider a timing-driven router based on the Pathfinder algorithm[110]. For timing calculations we use Elmore delay model[47], that uses the architectural parameters to compute the source-to-sink delay. In timing-driven routing, the timing constraints are specified as arrival time (A_t) and required time (R_t) at primary inputs or outputs of storage elements. A timing graph, $G_t(V_t, E_t)$, is created from the placed application netlist to represent these informations.

The goal of the timing-driven routing problem is to route all nets such that the delay on the critical path is minimized while satisfying the delay and exclusivity constraints.

From the MDE perspective, the routing problem is a refinement of the partial ArchM instance. This refinement consists in the identification of the routing paths in the ArchM instance for the nets from the AppM instance. The transformations required to solve the routing problem are classified in 3 categories: pre-routing transformations, core routing, and post-routing transformations.

The pre-routing transformations, are a preprocessing step that identifies and extracts the elements directly needed by the core routing transformation, G_r, G_t, and the nets in our case. The pre-routing transformations are simple transformations, concern extraction (as we called them in the last section), that can be implemented using transformation engines.

The *RRGraph extraction* transformation creates the routing graph G_r from the ArchM instance. It also computes the delay for the possible connections by applying the delay model on the RC parameters of the ArchM instance. The *nets* are extracted from the AppM instance using the placement results to identify the architectural elements corresponding to source and sinks. A partial instance of the *timing graph* is created G_t from the placement results and the AppM instance. We call G_t a partial instance because the delays of the routing paths between the logic blocks are not known, being computed during the core routing transformation.

The core routing transformation, is a composite transformation (see Figure 4.17) of the main algorithmic aspects of the routing problem. For the timing-driven pathfinder algorithm these aspects are: signal routing, and timing graph refinement. If these aspects are treated as transformations, they can be implemented apart from the core algorithm, thus decomposing the routing problem event further. The principal roles of the routing, in the case of Pathfinder, is to produce the routed nets according to the timing and exclusivity constraints, to refine the timing graph model, by adding the cost of the routing paths, and to update the costs of the routing resources in G_r, since the exclusivity constraint is solved by resource negotiation.

Once the core-routing transformation done, the post-routing transformation refines the ArchM instance injecting the routed nets in the model. The ArchM instance obtained represents the solution of the physical synthesis flow, the mapping of the AppM instance on the architecture.

Using this methodology we were able to experiment three different routing algorithms:

- the timing-driven pathfinder, described in this paper;
- a routability-driven router realized by removing, from the flow in Figure 4.17, the *TGraph Extraction, AT Refinement and RT Refinement* transformations;
- a lagrangian-relaxation based timing-driven router[99];
- the pathfinder-based router implemented in the VPR tool[9].

In the case of the VPR tool, the pre-routing transformations create an instance of the VPR architectural model using the XML-based description language, and a netlist using the .net file format needed by the VPR tool. The post-routing transformation reads the routing results file and injects the routes in the initial ArchM model instance. This shows that core-routing transformation, the Pathfinder router in the example, can be easily replaced by any other domain independent tool having the required functionality. Moreover this result is valid for all the other core algorithms needed in the physical synthesis flow, presented at the beginning of this section.

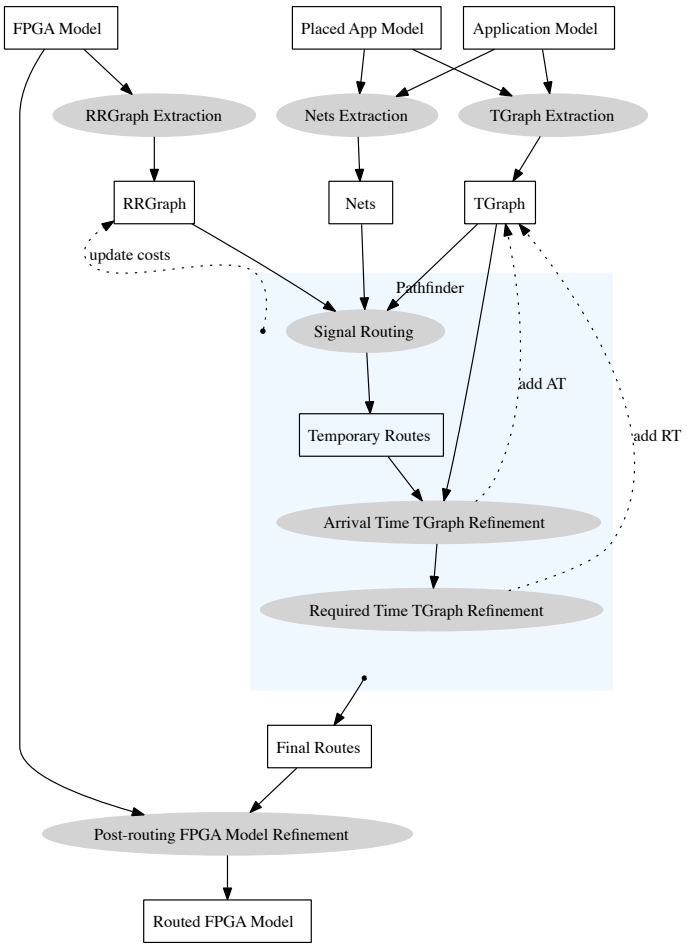


Figure 4.17: DAG representation of the composite routing transformation. The ellipses represent primitive transformations. The core-routing transformation is shown as the shaded area. Dotted lines represent model refinement (cost update, property injection), they do not alter the DAG structure, but represent updates of shared models

4.4 Tool-flow Modeling

The physical design is responsible for allocating all design components for creating the configuration bitstream for FPGAs, or the physical layout for ASIC. This means that each gate of the application netlist will be assigned a spatial location (placement). Then, the interconnection signals will be reified using appropriate routing structures. Physical design has a direct impact on the circuit characteristics (performance, area, power, etc). The main steps of physical design are: partitioning, floorplanning, placement, and routing[79].

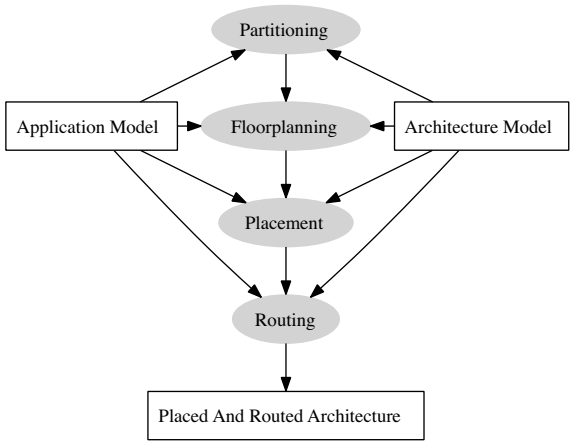


Figure 4.18: A standard physical design flow. The rectangles represent models, while the ellipses represent algorithms.

Figure 4.18 shows a typical FPGA physical synthesis flow. It starts from the description of a particular FPGA architecture as an architectural model (ArchM) instance, and a target application as an application model (AppM) instance. These models are domain specific, fully independent from any algorithms that are used for physical design. The output is a refined ArchM instance, configured to implement the AppM instance.

The flow is an endogenous transformation realized via four composite transformations (see Figure 4.18): partitioning, floorplanning, placement, and routing. Each of these four transformations is a composition of more-elementary transformations. Thus, the physical synthesis tool flow is a hierarchical directed acyclic graph (DAG) of transformations, where the nodes represent the transformations to be done, and the edges represent the dependencies between the tools. The execution of the transformations happens in topological order from the DAG inputs (ArchM and AppM instances) to the outputs (AppM instance mapped on the ArchM instance).

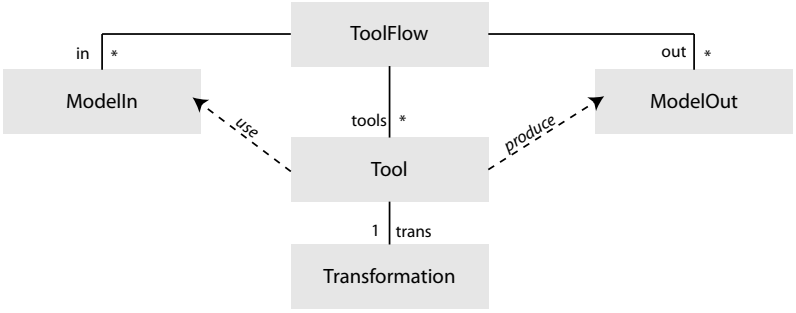


Figure 4.19: The abstract toolflow meta-model

Figure 4.19 shows the abstract toolflow model. A *ToolFlow* instance takes a number of *ModelIn* as input and produces a number of *ModelOut* as output using any number of tools to do it. Each

tool is uniquely associated with a *Transformation* (either composite, or atomic). In our case the a *ToolFlow* instance is created with AppM and ArchM models as inputs (*in*) and four *Tool* instances corresponding to the four automation steps. The result produced would be a refined ArchM instance.

The principal advantage of this flow is the capacity to easily replace any of the physical synthesis algorithms with different implementations, with no constraint on the implementation language or formalism. But this also has a drawback, the high number of simple transformations (concern extraction) needed to implement the flow. However, such a view of the physical design automation, that isolates and encapsulates the different steps of the flow, poses the bases for future standardization, which that can cut down the number of intermediate transformations.

4.5 From Legacy to MDE Toolkit

This section provides an account for some of the steps we followed toward the MDE-based framework. Moreover it shows the flexibility of the approach as the environment remained usable during the whole evolution process.

4.5.1 Improving on Legacy – First steps

The first modifications that were integrated in the toolkit focused on improving some of the optimization routines already present in our legacy code-base, namely the floorplanning and routing routines.

In the case of the floorplanning routine, we have chosen to replace the TCG-based[103] heuristic present by an improved version relying on a different floorplan representation, namely TCG-S[102]. From the implementation perspective we tried to decouple as much as possible the heuristic from our domain models, so that it can be reused in other context with no modifications. The integration into the toolkit was done by redirecting the automation flow toward the newly created module. Concern extraction was used to instantiate the TCG-S specific floorplan model from the AppM using ArchM geometrical information. Once the floorplan model instantiated, the optimization goals (metrics) where added as closures (smalltalk blocks) independent of the heuristic implementation.

For the routing routine we have refactored the existing routing algorithm (Pathfinder[110]) decoupling it from the architectural model with which it had numerous dependencies, and we created a transformation-based version (see Section 4.3 for more details). As for the TCG-S algorithm, the architecture specific optimization goals are set using closures. The results using this new implementation were impressive in terms of execution speed (over 40% faster), principally due to the possibility to prune the routing resource graph, thus reducing considerably ($\geq$50%) the number of nodes explored during the execution. One negative aspect of using this approach is the increase in the memory footprint due to the duplication of some ArchM aspect in the routing specific model.

4.5.2 Extensions for Nanoscale Physical Design

In [96, 40] the extensibility of the MADEO framework was put to a test for the first time with the advent of emerging technologies. The core concepts of the NASIC fabric[115], see Figure 4.20, were introduced into the framework, and a reconfigurable nanoscale architecture, called NFPGA, was designed. This required to extend both the reconfigurable architecture model and its associated tools in such a way that NASIC can be modeled and programmed. Process that goes through several steps:

1. The generic model must evolve to describe new potential components (nanogrid, nanowire, etc. . .) with specific characteristics.

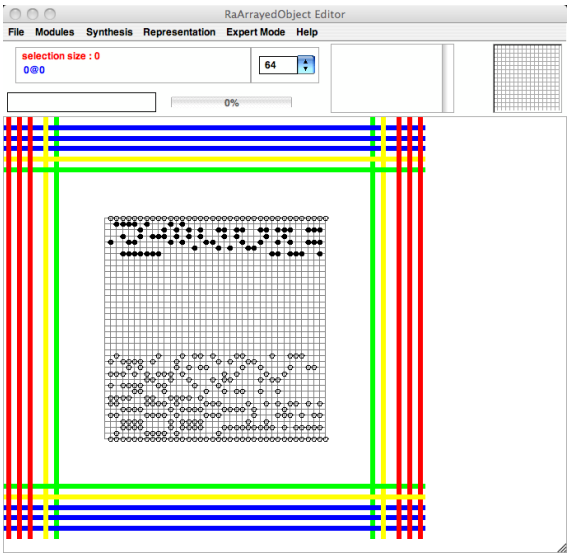


Figure 4.20: Madeo viewer on an nanoscale tile instance

2. This generic model must be instantiated using a proprietary HDL. As the HDL expresses the model, any in-depth change within the model leads to an evolution of the HDL(i.e. new keywords).

3. Some algorithms must be adapted or substituted for technology-specific counterparts while preserving the API. For example, the logical functions are implemented using a 2 level logic rather than FPGAs LUTs or processor μ-instruction.

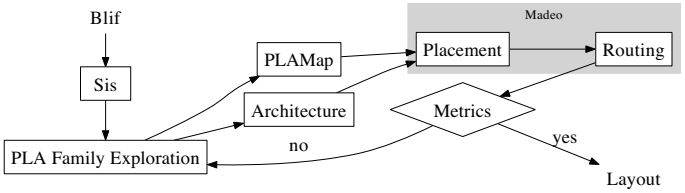


Figure 4.21: The R2DNasic CAD flow.

More recently[172, 171] the methodology presented in this study was used to propose a complete physical synthesis tool-flow for a new nanoscale architecture template. As it can be seen in the Figure 4.21 the Madeo toolkit legacy was used for placement and routing (as well as for archM description, instantiation and visualization), external tools like Sis[150], PLAMap[19] where seamlessly integrated with new internal tools for pla family exploration, and metric computing. Different tool-flows were created using these tools, each one having different optimization goals, and working on different architecture variants. Moreover by opening the toolbox the design-space exploration (DSE) was bootstrapped relying on standard reconfigurable place & route routines, thus enabling a baseline evaluation which showed the need for more optimized routing. Once the new routing algorithm was developed it was integrated into a new tool-flow, specializing the baseline tool-flow via inheritance.

The main conclusion of this experiment is that using this MDE approach effective incremental design-space exploration is enabled, and a new tool-flow exploration axis is added to the typical application/architecture trade-off, while the tool-flow specialization reduces the development effort.

4.5.3 Refactoring Domain-Models

The most extensive evolution of our legacy code-based was the replacement of old domain-specific OO models with a newly engineered set of FAME-based domain models, relying on the hierarchical port-graph abstraction, described in Section 4.2.1. The preservation of legacy functionality is the principal constraint in this case.

To this purpose we engineered the new models to replicate the old-model entities and then we merged the two models in such a way to factorize the available functionality of the two.

Two automated methods of merging the two models were devised: copydown method, and doesnotunderstand method. They are both explained in detail in the following paragraphs along with their advantages and constraints.

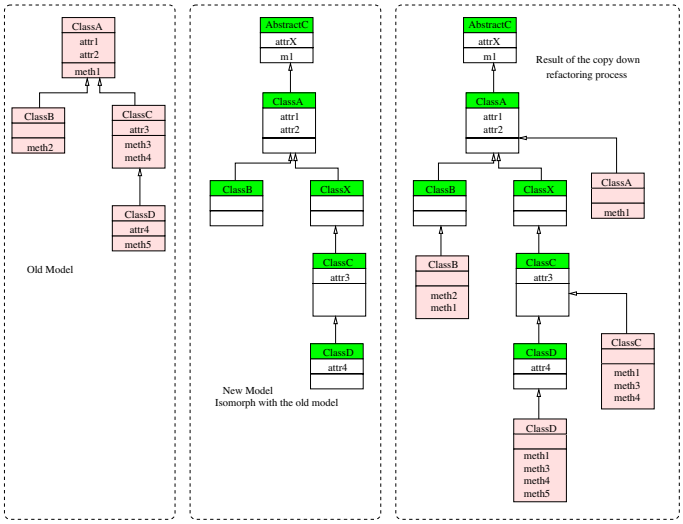


Figure 4.22: Example model transformation using CopyDown method

CopyDown Method Starts by inlining the calls to super methods in order to obtain inheritance hierarchy independent methods that can be safely copied to all subclasses of a specific node. After the inlining step the CopyDown step proceeds where all superclass methods will be copied recursively to all subclasses. The next step is to remove the duplicated instance variables from the old model. This step is required because the new model already contains some instance variables and they will be accessible from the future superclasses. This step being done the old class hierarchy is destroyed, and the new designed superclasses are assigned to the old model classes. Because the two models have some classes with identical names the name clash is prevented by isolating the classes in different namespaces. Figure 4.22 shows the result of this refactoring method applied on an example. The different colors in class representation represent the different namespaces that isolate the classes with identical name. The rightmost diagram shows the method duplication through the classes of the old hierarchy in order to preserve their inherited functionality.

