11-9-2004

Low Power Technology Mapping and Performance Driven Placement for Field Programmable Gate Arrays

Hao, Li
University of South Florida

Low Power Technology Mapping and Performance Driven Placement for Field Programmable Gate Arrays

by

Hao Li

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Srinivas Katkoori, Ph.D.
N. Ranganathan, Ph.D.
Miguel Labrador, Ph.D.
Wilfrido Moreno, Ph.D.
Stephen Suen, Ph.D.

Date of Approval:
November 9, 2004

Keywords: physical synthesis and design, network flow, high level synthesis, logic synthesis, power minimization, FPGA

DEDICATION

To my dearest grandparents, my parents, my wife,

and those who have helped and encouraged me in my life.

In memory of my father-in-law, Li Binsheng.

ACKNOWLEDGEMENTS

I would like to express gratitude for my major professor, Dr. Srinivas Katkoori, for his guidance, support and encouragement throughout my doctoral degree program. I also sincerely thank Dr. Wai-Kei Mak, my previous co-major professor at the University of South Florida for his kindness, help and instruction. They led me into the seemingly arduous but actually exciting research world. And they made me believe that I am a Ph.D. material. Special thanks to Dr. Ranganathan, Dr. Labrador, Dr. Moreno, and Dr. Suen for being on my Ph.D. committee and providing valuable advice. I also want to thank Dr. Carnahan for chairing my defense. I would also like to thank the faculties and staffs of the CSE department at USF for giving me enormous help.

I thank all the members of the VCAPP group who have taught me a lot of things, just name a few: Suvodeep, Saraju, Mouli, Stelian, Ashok, Sanjukta.

I would also thank my friends: Tong, Guitao, Yanfei, Zhibin, and Dongqing for their help and friendship.

TABLE OF CONTENTS

LIST OF TABLES iv

LIST OF FIGURES v

ABSTRACT vii

CHAPTER 1 INTRODUCTION 1
- 1.1 VLSI Design and Computer-Aided Design 2
  - 1.1.1 Typical VLSI Design Cycle 3
  - 1.1.2 New Trends in VLSI Design 4
- 1.2 Overview of FPGA-based Designs 5
  - 1.2.1 FPGA Interconnect Architecture 6
  - 1.2.2 FPGA Logic Block Architecture 8
- 1.3 Low Power Design 8
  - 1.3.1 Static Power Dissipation 9
  - 1.3.2 Dynamic Power Dissipation 9
  - 1.3.3 Total Power Dissipation 10
- 1.4 High Level Synthesis 10
  - 1.4.1 Basic Concept of High Level Synthesis 10
  - 1.4.2 Motivation of High Level Synthesis 11
  - 1.4.3 Phases of High Level Synthesis 12
- 1.5 Dissertation Contributions 14
- 1.6 Dissertation Overview 15

CHAPTER 2 BACKGROUND AND RELATED WORK 16
- 2.1 FPGA Architecture 16
- 2.2 CAD Flow for FPGA Physical Design 19
  - 2.2.1 Design Entry 19
  - 2.2.2 Initial Synthesis 21
  - 2.2.3 Functional Simulation 21
  - 2.2.4 Logic Synthesis and Optimization 22
  - 2.2.5 Physical Design 23
    - 2.2.5.1 Partitioning 24
    - 2.2.5.2 Floorplanning and Placement 25
    - 2.2.5.3 Routing 25
    - 2.2.5.4 Compaction 26
    - 2.2.5.5 Extraction and Verification 26

2.2.6 Timing Simulation 27
2.3 Technology Mapping for LUT-based FPGAs 27
2.3.1 Technology Mapping for Delay Optimization 28
2.3.2 Technology Mapping for Area Minimization 30
2.3.3 Technology Mapping for Routability and Low Power 32
2.4 FPGA Placement 32
2.4.1 Optimization Objectives of Placement 33
2.4.1.1 Estimation of Wirelength 33
2.4.1.2 Minimize Total Wirelength 34
2.4.1.3 Minimize Maximum Density 36
2.4.2 Placement Approaches 36
2.4.2.1 Partitioning-based Placement Algorithms 37
2.4.2.2 Analytic-based Placement Algorithms 39
2.4.2.3 Simulated Annealing Placement Algorithms 41
2.4.2.4 Summary of Different Placement Algorithms 43
2.5 Conclusion 44

CHAPTER 3 LOW POWER TECHNOLOGY MAPPING FOR LUT-BASED FPGAS 45
3.1 Introduction 46
3.2 Problem Formulation 47
3.3 Power Estimation Model 49
3.4 Power Minimization Algorithm 52
3.4.1 Phase II: Computation of EP(v) 53
3.4.1.1 T-Bounded K-Feasible Cut 53
3.4.1.2 Incremental Network Flow Computation 55
3.4.2 Phase III: Mapping Generation 56
3.4.3 Computational Complexity of PowerMinMap 59
3.5 PowerMinMap-d: Simultaneous Power and Delay Optimization 60
3.5.1 Review of Flowmap Algorithm 61
3.5.2 PowerMinMap-d Algorithm 61
3.5.3 Computational Complexity of PowerMinMap-d 64
3.6 Experimental Results of Low Power Technology Mapping Algorithms 66
3.7 Conclusions on Low Power Technology Mapping 69

CHAPTER 4 PERFORMANCE-DRIVEN FORCE-DIRECTED PLACEMENT ALGORITHM FOR HIERARCHICAL FPGAS 71
4.1 Introduction 72
4.2 Hierarchical Xilinx FPGA Architecture 74
4.3 Proposed Placement Algorithm 75
4.3.1 Overview of the Algorithm 76
4.3.2 Net Cluster Floorplanning 76
4.3.3 Coarse Net-level Placement 81
4.3.3.1 Attractive and Repulsive Forces 81
4.3.3.2 Net Placement 83
4.3.4 Logic Cell Placement 85

4.3.5 I/O Pin Matching 87
4.3.6 Summary of the Proposed Algorithm 88
4.4 Experimental Results 89
4.4.1 Comparison with Xilinx Foundation Tool 90
4.4.2 Comparison with VPR 92
4.5 Conclusions and Future Work 94

CHAPTER 5 HIGH LEVEL SYNTHESIS FOR PERFORMANCE DRIVEN PLACEMENT 96
5.1 Automatic Design Instantiation System (AUDI) 97
5.2 Performance Driven Placement with High Level Synthesis 100
5.2.1 Overview of the Proposed Design Flow 102
5.2.2 Estimation of the Design Performance 103
5.2.3 Iterative Design Space Search 104
5.3 Experimental Results 105
5.4 Conclusions and Future Work 108

CHAPTER 6 CONCLUSIONS AND FUTURE WORK 111

REFERENCES 114

ABOUT THE AUTHOR End Page

LIST OF TABLES

Table 1.1	Transistors on an Intel Processor Over the Years.	2
Table 3.1	Comparison of PowerMinMap with [1] and [2] (PWR: mW).	67
Table 3.2	Comparison of PowerMinMap-d and Cutmap (Power: mW).	68
Table 3.3	Comparison of PMM-d and Cutmap with Randomly Generated Transition Densities for PI Nodes.	69
Table 4.1	List of Constants Used in Our Work.	88
Table 4.2	Characterics of Combinational Circuits.	91
Table 4.3	Comparison with Xilinx Foundation for Combinational Circuits.	92
Table 4.4	Characterics of Sequential Circuits.	92
Table 4.5	Comparison with Xilinx Foundation for Sequential Circuits.	93
Table 4.6	Comparison with VPR.	94
Table 5.1	Description of Behavioral Benchmarks for AUDI System.	106
Table 5.2	High Level Synthesis for Performance Driven Placement.	107
Table 5.3	Delay Estimation for "latt".	108

