A FRAMEWORK TO EXPLORE LOW-POWER ARCHITECTURE AND VARIABILITY-AWARE TIMING ESTIMATION OF FPGAS

LEE CHEE SING

(B.Eng.(Hons.), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2007

Acknowledgements

My sincere thanks go to my advisor, Assistant Professor Ha Yajun. Without his help, this work would never have been possible. I have enjoyed a wonderful research experience under his supervision as he has gone beyond the duties of a supervisor to act as a mentor as well as a supporter.

I would also like to give special thanks to Professor Ben Chen (M. Eng./Ph.D. Program Coordinator), who provided impetus for the project, laid down the initial specifications and gave advices. Also, I would like to give a special acknowledgment to Professor Jonathan Rose and Vaughn Betz (creators of VPR tool) from the University of Toronto as well as Professor Jorge Stolfi (creator of affine arithmetic model) for their help in formulating the technical aspects of this work. Their contribution of ideas and software had greatly aided in the development of my research.

In addition, during this Master's program, I have gained wonderful experience working with different groups of people. Special thanks to Dr Heng Chun Huat for his valuable contribution to the project on the designing of the reconfigurable buffer for a low-power FPGA architecture. Thanks to Pu Yu and Kumaran, with who have allow me to gain more insight to VLSI circuit designing in this project too. Next, thanks to my hardware timing analysis project team (Zhang Wenjuan, Chen Xiaolei and Loke Wei Ting), who have worked closely with me on the research on

timing estimation in FPGAs. Also, thanks to my fellow colleagues, Shakith, Teo Jenn Yue, Li Yanhui, Shefali, Zhang Wenjuan, Chen Xiaolei, Loke Wei Ting and Yu Heng for the various knowledge enriching sharing mini-seminars that are organized by our supervisor.

Last but not least, I would like to give special thanks to my family, friends and anyone who is not mentioned here but had helped in one way or another.

Contents

Abstract

This thesis is written in 3 main sections. First, a new CAD framework is designed. As semiconductor technology gets scaled down, more transistors will be allowed to be fabricated onto a single chip. There is a need for a new tool to handle the building of larger FPGAs. Heterogeneity is brought into the development phase to improve FPGAs' qualities. We propose a framework to allow researchers to design arbitrary architectures with the help of a graphical user interface. It enables the initialization of essential circuit parameters to obtain a basic architectural layout. Editing of the initial design can be performed to allow the creation of an arbitrary architectural design. It is built in with placement and routing capabilities to test the feasibility of the newly designed architecture. Different arbitrary architectures are being tested using a set of MCNC benchmarks. Furthermore, porting of the designed architecture's resource graph to the current state-of-art VPR for more complete testing is made available.

Second, we use the developed framework to investigate an alternative approach to minimize the short-circuit power of FPGA global interconnects without the luxury of

dual supply. A reconfigurable buffer, with programmable driving strength, is designed and integrated into the FPGA switch block. EDA support is built into our framework to test this new architecture. With our methodology, interconnect buffers can choose the right driving strength based on the exact wire load after detailed routing. Our simulation results show that, by applying larger driving strength along the critical paths and relaxing the driving strength along the non-critical paths, the proposed FPGA architecture can reduce the overall dynamic power by 6.10% - 10.05%, compared with the conventional FPGA architecture. Our approach is complementary to the existing dual supply voltage solution. Both techniques can be combined to further reduce the overall dynamic power consumption.

Third, we use a developed framework VPR to explore a fast and accurate interval-based timing estimator for variability-aware FPGA physical synthesis tools. As process variations of deep sub-micron technologies have created significant timing uncertainty, this generates the need for a new generation of variability-aware physical synthesis tools for FPGAs. Ideally, variability-aware tools should be able to perform both timing variability estimation during the synthesis and timing variability analysis after the synthesis. SSTA methods are being developed to perform the timing variability analysis after the synthesis, but they are computationally expensive and not fast enough to provide the timing variability estimation during the synthesis. Hence, we propose a fast and accurate interval-based method for the timing variability estimation. This method uses correlation-aware affine intervals instead of

probability density distributions to model timing uncertainties. Compared to Monte Carlo simulations, we estimate the mean of timing variation within the accuracy of 1%, the average looseness range of about 22.6% and 4.5% for the Uniform and Gaussian distribution respectively and a 1000X simulation speed-up. This work can be easily extended to ASIC flows. Furthermore, using our developed framework, this case study can be extended to non-regular architectures.

List of Figures

List of Tables

List of Abbreviations

AA	Affine Arithmetic
ASIC	Application-Specific Integrated Circuit
CAD	Computer-Aided Design
CLB	Configurable Logic Block
CMOS	Complementary MetalOxideSemiconductor
DLL	Delay-Lock Loop
EDA	Electronic Design Automation
FPGA	Field-Programmable Gate Array
GUI	Graphical User Interface
HDL	Hardware Description Language
I/O	Input/Output
IA	Interval Arithmetic
IOB	Input/Output Block
LE	Logic Element

LUT Look-Up Tables

MPGA Mask-Programmable Gate Array

MCNC Microelectronics Corporation of North Carolina

PLD Programmable Logic Device

RRG Routing Resource Graph

RTL Register Transfer Level

STA Static Timing Analysis

SSTA Statistical Static Timing Analysis

SOC System-On-a-Chip

VPR Versatile Placement and Routing tool for FPGAs

VLSI Very Large Scale Integration

TTL Transistor-Transistor Logic

Chapter 1

Introduction

For the past few decades, microelectronics has been the technology in demand for the development of both the hardware and software systems. With the continuous increase in the level of integration of electronic devices, this form of technology improves tremendously. The trend towards higher integration brings about the evolution of more sophisticated and faster systems to meet the increasing market demand. As a result, the final products become better and cheaper.

Field programmable gate arrays (FPGAs) are first introduced during the mid-1980s. At that time, FPGAs are only made up of transistor-transistor logic (TTL) equivalent logic gates. With enhancements in the very-large-scale integration (VLSI) processing technology, FPGAs have evolved to system-on-a-chips (SOC) with millions of logic gates being packed together. Ever since, FPGA becomes a widely adopted design at the heart of most electronic systems for its wide abundance of resources and

efficiency.

Moreover, with the discovering of new processing techniques over the recent years, the semiconductor technology has been seen scaling down as predicted by the Moore's Law. This results in more transistors to be able to get fabricated onto a single chip; and opens up more opportunities for researches to build larger and sophisticated FPGAs than ever. Furthermore, new features are continuously being discovered and added into these FPGAs to cater for different design needs. For example, power efficient FPGAs are being developed for portable electronic devices for which low power consumption is a key requirement. As of today, we have seen numerous researches with innovative ideas evolving and this has led to the development of FPGA architectures of higher qualities and efficiencies.

1.1 FPGA Architecture

An FPGA architecture is made up of several millions of logic gates fused together. In order to develop an optimized and efficient architecture is not an easy task. However, a good approach to start off is to first implement an architecture instance in all the selected classes of FPGAs and evaluates their performances. The architecture displaying the best combination of placement and routing results in terms of timing, area or power is deemed to be the best. Previous researches [1–5] have shown that a proper design of the routing architecture does play a major role in determining its quality. The description of the architecture plays an important role in determining

the overall efficiency of the FPGA too.

Different approaches in describing an FPGA architecture have been adopted in many of the existing frameworks. One brute force method to describe the routing architecture is by manually specifying all the interconnections between the logic blocks through the use of a routing resource graph (RRG). This enables researches to have the flexibility in describing different forms of architectures. However, this method is not practical as a typical FPGA RRG's size can go up to megabytes or even larger. Eventually, due to its inefficiency and impracticability, such a low level and detailed specification is not applied.

A more practical approach is to first design a basic tile with its interconnections manually and uses a program to automatically replicate that basic structure into an array to form a complete architecture. This technique is applied by George in [5] to design low energy FPGA architectures. Not only it is time consuming, this method also shows limitation in terms of flexibility as the whole architecture is a replica of the basic tile.

1.2 Process variation

With the continuous scaling of technology into the deep sub-micron regions, the amount of variability increases significantly in the process parameters that have to be accounted for. For example, more than 35% variations on the gate length are cited for 90nm processes and they are even larger for 65 nm processes [6]. Also, as shown

in Table 1.1 [7], the magnitude of the parameter variations does not scale down as fast as the nominal values. As such, the parameter variation, as a percentage of the nominal value, gets larger with decreasing technology.

Parameters	Nominal Values					3σ Values				
Years	1997	1999	2002	2005	2006	1997	1999	2002	2005	2006
Leff [nm]	250	180	130	100	70	80	60	50	40	33
Tox [nm]	5	4.5	4	3.5	3	0.4	0.36	0.39	0.42	0.48
Vdd [V]	2.5	1.8	1.5	1.2	0.9	0.25	0.18	0.15	0.12	0.09
Vth [mV]	500	450	400	350	300	50	45	40	40	40
W [μu]	0.8	0.55	0.5	0.4	0.3	0.2	0.17	0.14	0.12	0.1
H [μm]	1.2	1	0.9	0.8	0.7	0.3	0.3	0.27	0.27	0.25
p [mΩ]	45	50	55	60	75	10	12	15	19	25

Table 1.1: CMOS technology roadmap

Process variations [8, 9] can be classified as inter-die variations, which affect the entire chip, and intra-die variations, which are the results of layout-specific variations. These variations are normally accompanied with a complex spatial or temporal correlation structure. They create significant timing uncertainty and yield degradation. This growing problem brings about the need to build the next generation variability-aware electronic design automation (EDA) tools.

The above observation is especially important for FPGA vendors because they are almost always the first to use the most advanced technologies. For example, Xilinx is the first in the whole semiconductor industry to fabricate their Virtex-2 FPGAs in 130nm, Virtex-4 in 90 nm, and Virtex-5 in 65nm processes. As the process shrinks, variations in effective channel length, threshold voltage and gate oxide thickness become more prominent. This will greatly influence the timing performance

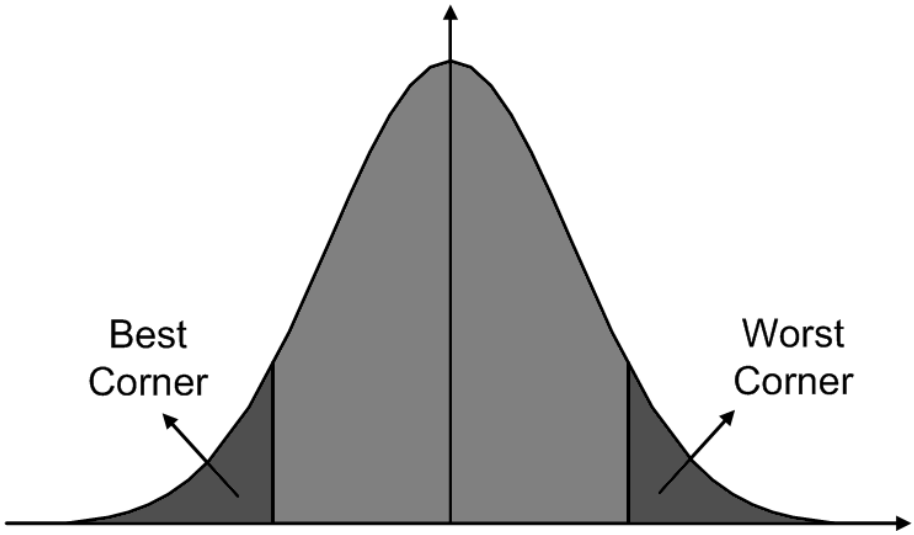
of FPGAs. Hence, the FPGA physical synthesis tools need to consider the impact of process variations on timing in order to help guide timing-driven optimizations.

1.2.1 Traditional corner-based timing method

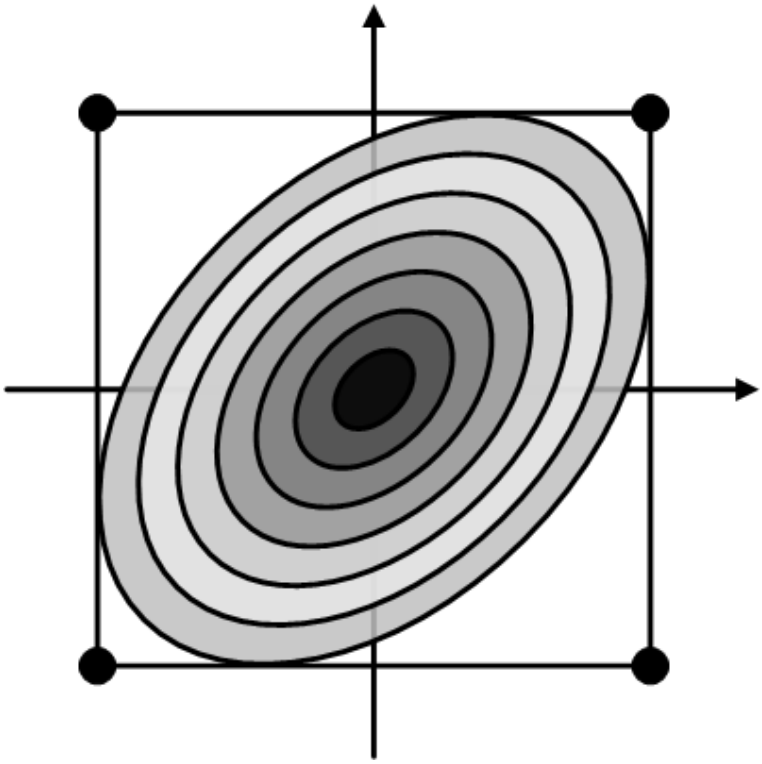
Process variations and their correlations have been studied over the years. Their importance accelerates as the technology continues to scale down. Traditionally, parameter variations and correlations are handled using the corner-based deterministic static timing analysis as shown in Figure 1.1 [7]. From Figure 1.1(a), two corners known as the worst case and best case are individually timed for a single parameter variation. However, if the two parameter variations are of significance, four corners are to be timed individually as shown in Figure 1.1(b). Hence, as the parameter variations increases, an exponential number of corners need to examine individually. This makes the approach to be cumbersome and inefficient. In addition, the corner-based approach only provides information on whether the circuit is able to function at the extreme corners and not on the quantitative yield information which is more critical.

1.3 Problem definition

Although there had been existing works which are efficient in describing FPGA architectures, reducing power usage and handling of process variations in FPGAs, there are still many problems that need to be solved to improve them.



(a) Two corners for single parameter



(b) Four corners for two parameters

Figure 1.1: Corner-based timing analysis: 2n corners for n parameters

1.3.1 Limitation of CAD tools

Currently, there have been several promising computer-aided design (CAD) tools [1–5] capable of describing routing architecture with enhanced design complexity, better cost-saving or even improved efficiency. For example, Emerald [1] makes use of the *WireC* schematics to describe its routing architecture. This method requires inputs like routing architecture description, logic block architecture description and architecture specific metrics in order to provide the basic features needed in placement and routing tools. In another example, the Versatile Placement and Routing tool (VPR) [2, 3] makes use of an FPGA architecture description language to describe its routing architecture. An "architecture generator" is used to convert this specification into a detailed and complete architecture for future work on optimization and visualization. However, both the Emerald and VPR CAD tools share a common limitation. Their architecture description techniques limit the range of architectures only to a selected class of templates. This limitation prevents the design of heterogeneous architectures.

1.3.2 Limitation of power reduction in interconnects

Among the routing resources in an FPGA architecture, switch buffers are the most important components that determine its performance. The buffer not only behaves as an intermediate repeater to regenerate the signal, it also breaks a long RC network to minimize the interconnect delay. Therefore, the buffer chosen must be large enough

to drive its downstream circuits. While buffers can be fully customized for various applications in application specific integrated circuits (ASIC) design, FPGAs do not have such freedom because they are pre-fabricated. Targeting at driving the worst case of load normally results in having unnecessarily large buffers within a FPGA chip. These oversized buffers can cause undesirable problems.

First, due to the non-zero rising and falling time of the input signal, a larger buffer will result in larger peak and average short-circuit currents during the transition period, hence resulting in an increase in the short circuit power. From the simulations, it can be shown that the short-circuit power accounts for roughly 10% of the total dynamic power, depending on the actual synthesized circuits. As a result, the dynamic power, which consists of both the switching power and the short-circuit power, is increased.

Second, a larger buffer creates more ground-bounce noise. In custom ASIC design, large transient current is avoided by using the minimum required buffers. This will minimize the ground bounce noise introduced by Ldi/dt, where L is the inductance associated with the package pins, bonding wires and on-chip metal lines for power routing. Ground bounce noise reduces the available noise margin for the digital circuits [10]. In addition, it also deteriorates the performance of the sensitive analog circuit on the chip, such as delay-lock loop (DLL), which is crucial for the functioning of large digital circuits. If an oversized buffer is used within the FPGAs, large transient current and thus large ground bounce noise are inevitable.

1.3.3 Limitation of SSTA techniques

In relation to process variation, there has been several works [11–21] considering the impact of variations on circuit performances using statistical static timing analysis (SSTA). These approaches are classified into various categories such as block-based, path-based, incremental, etc. In [12, 14], the authors propose techniques to get the bounds of the delay distributions instead of calculating the exact distributions using path-based or block-based analysis techniques. In [16], the proposed approach does an estimation based on a generic path analysis rather than evaluating every path statistically. However, many of these researchers have advocated complicated SSTA techniques, primarily due to handling correlation and path reconvergence during the MAX operation fundamental to static timing analysis (STA). This leads to undesirable high computation complexity and large CPU overhead. Furthermore, most of these statistical analysis techniques typically assume the circuit parameters as independent random variables with a Gaussian distribution. This is not true in most cases.

1.4 Proposed research approach

From the problem definitions above, we propose three approaches to solve each of them individually. First, a CAD framework capable of designing heterogeneous architecture is developed. Second, a FPGA architecture with reconfigurable buffer

is designed to allow different operating buffer modes to save power. Third, a novel idea is proposed to handle process variations while considering spatial correlation and path reconvergence.

1.4.1 Proposed CAD framework

In order to facilitate the designing of heterogeneous FPGA routing architecture, a graphical user interface (GUI) framework is proposed. In this framework, an interface is built to allow users to input essential parameters to generate a generic routing architecture. This removes the hassle to come out with a descriptive language to implement the architecture. After which, using the drawn architecture, users can click on any components and do editing to them. With this flexibility, users can design any kind of routing architectures that they desired. This eliminates any forms of restriction or constrain that are encountered in the existing CAD tools.

After the design is finalized, a RRG is generated. This RRG is a detailed internal representation of the routing architecture which specifies how each component in the architecture is connected with each other. Placement and routing algorithms are implemented to test the feasibility of the design architecture. The placer does the placing of the logic blocks in the physical position of the FPGA while the router finds the best path for all the nets.

1.4.2 Proposed low power FPGA architecture

In order to provide a way to minimize the transient current by using only the minimum required driving strength, the use of a reconfigurable buffer is proposed. The concept behind our methodology is that, a large buffer can be physically considered as a combination of smaller buffer cells. The different modes of driving strength can be obtained through the binary combinations of the small buffer cells. We integrate this reconfigurable buffer into the FPGA switch blocks. In this way, we are capable of choosing the right driving strength for each wire based on their exact load after detailed routing. By using larger driving strength along the critical paths and relaxing the driving strength along the non-critical paths, the overall dynamic power consumption and transient current can be reduced.

1.4.3 Proposed variability-aware timing estimation

In order to perform a fast and accurate timing estimation for the variability-aware FPGA physical synthesis tools, an interval-based method is proposed. Two models are initially suggested: interval arithmetic (IA) and affine arithmetic (AA). IA [22] is a surprisingly long-lived branch of range analysis. It makes use of intervals to represent uncertainties in variables. However, it does not consider correlation and dependency between the variables. On the other hand, AA [23], which is a novel refinement of interval analysis, can be applied to the problem of circuit timing analysis [24,25] and can preserve correlations among variables. With the motivation in

mind, we employ AA to propose a new interval-based timing estimation technique for FPGAs with correlation and dependencies among process parameters being accounted for. Furthermore, AA is chosen for its low complexity and distribution independent property, in contrast to the existing SSTA methods.

1.5 Contributions

The work done for this thesis makes the following contributions:

1. Designed a CAD framework capable of producing an arbitrary FPGA routing architecture.
2. Incorporated placement and routing algorithms to test the framework.
3. Designed a power efficient FPGA architecture.
4. Designed a fast interval-based timing estimator for FPGAs.

1.6 Thesis organization

The remainder of this thesis is organized as follows. The next chapter presents some general background on the research topic and related works. Chapter 3 describes the modeling of the proposed framework. Chapter 4 shows a design of an arbitrary architecture and some generated results. Chapter 5 discusses a case study to investigate a low-power FPGA switch block with reconfigurable buffers using our framework.

Chapter 6 presents another case study to investigate the use of the affine model to handle process variations using VPR. Finally, Chapter 7 presents the conclusions and suggestions for future work.

Chapter 2

Background and Related Works

This chapter begins with a general overview of the different types of the FPGA routing architectures used in the academic research as well as in the industry. Next, a description of a typical CAD flow for a FPGA design is illustrated. Finally, literature reviews on the existing power estimation and SSTA techniques are presented.