This refactoring method is a good solution to the model-refactoring problem encountered in the development of the framework since the old functionality is maintained, the old applications developed around the old model continue working without any modification, and the new model can be used freely without any execution delay. Another advantage of this method is that it does not change the execution mechanism of the underlying platform, and so it can be used almost unmodified with all OO languages. Still this method has its drawbacks principally because the classes of the old model contain duplicated behavior. This duplication decreases the maintainability of the system, and renders the old model entities less comprehensible.

A solution can be the introduction of another refactoring step that will push up in the new hierarchy the equivalent methods.

But, despite its drawbacks, this method can still be used provided that some necessary precautions are taken:

- If the tools using the old model are mature enough so they can be used in their actual state without modifications;
- If the developers intention is to replace the old model with the new one in all the tools using it. In this case the CopyDown method can be used as an intermediate evolution step, where some tools are ported to the new model and others are still using the old one.

DoesNotUnderstand Method Another method for merging the two models into one directly perform the last two steps of the previous technique (remove the instance variables; Cut the superclass link. Add the new classes as the superclass of all old classes) without copying-down the superclass methods. The old hierarchy must be stored (in a Dictionary for example) in order to be able to replicate the old hierarchy method inheritance. To be able to use the behavior declared in the old model classes, the #doesNotUnderstand method will be redefined. Once the error message is intercepted we start searching in the old inheritance hierarchy to find the implementation class of that message. If we find it we send the message to that class and return the result to the sending class. If the message was not found we simply throw the "does not understand" error.

This method solves the previous problems related to the message duplication throughout the old hierarchy. But it comes with new drawbacks like:

- The execution mechanism of the object oriented framework must be modified. While this is possible in an open context like Smalltalk in most object-oriented environments will be difficult to implement this method.
- Since all missing methods need to be searched in the old inheritance hierarchy this will add some overhead to the overall execution of code using this model.
- If the new model classes implement one of the methods implemented in a class of the old hierarchy, say class X, all subclasses of the class X will execute the new implementation of the method instead of executing the implementation in class X. Thus rendering the tools using the old model unusable. That happens because the "does not understand" error will not be triggered once a method with an identical signature is found in the new inheritance hierarchy.

4.6 Summary

This chapter has presented a model-driven physical design toolkit which enables multi-fabric modeling and incremental design space exploration. Section 4.1 reviewed the state of the art in model-driven development and its application to automated circuit design. Some important software engineering innovations were discussed as they pose the basis for the developments presented in this chapter. Section 4.2 presented the approach used in the MoNaDe framework for structural domain modeling. A hierarchical port-graph model was used for abstracting away the core domain entities to enable fine-grain reuse. After presenting re-configuration and fault-modeling/injection two important transversal concerns, the abstract model was specialized for modeling the target architecture at the fabric and circuit level as well as for application modeling. Section 4.3 introduced the transformation metaphor and detailed its use for software algorithm/tool design relying on the Pathfinder router[110] as a case study. In Section 4.4, the model-driven methodology was used to abstractly describe the automation tool-flow as a dependency graph, approach which reifies the tool-flow decoupling it from the execution platforms thus offering high-degree of flexibility and reuse. Section 4.5 provided some insights on the iterative development process used for refactoring an existing physical-design framework (Madeo Framework[90]) towards the presented model-driven toolkit.

5

Nanoscale Architecture Template and Associated Tools

This chapter presents a detailed case study built around a regular 2D nanoscale architecture template based on NASIC fabric building blocks. The R2D NASIC architecture is presented, along with its evaluation metrics and optimization tools. This architecture enables the creation of highly pipelined circuits while easing the delay estimation at the tool-flow level. At the end of this chapter the creation of highly pipelined circuit design at nanoscale is discussed, and the readers attention is drawn to other architectures similar in this matter with R2D NASIC.

5.1 Introduction

Some nanowire-based fabric proposals emerged which all exhibit some common key characteristics. Among these, their bottom-up fabrication process leads to a *regularity of assembly*, which means the end of custom-made computational fabrics in favor of regular structures designed with respect to the application needs. Hence research activities in this area mainly focus on structures conceptually similar to today's reconfigurable PLA[1] and/or FPGA[2] architectures[165, 160]. A number of different fabrics and architectures are currently under investigation, for example NanoPLA[31], CMOL[165], FPNI[3][160], NASIC[4][115]. They are based on a variety of devices such as field effect transistors (FET)[121], spin-based devices[152], diodes, and molecular switches[161]. All these fabrics include some support in CMOS: some like FPNI would move the entire logic into CMOS, others, like NASIC, would only provide the control circuitry in CMOS. The rationale for this varies but includes targeted application areas as well as manufacturability issues[122].

Apart from the fabrication issues, another limitation lies in the linkage between architecture and exploitation tools. This prevents algorithms/tools reuse, thus hindering shared improvements over fabric designs. This also slows the intrinsic performances comparison through devices, whereas such an ability to compare is the key, driving the domain-space exploration.

Hence, to summarize this short nano-computing landscape analysis, it is important to note that several proof-of-concept architectures exist that take into account some fabrication constraints and support fault-tolerance techniques. What is still missing is the ability to capitalize on these

[1] Programmable Logic Array
[2] Field-Programmable Gate Array
[3] Field Programmable Nanowire Interconnect
[4] Nanoscale Application Specific Integrated Circuits

experiments while offering a one-stop shopping point for further research, especially addressing new algorithms. Sharing metrics, tools, and exploration capabilities is the next challenge to the nano-computing community.

One of the principal developments presented in this chapter is a regular application-specific circuit architecture, based on the NASIC fabric architectures concepts. This architecture, named R2D NASIC, shows a number of very promising characteristics such as:

- full compatibility with the NASIC fabric technological framework and manufacturing pathway[121, 122];
- adaptability to a variety of technological and applicative constraints, such as nanowire length, logic density, physical delay, etc;
- compatibility with the fault-tolerance techniques presented in the context of NASIC fabric[115];
- regularity, which means easier fabrication process in the context of nanoscale technologies that have huge constraints in terms on custom placement and routing of wires;
- capacity to implement max-rate pipelined designs based on its pipelined routing architecture, which paves the way towards high-throughput digital circuits – approaching the theoretical limits of max-rate pipelining presented by Cotten in[26];
- simplified delay estimation, due to the dynamic logic evaluation and pipelined routing architecture.

In contrast to NanoPLA architecture[31], with which R2D NASIC design might seem close due its regular replication of blocks, this study presents an application-specific architecture template and not a reconfigurable architecture. Though, from the tool-flow perspective, this design is similar to anti-fuse configurable architectures (as the nanowire functionalization[122] can be seen as an one-time configuration step). Hence, in this study a design automation flow, based on standard tools used in the reconfigurable field, is proposed for physical synthesis. Moreover we show how this flow can be used without modification to provide a baseline evaluation of the architecture, thus bootstrapping the design-space exploration.

This chapter starts by presenting the R2D NASIC architecture, in Section 5.2, along with its evaluation metrics. The CAD flow used for circuit mapping on this architecture is detailed in Section 5.3. In Section 5.4 the architecture is evaluated and the results obtained by mapping circuits from the MCNC benchmark on the R2D Nasic architecture are reported. Section 5.5 investigates the need of pipelined routing beyond the scope of R2D NASIC, extrapolating the results to other nanoscale architectures, which due to the dynamic logic implementation need pipelined routing for creating high-speed designs. Section 5.6 concludes this chapter overviewing the principal results presented in this chapter, along with trade-offs and some future developments.

5.2 Regular 2D NASIC-based Architecture Template

The Regular 2D NASIC architecture (R2D NASIC) is a general purpose NASIC-based architecture template. It is based on a regular array of cells of identical size that are interconnected by a flexible routing architecture, which enables arbitrary circuit placement and routing while maintaining all timing and signal integrity constraints. Moreover the cell design enables logical application partitioning as interconnected two-level logic functions.

Figure 5.1 presents a high level view of this architecture, showing the main architectural components: the logic block (LB), the connection block (CB), the routing block (RB) and the CMOS I/O infrastructure. These components form a R2D NASIC cell detailed in Fig. 5.3. These cells are replicated regulary to form a cluster. Cells lacking the LB are added at the periphery to assure the structural completeness of the cluster.

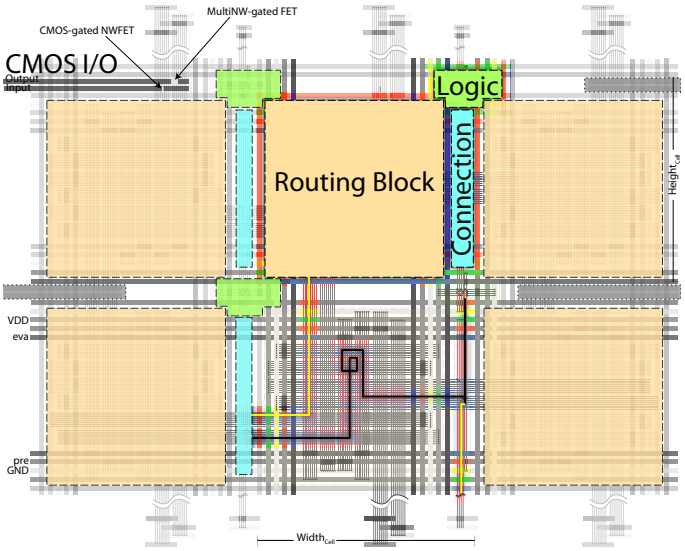


Figure 5.1: A R2D NASIC cluster, showing a 2x3 array of parametric cells. On the left and right sides the CMOS I/O circuitry is shown

5.2.1 Logic and Interconnect

The logic block uses a NAND-NAND two level scheme, implemented using two dynamic xnwNFET stages, forming a tile, as proposed in[120]. Such a tile is characterized by 3 parameters: the number of inputs, the number of minterms, and the number of outputs. These parameters are tunned according to the size of the application circuit, and to the physical constraints, then a custom tile instance is created and replicated in a grid. The inputs and the outputs of the tiles are connected to CBs which link the logic tiles with the routing infrastructure.

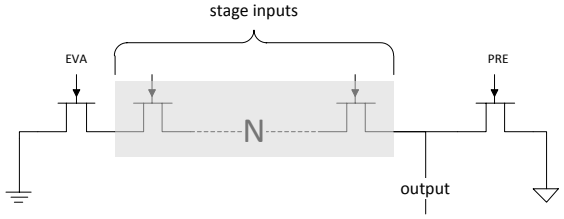


Figure 5.2: The electrical schema of a NASIC dynamic NAND stage

Figure 5.2 shows the electrical schema of a NASIC dynamic NAND stage. These stages are realized using one nanowire connected between VDD and GND, the control signals (EVA and PRE) are realized using xnwNFET transistors controlled using lithographic scale wires, and the inputs are fed-in from the output of other dynamic NAND stages created using NWs.

The Figure 5.3 presents the layout of a cell, which is composed of a LB (top-right), a CB, and a RB. The interface between the nanowires with the CMOS support circuitry is done by using CMOS controlled FETs, which are used especially for providing good control signals for the dynamic evaluation stages. In the top-left part the CMOS I/O interface is presented, which will be detailed in the following section.

The routing architecture is built using routing elements based on dynamic logic evaluation stages which operate by signal inversion. Figure 5.4 shows the electrical schema of a signal routed through two stages one realized on a horizontal stage followed by the second one on a vertical stage.

The RB assures the connection between different routing channels. One particularity of the

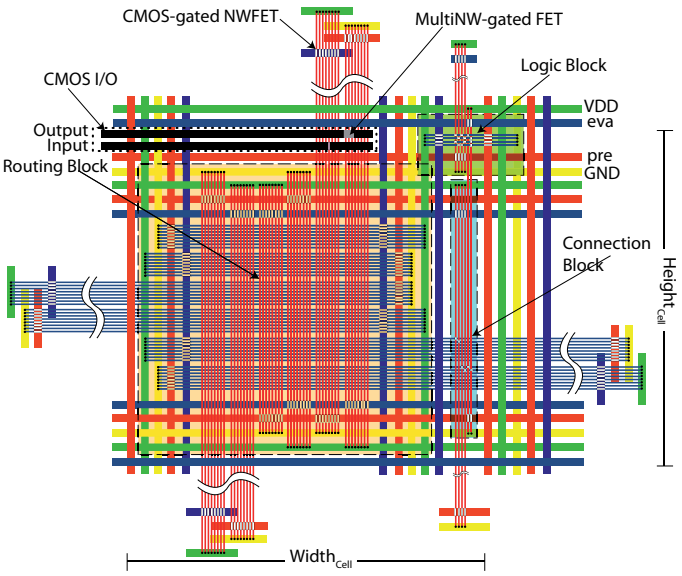


Figure 5.3: The layout of a R2D NASIC Cell. The thinner wires represent the nanowires.

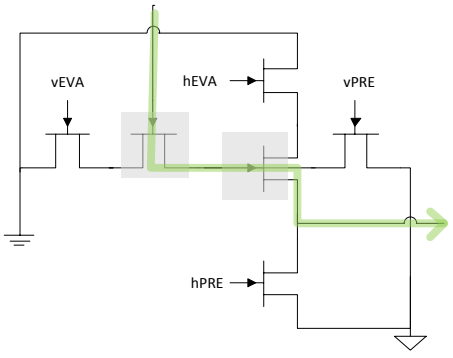


Figure 5.4: The electrical schema of a signal routed through two dynamic stages

RB is that it has one set of vertical (and one set of horizontal) directional routing tracks used to ease the signal routing inside the RB but also to delay a certain signal a number of stages (e.g. the signal a-c, in Figure 5.5). This feature can be used by the routing algorithm to balance the pipeline stages to create high-throughput circuits approaching the max-rate pipeline limits.

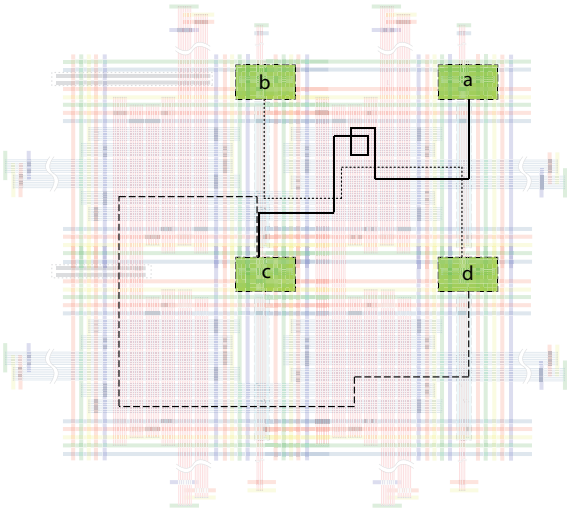


Figure 5.5: R2D NASIC signal routing example.

Figure 5.5 shows an example of routing 3 signals ($b-d$, $d-c$, and $a-c$) on a four cell array. The propagation latency of signal $b-d$ is 2, since it needs four evaluation stages to get from b to c and each evaluation period has 2 stages. The latency of signal $d-c$ is 3. In consequence, the signal $a-c$ needs to have a latency of 5 in order for the logic block c to issue one result each period (each *heva* assertion, see Sec 5.2.3). Thus the signal $a-c$, that could be routed with a latency of 2, has been delayed 6 evaluation stages, inside the SB, to satisfy the latency constraint.

5.2.2 Lithographic Cluster I/O

In the case of R2D NASIC cluster, the input and output signals will be provided via lithographic pitch wires. The lithographic circuitry and wiring (which is also used for the power, ground and control signals) provides a reliable structure for providing the input signals and for collecting the computed results.