LIST OF FIGURES

Figure 1.1	Moore's Law on Intel Processor Series.	1
Figure 1.2	Typical VLSI Design Cycle.	3
Figure 1.3	Typical Flow of High Level Synthesis.	12
Figure 1.4	An Example Data Flow Graph and a Schedule.	13
Figure 2.1	(a) An Example of 2-input LUT. (b) A Fine-grained Logic Block with a Flip-flop.	17
Figure 2.2	Multiplexer-based Logic Module Used by Actel.	17
Figure 2.3	Architecture of an Array-based FPGA.	18
Figure 2.4	Typical Design Flow of a CAD System.	20
Figure 2.5	Logic Synthesis Design Flow.	23
Figure 2.6	Physical Design Flow.	24
Figure 2.7	Global and Detailed Routing.	26
Figure 2.8	Various LUT Mappings for a Boolean Network ($K = 4$): (a) Original Boolean Network; (b) Duplication-free Mapping; (c) Mapping with Overlapping LUTs.	28
Figure 2.9	Different Techniques for Wirelength Estimation.	35
Figure 2.10	Vertical and Horizontal Cutlines for a Placement.	38
Figure 2.11	Different Sequences of Cut Lines for Min-cut Placement.	40
Figure 2.12	Outline of Simulated Annealing Placement Algorithm.	42
Figure 3.1	Power Dissipation of FPGAs.	46
Figure 3.2	A 3-feasible Cut for Node v.	48
Figure 3.3	A Mapped Network into 3-LUTs Rooted at l (Nodes *a, b, c, d*, and *e* are PI nodes).	52

Figure 3.4	(a) Selecting a 3-feasible Cut for Node v. (b) The Mapping Solution.	54
Figure 3.5	(a) Initial Flow Network (for T=5.2). (b) Residual Network after the Max-flow is Computed. (c) The Updated Residual Network with a New Augmented Path Shown in Bold Edges.	57
Figure 3.6	Cut Frontier Refinement for Network Rooted at v (Assuming K=4).	58
Figure 3.7	Pseudocode of PowerMinMap Algorithm.	59
Figure 3.8	Labels Computed for a Boolean Network Assuming $K = 3$.	62
Figure 3.9	Different Mappings Assuming $K = 3$: (a) Using Flowmap and (b) Using PMM-d.	64
Figure 3.10	Pseudocode of PowerMinMap-d Algorithm	65
Figure 4.1	Top Level View of Xilinx Hierarchical FPGA.	74
Figure 4.2	Simplified Architecture of an CLB.	75
Figure 4.3	Design Flow of the Proposed Placement Algorithm.	77
Figure 4.4	Example of Net Clustering: (a) Netlist (b) Net Dependency Graph.	79
Figure 4.5	(a) Star Model of a 5-pin Net. (b) Complete Graph Model of a 5-pin Net.	84
Figure 4.6	Force-Directed Performance-Driven Placement Algorithm.	89
Figure 4.7	Experimental Flow of Our Algorithm.	90
Figure 5.1	RT-Level Design Model of AUDI System.	99
Figure 5.2	Behavioral Description of Design “mx2”.	100
Figure 5.3	(a) DFG Representation of “mx2”. (b) A Scheduling for “mx2”.	100
Figure 5.4	Datapath Information.	101
Figure 5.5	Overview of Design Flow.	102
Figure 5.6	Delay Estimation and Cost Convergence for “latt”.	109

LOW POWER TECHNOLOGY MAPPING AND PERFORMANCE DRIVEN PLACEMENT FOR FIELD PROGRAMMABLE GATE ARRAYS

Hao Li

ABSTRACT

As technology geometries have shrunk to the deep sub-micron (DSM) region, the chip density and clock frequency of FPGAs have increased significantly. This makes computer-aided design (CAD) for FPGAs very important and challenging. Due to the increasing demands of portable devices and mobile computing, low power design is crucial in CAD nowadays. In this dissertation, we present a framework to optimize power consumption for technology mapping onto FPGAs. We propose a low-power technology mapping scheme which is able to predict the impact of choosing a subnetwork covering on the ultimate mapping solution. We dynamically update the power estimation for a sequence of options and choose the one that yields the least power consumption. This technique outperforms the best low-power mapping algorithms reported in the literature. We further extend this work to generate mapping solutions with optimal delay.

We also propose placement algorithms to optimize the performance of the placed circuit. Net cluster based methodology is designed to ensure closely connected nets will be routed in the same region. Net cluster is obtained by clique partitioning on the net dependency graph. Net positions and consequent cell positions are computed with a force-directed approach which drags nets connected to closer positions. We further study the performance-driven placement problem for high level synthesis. We use the Automatic Design Instantiation (AUDI) high level synthesis system to generate an register-transistor level (RTL) netlist. This RTL netlist is fed into an CAD tool for physical synthesis. We do not necessarily go through the entire physical design process which is usually quite time-consuming. Instead,

we have created an accurate wirelength/timing estimator working on the floorplan. If the estimated timing information does not meet the constraints, a guidance is generated and provided to AUDI system. The guidance consists of the estimated timing information and instructions to produce a new netlist in order to improve the performance. Finally the circuit is placed and routed on a *satisfying* design. This performance-driven placement framework yields better results as compared to a commercial CAD tool.

CHAPTER 1

INTRODUCTION

According to *Moore's law* [3], the total number of transistors per chip and the microprocessor's performance (measured by millions of instructions per second (MIPS)) will be doubled every 1.5 to 2 years. It has been a key trend indicator for the semiconductor industry correctly for the past 30 years. Figure 1.1 shows the number of transistors on an Intel processor has been increasing steadily as Moore's Law indicates. The corresponding processor type, year to appear in the market and transistor numbers are given in Table 1.1.

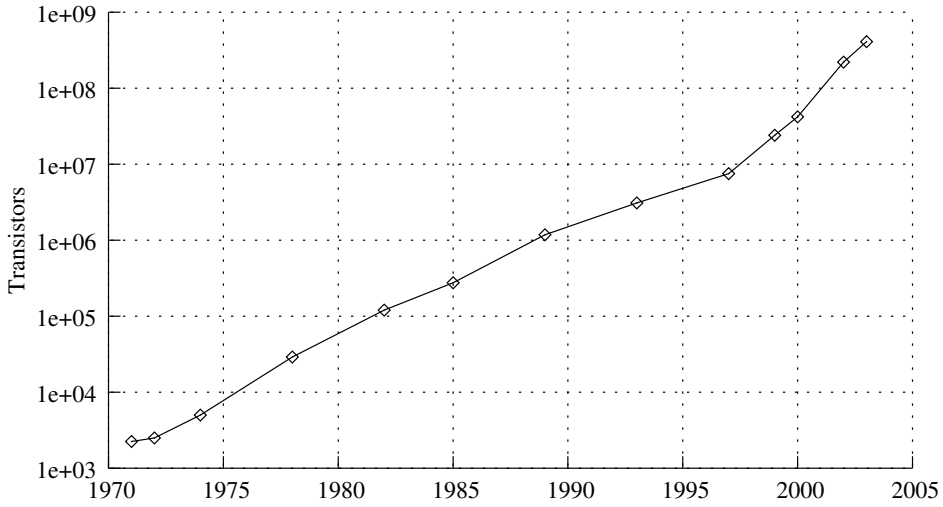


Figure 1.1. Moore's Law on Intel Processor Series.