2.1 FPGA routing architecture

There is a wide variety of FPGA architectures developed by various vendors. These vendors include Actel, Altera, QuickLogic, Xilinx, and so on. Although the exact structure of these FPGAs varies from vendor to vendor, all FPGAs consist of three fundamental components needed to define a typical architecture:

1. Logic blocks capable of implementing multiple logic functions.

2. Logic blocks which support wide range of I/O signaling standard.

3. Routing resources used to realize all interconnections among the blocks.

The complexity of the logic block is classified into two types: coarse-grained and fine-grained. A coarse-grained logic block contains substantial logic structures, look-up tables (LUTs), flip-flops or programmable logic device (PLD) modules. As the complexity of the logic block increases, more functions can be implemented. The 4-input LUT is most widely employed in coarse-grained architectures [26]. In fine-grained architecture, it is made up of a large number of relatively simple logic blocks, which consists of a few basic gates, multiplexes or transistors with programmable interconnect resources. In terms of logic block and routing resource layout, FPGAs can be further classified into four main architecture groups [26].

Row-based In row-based architecture, logic blocks are arranged in rows with its routing resources separated by routing switches. The routing resources consist of mainly horizontal wire segments of various lengths and a few vertical wire segments which are used for routing between rows. (See Figure 2.1(a))

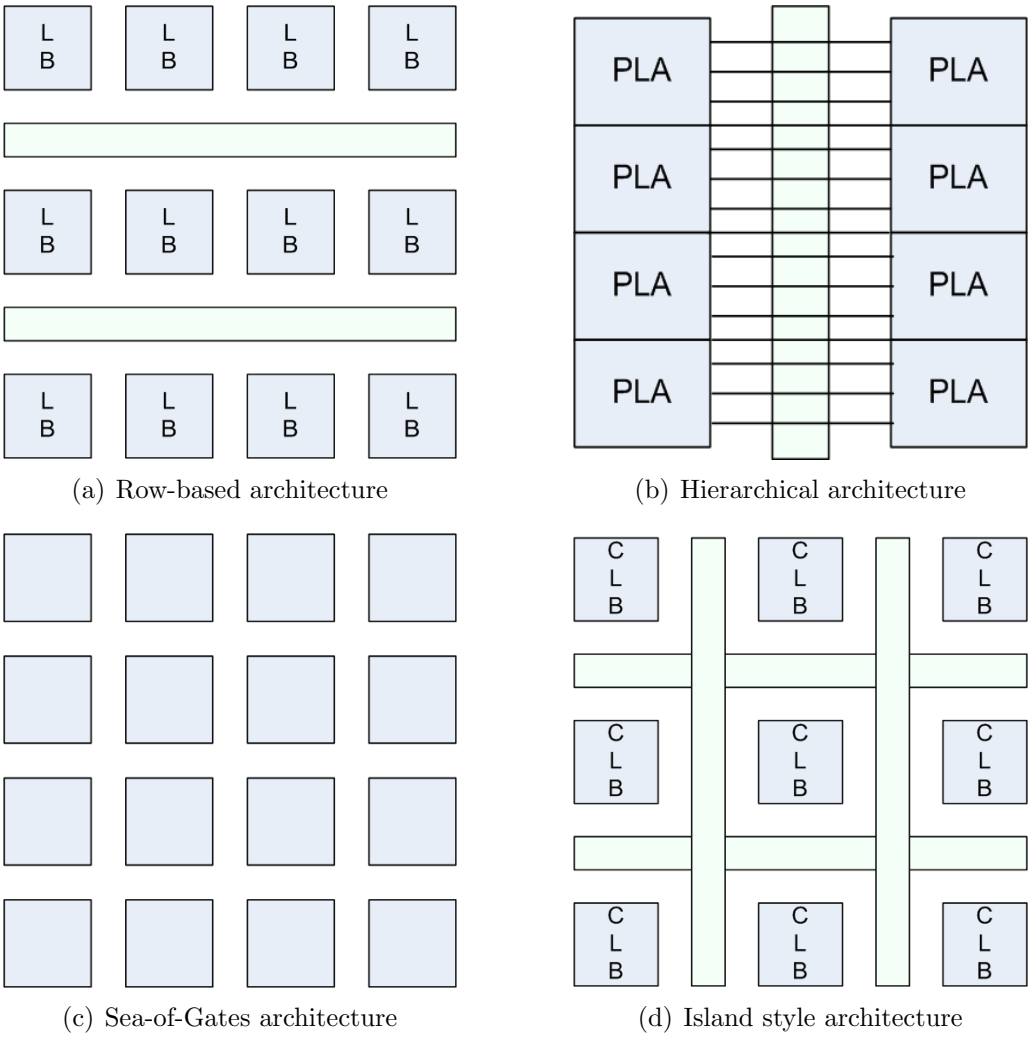
Hierarchical In hierarchical architecture, logic blocks and routing resources are displayed in a hierarchical mode. A two-dimensional array of programmable logic blocks is used to implement the multi-level logic functions. Intra-level and inter-level interconnections are used in this architecture. (See Figure 2.1(b)).

Sea-of-Gates In sea-of-gates architecture, fine-grained logic blocks are organized

in a symmetrical array manner. Routing resources are overlaid on top of these blocks. This structure resembles the architecture used in the mask programmable gate arrays (MPGAs). (See Figure 2.1(c))

Island-Style In island-style architecture, logic blocks, also known as configurable logic blocks (CLBs), are arranged in a symmetrical array with the Input/Output Blocks (IOBs) on the periphery of the chip. Routing tracks have Manhattan geometry, that is, they are either horizontal or vertical of various lengths. The CLBs are typically coarse-grained and are separated by programmable routing switches. (See Figure 2.1(d)).

Figure 2.2 shows the details of a typical island-style FPGA architecture which consists of three main routing resources: wire segments, connection block and switch box. The wire segments or routing tracks [27] are the paths taken by a signal transmitted from one source to its destinations (sinks). The length of a track may vary across the architecture and is determined by the number of CLBs it spans. A connection block connects a pin of a logic block to a specific track in the channel. The switch box [28] is a switch matrix that connects the tracks in a channel to other tracks in the adjacent channels. The connection blocks' and switch boxes' patterns may vary across the architecture.



(a) Row-based architecture

(b) Hierarchical architecture

(c) Sea-of-Gates architecture

(d) Island style architecture

Figure 2.1: Types of FPGA architecture

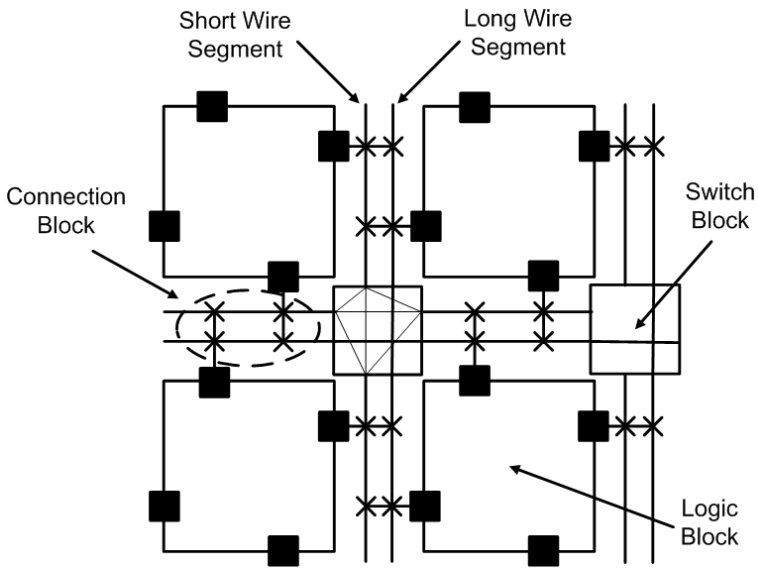


Figure 2.2: An island-style FPGA

2.2 CAD flow for FPGA design

To implement an FPGA architectural design, a series of steps is needed with each step assisted by a CAD tool. A typical design procedure employed by most commercial FPGA tools is shown in Figure 2.3.

Design Entry The description of a logic circuit can be specified using a register transfer level (RTL) description. A hardware description language (HDL) such as VHDL or Verilog can also be used. Alternatively, the circuit can be described using schematic drawing with the help of a state machine language or a schematic tool.

Synthesis & Optimization Logic synthesis does the generation of a detailed representation of the circuit with all the features required for fabrication. Optimization does the enhancement of the overall quality of the circuit in terms of performance, area and ease of testing. During synthesis, the target design, which is in terms of behavioral or logical description at the design entry level, is converted into a netlist of gates. If a schematic design is available, the logic design is already created. Using the logic design, an optimizer removes the redundant logic gates and simplifies the logic operations to minimize the set of gates used, while maintaining its functionality. This stage of the design phase is known to be technology independent as the type of elements used in the final circuit is not considered here.

Technology Mapping Once the design is generated, a technology-dependent mapping [29] tool is used to restructure the basic logic gates into k-LUT-sized groups, where k is based on the specific FPGA architecture on which the design is to be implemented. Conventional methods of technology mapping involve the use of standard cell library with pre-defined circuits. However, these methods require a large number of library cells. Hence, new algorithms for mapping are developed with the following criterion:

1. LUT number minimization
2. Routability
3. Delay Minimization

Logic Block Packing In clustered FPGA architectures, a logic block is normally made up of one or more logic elements (LE) [3]. A LE usually includes a k-LUT and a flip-flop. State-of-art architecture usually uses a 4-input LUT. The main objectives of the packing process are to combine the LUTs and latches into LEs and group the LEs into CLBs. This packing aims to maximize the number of LEs per CLB so as to minimize the number of signal connections between the CLBs [3].

VPack proposed by Betz and Rose [30], is one of the best known packing tools for clustered-based FPGAs. VPack first packs a flip flop and a LUT together into a LE using a matching based method. These LEs are then packed in a greedy manner into logic clusters by filling each cluster to its optimal capacity. In this way, the number of used inputs to each cluster is minimized.

Placement When the circuit has been reduced to a netlist which describes the connectivity between the logic blocks, a placement tool [31] is used to determine the physical location of these blocks within the target FPGA according to its physical view. During placement, parameters like overall layout size, total wire length and delay are optimized. Several placement techniques are available in the existing market. Wire-driven placement is placement which aims to optimize the routing cost. Timing-driven placement [32] is applied to reduce the length of critical path to meet timing constraints. Routability-driven placement [33] balances the wire density across the architecture. Most commercially available

placement tools uses timing-driven placement as it is more efficient in improving the speed of FPGA-based circuit as compared to wire-driven placement.

Routing Routing [34] is the process of assigning specific routing resources to each net based on the RRG to realize the connectivity between logic blocks. Routing a net corresponds to finding a path from a start node (source) to the end nodes (sinks) with the help of the RRG. The design is acceptable and workable if and only if the circuit is routable within the given resources available in the targeted architecture. Routing algorithms aim to fulfill two objectives. First, they aim to avoid congestion channels so that routing one net will not use up the routing resource that another net needs. Second, they aim to optimize propagation delay by routing critical nets with the shortest and fastest paths.

Simulation Simulation entails the analyzing of the circuit response to a set of input stimuli over a time interval. After placement and routing have been done, the implemented design is simulated to ensure its functionality. Any design errors found is corrected at this stage.

Create Bitstream File & Download to FPGA With all the previous steps being successfully completed, the bitstream files can be generated for downloading to the target FPGA architecture to implement the logic and interconnection configurations. Once the FPGA is successfully programmed, it is ready for use.

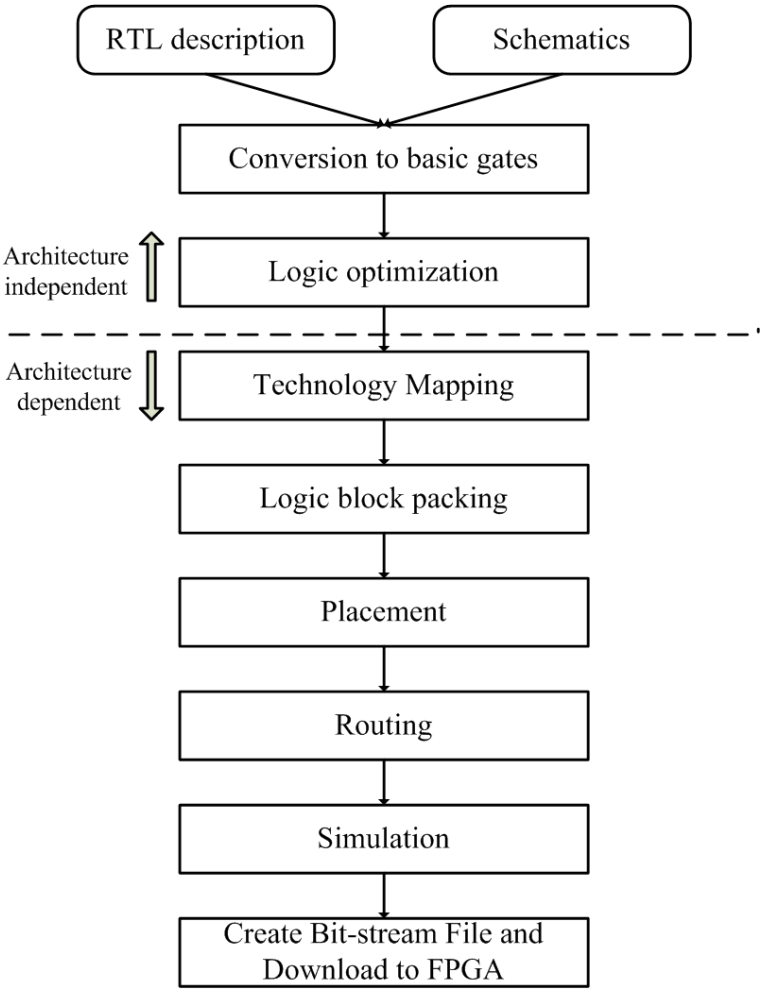


Figure 2.3: Typical FPGA CAD flow

2.3 Existing power estimation techniques

Over the years, different techniques had been explored for power efficient FPGAs to prolong battery life. Dual supply voltage schemes had been proposed to achieve lower dynamic power consumption. [35] presented a hierarchical interconnect architecture with low voltage swing signaling circuit. [36, 37] built the framework for FPGA power evaluation and analysis. [38] achieved power reduction by pre-defined dual-Vdd/dual-Vt fabrics. [39, 40] employed the configurable dual-Vdd supply to obtain a performance and power tradeoff. [41] proposed the voltage scaling scheme for commercial FPGAs. The benefit brought by dual supply voltages is obvious as the switching power is directly proportional to the square of the supply voltage. However, dual supply technique complicates the chip and system design. Either on-chip or off-chip regulators need to be provided for dual supply techniques and extra power routing is required. A huge number of configurable level converters are also needed to avoid a Vdd-Low interconnect switch from driving a Vdd-High interconnect switch. Hence, to explore new FPGA architectures like the above, a highly flexible design framework is required.

2.4 Existing SSTA techniques

As mentioned in section 1.2.1, the traditional corner-based timing analysis is unable to accurately perform timing predictions, thus SSTA is proposed to replace this

method. SSTA has the ability to capture circuit variability by modeling delays as statistical random variables and capture any possible correlation that exist between the circuit components [17]. In general, SSTA does offer fast and accurate timing predictions as compared to traditional corner-based timing analysis.

Existing SSTA approaches either assume Gaussian or non-Gaussian distributions. Others may add in consideration for correlation effects. Most of these proposed approaches are classified into two approaches: path-based SSTA [11–16] and block-based SSTA [17–21]. In path-based SSTA, it aims to provide an estimation of the circuit performance based on selected critical paths. This method is inefficient for large circuit as the worst case complexity of selecting the critical paths statistically grows exponentially with circuit size. Hence, path-based SSTA is not easily scalable to manage large circuits.

The block-based SSTA works by progressive computation. In this method, every component in the architecture is first treated as a timing block. Timing analysis is done from block to block using the timing graph in a forward manner, without ever tracking its history. Signals propagating through the timing blocks will sum up the delays into the arrival time. Delays and arrival times are called the timing variables of the circuit. Hence, the computation complexity for block-based SSTA is observed to grow linearly with circuit size.

The complexity comparison between the path-based and block-based approaches for a simple circuit is shown in Figure 2.4 [7]. From the figure, we notice that the

block-based approach shows 2.8x speedup even in such a simple case.

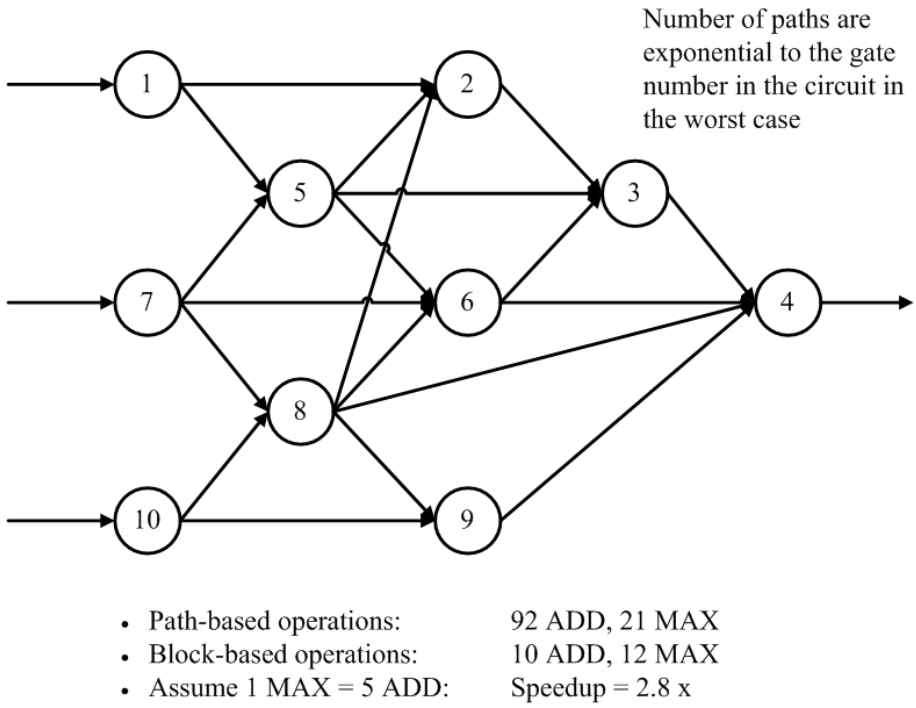


Figure 2.4: Complexity problem in path-based approach

However, to handle correlations among parameters, most approaches assume Gaussian distributions, which is not entirely the case in digital circuits. Other distributions require extensive computation, either for regression [42] or numerical integration [43]. On the whole, SSTA methods are computationally expensive and not fast enough to provide the variability-aware timing estimation during the synthesis optimizations.

Chapter 3

Modeling of the CAD Framework

As mentioned in chapter 1, existing CAD tools do not provide enough flexibility for users to design arbitrary FPGA routing architectures. We therefore need a tool to fulfill this capability.

3.1 Framework design approach

The proposed architecture that can be designed is the island-style architecture as seen in Figure 2.2. To specify this architecture, the number of CLBs and IOBs need to be defined. An approach is to ask users to key in the specific number of blocks required. However, this method does not allow users to specify the shape of the architecture. Instead, we adopt another method, that is, to allow users to specify the dimension of the architecture to achieve arbitrary shapes of the architecture to be created.

Furthermore, the layout of the island-style architecture will cause the number of IOBs to restrict the number of CLBs to be implemented or vice versa. This will result in excess blocks being declared when trying to accommodate the required number of CLBs or IOBs. Therefore, we allow users to have the flexibility to decide how many IO pads are needed at each physical location of an IOB. With the flexibility to define the dimension and number of IO pads, wastage of excess blocks declared is reduced and the dimension of the FPGA architecture is made more compacted.

In addition, the flexibility of each logic block can be further enhanced by allowing users to decide the number and the orientation of pins on the logic blocks. With this freedom to define arbitrary positions of the pins on the logic blocks, the routability and performance of the architecture can be improved.

Routing tracks play a critical role in routing performance, hence giving more freedom to define arbitrary wire segments should be allowed. However, this approach is too computational intensive. Instead, we constrain users to declare only single, double, hex and long tracks in the architecture. To enhance routability, the double and hex tracks are staggered [44]. In addition, we allow users to define the connectivity pattern of the connection block and switch box. For the connection box, users can decide how many tracks a pin can connect to in the channel. For the switch box, users can specify which tracks in the adjacent channel that a particular track can connect to.

After the basic parameters are defined, an initial architecture layout is generated.

In order for a more arbitrary architecture to be designed, we implement several pop-up interfaces for users to alter some parameters when they double-click on a specific resource of the architecture. These interfaces include the ability for users to change the number or orientation of pins on any logic blocks, change the pins' connectivity to routing tracks, change the switch pattern in the switch box or change the types of tracks in any channels.

3.2 Framework implementation approach

The proposed framework is developed in two phases. First, an interface that is used for initializing the basic parameters to create a basic routing architecture is created. Second, several additional interfaces are created which will pop-up by double-clicking on the components of the drawn architecture. These interfaces allow the editing of the corresponding clicked components. Once the design is finalized, placement and routing can commence.

3.2.1 Initializing the architecture template

To begin the design of the architecture, certain basic parameters need to be specified. An initializing interface is thus set up to allow users to key in the necessary parameters. Figure 3.1 shows the interface developed. Using this interface, it simplifies the task to describe a generic FPGA architecture. An explanation of each parameter settings is as follows.