The simplest arrangement uses the cells at periphery of the cluster to attach lithographic scale wires on the inter-switch vertical connections.

To drive an input to a certain nanowire a lithographic gated NWFET will be placed at the crosspoint. Since the lithographic-scale wires are wider the NWFET will have a wider gate, and thus a better control. In the case of the outputs a similar arrangement might be possible but this time the nanowires will act as gates for a lithographic FET. One variation of this scheme, as shown in the Figure 5.3, is to use multiple nanowires (carrying an identical signal) as a multiple gate FET to provide strong switching for the lithographic-scale FET. The principal advantage of this approach, besides its simplicity, is that the fabrication process presented in[122] is not altered, the I/O resources being just a particular case of control signals.

5.2.3 Sequencing schemes

Within a cluster the logic blocks implement a 2 level NAND-NAND logic style. The routing structure is based on cascaded inverting stages. Each routing element inverts the signal but the routing architecture is designed to guarantee the unchanged signal transmission between logic blocks. Using an even number of routing elements, the signal is routed always through $2n$ inverting stages.

R2D NASIC uses a three phase control scheme[118] (precharge, evaluate, hold) that precharges and evaluates a stage before the next, and the control signals are repeated every two stages. This offers the advantage of reusing the same control signals every two stages, and also enforces double signal inversion thus guaranteeing the correct signal transmission in the interconnect.

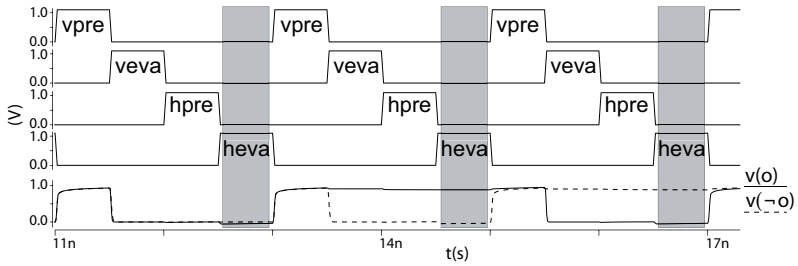


Figure 5.6: Pipelined R2D NASIC circuit HSpice simulation results using the 3 phase sequencing scheme.

Figure 5.6 shows HSpice simulation results of a multistage circuit based on the 3 phase control scheme. For this simulation the xnwFET device model, presented in [121], was used. During the first assertion of the *heva* signal the circuit does not produce any results because of the latency of the pipeline. After filling the pipeline one result is issued each evaluation period, as can be seen during the second and third assertions of the *heva* signal.

5.2.4 Parameters

Due to the simple design of R2D NASIC cells and their regular replication, a limited number of architectural parameters are used to describe the clusters:

- IN - the number of input of each logic block
- OUT - the number of outputs of each logic block
- $MTERMS$ - the number of minterms of each logic block
- W_x - the number of horizontal routing segments
- W_y - the number of vertical routing segments
- $N_{I/O}$ - the number of the I/O lithographic scale wires for each cell at the periphery.
- $Rows$ - the number of rows in the array
- $Columns$ - the number of columns in the array

For this study we will consider a simple topology with the same number of routing segments in every direction, and a segment length $L_{seg} = 2$ (the routing segments span between two routing blocks).

For the cluster layout construction the following technological parameters are used:

- P_{litho} - lithographic interconnect pitch.
- P_{nano} - nanowire pitch

5.2.5 Evaluation Metrics

The metrics, presented in this section, are analytical models of three different aspects of the R2D NASIC architecture: area, nanowire length, and performance. They provide a quantitative basis for the evaluation of R2D NASIC, based on the technological, and architectural parameters (see Section 5.2.4).

Area The area of each cell is derived as a function of the R2D NASIC parameters considering the cell layout proposed in Figure 5.3.

$$\begin{aligned} Height_{cell} &= 10 * P_{litho} + 6 * W_x * P_{nano} + \\ &\quad max(N_{I/O} * P_{litho}, MTERMS * P_{nano}) \\ Width_{cell} &= 10 * P_{litho} + (6 * W_y + IN + OUT) \\ &\quad * P_{nano} \\ Area_{cell} &= Height_{cell} * Width_{cell} \\ Area_{array} &= Rows * Columns * Area_{cell} \end{aligned}$$

The 10 * P_{litho} component, present in both the height and the width components of the cell, account for the 5 lithographic wires present all around the routing block. A high number of I/O wires at the periphery impacts negatively the cell height if the logic block has a small number of minterms. A fine grained, directional, tuning of the routing segments can improve the surface area.

Nanowire length The design of the R2D NASIC takes into account nanowire length constraints. In the case of hard manufacturing constraints on the length of the nanowires the architectural parameters are tuned to meet the constraints. The exact length of the longest nanowire in a cluster is computed using the following formula, derived from the cell layout:

$$\begin{aligned} NW_{length} = \\ max\Big\{ \Big[(2 * Width_{cell}) - (IN + OUT) * P_{nano} \\ - 2 * P_{litho} \Big], \\ \Big[2 * Height_{cell} - max(N_{I/O} * P_{litho}, MTERMS \\ * P_{nano}) - 2 * P_{litho} \Big] \Big\} \end{aligned}$$

Performance The delay component plays a secondary role in the application output frequency due to the signal routing using dynamic logic stages, that creates a pipeline structure between the cells. In consequence:

$$\begin{aligned} Latency &= \frac{S_{critical\_path}}{2} \\ P_{output} &= \frac{S_{critical\_path} - S_{shortest\_path}}{2} + 1 \end{aligned}$$

Where, *Latency* measures the pipeline latency, the time for an input signal to propagate to the output. P_{output} is the application output period, defined as the duration between two correct output results, in terms of *heva* assertions. $S_{critical\_path}$, and $S_{shortest\_path}$ represent the number of evaluation stages on the critical path and respectively on the shortest path from inputs to outputs.

P_{output} is inversely proportional to the output frequency. Moreover, along with *Latency*, it is greatly dependent on the quality of the circuit placement and routing, giving the motivation of the development of pipeline-aware placers and routers.

Nevertheless, the delay of the evaluation stage with the highest fanin (T_{HFIN}) imposes an upper-bound on the circuit frequency. There is a total order between the delays of different evaluation stages.

$$T_{HFIN} \geq T_{LS} \geq T_{RE}$$

Where T_{LS} is the delay of the other logic stages, and T_{RE} is the delay of a routing stage. T_{HFIN} is computed using the basic precharge-evaluate model:

$$T_{HFIN} = T_{precharge} + T_{eval}$$

For circuit delay estimation, in the case of this architecture, it is not needed to estimate the delay of the critical path, but it suffices to estimate the delay of only one logic stage (T_{HFIN}) to obtain the clock frequency at which to operate the whole cluster.

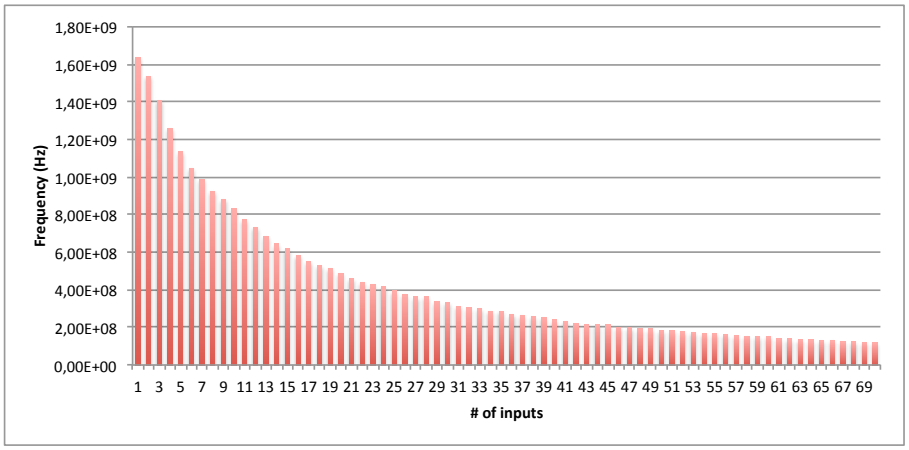


Figure 5.7: The frequency of one NAND stage as a function of the number of inputs

Figure 5.7 shows the frequency of one NAND stage with the number of inputs varied between 1 and 70. These results were obtained simulation using the xnwNFET device model. From the graph we can see that the frequency decreases as the number of inputs increases, and that a practical limit, in terms of the number of inputs to each stage, emerges as the frequency drops bellow a certain level (100MHz for example). Using these results the final frequency of the design is computed dividing the $HFIN$ frequency by the output period.

5.3 Physical-Design with MoNaDe and Madeo

We consider R2D Nasic at two levels, depending on how detailled the architecture must appear. The first level focuses on the interconnecion, with functions appearing as PLAs, see Figure 5.8. The second level gives acces to the insight of switches and functions.Besides the different abstraction levels for architecture modeling, in the context of the R2D Nasic we take advantage of the MoNaDe framework for implementing different design automation flows that enables fast design space exploration.

Placement and routing rely on traditional algorithms, that are parameterized through closures. This mechanism is flexible enough to support software plug and play within the framework.

Compared to the CMOL architecture [165], for which the authors had to perform drastic internal changes over the VPR structure, adapting MADEO to this new context, hence adding new algorithms, only brought a light development effort. The GUI is not affected by the underlying domain, nor is the internal architecture-algorithm loosely coupled scheme. From a practical point of view, adding a new algorithm only required to write an object oriented extra class that fits into the framework.

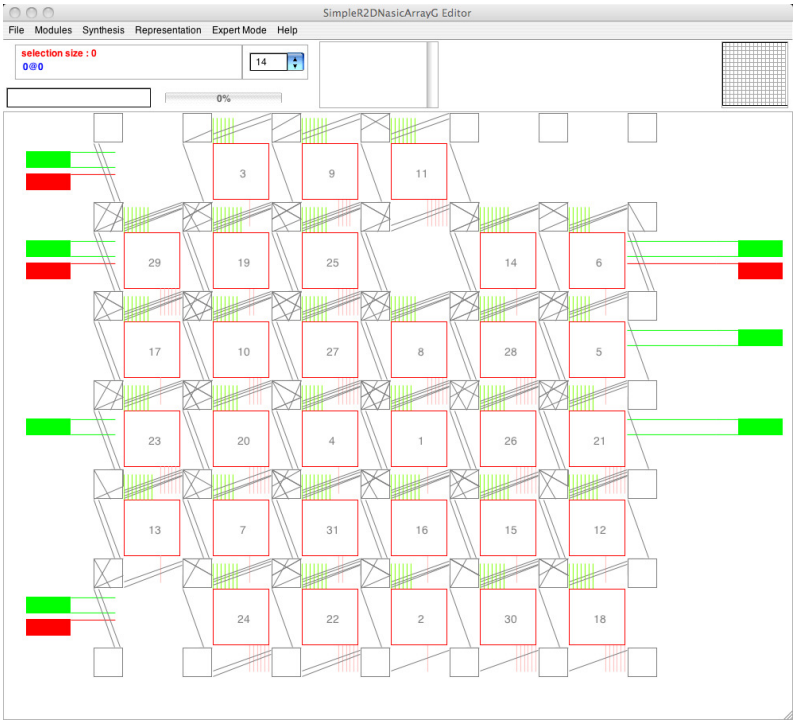


Figure 5.8: R2D NASIC tiles using MADEO visualization. Every compute node (e.g. red boxes) represents a basic NASIC tile implementing a single PLA

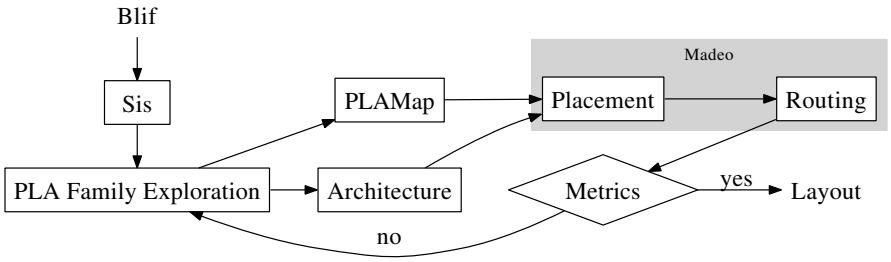


Figure 5.9: Design automation flow for R2D NASIC

5.3.1 FPGA CAD Flow for Nano-scale Architecture

The flow, presented in Figure 5.9, maps standard logic netlists (e.g. BLIF[150]) to R2D NASIC clusters.

SIS[150] performs technology independent logic optimizations and logic decomposition into small fanin nodes used for PLA family exploration and covering by PLAmap. The PLA Family Exploration step is based on the Run M Points algorithm, presented in [66], which explores different PLA families by breaking the 3D exploration space defined by (IN, MTERMS, OUT) into 3 1D spaces which are explored separately. At the end of this exploration step we obtain the PLA family, $\mathcal{F}^{PLA}_{best}$, which offers the best mapping quality ($Q^{best}_{mapping}$). For the purpose of this study, $Q^{best}_{mapping}$ is defined in terms of logic density and area as follows:

$$Q^{best}_{mapping} = \max_{1 \leq i \leq n} \left[(1 - \frac{Area^{i}_{array}}{A^{max}_{array}}) \cdot D^{i}_{logic} \right]$$
$$Area^{max}_{array} = \max_{1 \leq j \leq n} (Area^{j}_{array})$$

where n represents the number of different PLA families explored, $Area^{i}_{array}$ represent the R2D NASIC cluster area of the for the i^{th} family, $Area^{max}_{array}$ is the maximum area obtained during the exploration, and D^{i}_{logic} represent the logic density obtained by partitioning the application for the i^{th} PLA family.

Based on $\mathcal{F}^{PLA}_{best}$ an empty (no xnwFETs) R2D NASIC cluster is created, using the MADEO ADL. At the same time PLAMap[19] is used to cover the logic into PLAs defined by $\mathcal{F}^{PLA}_{best}$. These PLAs are then placed and routed on the empty cluster using Madeo framework [95], which implements VPR-like placement[9] and Pathfinder routing[110] algorithms.

Compared to the traditional FPGA design flow, in the context of R2D NASIC, the principal difference consist in replacing the FPGA specific packing tools, like T-VPACK, with the PLA-specific equivalent, PLAMap. The extra PLA exploration step is introduced due to the application specific nature of this architecture. The routability-driven flow described in this section, is used to bootstrap the design space exploration by providing a baseline evaluation of the architectural proposition, the mapping results obtained using this flow are named *baseline* results in the Section 5.4.

5.3.2 CAD Flow Tuning - Routing algorithm

Even though the CAD flow presented in the last section enabled us to quickly start the design space exploration for the R2D NASIC architecture, by reusing the already existing Madeo infrastructure. The standard P&R algorithms proposed by the framework are not suitable for the unique pipelined R2D NASIC designs. Principally the routing algorithm is not capable of taking advantage of the pipelined routing infrastructure in order to create high-speed application mappings[172].

To exploit the capacity of creating max-rate pipelined designs on the R2D NASIC architecture, the standard routing algorithm, used for the baseline evaluations, was replaced by a two steps routing tool. The first step reuses the classical A^* search under a Pathfinder-like[110] negotiation scheme. The second step refactors the routing solution by adding extra evaluation stages that balance the routing latency on the source-sink paths. Listing 5.1 shows the greedy policy used to compute the number of evaluation stages needed for balancing a particular route. This is mandatory to increase the circuit frequency. The extra delays are injected through looping wires into the switch blocks as illustrated by Figure 5.5.

Listing 5.1: Greedy delaying of a route to achieve 0 slack

```
delayingPolicy(Route route, int slack){
        int localSlack, slackRest;
        localSlack = floor(slack / route.size());
        slackRest = slack%route.size();

        for (each = 1; each < route.size(); each++){
                route.elementAt(each).addDelay(localSlack);
        }
        route.randomElement().addDelay(slackRest);
}
```

The mapping results obtained by using this modified version of the *baseline* are referred to as $max - rate$ in the Chapter 5.4, since the principal optimization target is creating max-rate pipeline designs.