As the process technology advanced to the deep sub-micron (DSM) region, the logic capacity and performance of a very large scaled integration (VLSI) circuit has grown rapidly. The latest commercially available microprocessor from Intel is manufactured with 90 nm technology and its operating frequency has reached 3.4 GHz while the supply voltage has decreased to 1.4 V. Designing VLSI circuits manually or from scratch has become virtually

Table 1.1. Transistors on an Intel Processor Over the Years.

Processor	Year	Transistors
4004	1971	2,250
8008	1972	2,500
8080	1974	5,000
8086	1978	29,000
286	1982	120,000
386 processor	1985	275,000
486 processor	1989	1,180,000
Pentium processor	1993	3,100,000
Pentium II processor	1997	7,500,000
Pentium III processor	1999	24,000,000
Pentium 4 processor	2000	42,000,000
Itanium processor	2002	220,000,000
Itanium 2 processor	2003	410,000,000

impossible and infeasible. Therefore, Computer-Aided Design (CAD) tools are essential to all VLSI chip designers in every design level. The CAD tools currently in use are highly sophisticated such that most design process can be taken care of automatically. This greatly expedites the design cycle and reduces the chances of design errors. As a result, the costs of VLSI chips have dropped dramatically, while their performance have increased significantly over the years.

In this chapter, we will provide the outline of VLSI design and computer-aided design. In particular, we focus on FPGA based design, power minimization, and high level synthesis. This chapter is organized as follows: We give an outline of VLSI design and corresponding new design trends in Section 1.1. We present an overview of FPGA design in Section 1.2. We show the motivation behind low power design in Section 1.3. We introduce the concept of high level synthesis and related topics in Section 1.4. We present the contributions of our work in Section 1.5. We give the outline of this dissertation in Section 1.6.

1.1 VLSI Design and Computer-Aided Design

The design of digital systems can be achieved at many different refinement levels from the most abstract architecture down to the most detailed layout. Given the enormous com-

plexity and various constraints required to be met at all levels, computer-aided design has been utilized extensively at each level in modern time. It is unimaginable to use manual design techniques for designing digital circuits, in which each layer is hand etched or composed by laying tape on film.

1.1.1 Typical VLSI Design Cycle

A typical VLSI design cycle is shown in Figure 1.2 [4].

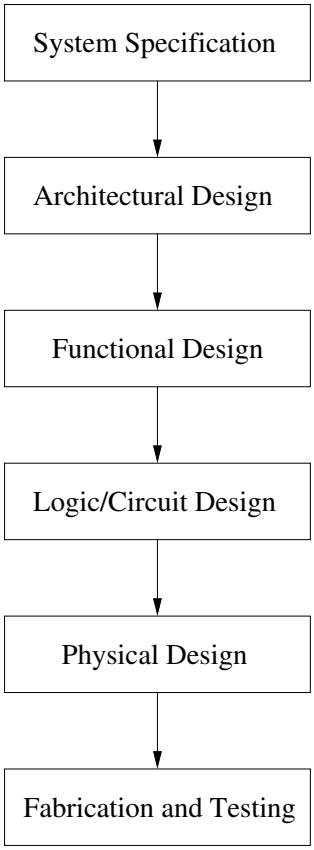


Figure 1.2. Typical VLSI Design Cycle.

Even though the details of various VLSI systems are different, we can briefly outline all the common phases of the VLSI design cycle as follows:

- System specification: A high level description of the digital system for the size, speed, power, and functionality of the system.

- Architectural design: A micro-architectural specification (MAS) includes number of ALUs, floating point units, number and structure of pipelines, choice of RISC or CISC, etc.

- Functional design: Also known as behavioral design. The behavioral aspects of the system and interconnect demands are identified without considering implementation related details.

- Logic and circuit design: In logic design phase, a register-transistor level (RTL) description is derived specifying the control flow, word widths, register allocation, arithmetic operations, and logic operations. Then, the RTL netlist is given to the logic synthesis tool to generate a detailed circuit diagram consisting of cells, macros, gates, transistors, and interconnection between these components.

- Physical design: The circuit representation or netlist is converted into a geometric representation called *layout*. The layout is obtained by converting each logic element into a geometric representation with specific shape. Interconnection between the logic components are also expressed as lines in multiple layers. Physical design is a complex process and hence it is typically broken down into various sub-steps. Verification and validation checks are performed on the layout during physical design.

- Fabrication and testing: Fabrication is the process to depose and diffuse various material on the wafer. Then each chip is packaged and tested to ensure that all design specifications are met.

In general, the objectives of VLSI CAD tools are to minimize the running time of each step discussed above, thus reducing time-to-market and optimize the performance of the system based on user-specified constraints.

1.1.2 New Trends in VLSI Design

With the dramatic increase in circuit complexity and decrease in technology geometry, there are many new trends in VLSI design that need to be taken into account at present.

- Interconnect delay dominates: As the fabrication technology advances to deep submicron (DSM) region, the interconnect is contributing more and more to the path delay. One solution to interconnect and signal integrity is to insert repeaters in long wires. Advanced planning becomes necessary because area overhead for repeaters must be allocated upfront.

- More metal layers: The number of metal layers is increasing to meet the need for interconnection. Three layers are commonly used while four or five layer process are adopted mainly for microprocessors. So a three dimensional view of the interconnect is necessary.

- Synthesis: Design time can be reduced if layout can be straightly synthesized from a higher level description. Depending on the level of design on which synthesis is utilized, there are two types of synthesis.

  I. Logic synthesis: It converts an HDL description into a circuit description (schematics) and then produces its layout. Logic synthesis is a well established technology for designing blocks of a chip, and for application specific integrated circuits (ASICs). It is not applicable for larger blocks, such as RAMs, ROMs, datapaths, and microprocessors because of slow speed and area inefficiency.

  II. High level synthesis: This process converts a behavioral aspects of the system into a layout or RTL description. We will provide more details on high level synthesis in Section 1.4.

1.2 Overview of FPGA-based Designs

The field programmable gate array (FPGA) was first introduced in 1985 by Xilinx. Presently, it has become one of the most popular devices utilized in current VLSI system and rapid system prototyping. The key to FPGA's popularity is their capability to implement *any* digital system due to its programmability. Compared with custom design technologies, such as Standard Cells or semi-custom designs such as Mask-Programmed

Gate Arrays (MPGAs), to use FPGAs has two apparent benefits: lower non-recurring engineering (NRE) costs, and faster time-to-market. This makes FPGAs the lowest cost implementation platform for small and medium volume designs. And if any fault is found in an FPGA based design, it can be corrected soon by reprogramming the FPGA. To meet today's compelling requirement of short product cycle, FPGAs are preferable due to the smaller time-to-market. However, FPGA also has its drawbacks mainly due to the interconnect technology it utilizes. Unlike standard cell technology where circuitry is connected via metal wires, FPGA connect circuitry using programmable switches. The switches have higher series resistance than the metal wires and add significant parasitic capacitance to interconnections, resulting in lower circuit speed. In addition, the programmable switch uses more area than metal wire does which makes FPGA implementations more expensive for high volume designs. To implement the same circuit, the one implemented using FPGA is usually 10 times larger and 3 times slower than the same circuit implemented in an MPGA in an equivalent process [5].

An FPGA chip consists of an array of uncommitted programmable logic blocks, programmable interconnections, and I/O pads. The lookup-table (LUT) based architecture is the most popular architecture used by several FPGA manufacturers, including Xilinx [6] and AT&T [7]. The programming technology determines the method of storing the configuration information, and comes in various flavors. This has a strong impact on the area and performance of the array. The main programming technologies are: Static Random Access Memory (SRAM), antifuse, and non-volatile technologies using floating gates. The selection of the programming technology is based on the computation environment in which the FPGA device is used.