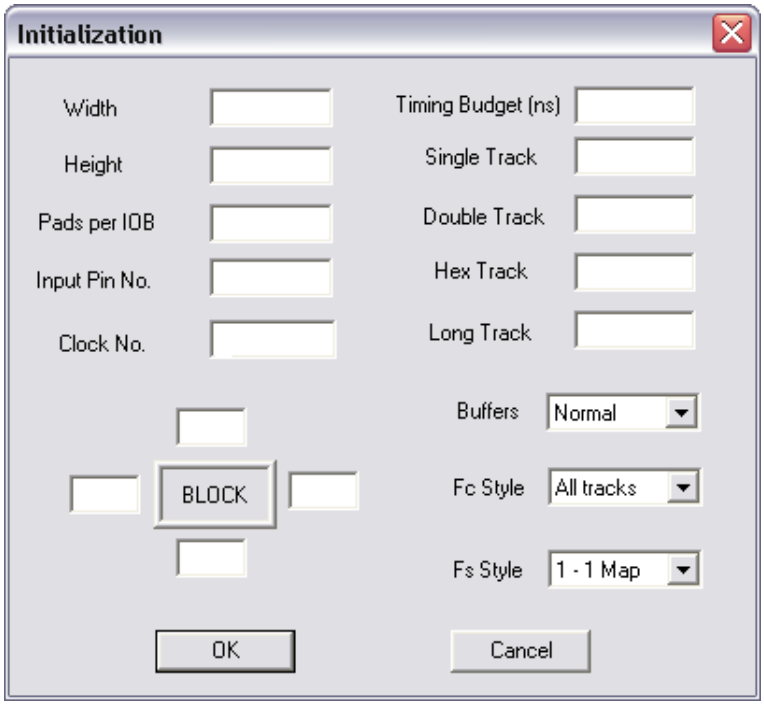


Figure 3.1: Interface for initialization

In the top left hand corner, we have five different parameters that need to be set. The first and second parameters state the width and height of the required architecture respectively. The third parameter states the required number of IO pads to be declared at a physical location of an IOB. The forth and fifth parameters indicate the number of input pins and clock pins to be declared, respectively.

Next, on the lower left corner, users can define the orientation of the pins desired by specifying the required number on each side of the logic block using the boxes provided. Using these parameters, our framework is programmed to distribute the input and output pins evenly around the entire perimeter of each logic block so as to enhance routability. This technique is known as the full-perimeter pin positioning [45]. An example of a logic block with one clock, four input and two output pins

is illustrated in Figure 3.2.

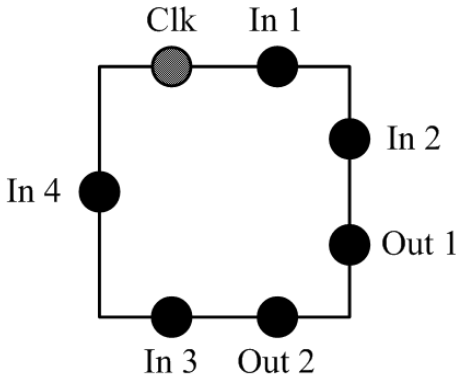


Figure 3.2: Logic block pins location

On the upper right corner, there are five parameters. First, the timing budget states the required timing constraint to be achieved for its critical path. This is explained further when we introduce the idea on reconfigurable buffers in chapter 5. The next four parameters allow different kinds of routing track to be specified by stating the preferred number corresponding to the desired tracks. Single track means that the track spans one logic block length, whereas the double and hex track spans two and six logic block lengths, respectively. The long track spans the entire width and height of the architecture.

On the lower right corner, three drop-down interfaces are shown. First, users can define the types of buffers required in the architecture, that is, normal buffers or programmable (reconfigurable) buffers. Second, we have the options to define the different types of block connectivity, F_c, needed for all the pins of the blocks. Three options on the style of the connection blocks are given. (See Figure 3.3)

1. Users can choose to have the pins connected to all available tracks.

2. Users can choose to have the pins connected to alternate tracks.

3. Users can choose to have the pins connected to 50% of the available tracks randomly.

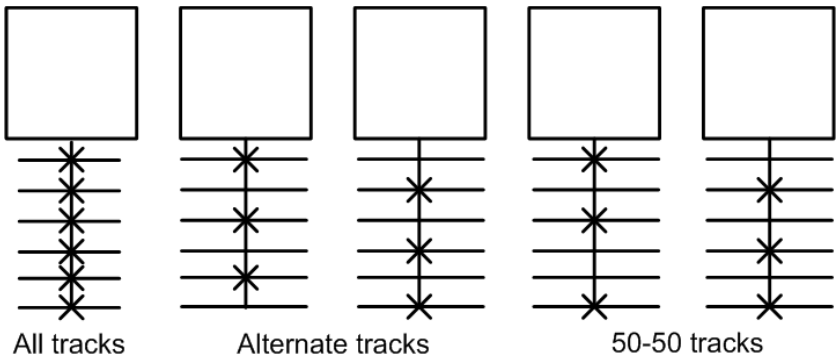


Figure 3.3: Types of connection block connectivity

For options 2 and 3, we ensure that the connection block pattern is pathologically good by fulfilling the following conditions:

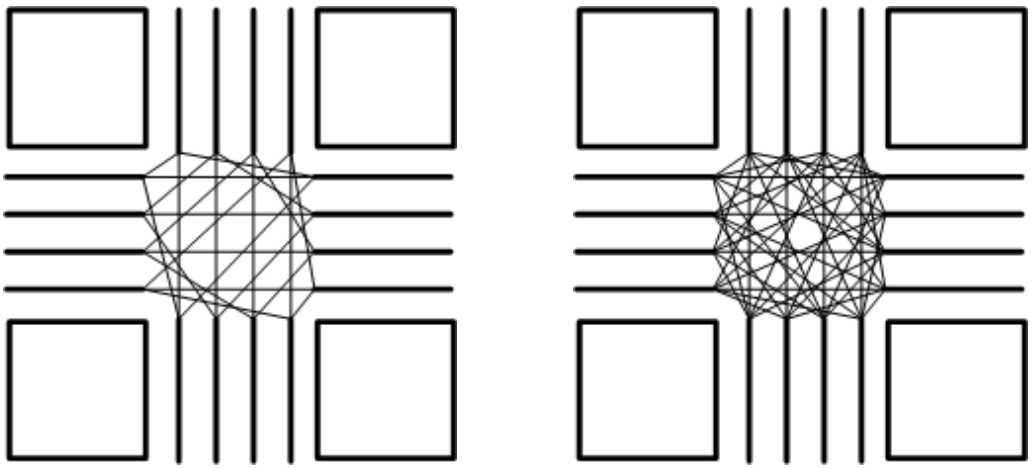
- Ensure each pin is connected to different wire types (if any).
- Ensure multiple pins on each side connect to alternate tracks.
- Ensure pins sharing the same channel share at least one common track.
- Ensure pins sharing the same channel connect to different tracks.

Third is the menu to define the switch connectivity, F_s, of the switch boxes. In this framework, all switch boxes defined consist of only buffers and no pass transistors. Users are given three options in this selection menu:

1. Tracks are connected to all tracks in the adjacent channels.

2. Tracks are connected to one track of each of the adjacent channels. (Figure 3.4(a))

3. Tracks are connected to alternate tracks of each of the adjacent channels. (Figure 3.4(b))



(a) 1-1 Connections (b) Alternate connections

Figure 3.4: Types of switch block connectivity

After the initial settings are done, the resulting FPGA routing architecture is drawn. Figure 3.5 shows the design of a FPGA routing architecture of dimension 3 x 2.

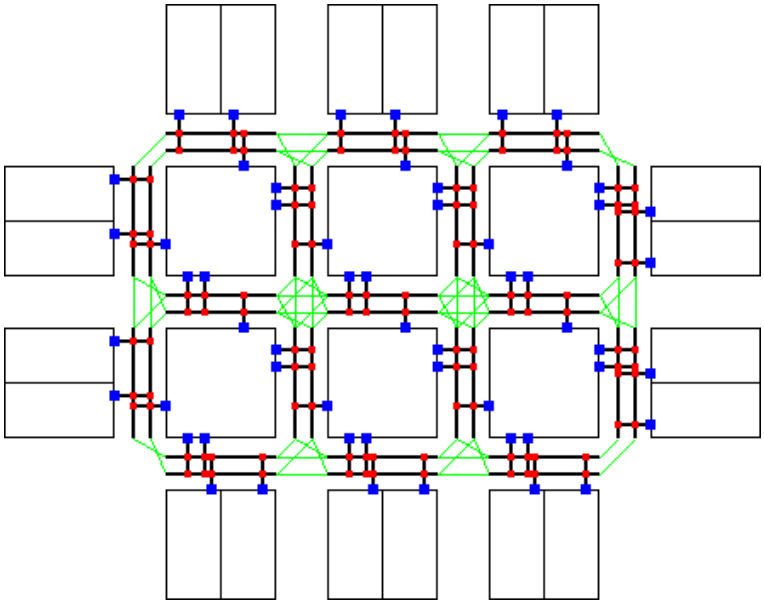


Figure 3.5: FPGA routing architecture template

3.2.2 Editing the architecture template

After the architecture is drawn, users can click any of the template components: logic blocks, routing channels, switch boxes or pins, to edit the original settings and create a customized routing architecture. Four different interfaces are illustrated next.

First, an interface to edit the logic blocks is described. Upon clicking on any of the CLBs, an interface is shown (See Figure 3.6). In this interface, users can re-initialize the number or orientation of pins of the selected block by keying a new set of data in the boxes provided. Three options are provided to allow users to specify whether to apply the changes made to the current CLB, all CLBs or to alternate CLBs.

Second, an interface to edit the routing channels is shown. Users can select any of the vertical or horizontal channel by clicking on it. Using the pop-up interface seen in Figure 3.7, users can re-specify the kind of tracks desired for the selected channel. This

can be done by adding in new values in the corresponding spaces provided. Also, users can specify whether to apply these changes to all vertical and/or horizontal channels by checking the required boxes.

Third, an interface to edit the pin's connectivity is illustrated. By clicking on a pin of any logic blocks, an interface is produced as shown in Figure 3.8. In this interface, it allows users to change the connectivity pattern of the selected pin to the routing tracks it is facing. Four options are provided for users to indicate how the changes can be applied to. The four options are: all pins of current block, all pins of all similar blocks, alternate pins of current block or alternate pins of all similar blocks.

Forth, an interface to edit the switch box connectivity is presented. Users can click on any of the switch box to reconfigure its switch pattern. Figure 3.9 shows the interface when a switch box is selected. Three options are given for users to choose whether to apply the change made to the current switch box, all switch boxes or alternate switch boxes. Having finalized the architecture, users can proceed to do placement and routing to test the feasibility of their new design.

3.2.3 CAD tool interface

In this section, we introduce the features that our framework is able to perform. Figure 3.10 shows the initial interface when the program is run. A description of the various menu bar options is listed in Table 3.1.

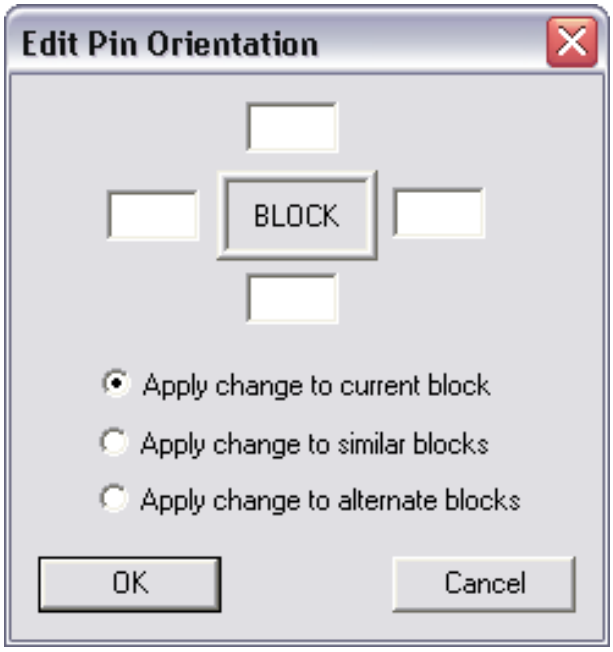


Figure 3.6: Edit CLB's pin orientation

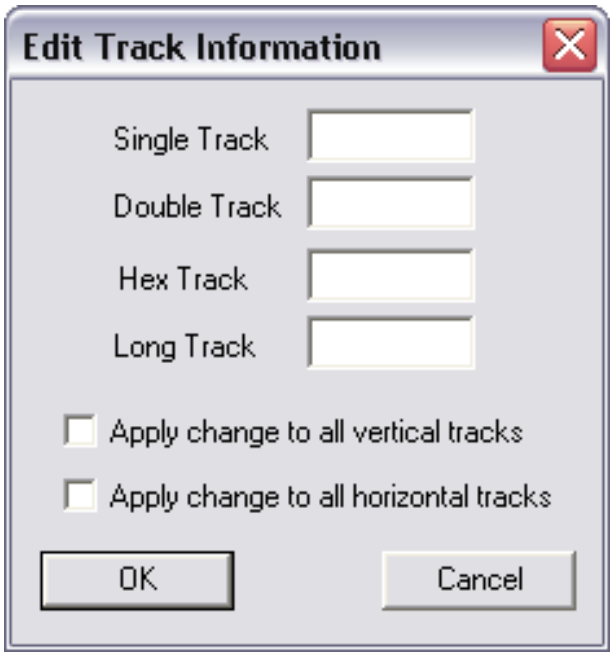


Figure 3.7: Edit track information

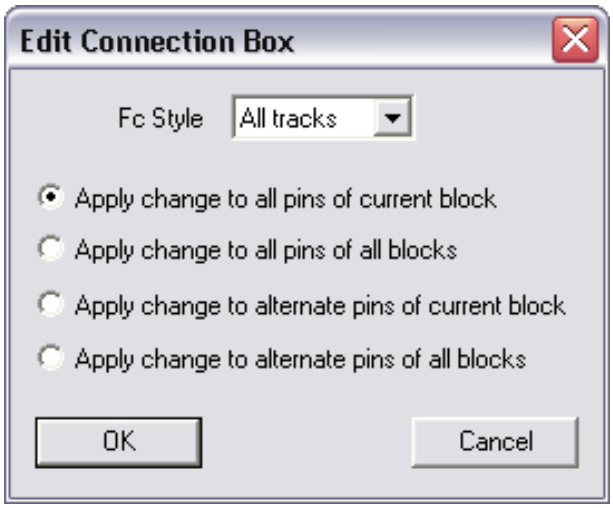


Figure 3.8: Edit connection box

Figure 3.9: Edit switch box connectivity

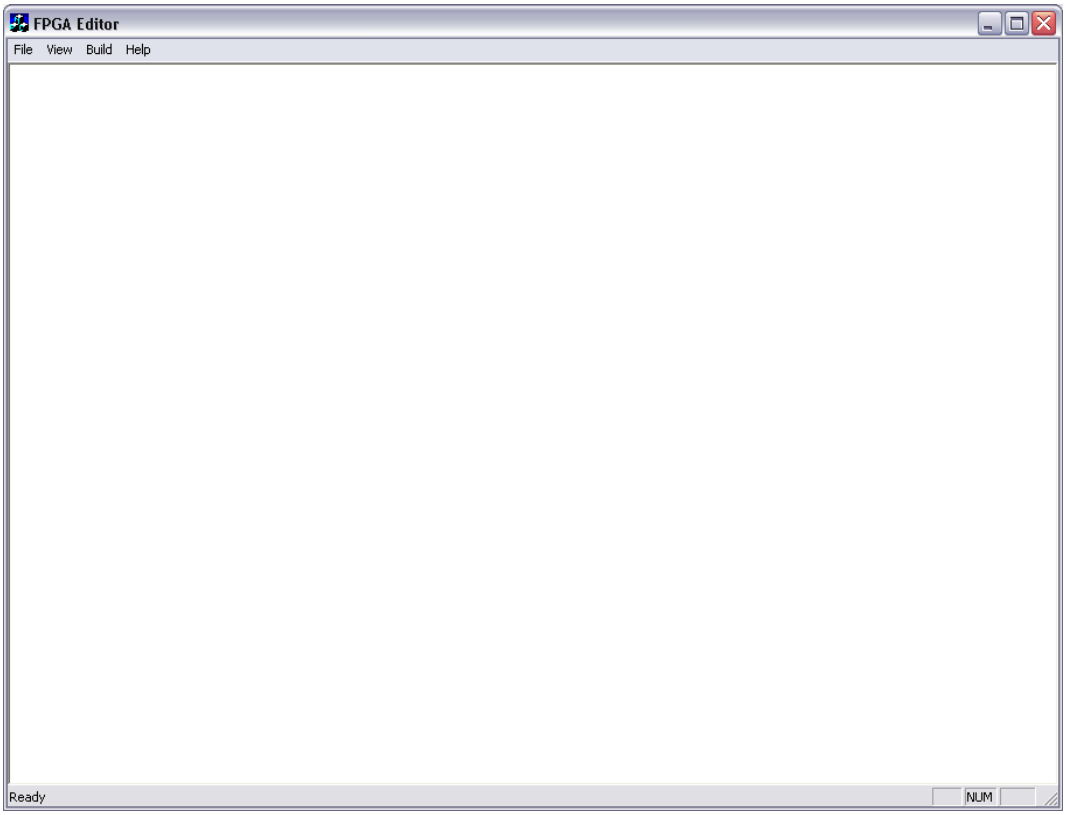


Figure 3.10: Program interface

Menu bar	Options	Descriptions
File	New	Initialize new architecture
	Exit	Quit the application; prompts to save documents
View	No display	Enable/Disable graphics drawing
	Toggle switch	Toggle the switch box visible/invisible
	Status bar	Show or hide the status bar
Build	Read Place File	Read existing placement file
	Read Net File	Read netlist file
	Create RRG	Create Routing Resource Graph for VPR
	Placement	Commence Placement
	Routing	Commence Global/Detailed Routing
Help	About Editor	Display program information

Table 3.1: Menu bar Options and descriptions

3.3 Routing resource graph

After the FPGA routing architecture is finalized, an RRG is generated to specify the resources in it. The RRG is essential as it contains all connectivity information such as wires to which a given wire segment can connect and is used by the router to make routing decisions.

The RRG [46] is described by a directed graph, *G=(V,E)*, where the set of nodes V corresponds to the functional block pins or wires in the routing architecture and the set of edges E corresponds to the switches that connect these nodes. In addition, the type of source and sink nodes are added to model the logically equivalent output and input pins, respectively. This RRG can be either manually generated once the routing architecture is being created or it will be automatically generated before routing commences. The generated RRG can be ported to VPR too. Figure 3.11 shows the RRG corresponding to a portion of the architecture whose functional block has 4 input pins and 1 output pin with 1 wire segment in both the vertical and horizontal channels.

3.4 Placement and routing processes

Placement and routing are two mutually dependent processes. Placement is the process of assigning functional blocks to their physical locations in the architecture. In the island style FPGA architecture, the CLBs are arranged in a two-dimensional

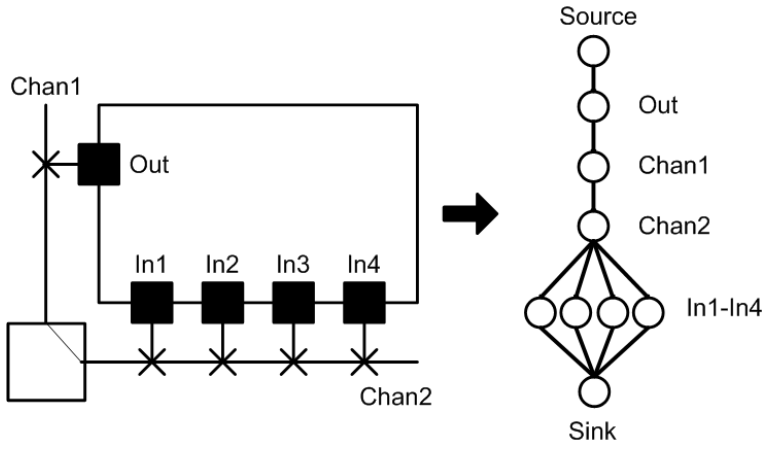


Figure 3.11: Modeling FPGA routing as a directed graph

array with the IOBs on the periphery of the chip. Hence, placement is done using a coordinate system to track both x and y directions. Routing is the process of finding a connected path from a source to its sinks using the available routing resources. The RRG is used to enable fast retrieval of the inter-connections of the various components. In typical CAD tools, placement and routing are carried out as two independent phases. However, there are still cases where both are carried out simultaneously.

In our framework, placement is carried out followed by routing. Global and detailed routings are done in the routing phase. The RRG is used by the router to locate physical paths available for each net. In global routing, all nets are routed without any constraints. Detailed routing is done to establish physical connections between the logic blocks by assigning distinct paths to individual nets, while considering constraints like congestion, signal performance, and resource utilization.

3.4.1 Placement process

In FPGA devices, there have been different placement techniques being adopted to bind logic elements to a physical unit on the device. Different algorithms have also been developed to suit the different types of architectures. All of these algorithms have the ability to handle constraints and cost minimization. Typical placement constraint includes user's constraints to fix certain logic blocks at a particular location. On the other hand, cost minimization includes the minimization of matrices like total wire lengths, usage of routing resources, etc and also to reduce congestion to enhance routability. Despite these constraints, all placement algorithms aim to present a good placement for the router to complete its routing efficiently.

Placement algorithm

In our framework, the simulated annealing placement algorithm [3, 47] is implemented. Its pseudo-code is shown in Figure 3.12. Initially, each block is randomly assigned to an available vacancy in the target FPGA architecture to generate an initial placement solution. An initial temperature, T, is computed in a similar manner to [48]. A range limiter R_{limit} is introduced to allow blocks with that range to be considered for swapping. Initially, R_{limit} is set to the entire chip.