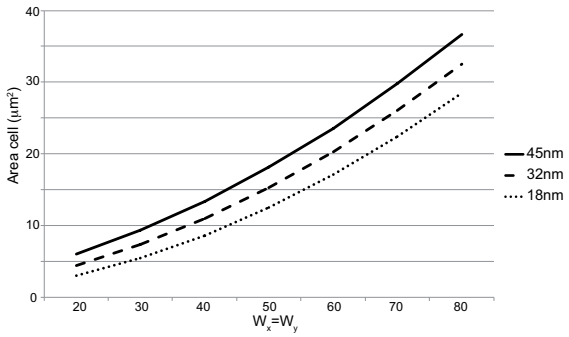


Figure 5.10: R2D NASIC Cell Area for 3 technology nodes as a function of routing segments

The principal advantage of reusing the Madeo FPGA design infrastructure resides in the capacity of rapid creation of design automation flows which offers the opportunity to incrementally design architecture specific tools, while having the capacity to evaluate the impact of changes on the fly, without having the need to build all the infrastructure from scratch. In our case, tuning the routing algorithm for creating the max-rate pipeline designs supposed adding one extra class to the system and its integration into the CAD tool flow already defined for the *baseline* case.

In the spirit of incremental development, again the $max - rate$ CAD flow proposed in this section is not to be considered as the final step in the R2D NASIC design space exploration, but just a step towards creating a high-quality CAD flow targeting this architecture. Future developments include will focus on finer tuning of all the design automation flow, first step being the creation of a pipeline-aware placement policy, further improving the routing algorithm and better application-specific PLA partitioning policy, see Chapter 5.4.2.3 for more details.

5.4 Results

In the following parts of this sections we present the routing segments impact on the metrics defined in Section 5.2.5, and the results of mapping circuits from the MCNC benchmark on the R2D NASIC architecture.

For the purpose of this study we assume that each cell at the periphery has at least two lithographic scale wires providing (reading) the inputs (the outputs). The nano pitch (P_{nano}) is set to 10 nm. The lithographic pitch (P_{litho}) is varied through 3 technology nodes (45nm, 32 nm, and 18nm) according to ITRS[74]. To simplify, in the context of this section, we consider $W_x = W_y$.

5.4.1 Routing Segments Impact

This section shows the relation between the number of routing segments and two of the metrics presented in Section 5.2.5, namely the area and the nanowire length.

Area Figure 5.10 shows that the difference in area between the 3 technology nodes is almost constant around 18% and that as the number of routing segments increases the area increases too.

Nanowire Length Figure 5.11 shows that the nanowire length increases linearly as the number of routing channels increase, and also that it stays inside the range of feasible technological ranges, as NWs of around $10 - 20\mu m$ are expected to be reliably assembled. Moreover the architectural parameters can be changed accordingly to accommodate different technological length constraints.

For example, Figure 5.11, shows that we cannot accommodate more than 65 routing segments for the 45nm node (or more than 75 in the case of the 18 nm node), if the nanowire length limit

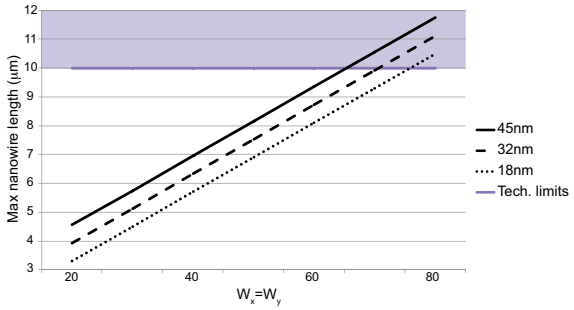


Figure 5.11: Maximum nanowire length for 3 technology nodes as a function of routing segments

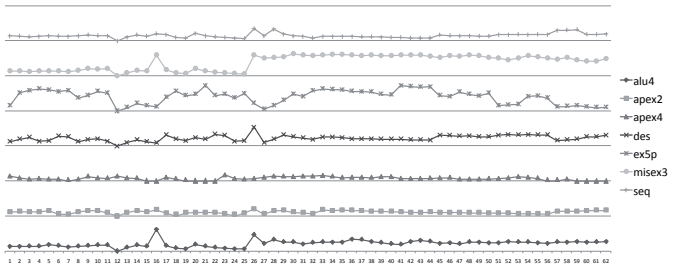


Figure 5.12: PLA exploration

is at $10\mu m$. This constraint is integrated into the CAD flow as: *a)* a PLA family size constraint; *b)* a constraint on the number of nets between PLAs, during PLA family exploration; *c)* an upper bound for W_x (W_y) exploration during routing;

5.4.2 Circuit Layout Exploration and Evaluation

To asses the benefits of the R2D NASIC cluster, we computed the layout of 7 combinatorial circuits from the MCNC-20-benchmark suite[193]. The other 3 combinational MCNC circuits (*ex*1010, *pdc*, and *spla*) are not included in this study due to execution time constraints exceeded during PLAmap execution.

The PLA family exploration decomposed the benchmark circuits into smaller PLA blocks, using the $Q^{best}_{mapping}$ metric, presented in the previous section. The results of the exploration for each benchmark are presented in Figure 5.12, and the most adapted PLA family was picked for each one of them.

Table 5.1 shows the obtained PLA family for each benchmark circuit, along with the number of logic blocks needed, the I/O requirements for the cluster, the initial number of routing segments ($W_{x\&y}$), the number of routing segments after optimizing the design for routability ($OW_{x\&y}$), and the percentage of improvement between $W_{x\&y}$ and $OW_{x\&y}$. The $\%improv.$ column shows the net advantage of running a binary-search routine to minimize the width of routing channels while guaranteeing the routability of the whole netlist.

Table 5.1: Mapped MCNC benchmark netlists

Netlist	PLA Family	# of PLAs	$N_{I/O}$	$W_{x\&y}$	$OW_{x\&y}$	$\%improv.$
alu4	(18, 36, 1)	44	3	29	9	69%
apex2	(18, 36, 2)	179	3	30	11	63%
apex4	(13, 48, 2)	43	3	25	7	72%
des	(18, 36, 2)	324	15	30	30	0%
ex5p	(8, 39, 17)	4	19	35	13	63%
misex3	(18, 29, 2)	61	2	30	9	70%
seq	(18, 36, 2)	127	4	30	15	50%

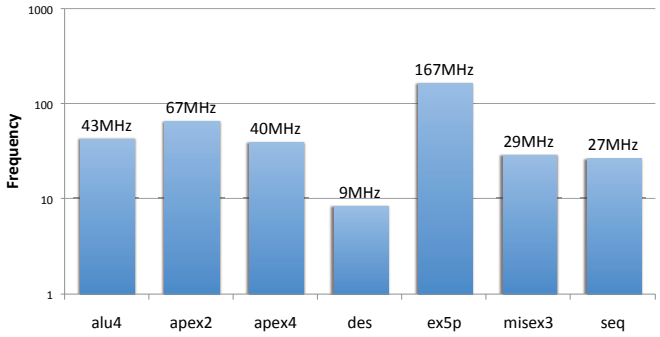


Figure 5.13: Resulting frequency for the place and routed benchmarks, assuming 1GHz logic-block frequency

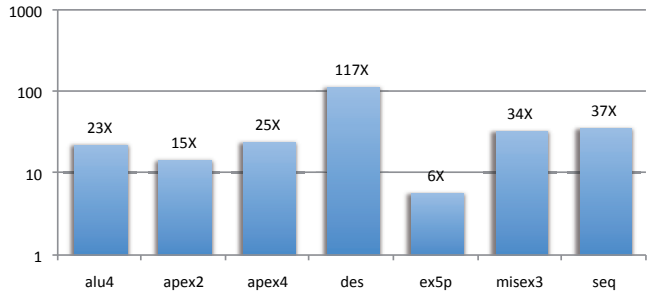


Figure 5.14: Frequency improvement over the baseline evaluation

For the purpose of this study we assume: *1)* The nano pitch (P_{nano}) is set to 10 nm. *2)* The lithographic pitch (P_{litho}) is set to 45nm according to ITRS[74].

5.4.2.1 Performance

By measuring the applicative output rate and the latency of the benchmark circuits in the case of the baseline CAD flow we observed that the quality of the P&R, as well as the size of the mapped application greatly influences these metrics. For example, for simple designs, max-rate pipelined systems could be obtained by using the baseline flow, but as soon as the size of the netlist increases the output rate plummets. To show the impact of this output rate degradation on the overall speed of the circuits, assumed 1GHz the frequency of the slowest logic block, and then we computed the circuit frequency on the benchmark circuits. The results, presented in Figure 5.13, show the negative impact of routing through dynamic-stages on the output frequency. The obtained output frequency is on average 18X lower than the slowest logic block frequency. The max-rate pipelining CAD flow solves this problem by equilibrating the pipeline stages over the netlist.

Figure 5.14 shows the improved in applicative output frequency of the max-rate pipelining flow over the baseline. For all the benchmark circuits the max-rate designs issue one output each clock period. Since the clock period is defined based on the delay of the largest fan-in evaluation stage, using the xnwFET[121] devices, the operating frequency of the cluster can get to GHz range according to the PLA logic size. We obtained a 35X average frequency improvement over baseline for the mapped benchmarks. Moreover, as it can be seen in the Figure 5.14, this improvement is proportional to the size of the application.

The high output rate of the max-rate pipelined designs impacts negatively the layout area (aspect discussed in the next section), and the pipeline latency of the design. Figure 5.15 shows the latency overhead obtained by using the max-rate pipelining flow over the baseline flow. From the figure we can see that for the *des* netlist, the largest circuit in our benchmark the latency

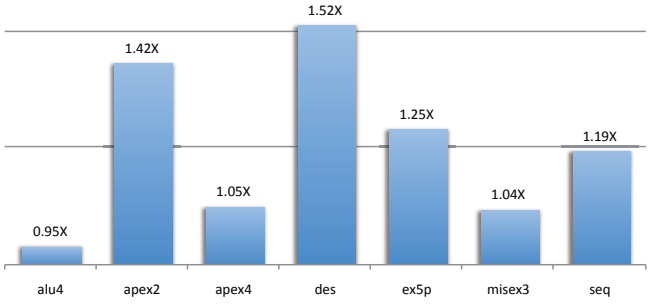


Figure 5.15: The impact of pipeline equalisation on the circuit latency

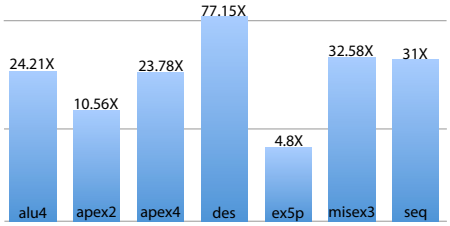


Figure 5.16: Net performance gain of the pipelined version over baseline

can increase with 52% over the baseline latency. The latency of the designs is strongly influenced by the logic depth of the partitioned netlist and by the quality of the placement on the R2D NASIC cluster. For the benchmark circuits an average 27% higher latency than the baseline can be observed. However from the performance point of view even with the high latency impact the overall performance of the mapped netlists are encouraging with a net 26X average performance improvement over the baseline performances.

Figure 5.16 shows the net performance improvement obtained by the max-rate pipelined design over the baseline evaluation. This shows the frequency improvement divided by the latency overhead. In conclusion, even with the latency overhead, the performance of the max-rate pipelined designs are significantly higher than the baseline. Moreover there is a correlation (0.98 correlation coefficient) between the netlist size and the performance gain achieved, which is normal since the output period using the baseline evalution is directly proportional to the size of the application netlist. In the case of the *apex*2 netlist the low performance improvement is explained by a particularly poor placement result.

5.4.2.2 Surface

As it was mentioned in the last section the performance gained by creating a max-rate pipelining design has a negative impact on the surface area of the mapped application. This section analyses the area overhead for the max-rate designs compared to standard cell CMOS design, to the baseline evaluation results, and to the projected results which give a lower bound on the surface area of a max-rate pipelined system.

For the comparison with the CMOS area we used Cadence tools to compute the layout of the circuits using the Oklahoma State University FreePDK 45nm standard cell library[164]. In this case the MCNC benchmark netlists were converted from blif to verilog. The verilog netlists were synthesized using Cadence RTL compiler and the results were P&R using Cadence Encounter.

Figure 5.17 compares the density advantage of the selected benchmark circuits with the 45 nm CMOS standard cell implementation. As it was expected the baseline mapping (*Baseline*) produces the densest designs at the expense of the huge performance drop presented in the last section. The projected density advantage (*Projected*) over standard cell CMOS lowers by a factor of 0.7 compared to baseline. The mapping results obtained using our max-rate pipelining

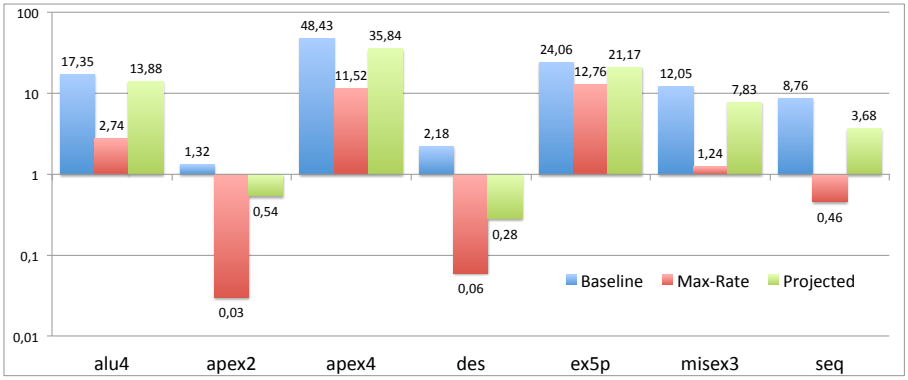


Figure 5.17: Normalized density advantage of R2D NASIC over 45nm standard cell CMOS and CAD flow impact.

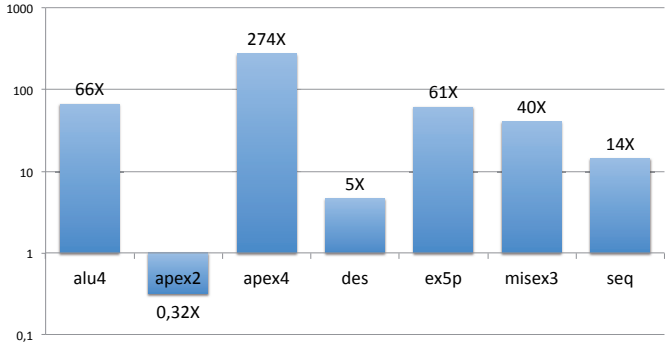


Figure 5.18: The performance per unit area advantage of the max-rate pipelined designs

flow ($Max - Rate$) are lower than the *Projected* results, but stay at around 4X average density advantage over CMOS, which represents around 35% of the lower bound. As can be seen from the figure, bigger the application netlist higher the density loss, result which can be explained by the direct correlation between the size of the netlist and the output period. As discussed earlier to reduce the output period the signals are delayed in the RB, which implies that more routing resources are needed to pipeline slower designs. But when equating the area overhead with the performance gains of the max-rate pipelined designs, results shown in Figure 5.18, the performance gains outweighs the density loss.

The max-rate pipeline designs have almost 3X better average performance per unit area compared to *baseline*. In Figure 5.18, the performance advantage for each of the benchmark circuits can be seen. Since the area density of the mapped netlist is influenced by the pipeline equilibration step of the CAD flow, which in turn is strongly influenced by the netlist partitioning, and the quality of the placement, there is much less correlation (-0.48 coeficient) between performance gain per unit area results and the size of the application netlist.

5.4.2.3 Room for improvement

Figure 5.19 shows the difference between the projected lower bound and the area obtained by our CAD flow. For the small size netlists (*alu*4, *ex*5*p* - in our case) the deviation stays under a factor of 2X, but for larger application netlists it gets to a factor of 4-5X, with an average at 3.5X. The line in the figure show the routing block resource usage overhead compared to the projected lower bound, which corresponds to uniformly distributed RB usage over the design surface.