1.2.1 FPGA Interconnect Architecture

In FPGAs, the interconnect architecture is realized using programmable switches to implement different connections. The method of providing the connectivity between the logic blocks has a strong impact on the characteristics of the FPGA architecture. The

arrangement of the logic and interconnect resources can be broadly classified into four categories: island style, row-based, sea-of-gates, and hierarchical.

- *Island style architecture*: It consists of an array of programmable logic blocks with vertical and horizontal programmable routing channels. The number of segments in the channel determine the resources available for routing. The XC4000 and XC3000 series from Xilinx [6] are examples of this type of architecture.

- *Row-based architecture*: It has logic blocks arranged in rows with horizontal routing channel between successive rows. The routing tracks within the channel are divided into one or more segments. The length of the segments can vary from the width of a block pair to the full length of the channel. Other tracks run vertically through the logic blocks. They provide connections between the horizontal routing channels. The ACT3 family of FPGAs from Actel [8] is an example of this architecture.

- *Sea-of-gates architecture*: Unlike the previous architectures, it is not an array of logic blocks embedded in a general routing structure. This architecture consists of fine-grain logic blocks covering the entire floor of the device. Connectivity is realized using dedicated neighbor-to-neighbor routes that are usually faster than general routing resources. The SX family of FPGAs from Actel [9] is a an example of this class of architecture.

- *Hierarchical Architecture:* Most logic designs exhibit locality of connections, which implies a hierarchy in the placement and routing of the connections between the logic blocks. The hierarchical FPGA architecture exploits this feature to provide smaller routing delays and a more predictable timing behavior. It is created by connecting logic blocks into clusters. These clusters are recursively connected to form a hierarchical structure. The speed of a net is determined by the number of routing switches it has to pass through. The hierarchical structure reduces the number of switches in series for long connections. The Virtex family FPGAs from Xilinx [10] is an example of this kind of architecture.

1.2.2 FPGA Logic Block Architecture

The logic blocks on FPGAs are responsible for implementing the gate level functionality for each design. The functionality can be defined by the number of different functions it can implement. It has a direct impact on the routing resources. As the functional units of the logic block increases, it reduces the amount of external routing resources. On the other hand, it may result in logic wastage because the logic block cannot be fully utilized. Therefore, there exists a tradeoff between optimizing the area and speed for different logic block structures.

The functionality of the logic blocks is derived by controlling the connectivity of some basic logic gates or by using lookup-tables (LUTs). In LUT-based FPGAs, a K-input lookup table (K-LUT) is the basic programmable logic block which can be programmed to implement any Boolean function of up to K variables [11].

Currently, FPGA's performance is approaching that of an ASIC. In generally, the following factors affect the performance of an FPGA:

- the quality of the FPGA architecture
- the electrical design of the FPGA
- the quality of the CAD tools used to map, place, and route a design onto the FPGA.

1.3 Low Power Design

The proliferation of mobile computing platforms and portable electronic devices has made low power design a key issue in VLSI system design and CAD. The total power consumption continues to increase because of higher operating frequencies, higher overall interconnect capacitance and resistance, and the gate leakage power of the on-chip transistors. In a CMOS circuit, the amount of power dissipation is determined by the following two components [12]:

- Static power dissipation due to the leakage and standby power dissipation.
- Dynamic power dissipation due to short circuit (transient switching current) and charging and discharging load capacitances (capacitive switching).

1.3.1 Static Power Dissipation

Leakage power dissipation happens because of reverse bias leakage between diffusion regions and the substrate. In addition, sub-threshold conduction also contributes to leakage power dissipation. *Standby power dissipation* occurs due to the current drawn continuously from the power supply. For example, when both the nMOS and pMOS are continuously on in a pseudo-nMOS inverter, standby power dissipation happens. Practically, standby power is negligible compared to leakage power and static power is measured by leakage power as shown in Equation 1.1.

$$P_s = \sum_{i=1}^{n} \textit{leakage current} \star \textit{supply voltage} \tag{1.1}$$

where n is the total number of transistors in the CMOS circuit.

1.3.2 Dynamic Power Dissipation

Capacitive switching power is caused by charging and discharging the output capacitive load in the circuit and it is measured as in Equation 1.2.

$$P_d = \frac{1}{2} C_L V_{dd}^2 N f_p \tag{1.2}$$

where C_L is the load capacitance, V_{dd} is the supply voltage, N is the average number of transitions per clock cycle (switching activity), and f_p is the clock frequency. This indicates that the power dissipation is proportional to the switching activity but independent of the device parameters. It is straightforward that circuit with slower clock frequency result in

less power dissipation, but this may not be desirable for designing high speed system. So reducing the supply voltage is preferred in low power design.

During the transition from 0 to 1 or 1 to 0, both nMOS and pMOS are switched on temporarily. Therefore, there exists a short current from V_{dd} to V_{ss}. The power dissipation because of this is called *short circuit power*, and it is computed as in Equation 1.3:

$$P_{sc} = \frac{\beta}{12} (V_{dd} - 2V_t)^3 \frac{t_{rf}}{t_p} \tag{1.3}$$

where β is the transistor gain factor, V_{dd} is the supply voltage, V_t is the threshold voltage, t_{rf} is the rise/fall time (assuming $t_r = t_f$), and t_p is the period of the input waveform.

1.3.3 Total Power Dissipation

The total power dissipation is the sum of the static power, short current power, and dynamic power dissipation.

$$P_{total} = P_s + P_{sc} + P_d \tag{1.4}$$

In practice, a large portion of the total power dissipation is due to the dynamic power. So research work on reducing power consumption has been mainly focused on reducing the dynamic power.

1.4 High Level Synthesis

High level synthesis (HLS) is a popular research topic in academia. In this section, we first introduce the basic concept of HLS. Then, we discuss the advantages of HLS. Lastly, we present the design flow of high level synthesis tool.

1.4.1 Basic Concept of High Level Synthesis

High level synthesis (HLS) is the process generating digital system from an abstract behavioral specification to its structural description [13] [14]. It is analogous to a "compiler" which translates a high level language such as C/C++ to an machine-dependent executable

program. High level synthesis finds a register-transfer level (RTL) structure that implements the behavior while satisfying a set of constraints. The constraints in HLS include performance, area, cost, power, reliability, testability, etc. *Behavior* refers to the way the system or its components interact with their environment (mapping from inputs to outputs). *Structure* refers to the set of interconnected components that construct the system. High level synthesis is also known as algorithm-level or behavioral-level synthesis.

The inputs to HLS usually consist of the following:

- The behavioral specification of the system
- Constraints such as cost, performance, power, etc.
- The optimization function
- A module library representing the available components at RTL.

And the outputs are composed of:

- RTL implementation structure (netlist)
- Controller which is usually captured as a symbolic FSM
- Geometrical information

1.4.2 Motivation of High Level Synthesis

Generally, decisions made at the higher level of a design have greater impact on the design than the decisions made at lower levels. High level synthesis is becoming popular due to the following reasons [13]:

- *Shorter design cycle*: If more of the design process is automated, the product can be available to the market faster at a lower cost.
- *Fewer errors*: If the synthesis process can be verified correctly, the possibilities of having errors will be fewer.

- *Ability to search the design space*: The synthesis system produces multiple designs for the same specification in a short time. The designers have the options to choose the design with different trade-offs such as cost, speed, power, etc.

- *Self-documenting design process*: An automated system can keep track of the design decisions and the effects of the decisions.

- *Availability of IC technology to more people*: More design expertise is moved into synthesis system, it becomes easier for a non-expert to produce a chip that meets a given set of specifications.