Next, pair-wise swapping is done and evaluated against a temperature schedule [30]. A temperature schedule is computed as $T_{new} = \alpha\ T_{old}$, where α depends on the fraction of attempted moves that were accepted (R_{accept}) at T_{old} as shown

in Table 3.2. At each temperature, there is a total of $10*(N_{blocks})^{1.33}$ swaps. The placement cost which determines the acceptance of a swap is determined by a cost function:

$$cost = \sum_{n}^{N_{nets}} q(n) * [bb_x(n) + bb_y(n)] \tag{3.1}$$

```
1   S = Init_Placement();
2   T = Init_Temperature();
3   R_limit = Init_Range();
4   while(Exit_Criterion() == false) /* outer loop */
5   {   while(Inner_Loop_Criterion() == false) /* inner loop */
6     {   S_candidate = Generate_Move(S_current, R_limit);
7         ΔC = Cost(S_candidate) - Cost(S_current);
8         /* calculate the changes in cost */
9         ran = random(0, 1);
10        /* create random float number between (0, 1) */
11        if(ran < e^(ΔC/T))
12           S_current = S_candidate;
13    }
14    Update(T);
15    Update(R_limit);
16  } /* end of outer loop */
17  /* get final placement S */
```

Figure 3.12: Pseudo-code for the simulated-annealing algorithm used in the placement step

Fraction of moves accepted (R_{accept})	α
$R_{accept} > 0.96$	0.5
$0.80 < R_{accept} \leq 0.96$	0.9
$0.15 < R_{accept} \leq 0.80$	0.95
$R_{accept} \leq 0.15$	0.8

Table 3.2: Temperature update schedule

For each net, bb_x and bb_y are defined as the horizontal and vertical spans of its bounding box. This is commonly known as the half-perimeter model (Figure 3.13)

which is widely used to estimate the wire-length of a net [49]. Given a block b with coordinates (x_b, y_b), the half-perimeter of a net n is calculated as follows:

$$half-perimeter = [MAX_{b\epsilon n}(x_b) - MIN_{b\epsilon n}(x_b) + 1] + [MAX_{b\epsilon n}(y_b) - MIN_{b\epsilon n}(y_b) + 1] \tag{3.2}$$

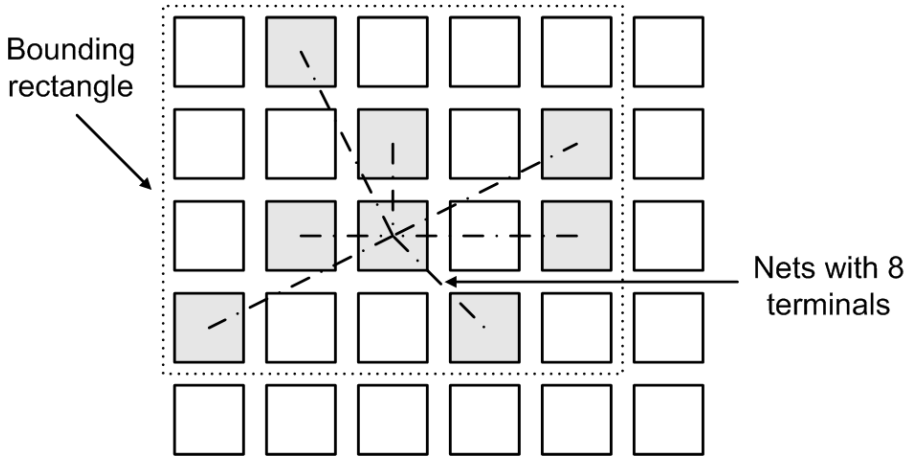


Figure 3.13: Half-perimeter wavelength model

A *q(n)* factor [50] is introduced to compensate for any underestimation of the wire-length by this model. This value depends on the number of sinks a net has to connect. For nets with less than 3 terminals, *q(n)* is set to 1 and slowly increases to 2.79 for nets with 50 terminals. For large fan-out nets having more than 50 sinks, the value of *q(n)* increases linearly as follows [30]:

$$q(n) = 2.7933 + 0.02626 * (TerminalNumber - 50) \tag{3.3}$$

An illustration showing the swapping of two CLBs during placement is shown in Figure 3.14. Finally, the program will terminate when $T <$ *0.005 * Cost /* N_{nets}. A placement file is then generated which shows the physical locations of all the logic

blocks. The format of this file is discussed in the following section.

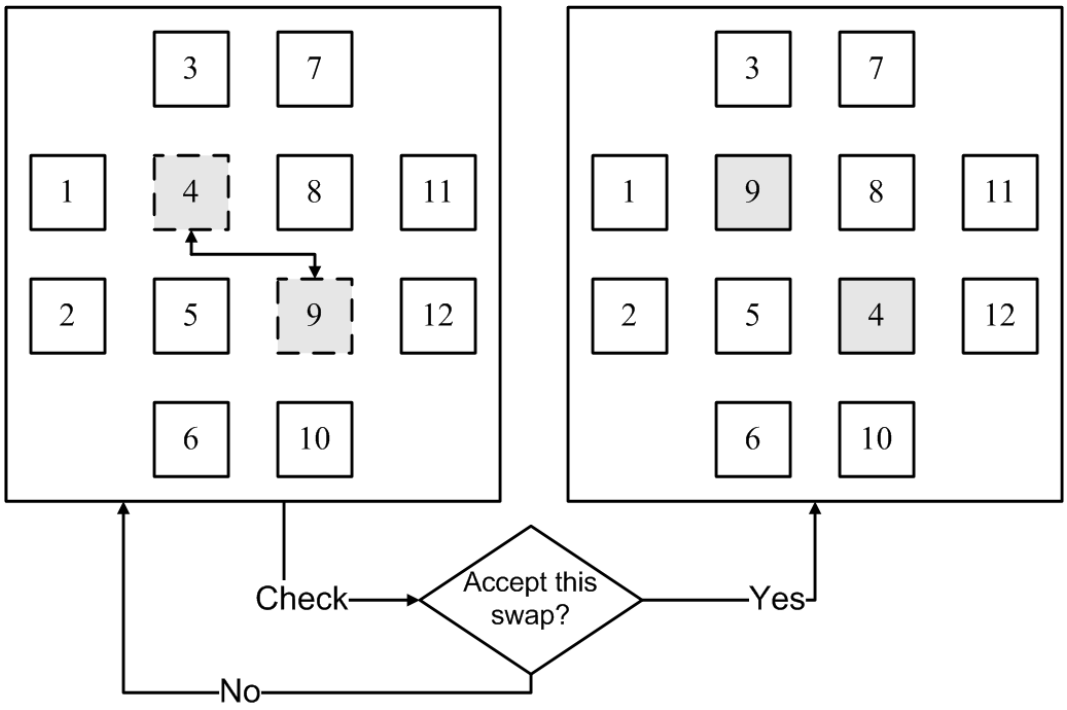


Figure 3.14: Swapping between two logic blocks

Placement file format

After placement has been completed, a placement file is generated as shown in Figure 3.15. The first line of the placement file lists the netlist used during the placement. The second line of the file gives the size of the logic block array as per designed. All the following lines have the following format:

block name x y *sub-block* *physical location*

The block name is the name of this block, as given in the input netlist file. x and y are the row and column coordinates in which the block is placed, respectively. The sub-block is applicable to IOBs. As we can have different number of pads on the

IOB, this sub-block number specify the pad position given the x and y coordinates. For CLBs, the sub-block number is always zero. The physical location indicates the physical placement of the block name on the architecture. Figure 3.16 shows the coordinate system used via a 2 x 2 FPGA architecture.

Netlist file: sample.net
Array size: 2 x 2 logic blocks

#block name	x	y	subblk	block number
#----------	-	-	------	----------
i_31_	0	1	0	#1
i_56_	0	1	1	#2
i_47_	2	3	0	#3
i_34_	0	2	0	#4
out: o_9_	0	2	1	#5
o_1_	2	1	0	#6
o_9_	1	2	0	#7
o_2_	1	1	0	#8
[93]	2	2	0	#9

Figure 3.15: Sample placement file

3.4.2 Routing process

In most FPGA devices, routing is carried out in two steps: global routing and detailed routing. In global routing, the router assigns paths that a net can transverse from its source to its destination(s). Its objective is to find the shortest possible connection path with the least usage of routing elements and avoid congestion. Once global routing is done, the regions or routing areas where congestion are likely to occur are identified. Detailed routing then commences by assigning specific tracks to the nets within the given routing regions and avoiding the identified congestion

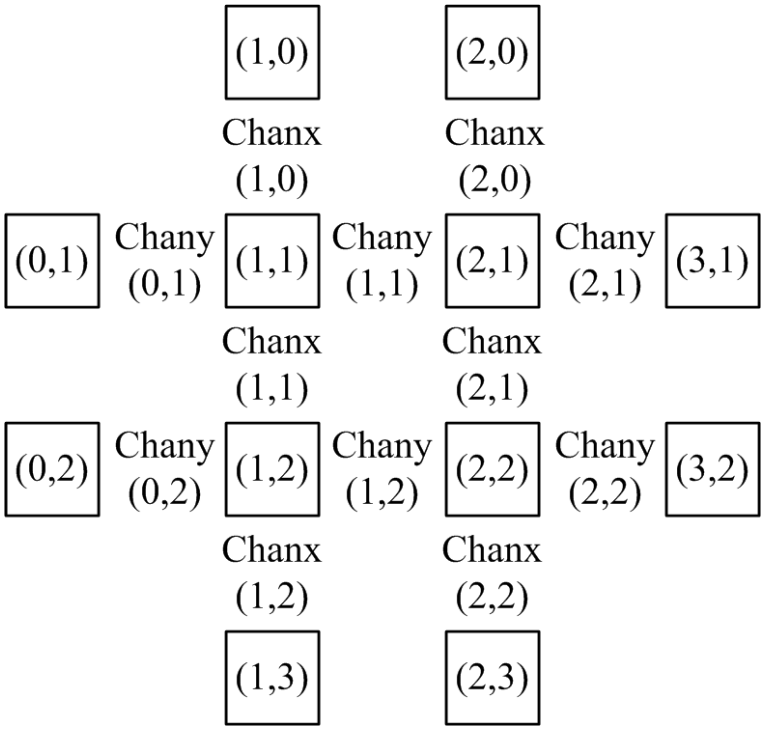


Figure 3.16: Coordinate system used

regions where possible.

At present, there are several routing algorithms. These algorithms aim to satisfy two criterions. First, they aim to minimize congestion cost such that each net will not use up the routing resource that another net needs. These algorithms are known to be routability-driven. An example is the Pathfinder negotiated congestion algorithm [45] used in VPR. This algorithm aims to minimize the propagation delay by routing critical nets first with the shortest path while minimizing the use of routing resources. Second, they aim to minimize the critical path delay. Such algorithms are called timing-driven. One example is the maze-routing algorithm based on Dijkstra's shortest path algorithm [51].

Routing algorithm

In our framework, the pathfinder negotiated congestion algorithm is implemented as the routing algorithm. Its pseudo-code is shown in Figure 3.17 [45, 46]. Global routing is first carried out followed by detailed routing. First, the router takes in the starting point and computes the shortest path to the destination using breath-first search. An evaluation cost function is formulated to measure the cost of a route from its source to its targeted sink(s) [45, 46]:

$$cost = (1 + h_n.h_{fac}) * (1 + p_n.p_{fac}) + b_{n,n-1} \tag{3.4}$$

The p_n term measures the current congestion at a node. It is updated every time any net is being ripped up and rerouted. h_n refers to the past congestion at the current node. It is updated after each iteration. Initially, h_n is 0 and is increased after each routing iteration. The $b_{n,n-1}$ terms penalizes bends. Two controlling factors, h_{fac} and p_{fac}, are used to force nets with alternative routes to avoid using congested paths.

Some modifications are done to the breadth-first search algorithm to improve its efficiency [30]. The default algorithm is time-consuming as it requires k-1 iteration to find the paths of k terminals of the net as after each path is found, its wave-front is emptied and a new wave-front is formed for the subsequent sink. To remedy this, all the terminals of each nets are arranged in terms of Manhattan distance from their source, starting from the nearest to the furthest, prior to routing. After each successful routing, the wave-front is not emptied but stored in a sorted order. When a new iteration begins, the router will search through this set of data to find the

```
1    While shared resources exist (global router)
2      Loop over all signals i (signal router)
3        Rip up routing tree RT_i
4        RT_i ← s_i
5        Loop until all sinks t_ij have been found
6          Initialize priority queue PQ to RT_i at cost 0
7          Loop until new t_ij is found
8            Delete lowest cost node m from PQ
9            Loop over fanouts n of node m
10             Add n to PQ at cost c_n + P_im
11           End
12         End
13         Loop over nodes n in path t_ij to s_i (backtrace)
14           Update c_n
15           Add n to RT_i
16         End
17       End
18     End
19   End
```

Figure 3.17: Pseudo-code for the Pathfinder negotiated congestion algorithm used in the routing step

nearest starting point and expand out to its designated sink. In this way, the router takes a shorter amount of time to reach to its next sinks then having to expand its wave-front from scratch.

Moreover, instead of allowing the algorithm to go through all the adjacent vertexes of the map and to avoid overly circuitous routes, we reduce the search by the router to at most 3 channels outside the boundary of the source pin and the sink pin. Once routing is successfully completed, a routing file is generated.

Routing file format

The first line of the routing file [3] gives the FPGA architecture size. The remainder of the routing file lists the detailed routing for each net. Each routing net begins with the word *net* and its count, followed by the name of the net given in the netlist file in brackets. The following lines define the routing of the net. Each line begins with a keyword that identifies a type of routing segment.

The keywords used are *SOURCE* (output pin from which the net starts), *SINK* (input pin where the net terminates), *OPIN* (output pin), *IPIN* (input pin), *CHANX* (horizontal channel) and *CHANY* (vertical channel). Each routing net begins with a *SOURCE* and ends in a *SINK*. In brackets after the keyword is the coordinates of this routing resource based on Figure 3.16.

Next, the pad number (if the *SOURCE*, *SINK*, *IPIN* or *OPIN* is on an IOB), pin number (if the *IPIN* or *OPIN* was on a CLB), class number (if the *SOURCE* or *SINK* was on a CLB) or track number (for *CHANX* or *CHANY*) is listed. The pad or pin number indicates the pin reference number given to each pin of each logic block. Class number '0' indicates the CLB is a *SINK* and '1' indicates the CLB is a *SOURCE*. As for the track number, track 1 is the topmost track of *CHANX*, while in *CHANY* track 1 is the leftmost track.

For an N-pin net, we need N-1 distinct wiring paths to connect all the pins. The first wiring path goes from a *SOURCE* to a *SINK*. The routing segment listed immediately after the *SINK* is the part of the existing routing to which the new path

attaches. An example of the routing file for a net is listed in Figure 3.18.

```
Net 1 (i_31_)

SOURCE   (1,0)   Pad: 1      # This source is an input pad at (1,0)
  OPIN   (1,0)   Pad: 1
 CHANX   (1,0)   Track: 1
 CHANY   (1,1)   Track: 2
  IPIN   (2,1)   Pin: 5
  SINK   (2,1)   Class: 0    # Sink for pins of class 0 on a CLB
 CHANX   (1,0)   Track: 1    # Note: Connection to existing routing
  IPIN   (1,1)   Pin: 1
  SINK   (1,1)   Class: 0
```

Figure 3.18: Sample route file

Chapter 4

Framework Experimental Results and Analysis

In this chapter, three experiments have been set up to show the capabilities and qualities of our design framework. The first experiment shows a sample of an arbitrary FPGA architecture which our framework can design. The second experiment shows the placement, routing and user interface that the framework can support to develop heterogeneous FPGA architectures. The third experiment compares our framework to VPR and demonstrates that our framework can further optimize an architecture for better performance.

4.1 Display of generic FPGA architecture

Figure 4.1 shows a sample of a generic FPGA routing architecture. It shows the entire architecture of size 6 x 6, with IOBs surrounding the CLBs and each channel is of width four, which consist of one of the four kinds of the routing tracks. Each IOB location is assigned two pads while each CLB has the pin configuration as shown in Figure 3.2. Figure 4.2 shows the same FPGA routing architecture with the routing switches not displayed. Notice that the "start points" of the longer tracks (double, hex and long) are staggered [44] to enhance routability.

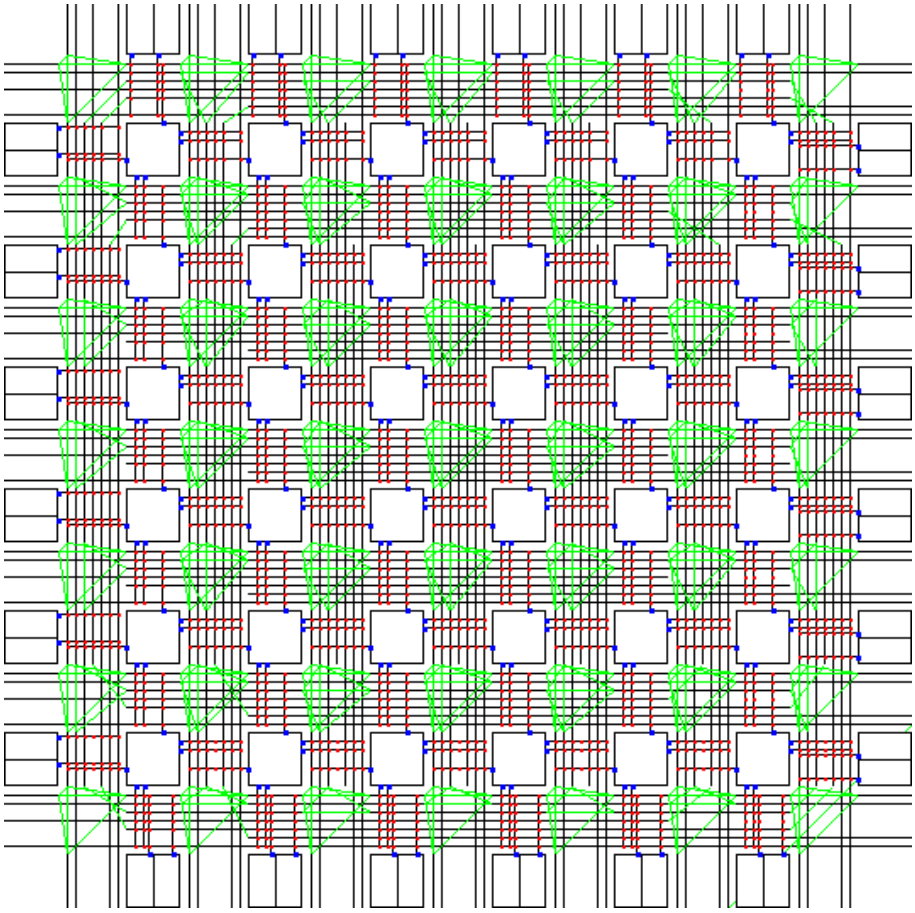


Figure 4.1: Graphical view of a sample of FPGA routing architecture

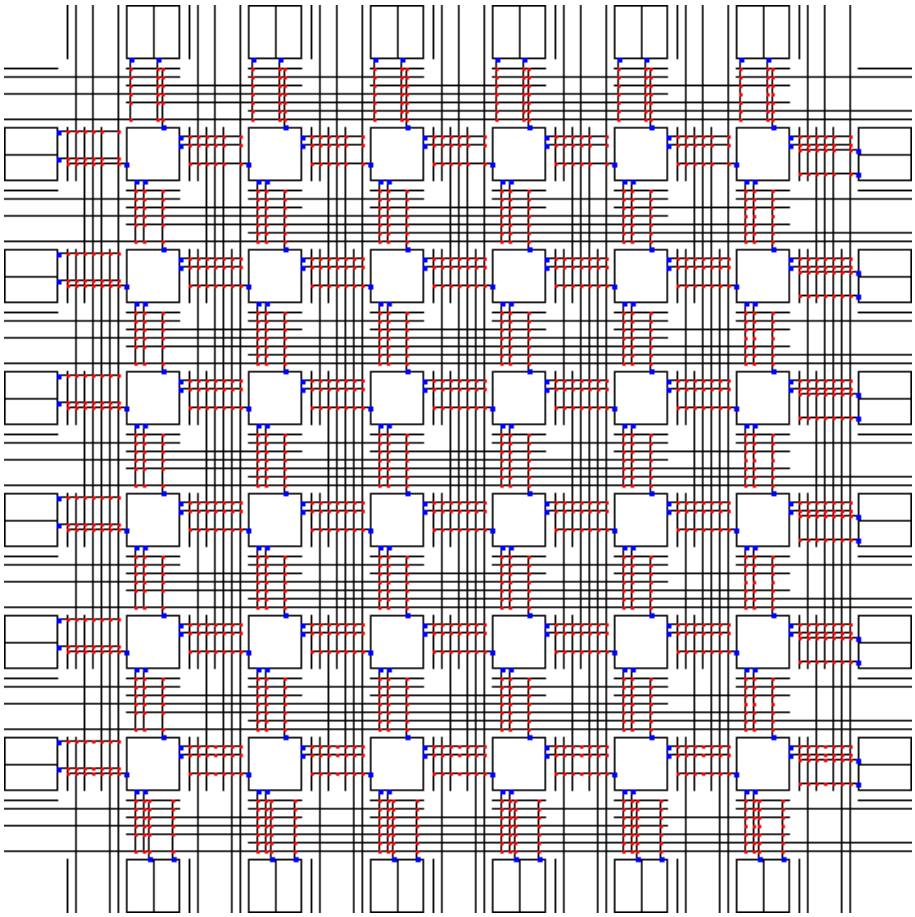
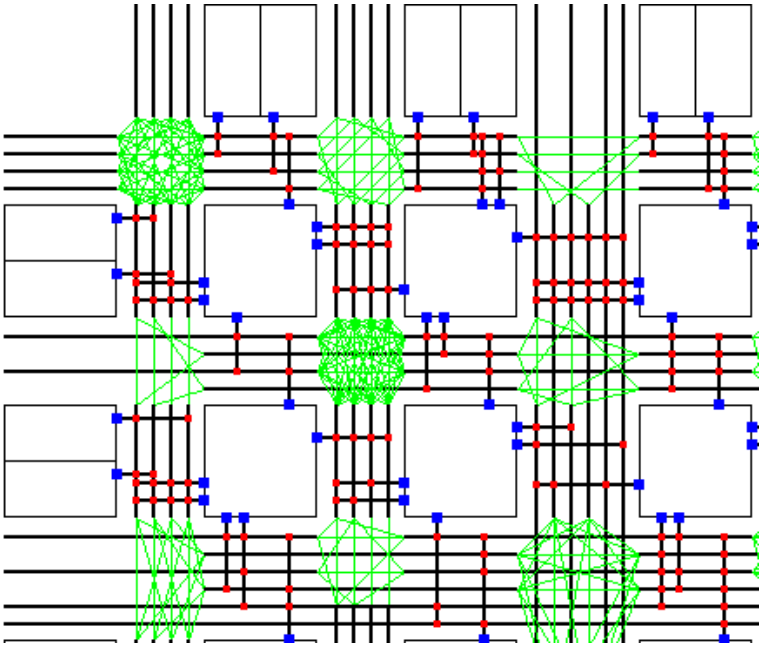