In Figure 5.20 we can see the standard deviation of switch resources for the benchmark circuit. This metric shows that there is a large overall difference between the switch utilization and the

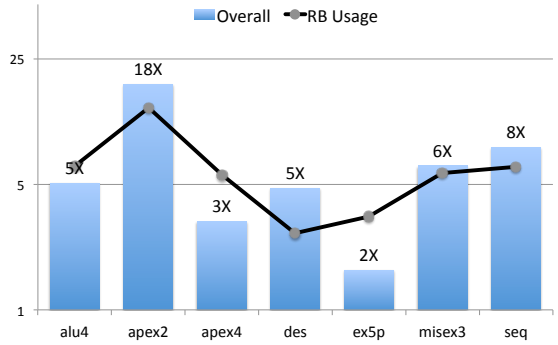


Figure 5.19: Deviation of the computed layout area from the projected bound

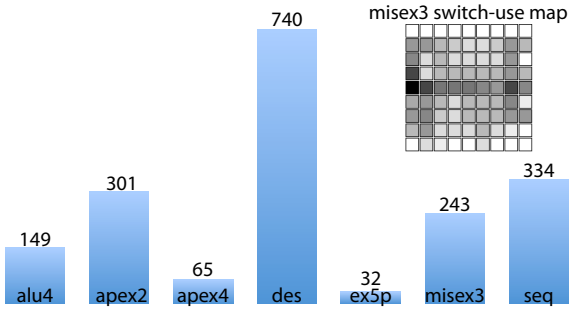


Figure 5.20: Standard deviation of RB usage for the benchmark circuits. The top-right matrix shows the real resource usage for *misex*3, darker (lighter) squares represent over-used (under-used) switches

average routing requirements. For the *misex*3 netlist benchmark we present, in Figure 5.20 top-right corner, the real usage of the switch resources. It should be noted that since the R2D NASIC architecture is a regular architectural template, the size of the darkest rectangle gives the size of the RB that is replicated around the cluster.

These results show that even though the current max-rate pipelining design flow improves considerably the performances of R2D NASIC cluster over the baseline evaluation, there is still room for improvement from the CAD tool perspective: *a)* RB-resource usage negotiation during routing, to reduce the degree of heterogeneity in terms of routing resources usage, which in turn will positively impact the area density of the designs. In which case the maximum RB usage will be closer to the mean, thus approaching the projected lower bound presented in this study. *b)* Better PLA Family exploration, to improve the partitioning of the application netlist and optimize the results in terms of logic-density, area and performance, constrained on the fan-in bound imposed by the underlying nanoscale technology. A better partitionment will positively impact the clock frequency of the R2D NASIC cluster which, with the max-rate pipeline designs, directly impacts the application output frequency. Larger, high-density PLAs will have a positive impact on the surface area but their size is limited by the fan-in bounds. *c)* The integration of a pipeline-aware placement step in the tool flow will decrease even more the latency and the area of the designs.

5.5 Pipelined Routing at Nanoscale

As already stated, the fabrication process of nanoscale devices makes practically impossible to arbitrarily place devices and wires on the surface. This constraint led the different research group to investigate fabric architectures based on highly regular building blocks such as crossbar-based PLAs. Moreover the lack of fine grain control during fabrication limits the number of

different devices (NFET, PFET, diodes) that can be successfully integrated on the fabric. While complementary FETs have been demonstrated [28, 57, 126] there are large differences in the intrinsic characteristics of these devices, notably in terms of transport properties. This lack of symmetry between N-FET and P-FET devices would certainly complicate the timing closure thereby making it harder to manufacture reliable circuits. Therefore one of the objectives of nanoscale fabric architecture is to drastically reduce the number of different devices used, thus minimizing the manufacturing requirements.

Dynamic logic evaluation provides the opportunity to implement logic functions using only one type of FETs, it also has the advantage of providing implicit latching between different logic stages, moreover it minimizes the leakage power consumption by eliminating the direct path between power supply and ground. Hence dynamic logic evaluation is presented as a viable candidate for logic evaluation in the context of different nanoscale architectures. Besides NASIC, other architectures making use of dynamic logic are NanoPLA[32], Self-Timed NanoPLA[195], and the matrix architecture using DG-CNT FET devices (MCNT) proposed in[131]. All three designs aim at creating high-density reconfigurable architectures. However while the first two approaches are build around NW crossbar structures, the last example uses custom assembled double-gate carbon nanotube FETs (DG-CNT FET) to create a reconfigurable cell[131]. While the NanoPLA and the MCNT both can use both static and dynamic logic for evaluation, the Self-Timed NanoPLA design implements a dynamic logic evaluation style only.

While dynamic logic has its advantages it comes with its challenges, some of which are unexpected and pose important constraints on the overall circuit design. Besides the control signal variability, which can lead to hazards during evaluations, our experience with the R2D NASIC architecture led to the conclusion that dynamic logic can lead to very low performance circuits if their are not designed carefully (see Figure 5.13). This conclusions refers principally to the need of storage for the intermediate results to synchronize the arrival of inputs to a logic block, and thus to aggressively pipeline the design, an approach similar to C-slowing.

Due to the regularity of these nanoscale architecture, they tightly related to reconfigurable architectures, thus we will present some reconfigurable architectures that provide the support for aggressive pipelining, approaches that gives us some insight of the architectural and tooling costs for addressing this problem more generally than the approach used in the case of R2D NASIC.

Pipelined FPGAs

In the context of traditional FPGA architectures the designers try to improve the lower clock frequencies of these architectures by heavily pipelining, retiming and C-slowing. Unfortunately these techniques produce a large number of registers, resources that are not available on the architectures, and that are hard to be optimally used by automation tools. Multiple research groups have tried to address this problem with specialized FPGAs. One solution being to incorporate storage resources in the interconnect with the purpose of minimizing the interconnect delay which dominates the mapped circuits. hierarchical synchronous reconfigurable array (HSRA)[178], synchronous and flexible reconfigurable array (SFRA)[186], and reconfigurable pipelined datapath(RaPiD)[45] are some examples of this approach.

HSRA and SFRA are two designs that guarantee the execution of an application at a predefined fixed frequency. HSRA has a hierarchical routing structure while SFRA uses an island-style routing. The applications mapped on these two architectures are heavily C-slowed to reduce the clock-cycle increasing the register count. To ease the problem of locating the needed registers, and to reduce the impact on the tool-flow, these two architectures provides large retiming register-banks at the input of the logic blocks and a number of registered switching points. This register reach structure alleviates the need for optimized register allocation during placement and routing. However this flexibility is payed, for both these architectures, by a large area overhead.

Another approach for aggressive pipelining using C-slowing was proposed in [187] for Xilinx Virtex FPGAs. This approach tries to benefit from unused logic blocks of the target FPGA by using them to place the registers needed for C-slowing. The approach iteratively searches for unused logic blocks in the bounding box of the net, this is advantageous if the placement result is

sparse. In the case of dense placement results this methodology might allocate registers that are too far from the net, hence possible canceling the benefits of C-slowing.

The RaPiD architecture represent yet another approach for tackling the performance of reconfigurable ICs. RaPiD is a coarse-grain, linear array with word-size interconnect. For pipelining this architecture provides registered switch-points (bus connectors) which can delay a signal up to 3 retiming latencies. By constraining the placement of retiming registers only in the routing architecture this approach motivated the research on pipelining routers.

To make best use of the registers provided in the routing architecture for creating pipelined designs a solution to the N-Delay routing problem is required. But in [154] this problem is shown to be NP-complete by reduction to the traveling salesman with triangle inequality problem. However a number of different heuristics have been proposed in the literature, like PipeRoute[154], QuickRoute[101], and Armada[46].

5.6 Summary

In this chapter, a general purpose Nasic-based architecture was proposed. Relying on a regular array of parametric cells, interconnected by a flexible routing architecture, it enables arbitrary circuit placement and routing and paves the way towards high-throughput nanoscale circuits through its unique pipelined routing architecture. One important trade-off offered by this architecture is the possibility to increase the output rate of the circuit at the expense of higher area and latency, by implementing pipelined designs approaching the max-rate pipeline bound. Trade-off which is impracticable using todays technology for generic applications due to the high area overhead, but which becomes realistic in the context of nanoscale technologies. The exploration of this trade-off is the principal axis of on-going works in the context of R2D Nasic.

Along with it, a design automation flow was presented, based extensively on standard tools used currently in the reconfigurable architecture field. In the context where the regularity of assembly is one of the principal constraints imposed by the nanoscale technologies, this showed a way of transposing the experience from reconfigurable research to application specific nanoscale circuits. Exploiting the extensibility of the MoNaDe toolkit (introduced in Chapter 4) a sound architecture-specific solution was created, that enables incremental development and assures development convergence based on iterative quantitative evaluations. Moreover, reusing parts of the already proved Madeo infrastructure[90] reduced considerably the software development effort.

The last section of this chapter reviewed the principal characteristics that enabled the creation of the max-rate pipeline designs, showed the potential of the approach in the context of other nanoscale architecture and discussed the use of pipelined routing in the context of traditional CMOS architectures while showing some important results in terms of automation complexity.

6 Conclusion & Perspectives

This thesis advocates the importance of a generic physical-design toolkit for managing in an unified fashion the automated physical-design of application on a wide range of nanoscale architectures. The main concern being the requirements analysis and the design of such a toolkit in the context of numerous diverging technologies that fight for adoption as replacement for the traditional CMOS technology. The solution proposed relies on the model-driven software engineering methodology, which pushes for the use of high-level models reifying domain-specific concepts. This thesis shows that the adoption of such a development methodology enables the factorization of common domain-modeling concepts, which then offers a high-degree of reuse and flexibility. Besides structural domain-modeling the whole physical design toolkit presented relies heavily on the use of models and model transformations for algorithm and heuristic design as well as for tool-flow creation. This approach decouples the structural models used to instantiate domain concepts from the tool-specific data-structures enabling their independent evolution and reuse. Moreover, a new regular 2D nanoscale architecture is designed along with its physical-design tool-flow. This tool-flow was created relying on the presented methodology. The design process of this architecture, named R2D NASIC, was directly driven by rapid evaluations through the created tool-flow, which enabled the incremental parallel evolution of the architecture and of its design tools. This design experience has shown the success of our approach by bridging the gap between technology-specific architectural concerns and circuit physical-design automation, and by assuring design convergence based on iterative quantitative evaluations.

6.1 Summary of Contributions

Today, many emergent technologies are proposed as possible alternatives for replacing the CMOS technology, which approaches its limits. During the last years tremendous improvements have been made on the technological side for the fabrication and the assembly of novel electronic devices, on the theoretical side to better understand their behavior and characteristics and on the practical side for the creation of competitive computing architectures, see Chapter 2 for more details. Crossbar-based structures using Silicon nanowire technology is one of the most promising approaches for addressing the current technology gap. Efforts like CMOL, NanoPLA, and NASIC leverage the NW crossbar structure to create novel computational bricks assembled into high-density reconfigurable and application-specific architectures.

The research presented in this thesis was particularly focused on the physical synthesis part of the circuit design automation tool flow, mainly due to the high-impact these novel technologies has

on these tools. Chapter 2 presented the state of the art electronic CAD with a focus on the specific needs of the crossbar-based nanoscale architectures. The design space exploration problem in the context of these architectures was presented with an accent on the adequacy architecture/tool-flow and the methodology used to bootstrap the exploration, critical point for the study of prospective architectures.

The results presented in this thesis rely most notably on the exploration and the analysis of the interdependence between the crossbar-based nanoarchitectures and the physical design tools, with the purpose of providing answers that would help reduce the design and exploitation costs for new technologies. The principal contributions of this work are discussed below.

The architectures studied are designed based on regular SiNW crossbars, which have the advantage of providing a simple, high-density structure enabling arbitrary logic designs. The principal constraint imposed by the bottom-up fabrication process of these crossbars is the regularity of assembly which imposes the creation of highly regular fabrics tuned according to the needs of the applications. The majority of crossbar-based nanoscale fabrics, with the exception of NASIC, have structures conceptually similar to today's reconfigurable PLA and/or FPGA architectures. This regular organization complemented by the reconfigurable design enables arbitrary placement and routing, feature much needed for the implementation of arbitrary circuits on these computing supports. In the case of NASIC the authors traded-off the reconfigurability to ease the fabrication process and branded their fabric as an "application specific integrated circuit"(ASIC) at nanoscale. However, the nanoscale tile placement and signal routing between heterogeneous 2D rectangular tiles proved to be very challenging. The R2D NASIC design solves this problem by creating a 2D array of identical NASIC tiles, interconnected using a flexible routing architecture build also around the concept of NASIC tile. The R2D NASIC architecture template is still an ASIC with the particularity that the fabric surface is completely regular at the nanowire level, but the active FET devices are placed differently in different tiles (logic, routing) according to the application logic. Thus, through its regularity, this architecture enables arbitrary logic placement and routing. This architecture is compatible with the NASIC technological framework and fabrication process, and can easily be adapted according to the technological and application constraints. Moreover it offers the possibility of implementing max-rate pipeline designs with an average 35X higher-frequency than non-pipelined versions, paving the way to high-performance nanoscale circuits.

Early quantitative architectural evaluations drive the design-space exploration, enabling the unbiased analysis of results according to different architecture-specific metrics. As already stated, most of the nanoscale crossbar-based architectures are structurally similar to today's reconfigurable architectures. Moreover, their respective tool-flows rely heavily on generic reconfigurable tool-flows as VPR or Madeo. However, the generic physical-design algorithms implemented in these frameworks are replaced with architecture-specific solutions. In the case of R2D NASIC, instead of replacing these algorithms with specific solutions we re-used them for obtaining a first quantitative evaluation of our architectural proposition. Relying on this evaluation, we could objectively identify and improve different architectural and automation aspects. One example of such improvement is the design of an architecture specific routing algorithm as a direct consequence of the frequency results obtained based on this first architecture agnostic exploration. This proves, the interest for algorithm reuse in the context of novel architecture design, especially in the context of crossbar-based architectures, which could directly benefit from the maturity of FPGA SDKs for creating early architectural evaluations. The possibility to perform early evaluation enhances the productivity by pruning the search space of eventual design errors and/or inadequacies. In consequence the design process converges faster directed by quantitative measures tuned through metrics according to the imposed constraints.

With the study of the R2D NASIC architecture, we have seen that if the crossbar-based nanoscale fabric implements a dynamic logic evaluation strategy the execution speed of the design can be very slow principally due to the inequality between the routing paths associated to a logic-block fan-in. If in the context of traditional CMOS design approaches like C-slowing and pipelined routing, which aim at increasing the output frequency by aggressively pipelining the design, are very expensive due to the high-area overhead of the storage elements (e.g. registers, flip-flops, latches). In the context of nano-crossbar fabrics, the high-density fabric design and latching on

the wire[114] opens the opportunity to cheaply implement storage for the intermediate result, hence the ability to synchronize the arrival of inputs at each logic block. The unique design of the R2D NASIC routing architecture, enables the creation of fully balanced pipelines - approach dubbed max-rate pipeline - which besides increasing the output frequency (at the expense of small latency increase) decouples the evaluation stages, thus simplifying the circuit delay estimation. However, this approach puts pressure on the routing routine which needs to solve an N-Delay routing problem for each net. To address this problem we have designed a simple heuristic relying on the Pathfinder router, which functionally balances the routing paths according to the N-Delay routing constraints. Using this heuristic the performances obtained were improved up to 77X, with 3X better performances per unit area compared to the non-optimized designs.

For the architectures presented in Chapter 2.5.5 the nanoscale devices play different roles, in NASIC they are used for logic and routing, in CMOL for OR logic and routing, in NanoPLA for logic, and in FPNI just for routing. Each approach has its advantages and disadvantages but it is almost impossible to compare the result presented for each one. The main reasons for this are: *1*) architectural differences, *2*) different physical parameters used for evaluation, *3*) different evaluation strategies, *4*) different hypotheses during evaluation for the aspects not being investigated. However, except the architectural differences all the other reasons point clearly to the lack of a common vocabulary along with a set of integrated tools targeted for these specific architectures, which imposes the development of specific tools for analyzing each architecture alone using different metrics and under different sets of assumptions. To address this issue we have introduced a common vocabulary for nanoscale architecture modeling at different abstraction levels. This vocabulary is based on an abstract meta-model relying on a hierarchical port-graph structure. This meta-model is used for architecture and application modeling, for the specification of different simulation models and can be extended to address defect and fault modeling and injection, as discussed in Chapter 4.2. This approach proved to be highly efficient for modeling and evaluating FPGA architectures, see VPR[9] and Madeo[90], and we argue that applying it to nanoscale architectures will certainly increase our understanding of these fabrics, and will bridge the gap between the technological advancements and their mass market exploitation.