1.4.3 Phases of High Level Synthesis

A typical high level synthesis tool consists of the following phases as shown in Figure 1.3. The behavior of the system to be synthesized is usually specified at the algorithmic level

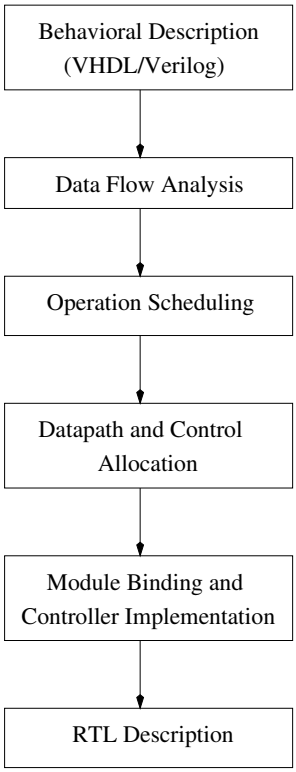


Figure 1.3. Typical Flow of High Level Synthesis.

using a hardware description language like VHDL or Verilog. Unlike general-purpose programming languages such as C/C++, VHDL/Verilog support concurrency. The behavioral specification is transformed into graphical representation which is a data flow graph (DFG) and control flow graph (CFG). The data flow graph is a directed graph indicating the data moves, and the control flow graph is a directed graph representing the sequence of operations. An example DFG is shown in Figure 1.4 (a). During data flow analysis, several transformations are involved including: parallelism extraction, eliminating high level language constructs, loop unrolling, and common subexpression detection. Scheduling is the

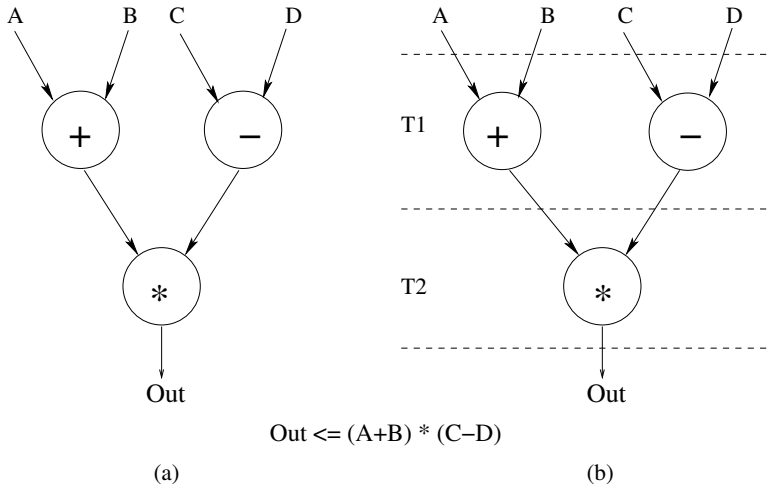


Figure 1.4. An Example Data Flow Graph and a Schedule.

process of assigning arithmetic and logical operations to control steps. A control step is defined as the fundamental sequencing unit in synchronous systems and it typically corresponds to a clock cycle. The objective of scheduling is to minimize the amount of time or the number of control steps needed for completion of the program, given a certain of constraints on the available hardware resources. Numerous scheduling algorithms have been proposed by previous researchers [15] [16] [17] [18] [19] [20]. These algorithms can be categorized as: integer linear programming, as soon as possible (ASAP), as-late-as-possible (ALAP), force-directed scheduling, and list-based scheduling, etc. One schedule for the DFG shown in Figure 1.4 is shown in Figure 1.4 (b).

Resource allocation refers to the process of determining the types of hardware components (functional units, storage, and communication paths) and the number of each type to be used in the final implementation. Usually, resource allocation can be divided into datapath and control allocation. Data-path allocation refers to operation selection, register/memory allocation, interconnection generation and hardware minimization. Control refers to the selection of control style (PLA, microcode, random logic, etc.) and controller generation. Actually, scheduling and allocation are closely interrelated and depend on each other.

Binding is the process of assigning operations to the allocated hardware components. In this phase, physical modules are selected meeting the specification of module parameters and constraints. Meanwhile, controller implementation is generated. Controller is implemented as a finite state machine (Mealy or Moore machine) that controls the flow of data in the datapath. The datapath and the controller communicate through a set of registers.

In the final phase of HLS, design output is generated in a form such that it can be understood by the logic-level synthesis tools. The generated output is usually described with a low level hardware language such as structural VHDL or EDIF [21].

1.5 Dissertation Contributions

The major contributions of this dissertation include:

- The low power driven FPGA technology mapping approach significantly reduces the dynamic power consumption over previous works.
- The extension of the proposed technology mapping algorithm guarantees to generate delay optimal mapping solutions while still reducing the power consumption.
- The top-down design flow for placement is effective to enhance the performance of the placed circuit.
- Force-directed placement scheme was mainly used for ASIC-based designs. We show that it can be utilized for FPGA placement and the experimental results are satisfying.

- Layout aware high level synthesis and placement algorithm is capable of efficiently predicting the timing information at the physical design level. Further, it increases the performance of the placed circuit.

1.6 Dissertation Overview

The rest of this dissertation is organized as follows:

In Chapter 2, we provide the background on VLSI CAD, FPGA physical synthesis, and discuss related research work that has been done in the area of technology mapping and placement.

In Chapter 3, we propose a low power driven technology mapping framework for LUT-based FPGAs. And extension of this work is also presented so as to generate delay-optimal mapping solutions while reducing the power consumption as well.

In Chapter 4, we present a force-directed performance driven placement algorithm for hierarchical FPGAs.

In Chapter 5, we propose a layout aware performance-driven placement flow with high level synthesis.

In Chapter 6, we draw the conclusions and discuss future research directions.

CHAPTER 2

BACKGROUND AND RELATED WORK

The *Field Programmable Gate Array* (FPGA) was first introduced in 1985 by Xilinx as an alternative to application-specific integrated circuit (ASIC) designs. It has become one of the most popular devices for digital system prototyping and has grown into a multi-billion industry. VLSI system designers can implement their designs by *programming* the FPGA chip in the field, thus reducing the lengthy and expensive fabrication cost. With the performance of the FPGA chips increasing rapidly, the computer aided design (CAD) for FPGAs is of great importance currently. In this chapter, we provide the background information on CAD for FPGA-based designs and then we go over the related research work. This chapter is organized as follows: In Section 2.1, we give the background information on FPGA architectures. In Section 2.2, we present the CAD design flow for FPGA physical design. In Section 2.3, we describe the previous work on FPGA technology mapping. In Section 2.4, we review the prior work on placement for FPGAs. In Section 2.5, we conclude this chapter.

2.1 FPGA Architecture

An FPGA is an off-the-shelf VLSI chip consisting an array of configurable logic blocks (CLBs), vertical and horizontal routing channels, and programmable input/output blocks. The most popular FPGA technology used nowadays is the *static random-access memory* (SRAM) based FPGA. The programmability is achieved by using the SRAMs to implement programmable logic blocks and to control programmable routing resources. Due to the fact that SRAM cells can be rewritten by the users, the SRAM based FPGAs have the feature called *field re-programmability*. The basic logic blocks are commonly realized by a 2^K-bit

SRAM cell, which represents a K-input one-output *lookup-table* (K-LUT). A K-LUT can be programmable to implement any Boolean function for up to K variables. An example of 2-input LUT is shown in Figure 2.1 (a). Flip-flop can be incorporated into a LUT-based logic cell to implement sequential functions. A *fine-grained* logic cell with a single flip-flop is shown in Figure 2.1 (b). A *coarse-grained* logic cell typically contains multiple LUTs, multiplexers, and flip-flops.