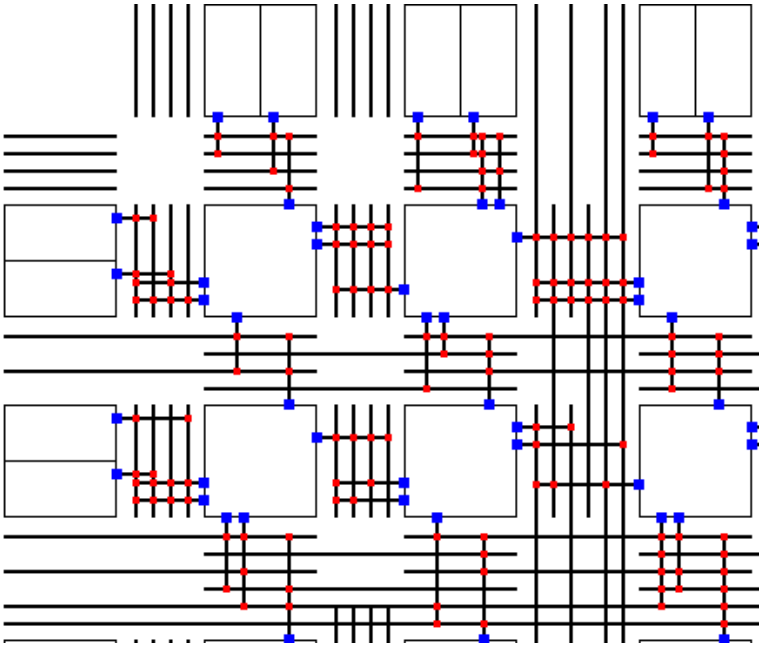
Figure 4.2: Segmentation view of a sample of FPGA routing architecture

4.2 Display of edited FPGA architecture

Figure 4.3 shows an edited architecture from the first experiment. It features different channel widths, connectivity and switch box patterns. Notice that the changes applied have made the architecture to be non-uniformed, which are not possible using present tools.



(a) Edited FPGA architecture



(b) Edited FPGA architecture without routing switches

Figure 4.3: An edited FPGA architecture with heterogeneity

4.3 Display of architecture after placement and routing

A sample designed architecture from the second experiment after placement and routing is shown in Figure 4.4. Notice that there are some blocks in white. These are the unused blocks. Next, to show the connectivity of each block, users can click on the desired block. The selected block with all its nets will be highlighted as shown in Figure 4.5. The selected block is in grey. Its fan-out blocks are indicated using cross-stitches and its fan-in blocks are indicated with diagonal lines.

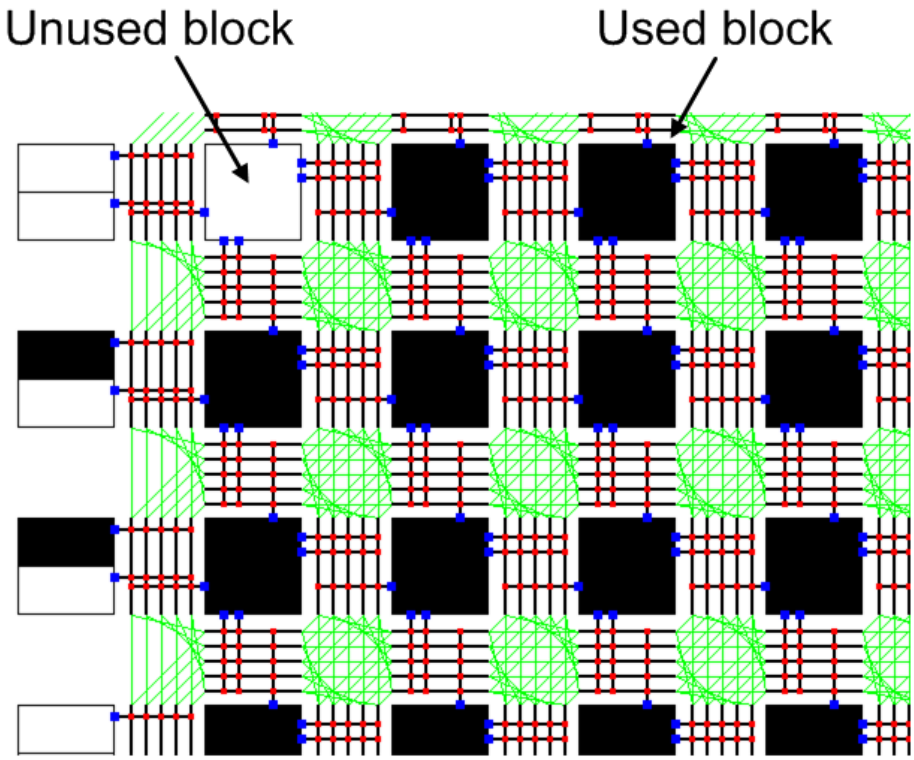


Figure 4.4: An architecture after placement and routing

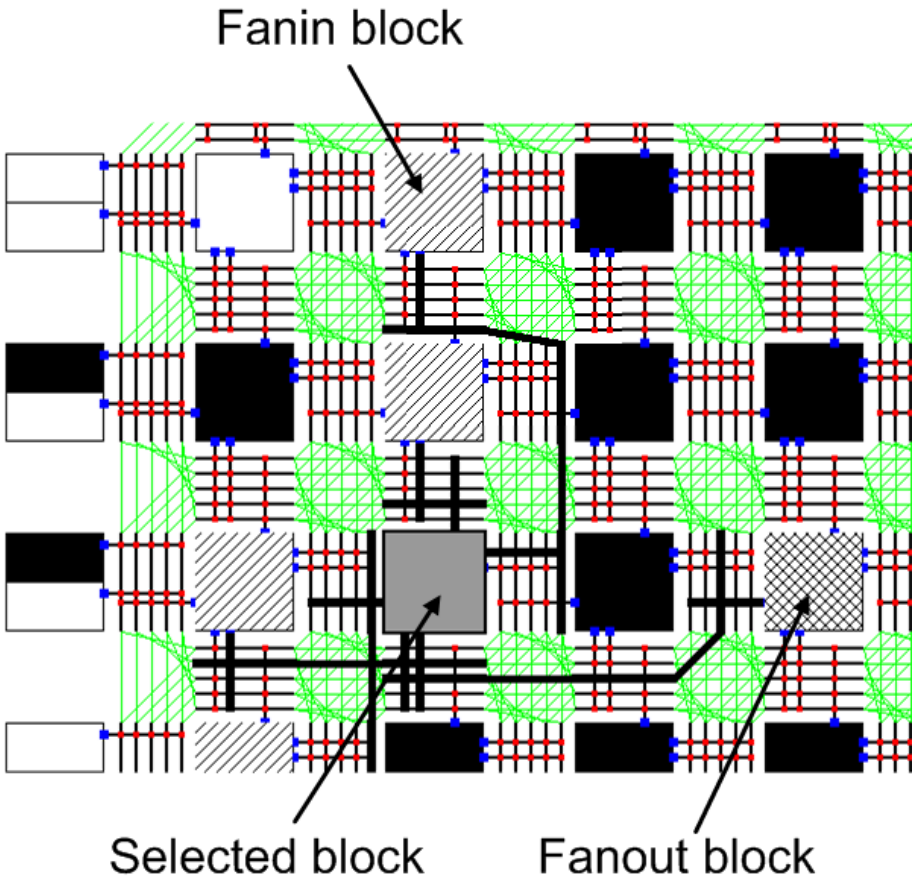


Figure 4.5: A selected CLB with its connectivity

4.4 Placement and routing results

We have performed a third experiment to compare the quality of our framework with VPR [30]. Table 4.1 shows the comparison of the minimum number of tracks per channel for a successful routing between our framework and VPR on a set of 20 MCNC benchmark circuits. All the results shown were obtained with a logic block consisting of a 4-input LUT and a flip flop. Each logic block's pin can be connected to all tracks in the adjacent channel(s) (F_c = W). Each wire segment is of length one and can connect to one wire segment of each of the adjacent channels through

disjoint switch boxes ($F_s = 3$). Each benchmark is routed with a maximum of 100 iterations. If circuit has not successfully routed in the given number of tracks in 100 iterations, it is assumed to be unroutable with channels of that width. When running the experiment using VPR, the placement cost function is set to use the bounding-box calculation and the router uses the breadth-first search algorithm.

Circuits	VPR	GUI	Circuits	VPR	GUI
alu4	9	9	bigkey	6	6
apex2	10	10	clma	10	11
apex4	11	11	diffeq	7	7
des	7	7	dsip	5	6
ex5p	11	12	elliptic	9	9
ex1010	9	10	frisc	11	11
misex3	10	10	s298	6	7
pdc	15	15	s38417	6	7
seq	10	10	s38584.1	7	8
spla	12	12	tseng	6	6

Table 4.1: Minimum channel widths required to place and route 20 large benchmark circuits

As shown in Table 4.1, it has proven that our framework is capable of performing reasonably good placement and routing that is comparable to that of the current state-of-the-art VPR. From the results, out of the 20 benchmark circuits, 13 of them are able to route within the same minimum channel width compared to VPR. 7 of the benchmark circuits (clma, dsip, ex5p, ex1010, s298, s38417, s38584.1) manage to route with an addition channel. Though both experiments are carried out using the same algorithm, the results differ due to that the placement algorithm only guarantee a local minima instead a global minima solution. With a random initial placement, it leads to a different placement solution and hence, a different routing solution.

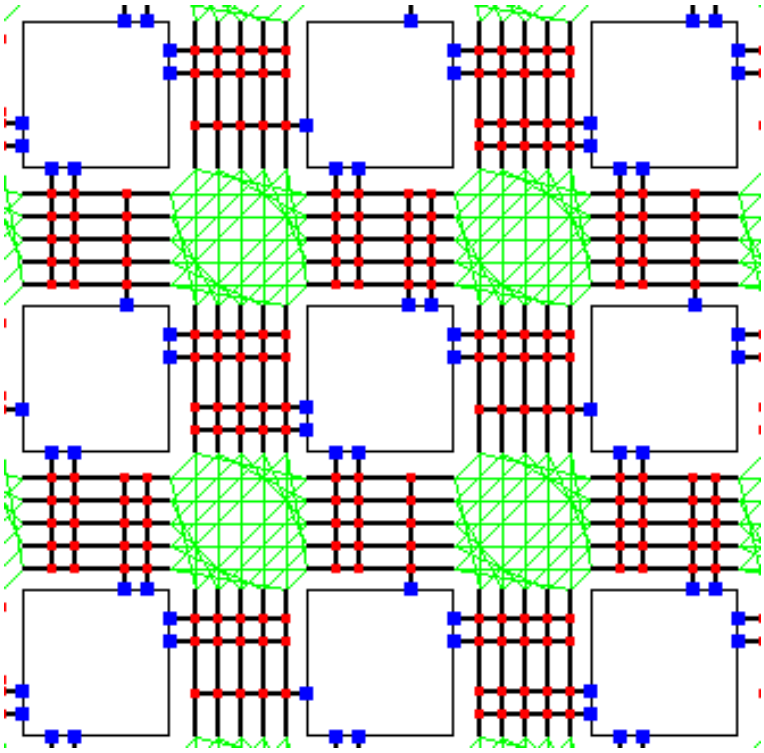


Figure 4.6: A modified FPGA routing architecture template

Next, we perform another experiment with the exact same settings but with a modified architecture using our framework. In this architecture, we change the orientation of the pins of all the alternate logic blocks as seen in Fig. 4.6. In Table 4.2, it shows the minimum number of tracks per channel to route the 20 MCNC benchmark circuits using the modified architecture.

From the results shown, we had obtained a better minimum channel width in 7 of the benchmark circuits comparing to our previous experiment. But in comparison to VPR, we obtained the same total track count number in all benchmark circuits. This is partially due to having more options being opened up for routability with a heterogenous architecture. Hence, this proves that with our framework, we can

further optimized an architecture to obtain better results compared to what VPR can do.

Circuits	VPR	GUI	Circuits	VPR	GUI
alu4	9	9	bigkey	6	6
apex2	10	10	clma	10	11
apex4	11	11	diffeq	7	6
des	7	7	dsip	5	5
ex5p	11	12	elliptic	9	9
ex1010	9	9	frisc	11	11
misex3	10	9	s298	6	6
pdc	15	15	s38417	6	7
seq	10	10	s38584.1	7	7
spla	12	11	tseng	6	6

Table 4.2: Minimum channel widths required to place and route 20 large benchmark circuits using modified architecture

Chapter 5

Case Study 1: A Low-power FPGA Architecture

In this chapter, we investigate a power efficient architecture that targets at minimizing the short-circuit power of FPGA global interconnects without the luxury of dual supply. The proposed architecture, EDA support and power analysis are covered in the following sections.

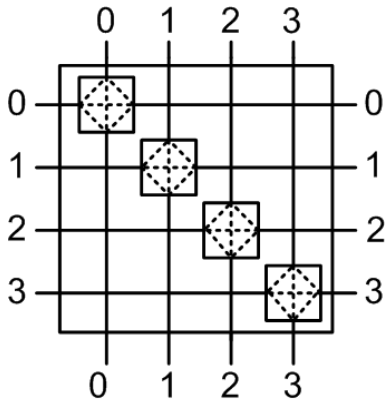
5.1 Conventional switch block

A routing switch block is located at the intersection of a horizontal channel and a vertical channel. Figure 5.1(a) illustrates the structure of a subset switch block, which consists of many switch points. Over the years, three basic types of switch points have been presented in literature.

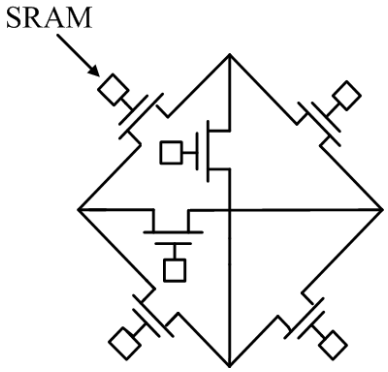
The switch point shown in Figure 5.1(b) is an un-buffered switch [52,53]. It is very area-efficient and works quite well for short connections. However, its performance degrades significantly for long connections which cause the delay to grow quadratically with the wire length. In addition, as pass transistor switch does not provide signal regeneration, the signal amplitude is reduced for long connections. Hence, this lowers the available noise margin at the subsequent input stage. Using try-state buffer as an intermediate repeater to avoid the quadratic delay and regenerate the signal is essential in modern FPGA design.

Figure 5.1(c) and Figure 5.1(d) show two buffered switches used in VPR [30]. The tri-state buffer consists of an inverter (or inverter chain) and a pass transistor [54]. When the pass transistor is switched off, the tri-state buffer produces a floating output node that is totally disconnected from its input. Figure 5.1(c) shows a bi-directional tri-state buffer. This buffer provides the best fan-out capability but it increases the area significantly. In addition, it is slower in short connections as compared to the un-buffered switch.

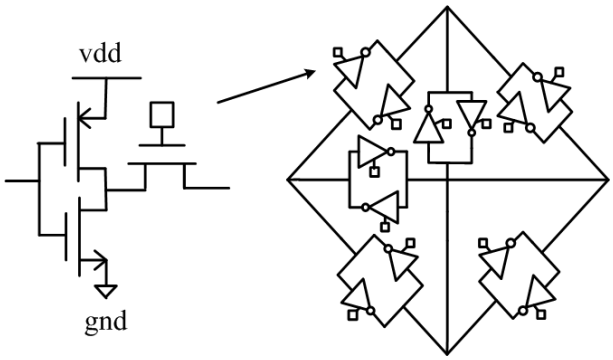
The switch shown in Figure 5.1(d) combines the advantages of both the pass transistors and the buffers to achieve a better area and fan-out tradeoff; and makes compromise between long and short connections. Other designs of routing switches based on Figure 5.1(d) have been presented in [55]. Because of its popularity in academic work, this buffer is used as a comparison to our proposed reconfigurable buffered switch.



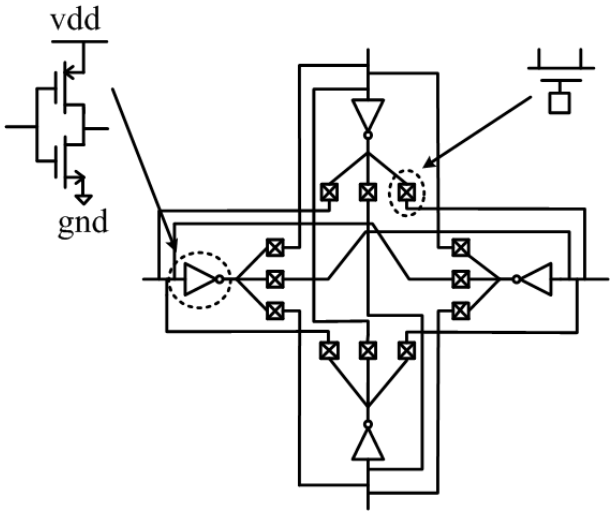
(a) Switch Block Schematic



(b) Un-buffered switch point



(c) Bi-directional buffered switch point



(d) Mixed switch point

Figure 5.1: Conventional switch blocks

5.2 Reconfigurable switch block

The proposed reconfigurable buffer is shown in Figure 5.2. Different driving strengths can be obtained through the binary combinations of the buffer cells. The proposed buffer consists of one mixed buffer cell with minimum driving strength of f_0 in parallel with n reconfigurable buffer cells; each has a driving strength of f_1, f_2, ... and f_n respectively. A total of 2^n programmable driving strengths can be achieved for the proposed reconfigurable buffer. The resulting driving strength can be expressed as follows:

$$f = f_0 + \sum_{k=1}^{n} b_k f_k \tag{5.1}$$

where b_k is the control bit for the k^{th} buffer cell.

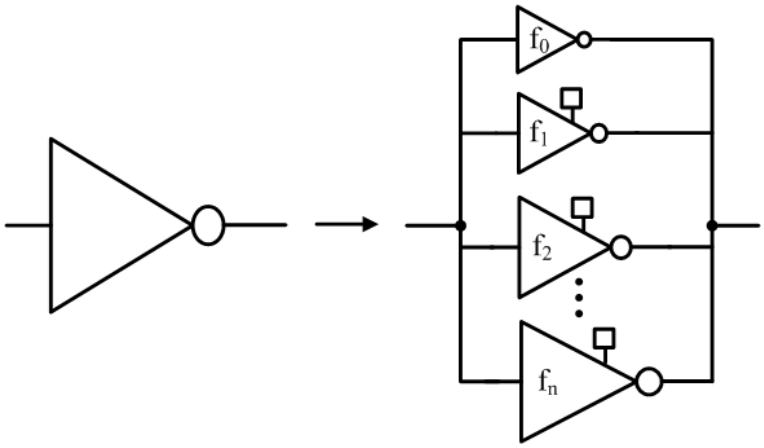


Figure 5.2: Reconfigurable buffer schematic

Three possible implementations of a reconfigurable buffer cell are shown in Figure 5.3. The buffer cell in Figure 5.3(a) is the most area-efficient but it has the worst delay performance. Both buffer cells in Figure 5.3(b) and Figure 5.3(c) employ the

stacked architecture design, which result in usage of an additional PMOS transistor. However, they both give a good delay performance. The buffer cell in Figure 5.3(b) has poor input and output isolation with slightly faster switching. The buffer cell in Figure 5.3(c) provides better isolation with slightly slower switching. Comparing the merits of the candidate buffers, the buffer cell in Figure 5.3(c) proves to be most suitable for our design. A detailed implementation of the whole reconfigurable buffer is shown in Figure 5.4.