The physical-design toolkit presented in this thesis aim principally at decoupling and orthogonally composing the core aspects of such an automation toolkit. To address the algorithm design problem for the MoNaDe toolkit we propose a technique that we call "Transformation metaphor". This technique appears as a conceptual framework; the algorithm designer looks at the algorithm implementation as it was a model-transformation problem. This approach mainly reifies the tight implicit dependency between algorithms and structural domain models, through explicit transformations, isolating their respective concerns, thus increasing the flexibility and the expressivity of the toolkit. Moreover, the MoNaDe toolkit employs a model-driven engineering methodology for tool-flow modeling. This approach reifies the tool-flow and its elements and creates the ideal conditions for the independent evolution of architecture, algorithms and tool-flow. This flow improves the algorithm reutilization, eases the agile development of the design-flow, and creates the necessary conditions for incremental design space exploration. Moreover the use of the Model-Driven Development in the context of the physical design opens the toolbox offering an unprecedented flexibility and support for high-performance physical-design in the context of dynamic technological targets.

6.2 Perspectives

This work opens various directions of research in circuit physical design automation and nanoscale architecture design. Some of them are discussed below.

MoNaDe Toolkit - Towards Modular, Re-targetable, Model-based Physical Design Toolkit

The MoNaDe toolkit opens the physical-design toolbox towards creating an open environment for research targeting physical-design automation on multiple technological targets. Through the adoption of a model-driven engineering methodology this toolkit prototype sketches the roadmap to providing a common way of modeling all structural aspects of the physical-design problem at multiple abstraction levels, and provides the building-blocks needed for interoperability between development tools used for the specification, the design, and the verification of integrated circuits. However, this prototype doesn't provide a complete physical-design environment needed for industry-level applications. To reach such a goal further research is needed especially on formalizing the models, and meta-models presented in order to offer a stable and expressive vocabulary for specifying all aspects of the physical-design process.

The "transformation metaphor", used for algorithm design and specification, provides the infrastructure for creating highly modular optimization routines. Moreover, it doesn't constraint the algorithm designer to use any particular programming language or formalism, as long as the resulting module provides clean and well-defined interfaces. This opens the way for future integration of hardware accelerated routines into the tool-flow, and along with the tool-flow reification can provide the infrastructure for side-by-side execution of hardware and software tasks to speed-up the physical-design process.

The reification of the tool-flow and its specification using an object-oriented model improves on the state of the art, transforming the tool-flow design process from an "ad-hoc" script-based textual description to an executable specification based on a graph model. Relying on such an abstraction, the tool-flow and the physical-design algorithms are decoupled from the execution environment. Relying on the ideas presented in this thesis future research can focus on the virtualization of the tool-flow, which by creating a physical-design execution engine can directly execute the tool-flow while performing just-in-time optimizations, like the parallel executions of different automation task on different processors. This aspect is very appealing in the current context, where the adoption of cloud computing provides the needed infrastructure for deploying massively parallel applications.

High-Performance Max-Rate Pipeline Circuit Design at Nanoscale

The study of the R2D NASIC architecture template presented in this thesis offers real perspectives for the creation of high-speed nanoscale circuits. However, it poses a number of unique challenges opening interesting research perspectives at both the technological level, and the physical-design level. As already stated the presence of defects at nanoscale poses serious challenges and require the integration of defect-tolerance techniques at all level of design. While the self-healing approach proposed for the NASIC fabric[115] is directly transposable for the R2D NASIC design there is a need to evaluate the effectiveness of these techniques especially for the implementation of the routing infrastructure. The dynamic logic evaluation which enables the creation of the aggressive pipeline scheme presented in this thesis might have a negative impact on the reliability of the design, most notably due to process variations and control distribution issues. While the process variations and parameter variability were studied in the context of simple NASIC tiles in[119], certainly there is a need to better understand the impact of these variations in the case of large scale integration of tiles. As for the clock(control) distribution, while in the context of this work we assumed an ideal clock tree network, there is a need for detailed analysis of this issue especially since the placement and routing control signals is highly constrained for crossbar-based nanoscale designs. From the physical-design automation perspective, future work will focus on integrating the pipeline constraints also in the PLA family exploration and the placement routines to further improve the density advantage and the performances of this architecture. Moreover, as it was discussed in Chapter 5.4.2.3, the routing algorithm can be further improved by a resource negotiation step, to reduce the degree of heterogeneity in terms of routing resource usage, which in turn will positively impact the area density of the designs.

Dynamic logic evaluation provides the opportunity to implement logic functions using only one type of FETs, it also has the advantage of providing implicit latching between different logic stages, moreover it minimizes the leakage power consumption by eliminating the direct path between power supply and ground. Hence dynamic logic evaluation is presented as a viable candidate for logic evaluation in the context of different nanoscale architectures. Besides NASIC, other architectures making use of dynamic logic are NanoPLA[32], Self-Timed NanoPLA[195], and the matrix architecture using DG-CNT FET devices (MCNT) proposed in[131]. Thus, the performance degradations observed in the baseline evaluation of the R2D NASIC, which come mainly from the inequality between different dynamic routing path leading to a logic block, will certainly affect other architectural propositions that implement their routing infrastructure relying on dynamic logic. Hence, the solutions presented in this thesis will extend to other nanoscale fabric proposition. In consequence, the study of max-rate pipeline design, and pipelined routing in nanoscale architectures is an important axis for future investigations, especially since it imposes huge constraints on the signal routing routines rendering it NP-complete.

Bibliography

[1] D. A. Allwood, G. Xiong, C. C. Faulkner, D. Atkinson, D. Petit, and R. P. Cowburn. Magnetic domain-wall logic. *Science*, 309(5741):1688–1692, 2005.

[2] M. Altun, M. Riedel, and C. Neuhauser. Nanoscale digital computation through percolation. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 615 –616, july 2009.

[3] J. Appenzeller, J. Knoch, M. Bjork, H. Riel, H. Schmid, and W. Riess. Toward nanowire electronics. *Electron Devices, IEEE Transactions on*, 55(11):2827 –2845, nov. 2008.

[4] C. Atkinson, M. Gutheil, and B. Kennel. A flexible infrastructure for multilevel language engineering. *IEEE Trans. Software Eng.*, 35(6):742–755, 2009.

[5] C. Atkinson and T. Kühne. Concepts for comparing modeling tool architectures. In L. C. Briand and C. Williams, editors, *MoDELS*, volume 3713 of *Lecture Notes in Computer Science*, pages 398–413. Springer, 2005.

[6] D. Bacon and W. van Dam. Recent progress in quantum algorithms. *Commun. ACM*, 53:84–93, February 2010.

[7] R. I. Bahar, J. Mundy, and J. Chen. A probabilistic-based design methodology for nanoscale computation. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, ICCAD '03, pages 480–, Washington, DC, USA, 2003. IEEE Computer Society.

[8] B. Bary. Smalltalk as a development environment for integrated manufacturing systems. In *International Conference on Object-Oriented Manufacturing Systems*, 1992.

[9] V. Betz, J. Rose, and A. Marquardt, editors. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[10] A. Black, S. Ducasse, O. Nierstrasz, D. Pollet, D. Cassou, and M. Denker. *Pharo by Example*. Square Bracket Associates, 2009.

[11] G. Bracha, P. Ahe, V. Bykov, Y. Kashai, and E. Miranda. The newspeak programming platform. Technical report, Cadence Design Systems, 2008.

[12] J. Brant and D. Roberts. Smacc, a smalltalk compiler-compiler, 2011.

[13] U. Brenner, M. Struzyna, and J. Vygen. Bonnplace: Placement of leading-edge chips by advanced combinatorial algorithms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(9):1607 –1620, sept. 2008.

[14] A. D. Brown and M. Zwolinski. Lee router modified for global routing. *Comput. Aided Des.*, 22:296–300, June 1990.

[15] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie. mpl6: enhanced multilevel mixed-size placement. In *Proceedings of the 2006 international symposium on Physical design*, ISPD '06, pages 212–214, New York, NY, USA, 2006. ACM.

[16] Y.-L. Chang and S. S. Yi. Controlled formation of individually addressable si nanowire arrays for device integration. In Z. M. Wang, editor, *One-Dimensional Nanostructures*, volume 3 of *Lecture Notes in Nanoscale Science and Technology*, pages 79–96. Springer New York, 2008. 10.1007/978-0-387-74132-1_4.

[17] R. Chau, M. Doczy, B. Doyle, S. Datta, G. Dewey, J. Kavalieros, B. Jin, M. Metz, A. Majumdar, and M. Radosavljevic. Advanced cmos transistors in the nanotechnology era for high-performance, low-power logic applications. In *Solid-State and Integrated Circuits Technology, 2004. Proceedings. 7th International Conference on*, volume 1, pages 26 – 30 vol.1, oct. 2004.

[18] A. Chen, A. Jacob, C. Sung, K. Wang, A. Khitun, and W. Porod. Collective-effect state variables for post-cmos logic applications. In *VLSI Technology, 2009 Symposium on*, pages 132 –133, june 2009.

[19] D. Chen, J. Cong, M. Ercegovac, and Z. Huang. Performance-driven mapping for cpld architectures. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(10):1424 – 1431, oct. 2003.

[20] D. Chen, J. Cong, and P. Pan. Fpga design automation: A survey. *Found. Trends Electron. Des. Autom.*, 1:139–169, January 2006.

[21] X. Chen, M. Hirtz, H. Fuchs, and L. Chi. Fabrication of gradient mesostructures by Langmuir-Blodgett rotating transfer. *Langmuir : the ACS journal of surfaces and colloids*, 23(5):2280–2283, Feb. 2007.

[22] C. P. Collier, E. W. Wong, M. Belohradsk, F. M. Raymo, J. F. Stoddart, P. J. Kuekes, R. S. Williams, and J. R. Heath. Electronically configurable molecular-based logic gates. *Science*, 285(5426):391–394, 1999.

[23] J. Cong and Y. Ding. Flowmap: an optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 13(1):1 –12, jan 1994.

[24] J. Cong and S. K. Lim. Performance driven multiway partitioning. In *Proceedings of the 2000 Asia and South Pacific Design Automation Conference*, ASP-DAC '00, pages 441–446, New York, NY, USA, 2000. ACM.

[25] L. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(10):1367 –1372, oct. 2004.

[26] L. W. Cotten. Maximum-rate pipeline systems. In *AFIPS '69 (Spring): Proceedings of the May 14-16, 1969, spring joint computer conference*, pages 581–586, New York, NY, USA, 1969. ACM.

[27] S. Cranefield and M. Purvis. UML as an Ontology Modelling Language. In *In Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99*, pages 46–53, 1999.

[28] Y. Cui, X. Duan, J. Hu, and C. M. Lieber. Doping and electrical transport in silicon nanowires. *The Journal of Physical Chemistry B*, 104(22):5213–5216, 2000.

[29] T. Dang, L. Anghel, and R. Leveugle. Cntfet basics and simulation. In *Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006. International Conference on*, pages 28 –33, sept. 2006.

[30] F. Darema, S. Kirkpatrick, and V. A. Norton. Parallel algorithms for chip placement by simulated annealing. *IBM Journal of Research and Development*, 31(3):391–402, 1987.

[31] A. DeHon. Design of programmable interconnect for sublithographic programmable logic arrays. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, FPGA '05, pages 127–137, New York, NY, USA, 2005. ACM.

[32] A. DeHon. Design of programmable interconnect for sublithographic programmable logic arrays. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, FPGA '05, pages 127–137, New York, NY, USA, 2005. ACM.

[33] A. DeHon, P. Lincoln, and J. Savage. Stochastic assembly of sublithographic nanoscale interfaces. *Nanotechnology, IEEE Transactions on*, 2(3):165 – 174, sept. 2003.

[34] A. DeHon and H. Naeimi. Seven strategies for tolerating highly defective fabrication. *Design and Test of Computers, IEEE*, 22(4):306–315, July-Aug. 2005.

[35] A. DeHon and M. J. Wilson. Nanowire-based Sublithographic Programmable Logic Arrays. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 123–132, New York, NY, USA, 2004. ACM.

[36] L. N. Denis Teixeira FRANCO, Jean-François NAVINER. Yield and reliability issues in nanoelectronics technologies. *Annals of Telecommunications*, 61(11-12):1247–1282, December 2006.

[37] D. Densmore, R. Passerone, and A. Sangiovanni-Vincentelli. A platform-based taxonomy for esl design. *IEEE Des. Test*, 23:359–374, September 2006.

[38] R. Devadoss, K. Paul, and M. Balakrishnan. A tiled programmable fabric using qca. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 9 –16, dec. 2010.

[39] C. Dezan, L. Lagadec, and B. Pottier. Object oriented approach for modeling digital circuits. In *Microelectronic Systems Education, 1999. MSE '99. IEEE International Conference on*, pages 51 –52, 1999.

[40] C. Dezan, C. Teodorov, L. Lagadec, M. Leuchtenburg, T. Wang, P. Narayanan, and A. Moritz. Towards a framework for designing applications onto hybrid nano/cmos fabrics. *Microelectron. J.*, 40(4-5):656–664, 2009.

[41] S. Ducasse and T. Gîrba. Using Smalltalk as a reflective executable meta-language. In *International Conference on Model Driven Engineering Languages and Systems (Models/UML 2006)*, volume 4199 of *LNCS*, pages 604–618, Berlin, Germany, 2006. Springer-Verlag.

[42] S. Ducasse, T. Girba, A. Kuhn, and L. Renggli. Meta-environment and executable meta-language using smalltalk: an experience report. *Software and Systems Modeling*, 8:5–19, 2009.

[43] S. Ducasse, O. Nierstrasz, N. Schärli, R. Wuyts, and A. P. Black. Traits: A mechanism for fine-grained reuse. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 28(2):331–388, Mar. 2006.

[44] R. Dutton and A. Strojwas. Perspectives on technology and technology-driven cad. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(12):1544 –1560, dec 2000.

[45] C. Ebeling, D. C. Cronquist, and P. Franklin. Rapid - reconfigurable pipelined datapath. In *Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, pages 126–135, London, UK, 1996. Springer-Verlag.

[46] K. Eguro and S. Hauck. Armada: timing-driven pipeline-aware routing for fpgas. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, FPGA '06, pages 169–178, New York, NY, USA, 2006. ACM.

[47] W. C. Elmore. The transient response of damped linear networks with particular regard to wideband amplifiers. *Journal of Applied Physics*, 19(1):55–63, 1948.

[48] O. Englander, D. Christensen, J. Kim, L. Lin, and S. J. S. Morris. Electric-field assisted growth and self-assembly of intrinsic silicon nanowires. *Nano Letters*, 5(4):705–708, 2005.

[49] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, pages 175–181, Piscataway, NJ, USA, 1982. IEEE Press.

[50] A. Gamatié, É. Rutten, H. Yu, P. Boulet, and J.-L. Dekeyser. Model-Driven Engineering and Formal Validation of High-Performance Embedded Systems. *Scalable Computing: Practice and Experience (SCPE)*, 10, 2009.

[51] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[52] E. Gautrin and L. Perraudeau. Madmacs: an environment for the layout of regular arrays. In *Proceedings of the IFIP WG10.2/WG10.5 Workshops on Synthesis for Control Dominated Circuits*, pages 345–358, Amsterdam, The Netherlands, The Netherlands, 1993. North-Holland Publishing Co.

[53] S. H. Gerez. *Algorithms for VLSI Design Automation*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1999.

[54] G. Gielen and R. Rutenbar. Computer-aided design of analog and mixed-signal integrated circuits. *Proceedings of the IEEE*, 88(12):1825 –1854, dec 2000.

[55] P. Graham and M. Gokhale. Nanocomputing in the presence of defects and faults: A survey. In S. Shukla and R. Bahar, editors, *Nano, Quantum and Molecular Computing*, pages 39–72. Springer US, 2004. 10.1007/1-4020-8068-9_2.