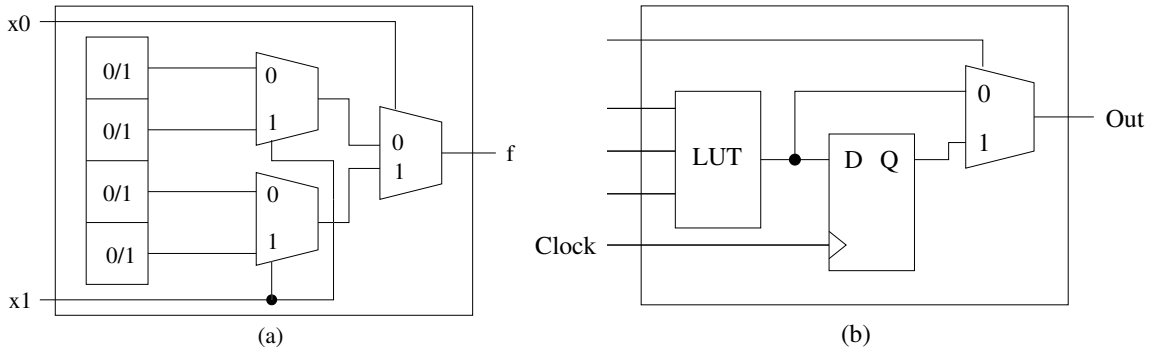


Figure 2.1. (a) An Example of 2-input LUT. (b) A Fine-grained Logic Block with a Flip-flop.

The LUT-based FPGAs are provided by several FPGA vendors, such as Xilinx [6], Altera [22], and AT&T [23]. Other than the LUT-based FPGAs, the logic cells can also be implemented by multiplexers which is usually composed of a tree of 2-to-1 MUXes. Figure 2.2 shows the multiplexer-based logic module employed by Actel [24].

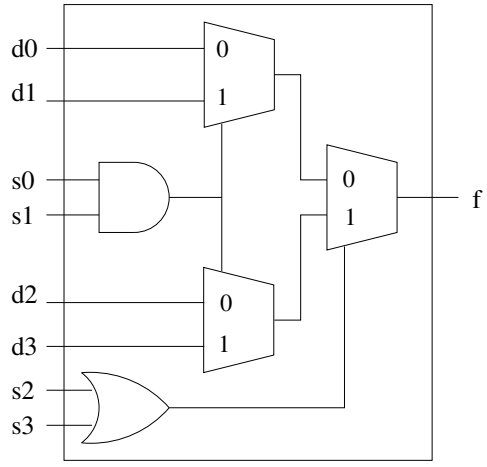


Figure 2.2. Multiplexer-based Logic Module Used by Actel.

Our research work is mainly concentrated on LUT-based FPGAs but can be extended to other architectures as well. While the FPGA manufactures's products have their own unique features, all FPGAs that are commercially available share some common characteristics such as an array of logic blocks and programmable routing resources. Figure 2.3 illustrates the common architecture of the array-based FPGA.

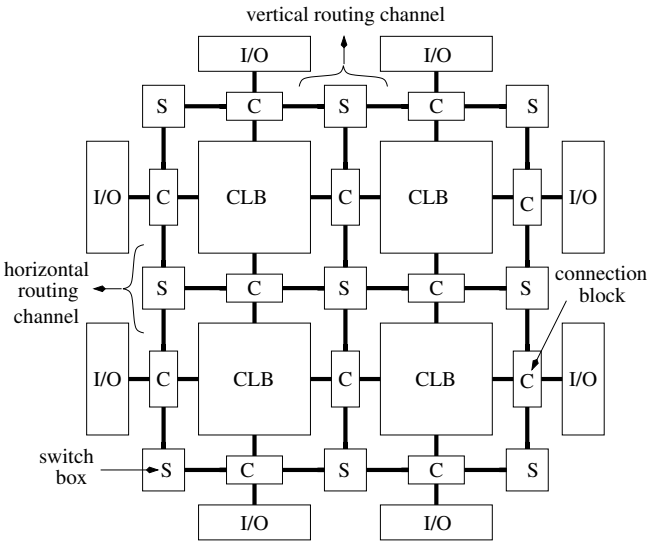


Figure 2.3. Architecture of an Array-based FPGA.

The configurable logic blocks, denoted as CLB in Figure 2.3, are customizable to implement the logic functions. The connection block, denoted as C in Figure 2.3, connect the CLB pins to the routing channels. A horizontal and vertical routing channel are connected via a switch box denoted as S in Figure 2.3. A switch block is comprised by a number of programmable switches which account for the connections of FPGA routing. Usually, the switches have higher resistance and capacitance, and hence result in significant delays. The routing channels are segmented in order to balance the circuit performance and routability. Routing tracks consist a set of wires with different lengths where longer wires are desirable for timing-critical nets and shorter wires are intended for short connections to save routing resources.

2.2 CAD Flow for FPGA Physical Design

Typically, a CAD system consists of design tools for performing the following tasks:

- *Design entry* provides the designers the options to enter the description of the digital system in the forms of schematic or Hardware Description Language (HDL) code.
- *Initial synthesis* creates an initial circuit according to the data entered in the design entry phase.
- *Functional simulation* is to verify the functionality of the design.
- *Logic synthesis and optimization* utilizes a certain optimization algorithms to obtain an optimized circuit, based on the optimization objectives specified by the designers.
- *Physical design* generates the layout on the optimized circuit for the given target technology (full-custom, semi-custom, FPGA, etc.).
- *Timing simulation* computes the expected propagation delay of the implemented circuit.
- *Chip configuration* configures the chip to implement the designed system.

In Figure 2.4, we show the typical design flow of a CAD system. Below we will describe the details of the CAD flow.

2.2.1 Design Entry

Design entry is the starting point of the CAD system where the designers give the description of the circuit being designed into the CAD tool. Currently, there are two major approaches of design entry: schematic capture and hardware description language (HDL).

1. Schematic Capture:
   - A schematic capture tool allows the users to draw a schematic diagram to describe the circuit using graphical symbols.

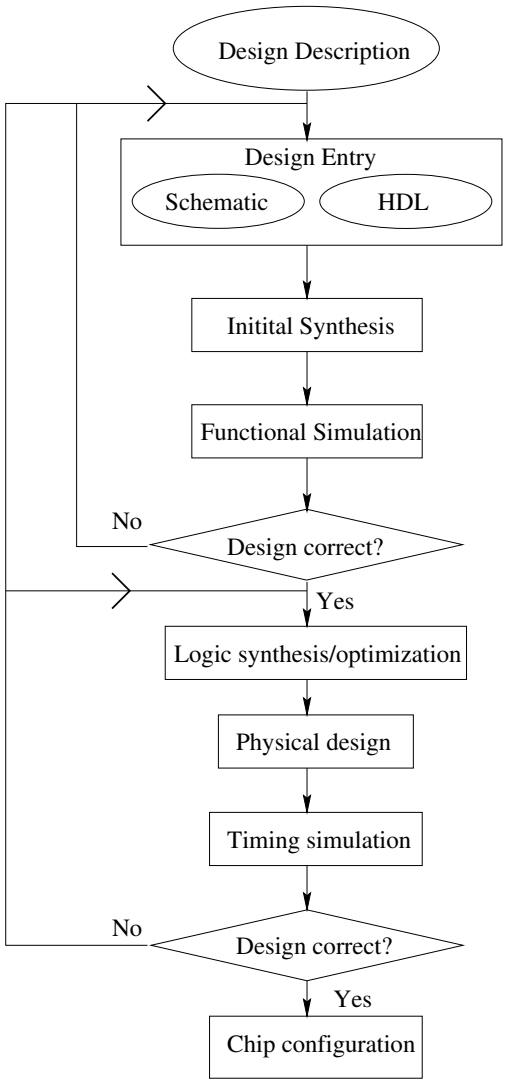


Figure 2.4. Typical Design Flow of a CAD System.