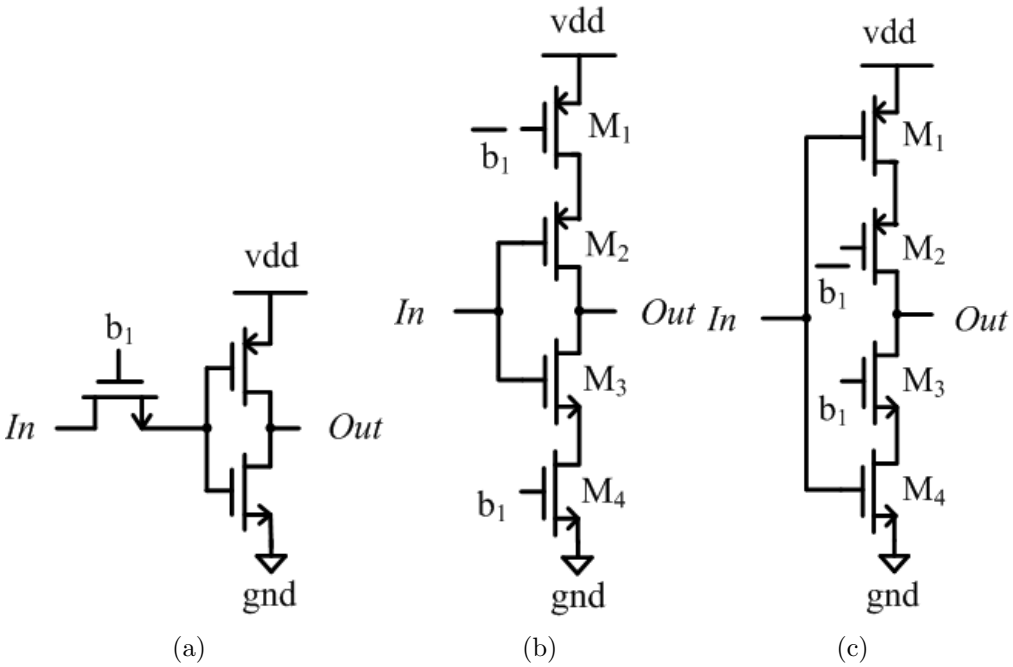


Figure 5.3: Candidate circuits for a reconfigurable buffer cell

To perform the timing analysis, the interconnect path in the FPGA is modeled as a first-order RC network by the EDA software. The reconfigurable buffer is modeled as an equivalent pull-up or pull-down RC network for low-to-high or high-to-low

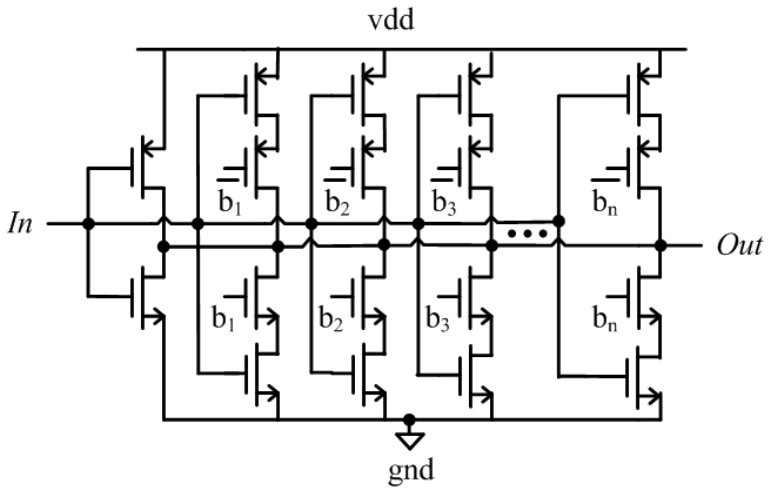
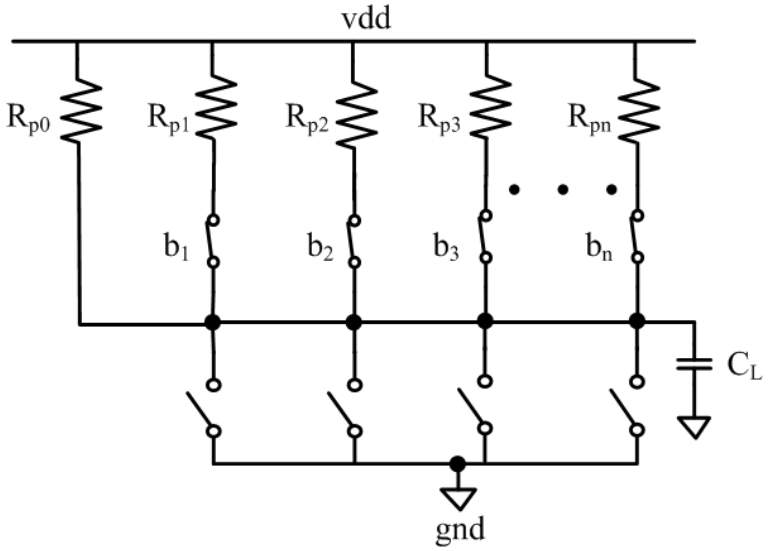


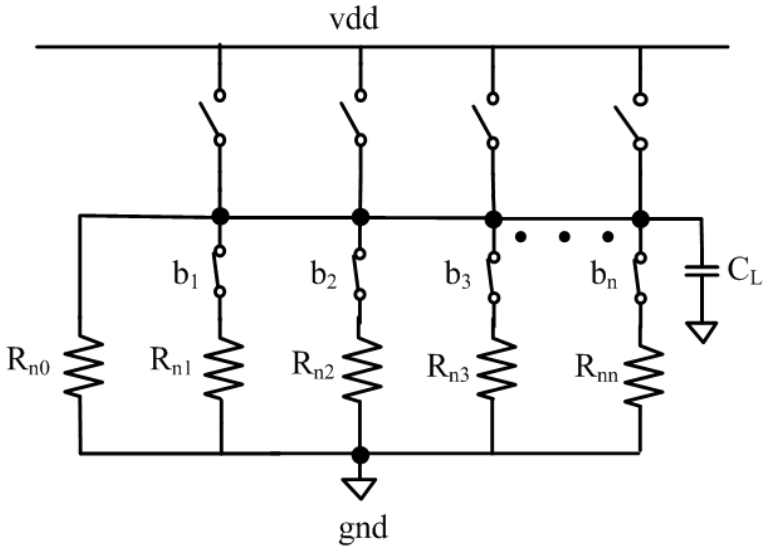
Figure 5.4: Circuit implementation of a reconfigurable buffer

transition respectively, as illustrated in Figure 5.5. The load capacitance (C_L) consists of the intrinsic and the extrinsic capacitance. The intrinsic capacitance, C_{int}, is made up of the total drain diffusion capacitances of the reconfigurable buffer. The extrinsic capacitance, C_{ext}, includes the wire load capacitance and the input gate capacitance of the subsequent stages. Whenever the reconfigurable buffer is switching between the different operating modes, the load capacitance can be treated as a fixed value despite the slight variations due to the different operating modes. Only the equivalent resistance is varied. The total resistance can be approximated as a parallel combination of the on-resistances of all the on-transistors. In our framework, the resistance values for each operating mode have been extracted earlier using the SPECTRE simulation and pre-characterized for subsequent timing analysis.

Since the reconfigurable buffer can be physically considered to be made up of a group of smaller buffer cells, our proposed reconfigurable buffer does not introduce



(a) low-to-high transition



(b) high-to-low transition

Figure 5.5: Equivalent circuits of configurable buffer

significant input gate capacitance or intrinsic capacitance. Hence, the switching power consumed by the reconfigurable buffer and the conventional buffer should be identical if their extrinsic load capacitances are the same. However, the reconfigurable buffer can lower the short circuit power and the ground bounce noise due to the reduced transient current.

5.3 Proposed switch block and FPGA architecture

In our FPGA design, the reconfigurable buffer is made up of three buffer cells, which is controllable with two SRAM bits so as to generate four different driving modes. Inside the proposed switch block, all the buffers are implemented using the reconfigurable buffers. As for the routing architecture, all the switch blocks are replaced with the new switch blocks. The schematic of the new switch is shown in Figure 5.6.

5.4 EDA support

In order to describe our proposed FPGA architecture, additional design parameters and modules are included into our framework. These modules include a Timing-Analyzer module that configures the buffer modes to meet timing requirements and a Power-Evaluator module that evaluates the power utilization of a circuit. An illustration of the design flow for the proposed architecture is shown in Figure 5.7.

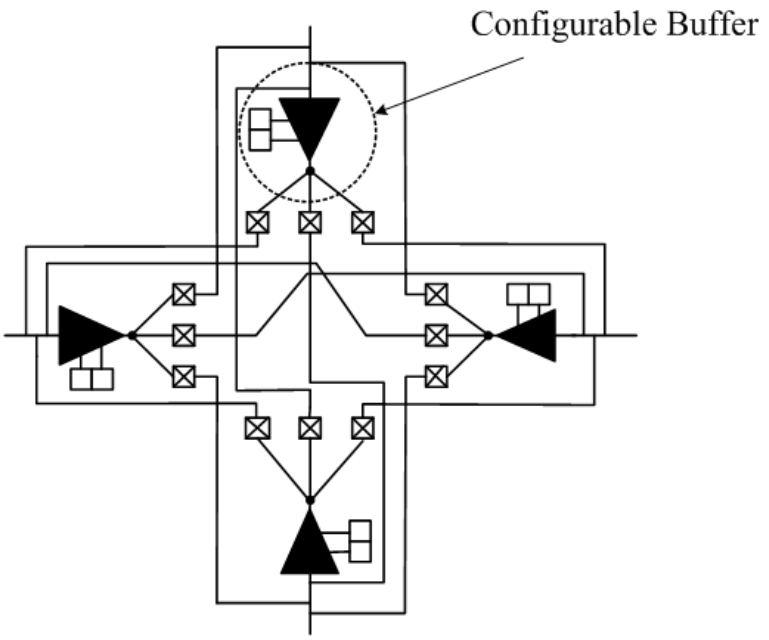


Figure 5.6: Switch point integrated with reconfigurable buffer

Initially, for a given circuit, all its buffers are set to the best mode with the largest driving strength. After the circuit has been successfully placed and routed, the Timing-Analyzer is initiated. It traces through all the nets in the circuit and obtains the circuit's critical path delay. This delay is checked against the preset timing budget. If the timing constraint is violated, the circuit is re-placed and re-routed. Once the timing budget is met, all buffers are tuned to their worst case mode with the smallest driving strength. Timing analysis is then performed again to check if the new critical path delay met the timing constraint. This procedure continues until the critical path delay falls within the indicated range. This reduces the computation time of the tool which tends to increase exponentially if the buffer operating mode is optimized individually. Upon the successful completion of the timing check and

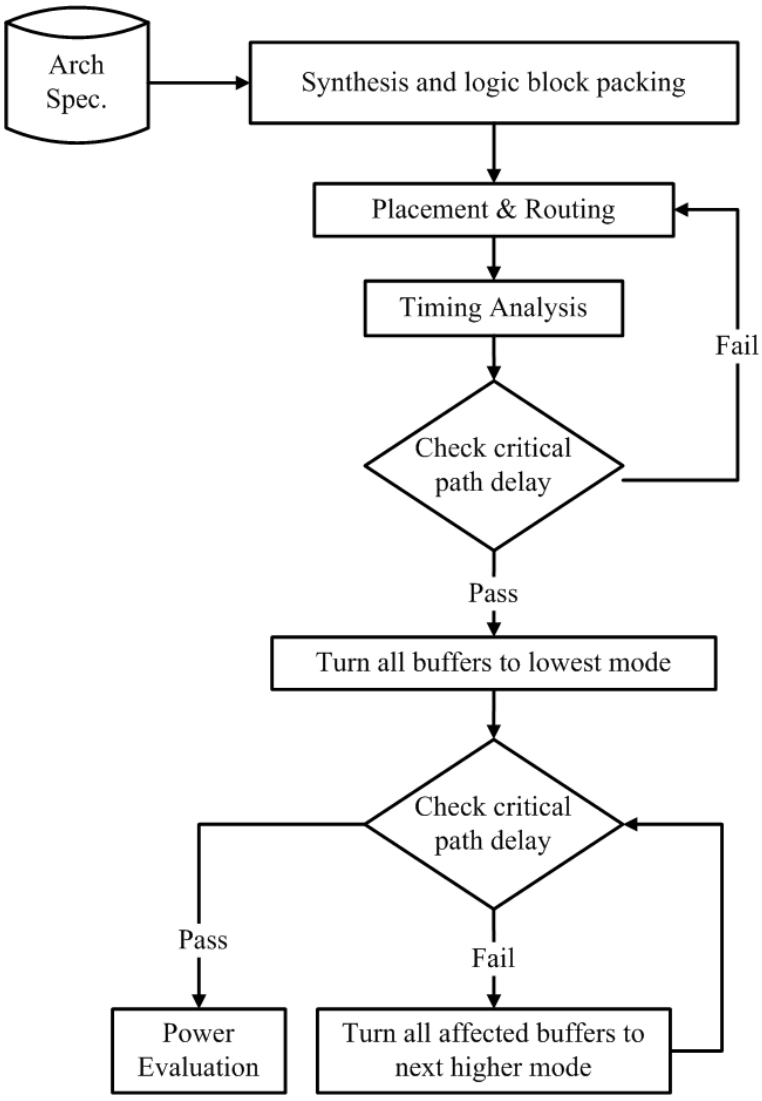


Figure 5.7: EDA flow for propose FPGA routing architecture

buffer tuning, the power evaluation is performed using our Power-Evaluator to report the overall power consumption.

5.5 Power analysis

When performing the power evaluation, two FPGA architectures are tested; one with the reconfigurable buffers and the other with the conventional buffers. In this simulation, the reconfigurable buffer is set to have four different driving strength modes. Each of them can provide a rail-to-rail output swing for a typical wire load capacitance. In relation to fan-out capability, the best operation mode (mode 3) of the reconfigurable buffer is similar to the conventional buffer.

The actual layout area is calculated by considering many implementation issues, such as the distribution of the routing wires, the SRAM bit lines, power routing, the clock tree layout, etc. In order to minimize the area overhead of the new switch block, we adopt the layout design proposed in [56]. Using STMicroelectronics' 0.18μm technology, the layout of the new switch block introduces 20.5% area overhead due to the additional SRAMs which is not present in the conventional buffered switch block.

Despite the additional area overhead, the new switch block does give some advantages. It increases the spacing of the routing wires, thus improving the interconnect delay and minimizing crosstalk noise. In addition, since the total chip area is dominated by the routing wires, this additional area overhead is relatively small as compared to the FPGA chip with conventional buffered switch blocks.

The same set of 20 MCNC benchmark circuits and conditions stated earlier in section 4.4 are chosen for simulation. A timing budget of 100ns is set using the option seen in Figure 3.1. The proposed reconfigurable buffer is set to its best mode (mode 3) as the initial buffer mode for the new architecture such that the two architectures are starting with the same driving strength. Table 5.1 summarizes the energy consumed by the conventional architecture and the new architecture.

Circuit	Buffers Used	Power (w)		
		Conventional Buffers	Reconfigurable Buffers	Saving (%)
alu4	19730	0.232	0.211	9.14
apex2	28377	0.321	0.291	9.49
apex4	20444	0.228	0.206	9.66
des	22739	0.352	0.328	6.94
ex5p	18025	0.199	0.179	9.77
ex1010	64184	0.732	0.663	9.44
misex3	20280	0.232	0.210	9.41
pdc	93315	0.998	0.898	10.05
seq	26086	0.295	0.267	9.51
spla	62436	0.689	0.622	9.74
bigkey	17549	0.262	0.243	7.25
clma	124161	1.405	1.271	9.52
diffeq	13964	0.176	0.161	8.61
dsip	12112	0.216	0.203	6.10
elliptic	42481	0.506	0.461	9.06
frisc	51806	0.587	0.531	9.52
s298	17792	0.227	0.208	8.47
s38417	57859	0.738	0.676	8.50
s38584.1	56368	0.726	0.665	8.43
tseng	9178	0.119	0.109	8.42

Table 5.1: New FPGA architecture energy consumption for 20 large benchmark circuits

As shown in Table 5.1, our new FPGA architecture reduces the total power by 6.10% - 10.05%, and 8.85% on average, when compared with the conventional FPGA

architecture. In our simulation, we observed that the majority of the buffers used for each benchmark circuit are located along the non-critical paths. Hence, these buffers can be relaxed to lower the fan-out mode without violating the specified timing constraint. Generally, the power reduction tends to increase when the number of reconfigurable buffers along the non-critical paths increases. However, since the reduced power only accounts for the short-circuit power of the global interconnects, which only contributes to about 10% of the total dynamic power, it sets the upper bound of the achievable power reduction in our approach.

Chapter 6

Case Study 2: A Interval-based FPGA Timing Estimator

In this chapter, we use VPR to explore a fast and accurate interval-based timing estimator for variability-aware FPGA physical synthesis tools. The proposed model and timing delay analysis are covered in the following sections.

6.1 Deterministic timing estimation

In the traditional deterministic FPGA timing estimation [57], a directed acyclic graph representing the circuit structure is utilized. Each node in the graph represents either the input pins or the output pins of basic circuit elements. Edges are added between the inputs and outputs of the logic blocks and between pins which the circuit netlist specifies. Each edge is annotated with the delay required to pass through the

circuit element. This delay is based on the Elmore delay model. The Elmore delay of a source-sink path is [58]:

$$T_d = \sum_{i \epsilon Source-sinkpath} R_i \cdot C(subtree_i) + T_{d,i} \tag{6.1}$$

where $T_{d,i}$, is the intrinsic delay of a buffer if element i is a buffer and 0 otherwise. R_i is the equivalent resistance of element i. $C(subtree_i)$ is the total downstream capacitance of the subtree rooted at element i.

To obtain the delay of a path, the transversal begins at nodes with no incident edges and each node is labeled with a signal arriving $T_{arrival}$, of 0. Each node which has incidents edges from the labeled nodes is marked with its arrival time as:

$$T_{arrival}(i) = MAX_{\forall \in fanin(i)}\{T_{arrival}(j) + delay(i,j)\} \tag{6.2}$$

where node i is the node being labeled and delay (i, j) is the delay value marked on the edge joining node j to node i. This procedure continues until every node is labeled.

6.2 Modeling of process variation

In this work, we will be focusing on modeling structural variations and delay variations. In structural variations, 4 components are considered: metal thickness (T), inter-layer dielectric (H), line-width (W) and gate length (L) (See Figure 6.1). These variations do result in adverse changes in the electrical properties which include the resistance (R) and capacitance (C). These electrical parameter variations bring

about direct impacts to the performance of the circuit. Hence, an accurate model of interconnect geometry variation is essential for an accurate circuit simulation.

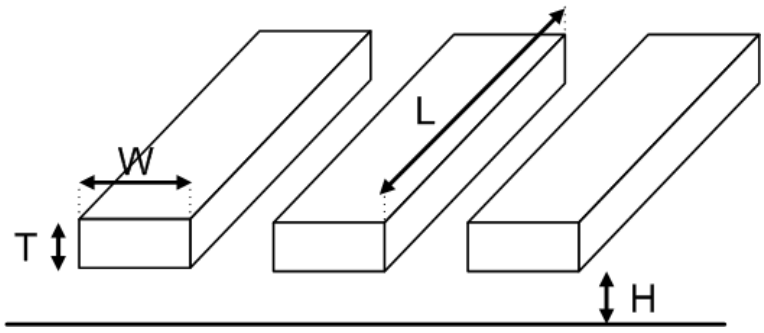


Figure 6.1: Geometry of wiring

In delay variations, variations in the buffer intrinsic delays, sub-block delays and logic delays are considered. These variations are due to the device variations which are not modeled in VPR [3, 49]. Instead, each of these delays is of a deterministic value defined in the architecture file obtained using SPICE simulations in VPR. These delay variations do affect the performance of the circuit significantly.

6.3 Introduction to interval arithmetic

IA was introduced by R.E. Moore in the 1960s [22]. It is a range-based model for numerical computation. In this model, each unknown quantity x is being represented in the form $x = [a, b]$, where the 'true' value of x lies in the range of a and b with $a \leq b$. Arithmetic operations (add, subtract, multiply, etc) can be applied such that each computed interval is guaranteed to contain the unknown value of the quantity

it represents. The rules to perform IA are as follows:

$$\begin{aligned} [a,b]-[c,d] &= [a-d, b-c] \\ [a,b]+[c,d] &= [a+c, b+d] \\ [a,b]\cdot[c,d] &= [min(ac,ad,bc,bd), max(ac,ad,bc,bd)] \\ [a,b]/[c,d] &= [a,b]\cdot[1/d, 1/c] \end{aligned} \tag{6.3}$$

Though IA provides a simple and relatively efficient solution to computational problems, its inability to provide precise data is one of its major setbacks. As IA tends to be too conservative, the computed interval of a quantity may be much wider than the expected range. This is particularly accentuated in long computation chains, where the intervals computed at one stage are the inputs for the next stage. An example to illustrate is when we evaluate the expression x-x where x is in the range [2, 8]. Using IA, we get [2-8, 8-2] = [-6, 6] instead of [0, 0], which is the true range of the expression. Hence, IA is not able to model any form of correlation or dependency between the quantities. To rectify this problem, much research have been carried out [59–61], but at the expense of additional computations.