[56] R. Grassi, A. Gnudi, E. Gnani, S. Reggiani, and G. Baccarani. Graphene nanoribbons fets for high-performance logic applications: Perspectives and challenges. In *Solid-State and Integrated-Circuit Technology, 2008. ICSICT 2008. 9th International Conference on*, pages 365 –368, oct. 2008.

[57] A. B. Greytak, L. J. Lauhon, M. S. Gudiksen, and C. M. Lieber. Growth and transport properties of complementary germanium nanowire field-effect transistors. *Applied Physics Letters*, 84(21):4176 –4178, may 2004.

[58] I. A. Grout. *Integrated Circuit Test Engineering: Modern Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[59] I. A. Grout. Test economics. In *Integrated Circuit Test Engineering*, pages 267–274. Springer London, 2006. 10.1007/1-84628-173-3_12.

[60] N. Haron and S. Hamdioui. Emerging crossbar-based hybrid nanoarchitectures for future computing systems. In *Signals, Circuits and Systems, 2008. SCS 2008. 2nd International Conference on*, pages 1 –6, nov. 2008.

[61] N. Haron and S. Hamdioui. Why is cmos scaling coming to an end? In *Design and Test Workshop, 2008. IDT 2008. 3rd International*, pages 98 –103, dec. 2008.

[62] R. He, D. Gao, R. Fan, A. Hochbaum, C. Carraro, R. Maboudian, and P. Yang. Si nanowire bridges in microtrenches: Integration of growth into device fabrication. *Advanced Materials*, 17(17):2098–2102, 2005.

[63] M. Healy, M. Vittes, M. Ekpanyapong, C. S. Ballapuram, S. K. Lim, H.-H. S. Lee, and G. H. Loh. Multiobjective microarchitectural floorplanning for 2-d and 3-d ics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(1):38 –52, jan. 2007.

[64] J. R. Heath and M. A. Ratner. Molecular electronics. *Physics Today*, 56(5):43–49, 2003.

[65] T. Hogg and G. Snider. Defect-tolerant logic with nanoscale crossbar circuits. *J. Electron. Test.*, 23:117–129, June 2007.

[66] M. Holland and S. Hauck. Automatic creation of domain-specific reconfigurable cplds for soc. *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, 0:289–290, 2005.

[67] X. Hu, M. Crocker, M. Niemier, M. Yan, and G. Bernstein. Plas in quantum-dot cellular automata. *in proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, 2006.

[68] C. Husband, S. Husband, J. Daniels, and J. M. Tour. Logic and memory with nanocell circuits. *IEEE Transactions on Electron Devices*, 50:1865–1975, 2003.

[69] B. Hutchings, P. Bellows, J. Hawkins, S. Hemmert, B. Nelson, and M. Rytting. A cad suite for high-performance fpga design. In *Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, FCCM '99, pages 12–, Washington, DC, USA, 1999. IEEE Computer Society.

[70] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, Netherlands, 1992.

[71] S. Inaba, K. Okano, T. Izumida, A. Kaneko, H. Kawasaki, A. Yagishita, T. Kanemura, T. Ishida, N. Aoki, K. Ishimaru, K. Suguro, K. Eguchi, Y. Tsunashima, Y. Toyoshima, and H. Ishiuchi. Finfet: the prospective multi-gate device for future soc applications. In *Solid-State Device Research Conference, 2006. ESSDERC 2006. Proceeding of the 36th European*, pages 49 –52, sept. 2006.

[72] International Organization for Standardization. Industrial automation systems and integration—product data representation—and exchange—part 1: Overview and fundamental principles, 1994.

[73] International Technology Roadmap for Semiconductors. [online]. *http://public.itrs.net/*, 2009.

[74] International Technology Roadmap for Semiconductors. [online]. *http://public.itrs.net/*, 2010.

[75] H. Iwai. Roadmap for 22nm and beyond (invited paper). *Microelectronic Engineering*, 86(7-9):1520 – 1528, 2009. INFOS 2009.

[76] A. Javey, Nam, R. S. Friedman, H. Yan, and C. M. Lieber. Layer-by-layer assembly of nanowires for three-dimensional, multifunctional electronics. *Nano Letters*, 7(3):773–777, 2007.

[77] X. Ji-Guang and T. Kozawa. An algorithm for searching shortest path by propagating wave fronts in four quadrants. In *Proceedings of the 18th Design Automation Conference*, DAC '81, pages 29–36, Piscataway, NJ, USA, 1981. IEEE Press.

[78] B. J. Jordan, Y. Ofir, D. Patra, S. T. Caldwell, A. Kennedy, S. Joubanian, G. Rabani, G. Cooke, and V. M. Rotello. Controlled self-assembly of organic nanowires and platelets using dipolar and hydrogen-bonding interactions. *Small*, 4(11):2074–2078, 2008.

[79] A. Kahng, J. Lienig, I. Markov, and J. Hu. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer, 2011.

[80] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: application in vlsi domain. In *Proceedings of the 34th annual Design Automation Conference*, DAC '97, pages 526–529, New York, NY, USA, 1997. ACM.

[81] T. Kempf, G. Ascheid, and R. Leupers. Principles of design space exploration. In *Multiprocessor Systems on Chip*, pages 23–47. Springer New York, 2011. 10.1007/978-1-4419-8153-0_3.

[82] A. Kennings and K. Vorwerk. Force-directed methods for generic placement. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(10):2076 –2087, oct. 2006.

[83] J. S. Kilby. Miniaturized electronic circuits (reprint of u. s. patent no. 3,138, 743). *Solid-State Circuits Newsletter, IEEE*, 12(2):44 –54, spring 2007.

[84] R. Kling. *Optimization by Simulated Evolution and its Application to cell placement*. PhD thesis, University of Illinois, Urbana, 1990.

[85] A. Kuhn and T. Verwaest. FAME, a polyglot library for metamodeling at runtime. In *Workshop on Models at Runtime*, pages 57–66, 2008.

[86] T. Kühne. Contrasting classification with generalisation. *Sixth Asia-Pacific Conference on Conceptual Modelling (APCCM 2009)*, January 2009.

[87] I. Kuon and J. Rose. Area and delay trade-offs in the circuit and architecture design of fpgas. In *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, FPGA '08, pages 149–158, New York, NY, USA, 2008. ACM.

[88] I. Kuon and J. Rose. Automated transistor sizing for fpga architecture exploration. In *Proceedings of the 45th annual Design Automation Conference*, DAC '08, pages 792–795, New York, NY, USA, 2008. ACM.

[89] I. Kuon, R. Tessier, and J. Rose. Fpga architecture: Survey and challenges. *Found. Trends Electron. Des. Autom.*, 2:135–253, February 2008.

[90] L. Lagadec. *Abstraction and modélisation et outils de cao pour les architectures reconfigurables*. PhD thesis, Université de Rennes 1, 2000.

[91] L. Lagadec, D. Lavenier, E. Fabiani, and B. Pottier. Placing, routing, and editing virtual fpgas. In G. Brebner and R. Woods, editors, *Field-Programmable Logic and Applications*, volume 2147 of *Lecture Notes in Computer Science*, pages 357–366. Springer Berlin / Heidelberg, 2001.

[92] L. Lagadec and D. Picard. Software-like debugging methodology for reconfigurable platforms. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1 –4, may 2009.

[93] L. Lagadec and D. Picard. Smalltalk debug lives in the matrix. In *International Workshop on Smalltalk Technologies*, IWST '10, pages 11–16, New York, NY, USA, 2010. ACM.

[94] L. Lagadec, D. Picard, and B. Pottier. *Dynamic System Reconfiguration in Heterogeneous Platforms*, chapter 13. Spatial Design : High Level Synthesis. Springer, 2009.

[95] L. Lagadec and B. Pottier. Object-oriented meta tools for reconfigurable architectures. In *Reconfigurable Technology: FPGAs for Computing and Applications II, SPIE Proceedings 4212*, 2000.

[96] L. Lagadec, B. Pottier, and D. Picard. Toolset for nano-reconfigurable computing. *Microelectronics Journal*, 40(4-5):665 – 672, 2009. European Nano Systems (ENS 2007); International Conference on Superlattices, Nanostructures and Nanodevices (ICSNN 2008).

[97] M. Lai and D. Wong. Slicing tree is a complete floorplan representation. In *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, pages 228 –232, 2001.

[98] L. Lavagno, G. Martin, and L. Scheffer. *Electronic Design Automation for Integrated Circuits Handbook - 2 Volume Set*. CRC Press, Inc., Boca Raton, FL, USA, 2006.

[99] S. Lee and D. F. Wong. Timing-driven routing for fpgas based on lagrangian relaxation. In *Proceedings of the 2002 international symposium on Physical design*, ISPD '02, pages 176–181, New York, NY, USA, 2002. ACM.

[100] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein. Quantum cellular automata. *Nanotechnology*, 4(1):49, 1993.

[101] S. Li and C. Ebeling. Quickroute: a fast routing algorithm for pipelined architectures. In *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pages 73 – 80, dec. 2004.

[102] J.-M. Lin and Y.-W. Chang. TCG-S: Orthogonal Coupling of P*-Admissible Representations for General Floorplans. In *DAC '02: Proceedings of the 39th conference on Design automation*, pages 842–847, New York, NY, USA, 2002. ACM.

[103] J.-M. Lin and Y.-W. Chang. Tcg: A transitive closure graph-based representation for general floorplans. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 13(2):288 –292, feb. 2005.

[104] Y. Liu, J.-H. Chung, W. K. Liu, and R. S. Ruoff. Dielectrophoretic assembly of nanowires. *The Journal of Physical Chemistry B*, 110(29):14098–14106, 2006.

[105] Y.-T. Liu, X.-M. Xie, Y.-F. Gao, Q.-P. Feng, L.-R. Guo, X.-H. Wang, and X.-Y. Ye. Gas flow directed assembly of carbon nanotubes into horizontal arrays. *Materials Letters*, 61(2):334 – 338, 2007.

[106] J. Luu, J. H. Anderson, and J. S. Rose. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, pages 227–236, New York, NY, USA, 2011. ACM.

[107] J. Mar. The application of tcad in industry. In *Simulation of Semiconductor Processes and Devices, 1996. SISPAD 96. 1996 International Conference on*, pages 139 – 145, sept. 1996.

[108] G. Martin, B. Bailey, and A. Piziali. *ESL Design and Verification: A Prescription for Electronic System Level Methodology*. Morgan Kaufmann, USA, 2007. 488p.

[109] G. Martin, L. Lavagno, and J. Louis-Guerin. Embedded uml: a merger of real-time uml and co-design. In *Proceedings of the ninth international symposium on Hardware/software codesign*, CODES '01, pages 23–28, New York, NY, USA, 2001. ACM.

[110] L. McMurchie and C. Ebeling. Pathfinder: A negotiation-based performance-driven router for fpgas. In *Field-Programmable Gate Arrays, 1995. FPGA '95. Proceedings of the Third International ACM Symposium on*, pages 111 – 117, 1995.

[111] T. Mens and P. V. Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125 – 142, 2006. Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005).

[112] K. Mikami and K. Tabuchi. A computer program for optimal routing of printed circuit connectors. *IFIPS Proceedings*, H47:1475–1478, 1968.

[113] G. Moore. No exponential is forever: but "forever" can be delayed! [semiconductor industry]. In *Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC. 2003 IEEE International*, pages 20 – 23 vol.1, 2003.

[114] C. A. Moritz and T. Wang. Latching on the Wire and Pipelining in Nanoscale Designs. *3rd Workshop on Non-Silicon Computation (NSC-3), ISCA'04, Germany*, june 2004.

[115] C. A. Moritz, T. Wang, P. Narayanan, M. Leuchtenburg, Y. Guo, C. Dezan, and M. Bennaser. Fault-Tolerant Nanoscale Processors on Semiconductor Nanowire Grids. *IEEE Transactions on Circuits and Systems I, special issue on Nanoelectronic Circuits and Nanoarchitectures*, november 2007.

[116] P. A. Muller, F. Fleurey, and J. M. Jézéquel. Weaving Executability into Object-Oriented Meta-Languages. In *LNCS*, Montego Bay, Jamaica, Oct. 2005. MODELS/UML'2005, Springer.

[117] H. Naeimi and A. DeHon. Fault-tolerant sub-lithographic design with rollback recovery. *Nanotechnology*, 19(11):115708 (17pp), 2008.

[118] P. Narayanan, J. Kina, P. Panchapakeshan, C. O. Chui, and C. A. Moritz. Integrated device-fabric explorations and noise impact and mitigation in nanoscale fabrics. *to be submitted to ACM Journal on Emerging Technologies in Computing Systems (JETC).*

[119] P. Narayanan, M. Leuchtenburg, J. Kina, P. Joshi, P. Panchapakeshan, C. O. Chui, and C. A. Moritz. Parameter variability in nanoscale fabrics: Bottom-up integrated exploration. *Defect and Fault-Tolerance in VLSI Systems, IEEE International Symposium on*, 0:24–31, 2010.

[120] P. Narayanan, M. Leuchtenburg, T. Wang, and C. Moritz. Cmos control enabled single-type fet nasic. In *Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual*, pages 191 –196, april 2008.

[121] P. Narayanan, C. A. Moritz, K. W. Park, and C. O. Chui. Validating cascading of crossbar circuits with an integrated device-circuit exploration. *Nanoscale Architectures, IEEE International Symposium on*, 0:37–42, 2009.

[122] P. Narayanan, K. Park, C. Chui, and C. Moritz. Manufacturing patway and associated challenges for nanoscale computational systems. In *9th IEEE Nanotechnology conference*, 2009.

[123] P. Narayanan, T. Wang, and C. A. Moritz. Programmable cellular architectures at the nanoscale. *Nano Communication Networks*, 1(2):77 – 85, 2010.

[124] K. Nepal, R. Bahar, J. Mundy, W. Patterson, and A. Zaslavsky. Designing nanoscale logic circuits based on markov random fields. *Journal of Electronic Testing*, 23:255–266, 2007. 10.1007/s10836-006-0553-9.

[125] A. N. Ng, I. L. Markov, R. Aggarwal, and V. Ramachandran. Solving hard instances of floorplacement. In *Proceedings of the 2006 international symposium on Physical design*, ISPD '06, pages 170–177, New York, NY, USA, 2006. ACM.

[126] H. T. Ng, J. Han, T. Yamada, P. Nguyen, Y. P. Chen, and M. Meyyappan. Single crystal nanowire vertical surround-gate field-effect transistor. *Nano Letters*, 4(7):1247–1252, 2004.

[127] M. Niemier and P. Kogge. The "4-diamond circuit"-a minimally complex nano-scale computational building block in qca. *IEEE Computer Society Annual Symposium on VLSI*, pages 3–10, 2004.

[128] M. Niemier, A. Rodrigues, and P. Kogge. A potentially implementable fpga for quantum dot cellular automata. *1st Workshop on Non-silicon Computation*, 2002.

[129] R. N. Noyce. Semiconductor device-and-lead structure, reprint of u.s. patent 2,981,877 (issued april 25, 1961. filed july 30, 1959). *Solid-State Circuits Newsletter, IEEE*, 12(2):34 –40, spring 2007.

[130] Object Management Group. *Meta Object Facility (MOF) Core Specification Version 2.0*, 2006.

[131] I. OConnor, J. Liu, D. Navarro, R. Daviot, N. Abouchi, P. Gaillardon, and F. Clermidy. Molecular electronics and reconfigurable logic. *International Journal of Nanotechnology*, 7(4-8):367–382, 2010.

[132] P. O'Connor. Future trends in microelectronics - impact on detector readout. *SNIC Symposium, Standford, CA*, pages 1–6, 2006.

[133] A. Pal, A. Sachid, H. Gossner, and V. Rao. Insights into the design and optimization of tunnel-fet devices and circuits. *Electron Devices, IEEE Transactions on*, 58(4):1045 –1053, april 2011.

[134] P. R. Panda. Systemc: a modeling platform supporting multiple design abstractions. In *Proceedings of the 14th international symposium on Systems synthesis*, ISSS '01, pages 75–80, New York, NY, USA, 2001. ACM.