- There are built-in libraries which contain the logic gates of different types. The users select the gates in the libraries to use in their schematics.
- Previously designed circuit or sub-circuit can be created as a symbol and hence can be reused. This mechanism makes it possible to design the system in a hierarchical manner.

2. Hardware Description Languages (HDLs):

- HDLs are similar to other programming languages but are designed to describe (structural or behavioral) the hardware. VHDL and Verilog are the most common HDLs in use today.
- The most distinct difference of HDL from other programming languages is its capability of representing parallel operation.
- HDLs are more popular than schematic capture for use in design entry, and CAD tools synthesize the HDL code into a hardware implementation of the described digital system.
- Advantages over schematic capture:
  * Portable: A circuit presented in HDL format can be synthesized and implemented by different CAD tools and chips from various companies. The HDL description need not to be changed.
  * Reusable: It is easy to share and reuse HDL-specified circuits.
  * Hierarchical design capable: HDL codes can be described in a modular way to facilitate hierarchical design.

2.2.2 Initial Synthesis

Once the design is entered through design entry, the HDL code or schematic diagram is translated into a logic gate network. The output of the initial synthesis tool is a lower-level description of the circuit in an appropriate form for use by succeeding tools. It consists of a set of logic expressions which describe the logic functions to be implemented.

2.2.3 Functional Simulation

Before the design can be logic optimized, logic expressions generated by the initial synthesis is fed into functional simulation tool to verify the functionality of the circuit. The designers specify the valuations of input test vectors and check the output of the simulation to make sure that the circuit is functioning correctly as anticipated. During functional simulation, the propagation delay associated with each signal is not being considered.

2.2.4 Logic Synthesis and Optimization

Logic synthesis tools map the design composed of simple gates or described in HDL code into an optimized circuit based on the type of logic resources available in the target device. For example, if the target device is a LUT-based FPGA, the number of inputs to the logic functions in the circuit cannot exceed the input size of the LUTs. Typically, logic synthesis can be divided into two phases:

1. Technology independent logic optimization:

   In this phase, the circuit is logic optimized without considering the resource availability in the target device. General techniques utilized to remove redundant logic and simplify the logic include factoring, decomposition, and extraction. The most popular tools used in academics are: Espresso for two-level logic optimization and SIS for multi-level circuit optimization. They are both developed by UC Berkeley. Commercial CAD tools such as Synopsys, Cadence Design Systems, and Mentor Graphics all have their own logic optimization modules.

2. Technology mapping:

   Technology mapping is to convert the original technology independent Boolean network into a functionally equivalent network such that it can be implemented by the target device. For example, for standard cell based designs, technology mapping is to map pieces of the original network with the logic cells available in the standard cell library. And for LUT-based FPGA technology mapping, the target device cell library is simply the LUTs. We will discuss the details of the algorithms on technology mapping for LUT-based FPGAs in Section 2.3. Generally, the optimization objectives of technology mapping include:

   - minimizing the total area of the cells needed covering the original network.
   - minimizing the maximum circuit level of the mapped network.
   - optimizing the routability of the mapped netlist.
   - minimizing the total power consumption of the mapped circuit.

The general design flow of logic synthesis is shown in Figure 2.5.

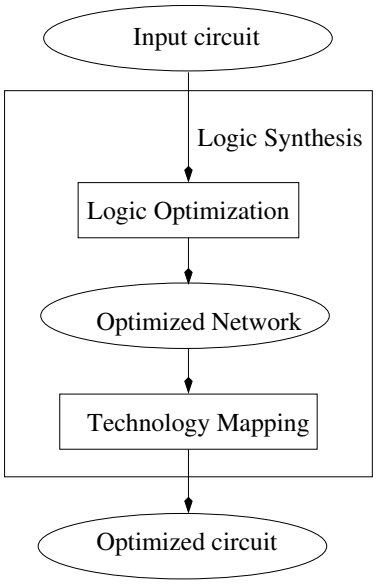


Figure 2.5. Logic Synthesis Design Flow.

2.2.5 Physical Design

Physical design is a process of transforming the *netlist* (a structural description) into the *layout* (a geometric representation). Layout is derived by converting the logic components (cells, macros, gates, transistors) into a geometric representation with specific shapes in multiple layers. The exact details of the layout also depend on design rules which are based on the electrical properties of the fabrication materials and the constraint on the fabrication process. Physical design determines where to put the logic components, how to deal with the interconnect, etc. The objectives of physical design include small area, high performance, low power consumption, feasibility, etc. In current DSM design regime, physical design has become a very challenging task because the circuits are composed of millions of transistors and interconnect, multiple objectives and constraints at the same time. Consequently, computer aided design (CAD) is very imperative for the chip designers. The physical design flow is shown in Figure 2.6.

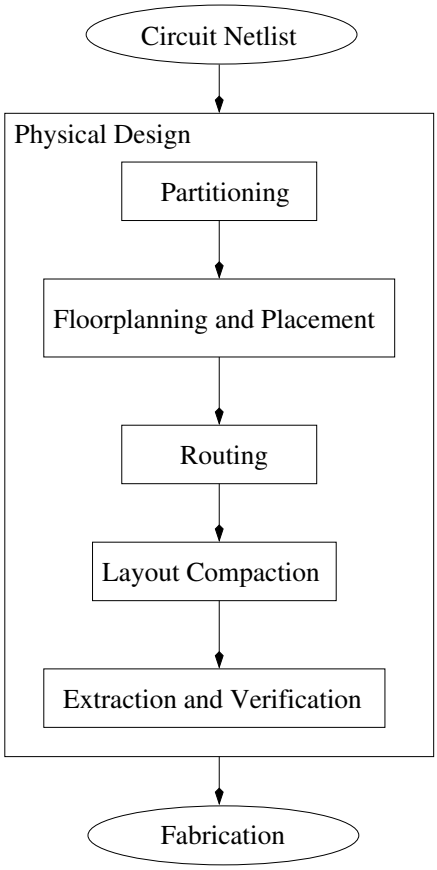


Figure 2.6. Physical Design Flow.

2.2.5.1 Partitioning

VLSI systems are becoming more and more complex and makes it almost impossible to implement a whole system on a single chip. And too large chip area will severely hurt the yield rate which in turn increases the cost of chips. So the system is usually partitioned into sub-systems (blocks). This gives the designers the flexibility of changing part of the system without worrying about the other parts. Subsystems can be designed independently and simultaneously to expedite the entire designing process. Since the subsystems' sizes are smaller, the designed complexity is reduced as well. For large circuits, the partitioning process is carried on in a hierarchical manner. For example, system level partitioning is executed, followed by board level partitioning and finally chip level partitioning. During the process of partitioning, several factors are considered such as the size of the blocks,

the number of the blocks, and the number of interconnections between the blocks. Some fundamental techniques used in partitioning include: Kernighan-Lin (KL) algorithm [25], Fiduccia-Mattheyses (FM) algorithm [26], Sanchis algorithm [27], and Simulated Annealing [28].

2.2.5.2 Floorplanning and Placement

Once the circuit is partitioned, the area of each sub-circuit can be estimated, the possible shapes of the blocks can be ascertained, and the number of pins needed by each block is also known. In addition, area overhead for routing needs to be taken into account. Both floorplanning and placement determine the block positions such that the area, total wire length, delay and routability for the blocks are optimized. Typically, floorplanning is carried on prior to placement. It is a planning step to deal with hierarchical design. In this phase, shapes of blocks (soft blocks) are not fixed, and pin assignment is not finalized. For placement, the shapes of blocks are fixed (hard block), and pin assignment is fixed too. For full custom design, floorplanning is followed by placement. While for standard cell design, floorplanning is the same as placement because the shape, pins of the cell are predefined. We will elaborate the algorithms for floorplanning and placement in Chapter 4.