6.4 Introduction to affine arithmetic

AA was introduced by Comba and Stolfi in the early 1990s [23]. This model is an improvement of the IA model but with an increased complexity and cost in representing the interval value. However, this extra information gives AA a tighter interval bound and keeps track of the correlations between quantities which IA is

unable to provide. In AA, the unknown quantity x is represented by an affine form which is a first-degree polynomial:

$$\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i \tag{6.4}$$

x_0 is defined as the central value of the affine form $\hat{x}$. The coefficient x_i are the partial derivatives defined using floating-point numbers. The ε_i are the noise symbols whose values are unknown but assumed to lie in the range [-1, 1]. Each noise symbol i represent an independent share of the total uncertainty of the variable x, while the corresponding coefficient x_i gives the magnitude of its error.

In AA, the arithmetic operations between the affine forms ensure that the dependencies between the quantities are preserved. These operations are divided into two categories: affine operations and non-affine operations. Examples of affine operations are:

$$\begin{aligned} \hat{x} \pm \hat{y} &= (x_0 \pm y_0) + \sum_{i=1}^{N} (x_i \pm y_i)\varepsilon_i \\ \alpha\hat{x} &= (\alpha x_0) + \sum_{i=1}^{N} (\alpha x_i)\varepsilon_i \\ \hat{x} \pm \alpha &= (x_0 \pm \alpha) + \sum_{i=1}^{N} x_i \varepsilon_i \end{aligned} \tag{6.5}$$

The non-affine operations include operations for which the representation of the result requires additional noise symbols on top of the same noise symbols of the operands. The result of a non-affine operation can be formulated as follows:

$$\hat{z} = f(x_0 + x_1\varepsilon_1 + \ldots + x_N\varepsilon_N, y_0 + y_1\varepsilon_1 + \ldots + y_N\varepsilon_N) \tag{6.6}$$

The key feature of the AA model is its ability to model correlation between two quantities through the sharing of common noise symbols. The magnitude and sign of the dependency is determined from the coefficients assigned to the noise symbols. For example, given two affine forms:

$$\begin{aligned} \hat{x} &= 10 + 2\varepsilon_1 + 1\varepsilon_2 - 1\varepsilon_4 \\ \hat{y} &= 20 - 3\varepsilon_1 + 1\varepsilon_3 + 4\varepsilon_4 \end{aligned} \tag{6.7}$$

From this data, we observed that x lies in [6, 14] and y lies in [12, 28]. As both x and y share the same noise symbols ε_1 and ε_4 with non-zero coefficients, they are not entirely independent of each other. Note that the signs of the coefficients are not meaningful in themselves as the sign of i is arbitrary. However, the relative sign of x_i and y_i defines the direction of the correlation. Hence, using AA, the pair (x, y) is constrained to fall within the dark shaded region as seen in Figure 6.2. However, if IA were to be used, this dependency would be lost. In fact, using IA, the pair (x, y) is only known to lie in the rectangle shown in light grey in Figure 6.2.

6.5 Interval-based timing estimation

As AA provides the abilities to model various variations using different noise symbols and at the same time maintain the correlation between these symbols, it is chosen as the ideal model. With the AA model, we are able to accurately track how each variation gets propagated during the placement and routing using the noise symbols. With the affine interval obtained, measurements to counteract these variations can

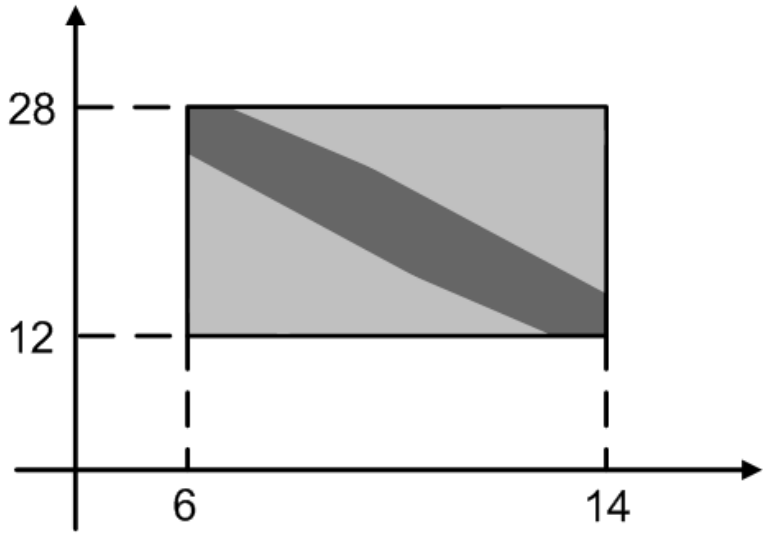


Figure 6.2: Joint range of two partially dependent quantities in Affine Arithmetic

be then developed to improve the quality of the circuit performance.

6.5.1 Modeling of Variation

To model the variation, the affine library is built into VPR. Parameters that need to account for the above variations are defined as affine variables in VPR. To facilitate users to define the degree of each variation, we modify VPR's graphics engine into a Windows MFC application and created an input interface for users to state the desired bounds of each variation. This allows greater flexibility for future researchers to test the effects of the different types and combinations of variations have on the circuit performance. Here, we name the modified VPR model as W_VPR.

To model the structural variations, variations in R and C are used. But initial results show that we are not able to handle correlation at this abstraction level. Instead, a low level implementation is applied to obtain the R and C using W,

H, T and L as stated in Section 6.2. The formulas used to obtain R and C are indicated in (6.8) and (6.9), respectively [10]. Using these equations, noise coefficients of different signs are generated and this creates correlations between the same noise symbols.

$$R = \frac{\rho \cdot L}{W \cdot T} \tag{6.8}$$

$$C = \varepsilon_{ox} \cdot \left[\frac{W}{H} + 0.77 + 1.06 \left(\frac{W}{H} \right)^{0.25} + 1.06 \left(\frac{T}{H} \right)^{0.5} \right] \tag{6.9}$$

Although correlation is handled, several extra noise symbols are created while using these equations. This affects the complexity significantly. To reduce the complexity, we sum up the positive and negative coefficients of the extra noise symbols of R and C and assign them into two new noise symbols respectively as demonstrated in (6.10). All similar parameters will share these two noise symbols and any other extra noise symbols that are generated during the program flow (place and route). To model spatial correlation as well, we have adopted the idea in [19,62], that is, each parameter is re-initialized to contain a unique noise symbol based on its grid position in FPGA. An example is illustrated in Figure 6.3.

$$\begin{aligned} x &= x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 - x_3\varepsilon_3 + x_4\varepsilon_4 - x_5\varepsilon_5 \\ x &= x_0 + x_1\varepsilon_1 + (x_2 + x_4)\varepsilon_2 - (x_3 + x_5)\varepsilon_3 \end{aligned} \tag{6.10}$$

Note: ε_1 is the unique noise symbol for R and C, while the rest are generated due to equations (6.8) and (6.9) and are global to all.

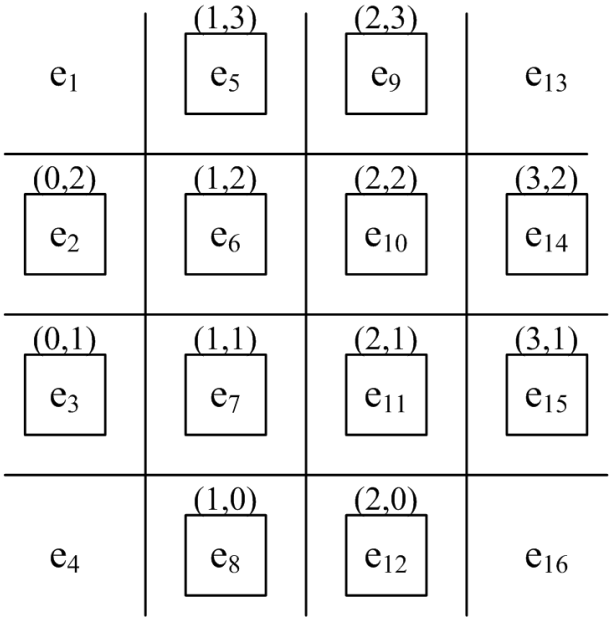


Figure 6.3: The grid-based model to model correlations

6.5.2 Comparison with Statistical modeling

In statistical modeling, variables are often modeled as a random variable that is represented using a probability density function (*p.d.f.*) or a cumulative distribution function (*c.d.f.*). The *MAX* and *ADD* operators [14] are often used to join these variables. Canonical timing model [20, 21] is also proposed to address the correlations through shared parameter variations. Using this model, a block delay and its covariance with another block delay can be evaluated.

Comparing with the existing techniques of statistical modeling, our work does take the form of the canonical timing model. However, using the AA model, we are unable to represent a variable using either a *p.d.f.* or *c.d.f.* as AA does not store the distribution of the variable. Although this is a drawback, AA allows quick

extraction and good estimation of the bounds of the variation without the hassle to handle distribution of the variations. Still, when comparing it with the traditional corner-based approach, AA is better in terms of runtime and accuracy.

6.5.3 Complexity

Having seen the existing works discussed in section 2.4 to solve SSTA problems, we realize that many of these researches have employed complicated SSTA techniques to handle correlations and path reconvergence. To the best of our knowledge, most SSTA techniques have a complexity ranging from $O(n)$ to $O(n^2)$ or have complexities which increase exponentially with the number of logic gates.

However, in our proposed technique, the AA model that is used to handle the process variation has a low complexity. In AA, the complexity of its arithmetic operators is of $O(m)$ where m is the max number of noise symbols in any of the operands. And this complexity does not increase exponentially with circuit size. More details are presented in Section 6.7.

Though our complexity is of $O(n)$, but in terms of runtime, our model is more efficient compared to any SSTA of the same complexity. This can be proven in terms of the number of operations at a node. In existing SSTA, the ADD and MAX operations are performed to obtain a variable's distribution before obtaining the required bound. This is computational intensive as the distribution of the variations is tracked. However, in the case of AA, no distribution information is involved. Its operations

mainly does simple arithmetics to alter the coefficients of the noise symbols. In another words, our proposed technique has a complexity of *O(n)* but with a faster computation of the bounds. Also its complexity is dependent on the nature of the circuit and independent of the circuit's size.

6.6 Design methodology

In this section, we describe the design flow that is undertaken. Our simulation methodology is summarized into the flow chart in Figure 6.4. First, we require users to extract the layout and connectivity variations from their actual model or test-bench circuits. A pop-up interface in W_VPR as shown in Figure 6.5 is created for users to input these values. These variations are to be indicated in terms of percentage deviation from their central values. Using these variations and the architecture specifications, the circuit is initialized.

To carry out AA simulations, its library is added into W_VPR. This library is slightly modified to account for the 3σ deviation. All affected parameters in W_VPR are declared as AA variables. The required test-bench circuit is first placed and routed. After which, the circuit's routed netlist is retrieved to build an affine timing graph. Transversal of the timing graph is done to obtain the critical path delay and its affine range. A pseudo code for the AA timing analysis is presented in Figure 6.6.

To validate our proposed methodology via AA to track process variation, a Monte Carlo [63, 64] (MC) Simulation is created. A MC library is included into W_VPR. In

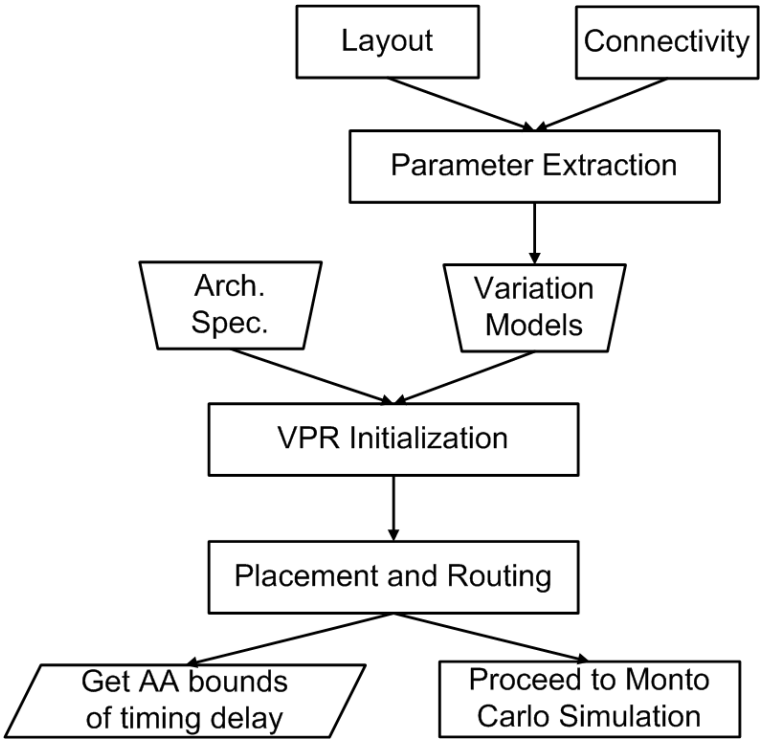


Figure 6.4: Design flow chart

the MC simulation, an interface as shown in Figure 6.7 is created to allow users to choose the type of distribution (Uniform/Gaussian) required, the number of iterations needed for the experiment and whether to use a single or multiple streams for the random number generation. For the Gaussian distribution, the confidence interval is set at 3σ deviation.

To start MC simulation, each circuit's routed netlist is first retrieved. Next, in each iteration, a new set of parameter values is generated and the timing graph is rebuilt. Transversal of the graph is performed to calculate the new critical path delay. Once the MC simulation is completed, a histogram of frequency distribution can be plotted using the new set of critical path delays obtained. Using this set of data, the

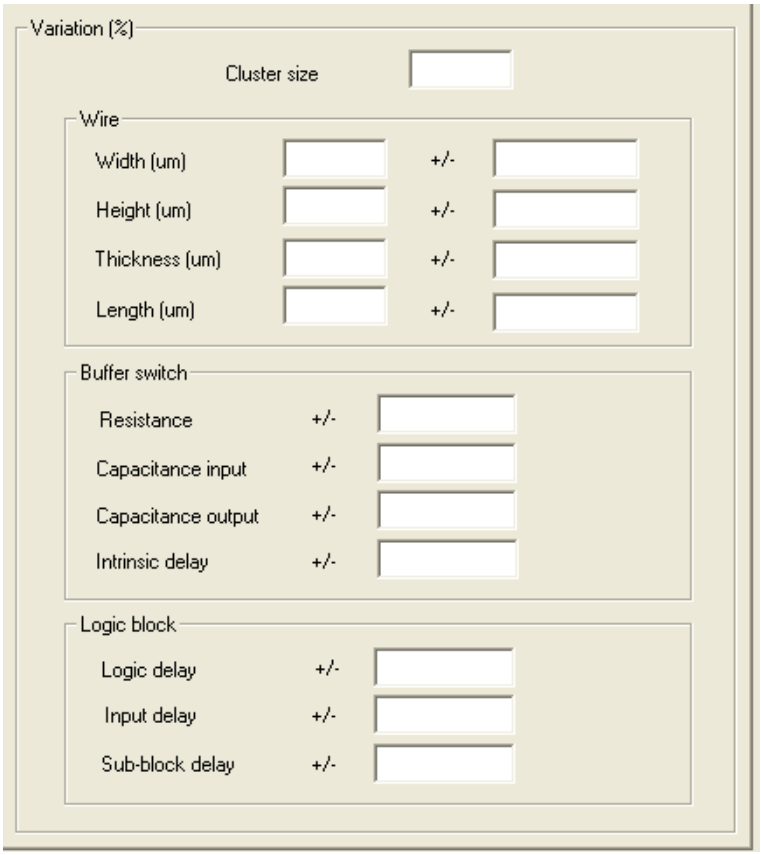


Figure 6.5: Variation initialization interface

bound for each circuit in the MC simulation is also extracted and compared with the one obtained from AA to test its accuracy.

6.7 Timing delay analysis

To prove the speed and accuracy of our method, we perform simulations to compare our estimation results with that of Monte Carlo simulations. Our simulations use the following VPR setup. Each FPGA logic block comprises of 2 slices. Each slice has 2 4-inputs LUT and 2 flip-flops. Each logic block's pin is connected to all

```
1   Declare affine sources l, t, w, h;
2   for ∀ d_i ∈ G |              d is a delay node
3                                G is the RRG
4   for ∀ (v_i,e_i) ∈ (V,E) |    V = {m|m is a node ∧ m∈T};
5                                E = {n|n is a edge ∧ n∈T;}
6                                T is the timing graph;
7   do {
8     Init_affine_model();
9     /* Initialize variable with affine value */
10    /* v ← affine model value (abstracted) */
11    /* l, t, w, h ← affine model value */
12    Calculate_wire_RC();
13    /* using equations (6.8) and (6.9) */
14    Compute_delay();
15    /* e ← Delay (Elmore delay) based on calculated R and C */
16  }
17  Update_RRG();              /* Re-initialize G */
18  Update_timing_graph();     /* Re-initialize T */
19  Get_critical_path();       /* Obtain affine model of critical path*/
```

Figure 6.6: Pseudo-code for AA timing analysis

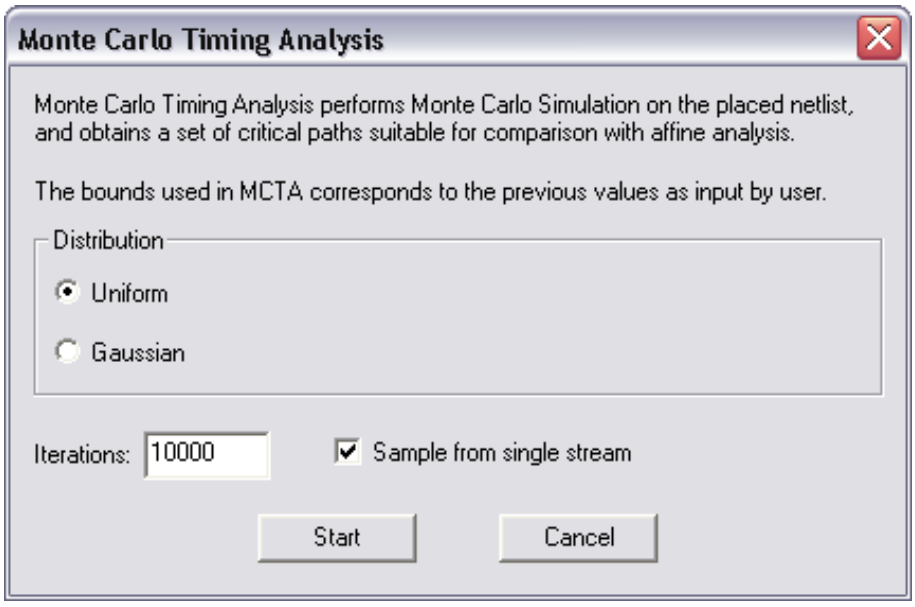


Figure 6.7: MC initialization interface

tracks in the adjacent channel(s) (F_c = W). Each wire segment is of length one and is connected to one wire segment of each of the adjacent channels through disjoint switch boxes (F_s = 3). All wire segments are connected by try-state buffers. Each benchmark is routed with a maximum of 30 iterations to obtain the minimal number of tracks per channel width. When running the experiment using W_VPR, the placement cost function is set to use bounding-box calculation, and the router uses the breadth-first search algorithm.

Parameter	Variation (%)
Length (L)	20
Width (W)	5
Height (H)	20
Thickness (T)	5
Sub-block delay (SD)	5
Logic delay (LD)	5
Buffer delay (BD)	5

Table 6.1: Parameter and its variation

Table 6.1 shows the percentage variations setting for each parameter. The same set of variations is applied to both the AA and MC simulation. The values are arbitrary set but more emphasis is on the interconnect variations for their importance in the deep sub-micron era. Figure 6.8 shows a histogram of the frequency distribution of a benchmark circuit *des* using the Gaussian distribution and single stream. Figure 6.9 shows the Uniform distribution of the same circuit. In both the figures, we observe that our implemented MC simulation gives a desired distribution of the delays and this justifies itself as a good base of comparison to our AA model.