[135] J. Park, A. N. Pasupathy, J. I. Goldsmith, C. Chang, Y. Yaish, J. R. Petta, M. Rinkoski, J. P. Sethna, H. D. Abruna, P. L. McEuen, and D. C. Ralph. Coulomb blockade and the kondo effect in single-atom transistors. *Nature*, 417(6890):722–725, 06 2002.

[136] T. Perry. For the record: Kilby and the ic. *Spectrum, IEEE*, 25(13):40 –41, dec 1988.

[137] D. Picard. *Méthodes et outils logiciels pour l'exploration architecturale d'unité reconfigurable embarqueés*. PhD thesis, Université de Bretagne Occidentale, Brest, 2010.

[138] A. Plantec and V. Ribaud. PLATYPUS : A STEP-based Integration Framework. In *14th Interdisciplinary Information Management Talks (IDIMT-2006)*, pages 261–274, Tchèque, République, Sept. 2006.

[139] B. Pottier and J.-L. Llopis. Revisiting smalltalk-80 blocks: a logic generator for fpgas. In *FPGAs for Custom Computing Machines, 1996. Proceedings. IEEE Symposium on*, pages 48 –57, apr 1996.

[140] W. Qian, J. Backes, and M. D. Riedel. The synthesis of stochastic circuits for nanoscale computation. *Theoretical and Technological Advancements in Nanotechnology and Molecular Computation: Interdisciplinary Gains. IGI Global*, 2011.

[141] I. R. Quadri, H. Yu, A. Gamatie, E. Rutten, S. Meftali, and J.-L. Dekeyser. Targeting reconfigurable fpga based socs using the uml marte profile: from high abstraction levels to code generation. *International Journal of Embedded Systems*, 4(3/4):204–224, 2010.

[142] Y.-A. C. Randal E. Bryant. Verification of arithmetic circuits with binary moment diagrams. In *Design Automation, 1995. DAC '95. 32nd Conference on*, pages 535 –541, 1995.

[143] S. Rao, P. Sadayappan, F. Hwang, and P. Shor. The rectilinear steiner arborescence problem. *Algorithmica*, 7:277–288, 1992. 10.1007/BF01758762.

[144] W. Rao, A. Orailoq, and R. Karri. Topology aware mapping of logic functions onto nanowire-based crossbar architectures. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 723 –726, 0-0 2006.

[145] L. Renggli. *Dynamic Language Embedding With Homogeneous Tool Support.* Phd thesis, University of Bern, Oct. 2010.

[146] C. Rousseau, Y. Saint-Aubin, C. Rousseau, and Y. Saint-Aubin. The dna computer. In *Mathematics and Technology*, Springer Undergraduate Texts in Mathematics and Technology, pages 1–43. Springer New York, 2008. 10.1007/978-0-387-69216-6_13.

[147] R. L. Rudell. Multiple-valued logic minimization for pla synthesis. Technical Report UCB/ERL M86/65, EECS Department, University of California, Berkeley, 1986.

[148] H. Schift and A. Kristensen. Nanoimprint lithography. In B. Bhushan, editor, *Springer Handbook of Nanotechnology*, pages 239–278. Springer Berlin Heidelberg, 2007. 10.1007/978-3-540-29857-1_8.

[149] C. Sechen and A. Sangiovanni-Vincentelli. Timberwolf3.2: a new standard cell placement and global routing package. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, DAC '86, pages 432–439, Piscataway, NJ, USA, 1986. IEEE Press.

[150] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, EECS Department, University of California, Berkeley, 1992.

[151] P. Shabadi, A. Khitun, P. Narayanan, M. Bao, I. Koren, K. Wang, and C. Moritz. Towards logic functions as the device. In *Nanoscale Architectures (NANOARCH), 2010 IEEE/ACM International Symposium on*, pages 11 –16, june 2010.

[152] P. Shabadi, A. Khitun, P. Narayanan, M. Bao, I. Koren, K. L. Wang, and C. A. Moritz. Towards logic functions as the device. In *Proceedings of the 2010 IEEE/ACM International Symposium on Nanoscale Architectures*, Nanoarch '10, pages 11–16, Piscataway, NJ, USA, 2010. IEEE Press.

[153] Y. Shan and S. J. Fonash. Self-assembling silicon nanowires for device applications using the nanochannel-guided "grow-in-place" approach. *ACS Nano*, 2(3):429–434, 2008.

[154] A. Sharma, C. Ebeling, and S. Hauck. Piperoute: a pipelining-aware router for fpgas. In *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, FPGA '03, pages 68–77, New York, NY, USA, 2003. ACM.

[155] V. V. Shende, S. S. Bullock, and I. L. Markov. Synthesis of quantum logic circuits. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, pages 272–275, New York, NY, USA, 2005. ACM.

[156] N. A. Sherwani. *Algorithms for VLSI Physcial Design Automation.* Kluwer Academic Publishers, Norwell, MA, USA, 3rd edition, 1998.

[157] A. Shrestha, S. Tayu, and S. Ueno. Orthogonal ray graphs and nano-pla design. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 2930 –2933, may 2009.

[158] S. K. Shukla and R. I. Bahar, editors. *Nano, quantum and molecular computing: implications to high level design and validation.* Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[159] A. Singh and M. Marek-Sadowska. Efficient circuit clustering for area and power reduction in fpgas. In *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, FPGA '02, pages 59–66, New York, NY, USA, 2002. ACM.

[160] G. S. Snider and R. S. Williams. Nano/CMOS Architectures Using a Field-Programmable Nanowire Interconnect. *Nanotechnology*, 18(3):035204 (11pp), 2007.

[161] M. Stan, P. Franzon, S. Goldstein, J. Lach, and M. Ziegler. Molecular electronics: from devices and interconnect to circuits and architecture. *Proceedings of the IEEE*, 91(11):1940 – 1957, Nov. 2003.

[162] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley Professional, 2 edition, Jan. 2008.

[163] D. W. Steuerman, H.-R. Tseng, A. J. Peters, A. H. Flood, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and J. R. Heath. Molecular-mechanical switch-based solid-state electrochromic devices. *Angewandte Chemie*, 116(47):6648–6653, 2004.

[164] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal. Freepdk: An open-source variation-aware design kit. *Microelectronics Systems Education, IEEE International Conference on/-Multimedia Software Engineering, International Symposium on*, 0:173–174, 2007.

[165] D. B. Strukov and K. K. Likharev. CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotechnology*, 16:888–900, April 2005.

[166] D. B. Strukov and K. K. Likharev. Cmol fpga circuits. In *In Proc. of Int. Conf. on Computer Design, CDES'2006*, pages 213–219, 2006.

[167] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams. The missing memristor found. *Nature*, 453(7191):80–83, 05 2008.

[168] S. Sugahara and J. Nitta. Spin-transistor electronics: An overview and outlook. *Proceedings of the IEEE*, 98(12):2124 –2154, dec. 2010.

[169] Y. Sun, Rusli, and N. Singh. Room-temperature operation of silicon single-electron transistor fabricated using optical lithography. *Nanotechnology, IEEE Transactions on*, 10(1):96 –98, jan. 2011.

[170] M. Tahoori. A mapping algorithm for defect-tolerance of reconfigurable nano-architectures. In *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, pages 668 – 672, nov. 2005.

[171] C. Teodorov and L. Lagadec. Fpga sdk for nanoscale architectures. In *6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC'11)*, 2011.

[172] C. Teodorov, P. Narayanan, L. Lagadec, and C. Dezan. Regular 2d nasic architecture and design space exploration. In *Nanoscale Architectures, IEEE / ACM International Symposium on (NanoArch'11)*, 2011.

[173] C. Teodorov, P. Narayanan, L. Lagadec, C. Dezan, and C. A. Moritz. Regular 2d nasic architecture and design space exploration. In *to be submited at Nanoarch '11 - 7th IEEE/ACM International Symposium on Nanoscale Architectures*, 2011.

[174] C. Teodorov, D. Picard, and L. Lagadec. Fpga physical-design automation using model-driven engineering. *6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC'11) 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC'11)*, 2011.

[175] M. Tommiska and J. Skyttä. Dijkstra's shortest path routing algorithm in reconfigurable hardware. In *Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, FPL '01, pages 653–657, London, UK, 2001. Springer-Verlag.

[176] P. D. Tougaw, C. S. Lent, and W. Porod. Bistable saturation in coupled quantum dot cells. *Journal of Applied Physics*, 74(5):3558 –3566, sep 1993.

[177] J. Tour, W. van Zandt, C. Husband, S. Husband, L. Wilson, P. Franzon, and D. Nackashi. Nanocell logic gates for molecular computing. *IEEE Transactions on Nanotechnology*, 1:100–109, 2002.

[178] W. Tsu, K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhani, V. George, J. Wawrzynek, and A. DeHon. Hsra: high-speed, hierarchical synchronous reconfigurable array. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, FPGA '99, pages 125–134, New York, NY, USA, 1999. ACM.

[179] Y. Vanderperren and W. Dehaene. UML 2 and SysML: An Approach to Deal with Complexity in SoC/NoC Design. In E. European design and Automation Association, editors, *Design, Automation and Test in Europe DATE'05*, volume 2, pages 716–717, Munich Allemagne, 03 2005. Submitted on behalf of EDAA (http://www.edaa.com/).

[180] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet. A co-design approach for embedded system modeling and code generation with uml and marte. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 226–231, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[181] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard. Programmable active memories: reconfigurable systems come of age. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 4(1):56 –69, march 1996.

[182] D. Wang, B. Sheriff, M. McAlpine, and J. Heath. Development of ultra-high density silicon nanowire arrays for electronics applications. *Nano Research*, 1:9–21, 2008. 10.1007/s12274-008-8005-8.

[183] T. Wang, M. Ben-Naser, Y. Guo, and C. A. Moritz. Self-healing wire-streaming processors on 2-d semiconductor nanowire fabrics. *NSTI (Nano Science and Technology Institute) Nanotech'06, Boston, MA*, may 2006.

[184] T. Wang, P. Narayanan, and C. Andras Moritz. Heterogeneous two-level logic and its density and fault tolerance implications in nanoscale fabrics. *IEEE Transactions on Nanotechnology*, 8:22–30, Jan. 2009.

[185] T. Wang, Z. Qi, and C. A. Moritz. Opportunities and challenges in application-tuned circuits and architectures based on nanodevices. In *Proceedings of the 1st conference on Computing frontiers*, CF '04, pages 503–511, New York, NY, USA, 2004. ACM.

[186] N. Weaver, J. Hauser, and J. Wawrzynek. The sfra: a corner-turn fpga architecture. In *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, FPGA '04, pages 3–12, New York, NY, USA, 2004. ACM.

[187] N. Weaver, Y. Markovskiy, Y. Patel, and J. Wawrzynek. Post-placement c-slow retiming for the xilinx virtex fpga. In *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, FPGA '03, pages 185–194, New York, NY, USA, 2003. ACM.

[188] D. Whang, S. Jin, and C. M. Lieber. Nanolithography using hierarchically assembled nanowire masks. *Nano Letters*, 3(7):951–954, 2003.

[189] Q. Xia, W. Robinett, M. W. Cumbie, N. Banerjee, T. J. Cardinali, J. J. Yang, W. Wu, X. Li, W. M. Tong, D. B. Strukov, G. S. Snider, G. Medeiros-Ribeiro, and R. S. Williams. Memristor-cmos hybrid integrated circuits for reconfigurable logic. *Nano Letters*, 9(10):3640–3645, 2009.

[190] X. Xiong, L. Jaberansari, M. G. Hahm, A. Busnaina, and Y. J. Jung. Building highly organized single-walled-carbon-nanotube networks using template-guided fluidic assembly. *Small*, 3(12):2006–2010, 2007.

[191] J. Xu, X. Hong, T. Jing, Y. Cai, and J. Gu. An efficient hierarchical timing-driven steiner tree algorithm for global routing. *Integr. VLSI J.*, 35:69–84, August 2003.

[192] H. Yan, H. S. Choe, S. Nam, Y. Hu, S. Das, J. F. Klemic, J. C. Ellenbogen, and C. M. Lieber. Programmable nanowire circuits for nanoprocessors. *Nature*, 470(7333):240–244, 02 2011.

[193] S. Yang. Logic Synthesis and Optimization Benchmarks User Guide, Version 3.0. Technical report, MCNC Technical Report, January 1991.

[194] J. W. Yoder and R. E. Johnson. The adaptive object-model architectural style. In *WICSA 3: Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture*, pages 3–27, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.

[195] M. Zamani and M. Tahoori. Self-timed nano-pla. In *Nanoscale Architectures (NANOARCH), 2011 IEEE/ACM International Symposium on*, pages 78 –85, june 2011.

[196] M. Zaveri and D. Hammerstrom. Cmol/cmos implementations of bayesian polytree inference: Digital and mixed-signal architectures and performance/price. *Nanotechnology, IEEE Transactions on*, 9(2):194 –211, march 2010.

[197] W. Zhao and Y. Cao. New generation of predictive technology model for sub-45nm design exploration. In *Proceedings of the 7th International Symposium on Quality Electronic Design*, ISQED '06, pages 585–590, Washington, DC, USA, 2006. IEEE Computer Society.

MODEL-DRIVEN PHYSICAL-DESIGN FOR FUTURE NANOSCALE ARCHITECTURES

Abstract

In the context where the traditional CMOS technology approaches its limits, some nanowire-based fabric proposals emerged, which all exhibit some common key characteristics. Among these, their bottom-up fabrication process leads to a regularity of assembly, which means the end of custom-made computational fabrics in favor of regular structures. Hence, research activities in this area, focus on structures conceptually similar to today's reconfigurable PLA and/or FPGA architectures[165, 160]. A number of different fabrics and architectures are currently under investigation, e. g.CMOL[165], FPNI[160], NASIC[115]. These proof-of-concept architectures take into account some fabrication constraints and support fault-tolerance techniques. What is still missing is the ability to capitalize on these experiments while offering a one-stop shopping point for further research, especially at the physical-design level of the circuit design tool-flow. Sharing metrics, tools, and exploration capabilities is the next challenge to the nano-computing community.

We address this problem by proposing a model-driven physical-design toolkit based on the factorization of common domain-specific concepts and the reification of the tool-flow. We used this tool-flow to drive the design-space exploration in the context of a novel nanoscale architecture, and we showed that such an approach assures design convergence based on frequent quantitative evaluations, moreover it enables incremental evolution of the architecture and the automation flow.

SYNTHÈSE PHYSIQUE DIRIGÉE PAR LES MODÈLES POUR LES ARCHITECTURES NANOMÉTRIQUE DU FUTUR

Résumé

Actuellement, comme la technologie CMOS arrive à ses limites, plusieurs alternatives architecturales nanométriques sont étudiées. Ces architectures partagent des caractéristiques communes, comme par exemple la régularité d'assemblage, qui contraint le placement de dispositifs physiques à des motifs réguliers. Par conséquence, les activités de recherche dans ce domaine sont focalisées autour des structures régulières similaires, d'un point de vue conceptuel, aux architectures réconfigurables de type PLA et FPGA[165, 160]. Parmi ces différents travaux, ont peut citer CMOL[165], FPNI[160], NASIC[115]. Ces prototypes architecturaux sont conçus pour répondre à des contraintes de fabrication et incluent des politiques de tolérance aux défauts. Par contre, il manque la possibilité d'exploiter ces expériences et d'offrir une solution qui, en capitalisant les résultats obtenus, puisse offrir une infrastructure unique pour les futurs recherches dans ce domaine. Ceci est vrai surtout au niveau du flot de conception physique ciblant l'automatisation du processus de création de circuit. Le partage de métriques, outils et supports d'exploration est le futur défi de la communauté nano-électronique.

On répond à ce problème en proposant un flot de conception physique, reposant sur une méthodologie de développement dirigé par les modèles, qui factorise les concepts métiers et réifie les éléments du flot de conception. Nous avons utilisé ce flot pour explorer l'espace de conception d'une nouvelle architecture nano-métrique et on a montré qu'une telle démarche permet la convergence du processus de conception à l'aide de fréquentes évaluations quantitatives. De plus, cette méthodologie permet l'évolution incrémentielle de l'architecture et du flot de conception.