In Figure 6.10, it shows the maximum number of noise symbols seen on an AA

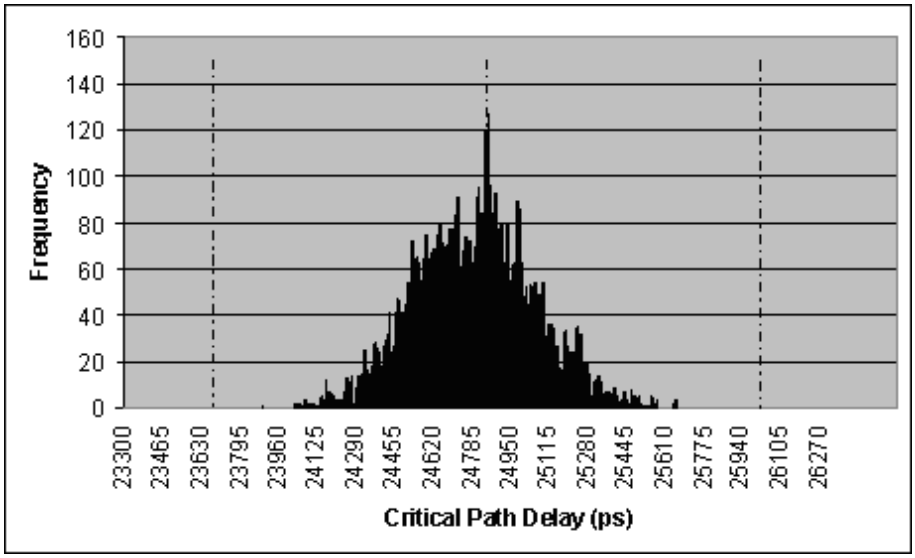


Figure 6.8: Frequency distribution of des using Gaussian distribution and single stream for 10000 iterations (MC)

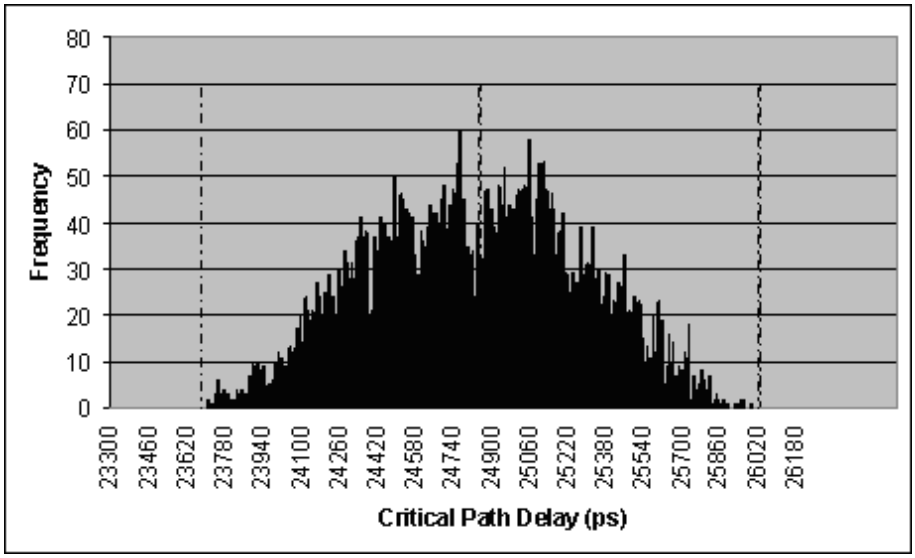


Figure 6.9: Frequency distribution of des using Uniform distribution and single stream for 10000 iterations (MC)

variable in each of the benchmark circuit in ascending order based on their architecture size. From the experiment, we observe that the AA variable with the largest number of noise symbols is always the last node on the critical path. From the results shown, this number does not grow proportionally with the circuit's size. As a critical path is defined as going from a primary input or latch output to a primary output or latch input, its length depends on how the circuit is described in the netlist. Hence, a small circuit may have a longer critical path delay compared to a large circuit. In the context of complexity, this proves that the AA model's complexity is independent on circuit's size but dependent on how the netlist of the circuit is described.

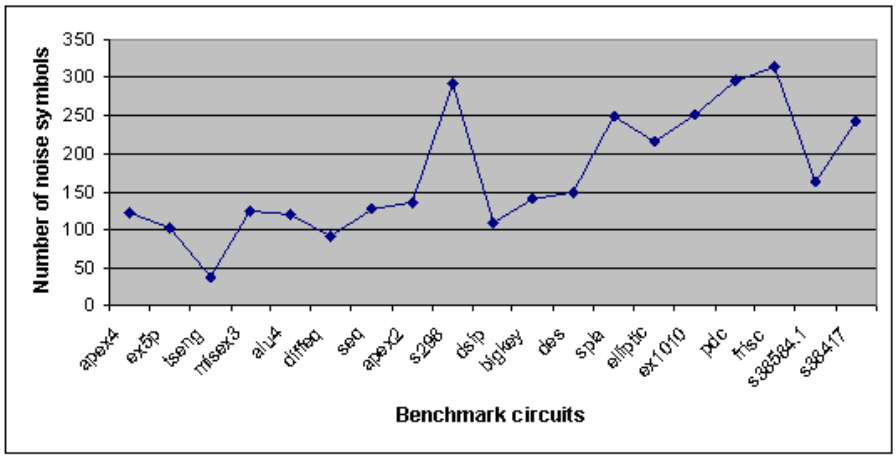


Figure 6.10: Max no. of noise symbols on an AA variable to illustrate that complexity does not grow with circuit's size

To evaluate the accuracy of the delay bound using our model, we compare it against the MC analysis using both Uniform and Gaussian distribution. Also, we define a metric looseness (6.11) to quantify the accuracy of our results. The looseness [65] indicates the ratio of the size of the MC interval to the size of the AA

Circuits	No. of nets	AA model		Uniform (MC)		Looseness (%)	Mean diff (%)
		Range	Mean	Range	Mean		
apex4	927	[23.5 , 25.9]	24.7	[23.9 , 25.6]	24.8	37.8	-0.2
ex5p	912	[19.8 , 21.9]	20.8	[20.1 , 21.5]	20.8	49.4	0.3
misex3	1019	[20.1 , 22.1]	21.1	[20.4 , 21.8]	21.1	40.1	-0.1
alu4	1029	[23.3 , 25.7]	24.5	[23.3 , 25.5]	24.4	6.9	0.4
s298	1287	[60.9 , 66.3]	63.6	[61.0 , 65.5]	63.2	19.3	0.5
dsip	1306	[16.1 , 17.5]	16.8	[16.0 , 17.5]	16.8	-7.3	0.1
bigkey	1649	[21.4 , 23.1]	22.2	[21.3 , 23.0]	22.2	1.7	0.3
des	1794	[23.7 , 26.0]	24.8	[23.9 , 25.7]	24.8	32.5	0.3
Average		-	-	-	-	22.6	0.2

Table 6.2: Comparison of bounds of critical path (ns) - Uniform

Circuits	No. of nets	AA model		Gaussian (MC)		Looseness (%)	Mean diff (%)
		Range	Mean	Range	Mean		
apex4	927	[23.5 , 25.9]	24.7	[23.6 , 25.7]	24.7	11.7	0.2
ex5p	912	[19.8 , 21.9]	20.8	[19.9 , 21.6]	20.7	19.2	0.3
misex3	1019	[20.1 , 22.1]	21.1	[20.1 , 22.0]	21.1	4.5	0
alu4	1029	[23.3 , 25.7]	24.5	[23.4 , 25.6]	24.5	6.1	0
s298	1287	[60.9 , 66.3]	63.6	[60.7 , 66.3]	63.5	-3	0.1
dsip	1306	[16.1 , 17.5]	16.8	[16.1 , 17.4]	16.7	9.4	0.3
bigkey	1649	[21.4 , 23.1]	22.2	[21.2 , 23.2]	22.2	-14.4	0.2
des	1794	[23.7 , 26.0]	24.8	[23.7 , 26.0]	24.8	2.5	0
Average		-	-	-	-	4.5	0.1

Table 6.3: Comparison of bounds of critical path (ns) - Gaussian

interval. The sign means that affine interval is smaller (negative) or larger (positive).

$$looseness = (\frac{AA\_Interval}{MC\_Interval} - 1) \times 100\% \tag{6.11}$$

With reference to Table 6.2 and Table 6.3, we observe that our AA model has an average looseness of 22.6% and 4.5% for the Uniform and Gaussian distribution using single stream respectively. The large value of looseness is partially due to that AA accounts for the worst case of the simulation. However, worst case scenario is

seldom reached in real situations. Hence, the interval obtained in AA is slightly over-pessimistic. Though our AA model gives a large interval, its mean is well-matched to about 0.2% and 0.1% deviation from that obtained in the Uniform and Gaussian distribution respectively. This demonstrates its accuracy in timing estimation.

Furthermore, having the need to only run an iteration with AA to obtain such accurate bound certainly proves its efficiency compared to running 10000 iterations to obtain a slightly tighter bound in a MC simulation. This speed-up can go as high as 1000X when running on a 2.6GHz Pentium PC.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

In this thesis, the work contribution is divided into 3 main sections. First, we have presented a new framework using a GUI interface to facilitate the designing and developing of a heterogeneous island-style FPGAs. This framework has the ability to generate an architecture template and allow editing to create an irregular architecture. The implementation is done in two phases: an initialization phase and an editing phase. In the initialization phase, a standard set of parameters is required for the tool to come up with an arbitrary design of the FPGA architecture. These parameters include the dimensions of the proposed architecture, number of desired pins on blocks, desired tracks to be included, connection box connectivity and switch block connectivity. In the editing phase, users are allowed to alter the above set

of parameters to their preference to create a more arbitrary architecture. Once the architecture is finalized, a RRG is generated to facilitate the routing decisions or for porting to VPR for more complete testing.

The placement and routing techniques are implemented in the framework to test the designed architecture. The placement technique implemented is the simulated annealing algorithm. In each iteration, the blocks are swapped against a temperature schedule. Placement stops when a local minimum solution is achieved. Routing is done using a pathfinder negotiated congestion algorithm. Global routing is done first followed by detailed routing. Ripping and rerouting of nets is carried out at every iteration till a physical route is found for all nets. Once placement and routing have successfully been completed, by clicking on a specified block, its nets and connected blocks are highlighted. This can be verified against the generated output files (xxx.place and xxx.route) before implementing it onto the FPGA.

Next, we have presented a case study using our framework to investigate the effectiveness of a power efficient FPGA architecture. Our preliminary simulation results have shown that, by applying larger driving strength along the critical paths and relaxing the driving strength along non-critical paths, the proposed architecture can reduce the overall dynamic power by 6.10% - 10.05%, and 8.85% on average, when compared with the conventional architecture. It also helps reduce the transient current and thus the ground bounce noise. The proposed technique is complementary to and can be combined with the existing dual supply to further improve the power

performance.

Lastly, we have presented a fast and accurate interval-based method for the timing variability estimation of FPGAs. The method uses correlation-aware affine intervals instead of probability density distributions to model timing uncertainties. Although affine arithmetic methods provide no indication of distribution owing to its intervalistic nature, it can quickly and accurately estimate the mean and range of timing variability for an iteration of physical synthesis optimization, so as to guide the optimization to the right direction. Compared to Monte Carlo simulations, we have shown that the mean of timing variation falls within an accuracy of 1%, the average range looseness is about 22.6% and 4.5% for the Uniform and Gaussian distribution respectively and a 1000X simulation speed-up. This work can also be easily extended to the case of ASICs. Furthermore, using our developed framework, we can extend this case study to non-regular architectures.

7.2 Future work

Suggestions for future improvement to this framework is to implement more functionality, enhance the flexibility and make the tool more user-friendly. A library with different templates of FPGA architectures can be implemented to give the designers more choices. Furthermore, the usefulness of this tool can be further enhanced by making it able to implement other architecture style and their non-uniform routing structures. An accurate power model can also be developed and integrated into this

framework so as to allow for more accurate power analysis to be done.

In addition, we may continue to study how to better integrate the process variations into the W_VPR model to permit correlation cancelation where applicable. This will tighten the bounds while not affecting its central value. Another suggestion for the future work is to add in more abstraction levels in order to model the different types of process variations to a much greater depth. A SSTA method can also be added into VPR to provide a full variability-aware FPGA timing estimation and analysis.

Bibliography

[1] D. Cronquist and L. McMurchie. Emerald - an architecture-driven tool compiler for fpgas. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 144 – 150, 1996.

[2] V. Betz. *Architecture and CAD for speed and Area Optimization of FPGAs.* PhD thesis, University of Toronto, 1998.

[3] J. Rose V. Betz and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs.* Kluwer Academic Publisher, 1999.

[4] V. Betz and J. Rose. Automatic generation of fpga routing architectures from high-level descriptions. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 175 – 184, 2000.

[5] V. George. *Low Energy Field-Programmable Gate Array.* PhD thesis, University of California, 2000.

[6] S. R. Nassif. Modeling and analysis of manufacturing variations. In *IEEE Custom Integrated Circuits Conference*, pages 223 – 228, 2001.

[7] L. Zhang. *Statistical Timing analysis for digital circuit design.* PhD thesis, University of Wisconsin-Madison, 2005.

[8] D. Boning and S. Nassif. Models of process variations in device and interconnect. *Design of High Performance Microprocessor Circuits*, 2000.

[9] S. Sapatnekar. *Timing.* Kluwer Academic Publishers,, 2004.

[10] A. Chandrakasan J. Rabaey and B. Nikolic. *Digital integrated circuits: a design perspective (2nd edition).* Prentice-Hall Publication, 2003.

[11] L. C. Wang J. J. Liou, A. Krstic and K. T. Cheng. False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation. In *Design Automation Conference*, pages 566 – 569, 2002.

[12] M. Orshansky and K. Keutzer. A general probabilistic framework for worst case timing analysis. In *Design Automation Conference*, pages 556 – 561, 2002.

[13] V. Zolotov S. Sundareswaran M. Zhao K. Gala A. Agarwal, D. Blaauw and R. Panda. Statistical delay computation considering spatial correlations. In *Asia and South Pacific - Design Automation Conference*, pages 271 – 276, 2003.

[14] V. Zolotov A. Agarwal and D. Blaauw. Statistical timing analysis using bounds and selective enumeration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1243 – 1260, 2003.

[15] M. Orshansky. Fast computation of circuit delay probability distribution for timing graphs with arbitrary node correlations. In *ACM/IEEE TAU Workshop*, 2004.

[16] F. N. Najm and N. Menezes. Statistical timing analysis based on a timing yield model. In *Design Automation Conference*, pages 460 – 465, 2004.

[17] A. Devgan and C. Kashyap. Block-based static timing analysis with uncertainty. In *International Conference on Computer Aided Design*, pages 607 – 614, 2003.

[18] S. B. Vrudhula S. Bhardwaj and D. Blaauw. Tau: Timing analysis under uncertainty. In *International Conference on Computer Aided Design*, pages 615 – 620, 2003.

[19] H. Chang and S. S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single pert-like traversal. In *International Conference on Computer Aided Design*, pages 621 – 625, 2003.

[20] D. Blaauw A. Agarwal and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *International Conference on Computer Aided Design*, pages 900 – 907, 2003.

[21] K. Ravindran C. Visweswariah and K. Kalafala. First-order parameterized block-based statistical timing analysis. In *ACM/IEEE TAU Workshop*, pages 17 – 24, 2004.

[22] R. E. Moore. *Interval Analysis.* Prentice-Hall Publication, 1966.

[23] J. Stolfi and L. H. de Figueiredo. An introduction to affine arithmetic. In *TEMA Tend. Mat. Apl. Comput.*, pages 297 – 312, 2003.

[24] C. L. Harkness and D. P. Lopresti. Interval methods for modeling uncertainty in rc timing analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1388 – 1401, 1992.

[25] James D. Ma and R. A. Rutenbar. Fast interval-valued statistical interconnect modeling and reduction. In *International Symposium on Physical Design*, pages 159 – 166, 2005.

[26] J. Rose S. Brown, R. J. Francis and Z. G. Vranesic. *Field-Programmable Gate Arrays.* Kluwer Academic Publishers, 1992.

[27] V. Betz J. Swartz and J. Rose. A fast routability-driven router for fpgas. In *International Workshop on Field-Programmable Gate Arrays*, pages 140 – 149, 1998.

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

[29] J. Cong and Y. Ding. Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-based fpga designs. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pages 1 – 13, 1994.

[30] V Betz and J Rose. Vpr: A new packing placement and routing tool for fpga research. In *International Workshop on Field-Programmable Logic and Application*, pages 213 – 222, 1987.

[31] W. Sun and C. Sechen. Efficient and effective placement for very large circuits. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pages 349 – 359, 1995.

[32] V. Betz A. Marquardt and J. Rose. A fast routability-driven router for fpgas. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 203 – 213, 2000.

[33] A. Mukherjee G. Parthasarathy, M. Marek-Sadowska and A. Singh. Interconnect complexity-aware fpga placement using rent's rule. In *International workshop on System-level interconnect prediction*, pages 115 – 121, 2001.

[34] M. Khellah S. Brown and G. Lemieux. Segmented routing for speed- performance and routability in field-programmable gate arrays. *IEEE Journal of VLSI Design*, pages 275 – 291, 1996.

[35] E. Kusse and J. Rabaey. Low-energy embedded fpga structures. In *International Symposium on Low Power Electronics and Design*, pages 150 – 160, 1998.

[36] L. He F. Li, D. Chen and J. Cong. Architecture evaluation for power efficient fpgas. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 175 – 184, 2003.

[37] L. He F. Li, D. Chen and J. Cong. Power modeling and characteristics of field programmable gate arrays. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pages 1712 – 1724, 2005.

[38] L. He F. Li, Y. Lin and J. Cong. Low-power fpga using pre-defined dual-vdd/dual-vt fabrics. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 42 – 50, 2004.

[39] Y. Lin F. Li and L. He. Fpga power reduction using configurable dual-vdd. In *Design Automation Conference*, pages 735 – 740, 2004.

[40] Y. Lin F. Li and L. He. Circuits and architectures for field programmable gate array with configurable supply voltage. *IEEE Transactions on Very Large Scale Integration Systems*, pages 1035 – 1047, 2005.

[41] P. H. Leong W. Luk C. T. Chow, L. S. M. Tsui and S. J. E. Wilton. Dynamic voltage scaling for commercial fpgas. In *International Conference on Field Programmable Technology*, pages 173 – 180, 2005.

[42] V. Khandelwal and A. Srivastava. A general framework for accurate statistical timing analysis considering correlations. In *Design Automation Conference*, pages 89 – 94, 2005.

[43] S. Narayan H. Chang, V. Zolotov and C. Visweswariah. Parameterized block-based statistical timing analysis with non-gaussian parameters, nonlinear delay functions. In *Design Automation Conference*, pages 71 – 76, 2005.

[44] S. Brown M. Khellah and Z. Vranesic. Minimizing interconnection delays in array-based fpgas. In *IEEE Custom Integrated Circuits Conference*, pages 181 – 184, 1994.

[45] V. Betz and J. Rose. Directional bias and non-uniformity in fpga global routing architectures. In *International Conference on Computer Aided Design*, pages 652 – 659, 1996.

[46] S. A. Hauck C. Ebeling, L. McMurchie and S. Burns. Placement and routing tools for the triptych fpga. *IEEE Transactions on Very Large Scale Integration Systems*, pages 473 – 482, 1995.

[47] C. Sechen and A. S. Vincente. The timber-wolf placement and routing package. *Journal of Solid-State Circuits*, pages 510 – 522, 1985.

[48] F. Romeo M. Huang and A. S. Vincentelli. An efficient general cooling schedule for simulated annealing. In *International Conference on Computer Aided Design*, pages 381 – 384, 1986.

[49] K. Shahookar and P. Mazumder. Vlsi cell placement techniques. *ACM Computing Surveys*, pages 143 – 220, 1991.

[50] C. Cheng. An accurate and efficient placement routability modeling. In *International Conference on Computer Aided Design*, pages 690 – 695, 1994.

[51] C. Lee. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, pages 346 – 365, 1961.

[52] J. Rose K. Chung G. Paez P. Chow, S. O. Seo and I. Rahardja. The design of an sram-based field programmable gate array-part i: architecture. *IEEE Transactions on Very Large Scale Integration Systems*, pages 191 – 197, 1999.

[53] J. Rose K. Chung G. Paez P. Chow, S. O. Seo and I. Rahardja. The design of an sram-based field programmable gate array-part ii: circuit design and layout. *IEEE Transactions on Very Large Scale Integration Systems*, pages 321 – 330, 1999.

[54] V. Betz and J. Rose. Circuit design, transistor sizing and wire layout of fpga interconnect. In *IEEE Custom Integrated Circuits Conference*, pages 171 – 174, 1999.

[55] G. Lemieux and D. Lewis. Circuit design of routing switches. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 19 – 28, 2002.

[56] H. Schimit and V. Chandra. Fpga switch block layout and evaluation. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 11 – 18, 2002.

[57] G. Smith R. Hitchcock and D. Cheng. Timing analysis of computer hardware. *IBM Journal of Research and Development*, pages 100 – 105, 1983.

[58] T. Okamoto and J. Cong. Buffer steiner tree construction with wire sizing for interconnect layout optimization. In *International Conference on Computer Aided Design*, pages 44 – 49, 1996.

[59] N. Femia A. Cirillo and G. Spagnuolo. An interval mathematics approach to tolerance analysis of switching converters. In *IEEE Power Electronics Specialists Conference*, pages 1349 – 1355, 1996.

[60] N. Femia and G. Spagnuolo. Identification of dc-dc switching converters characteristics for control systems design using interval mathematics. In *IEEE Workshop on Computers in Power Electronics*, pages 97 – 104, 1996.

[61] N. Femia and G. Spagnuolo. Genetic optimization of interval-arithmetic based worst case circuit tolerance analysis. *IEEE Transactions on Circuits and Systems - Part I*, pages 1441 – 1456, 1999.

[62] J. Singh and S. Sapatnekar. Statistical timing analysis with correlated non-gaussian parameters using independent component analysis. In *Design Automation Conference*, pages 155 – 160, 2006.

[63] Jianwu Lin Enver Yucesan Chun-Hung Chen, Karen Donohue. Efficient approach for monte carlo simulation experiments and its applications to circuit systems design. *Annual Simulation Symposium*, pages 65 – 71, 2001.

[64] M. H. Shi D. Zhou H. X. Gao, X. H. Ma and Y. T. Yang. A novel monte carlo method for fpga architecture research. In *International Conference Solid-State and Integrated Circuits Technology*, pages 1944 – 1947, 2004.

[65] J. D. Ma A. Singhee, C. F. Fang and R. A. Rutenbar. Probabilistic interval-valued computation: toward a practical surrogate for statistics inside cad tools. In *Design Automation Conference*, pages 167 – 172, 2006.