2.2.5.3 Routing

The routing process is to complete the interconnections between all blocks according to the specified netlist. Routing resources including wires and switch boxes are located in horizontal and vertical regions between blocks called *routing channels*. Generally, routing is done in two phases, referred to as the *Global Routing* and the *Detailed Routing*. In global routing, each net is assigned to particular routing regions without specifying the actual geometric layout of wires and pins. This is shown in Figure 2.7 (a). In detailed routing, the router determines the exact geometric layout of each net within the assigned routing regions as shown in Figure 2.7 (b). Due to the limited routing resources available on a chip, in many cases complete routing of all the interconnections cannot be guaranteed. Therefore,

rip-up and *re-route* is used to remove connection in congested area and reroute them in a different order.

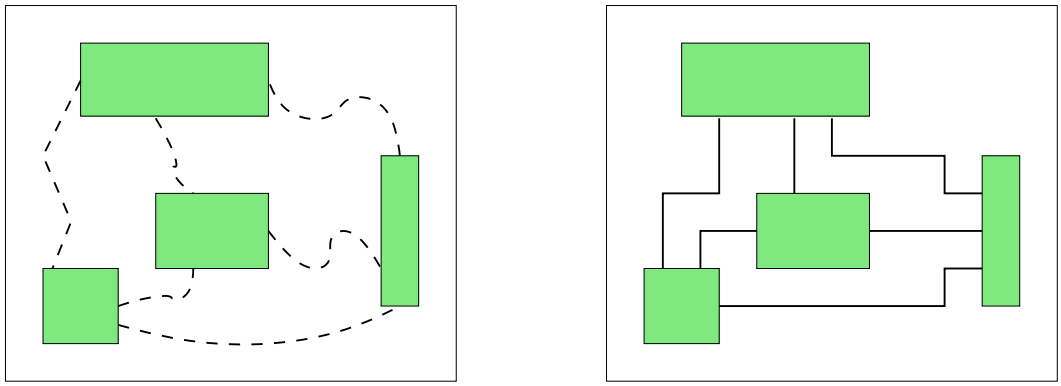


Figure 2.7. Global and Detailed Routing.

2.2.5.4 Compaction

After the detailed routing is done, the layout is ready for fabrication. Layout compaction is the task of reducing the chip area. It is necessary because the place and route tool may not generate optimal layouts. Some vacant space may exist in the layout. By moving objects closer to each other, area and wire lengths are decreased. And smaller chip area implies that more chips can be produced on a wafer, which in turn reduce the cost of manufacturing. Note that compaction should not violate any design rules.

2.2.5.5 Extraction and Verification

Before the chip is manufactured, design rule checking (DRC) is imposed to verify that all geometric patterns satisfy the design rules for the fabrication process. Once the design rules and checked and violations are removed, circuit extraction is executed to verify the functionality of the layout. It generates the circuit representation from the layout and the extracted description is compared with the original circuit description to verify its correctness. This process is known as *Layout Versus Schematics* (LVS) verification. Meanwhile,

it also calculates accurate timing of the components including interconnect to verify the performance. The extracted information is also used to check the reliability of the layout to ensure that the layout will not fail due to electro-migration, self-heat, and other effects [29].

2.2.6 Timing Simulation

Timing simulation is performed once the physical design is completed for the chosen technology device. It is to verify the circuit implemented in the target technology satisfies the anticipated timing requirement. Timing simulation simulates the actual propagation delays in the target technology. Hence, it gives a fairly precise indication of the performance before the chip is finally configured and fabricated.

2.3 Technology Mapping for LUT-based FPGAs

Logic synthesis for LUT-based FPGAs converts network of logic gates into functionally equivalent K-LUT network. This process is generally divided into two phases: *logic optimization* and *technology mapping*. Logic optimization reduces the complexity of the network based on a cost function. This is typically done by removing redundancies and common sub-expressions to reduce the circuit size or by resynthesizing the critical paths to reduce the circuit delay. For technology mapping, *gate decomposition* and *LUT mapping* are carried out. Large gates are decomposed into gates with at most K inputs (i.e., K-bounded). The K-bounded network is then covered to K-LUTs in the LUT mapping phase. Note that LUTs may overlap, which means the overlapped portion of logic will be duplicated into each of these overlapping LUTs. If no logic duplication is allowed, the mapping is called a *duplication-free mapping*. For example, for the Boolean network shown in Figure 2.8 (a), a duplication-free mapping is shown in Figure 2.8 (b) and a general mapping allowing overlapping is given in Figure 2.8 (c). Note that the separation of optimization and mapping is artificial. Some LUT synthesis algorithms such as [30] [31] decompose collapsed networks directly into LUT networks.

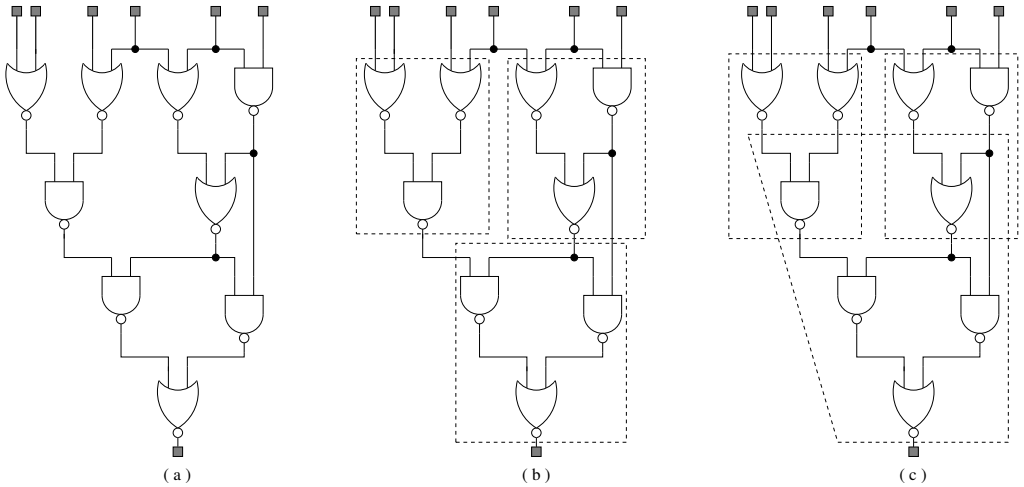


Figure 2.8. Various LUT Mappings for a Boolean Network ($K = 4$): (a) Original Boolean Network; (b) Duplication-free Mapping; (c) Mapping with Overlapping LUTs.

Previous LUT-based FPGA technology mapping algorithms can be broadly classified according to their primary optimization objectives: algorithms that minimize area (i.e., minimizing the number of LUTs) [32] [33] [34] [35] [36] [37], algorithms that minimize delay (i.e., minimizing the number of LUTs on the longest path) [38] [39] [40] [41], algorithms that focus on routability [42] [43], and algorithms that minimize both area and delay [44] [45] [46].

2.3.1 Technology Mapping for Delay Optimization

The delay of a LUT network can be estimated by the number of levels (or *depth*) in the network using the unit delay model. For depth and general static delay minimization, mapping for each node can be optimized independently without worrying about logic sharing. This is so, because, logic can be duplicated whenever needed. Therefore, the depth optimal mapping of node v only depends on the mapping of nodes in the subnetwork rooted at v, denoted as N_v. Because a mapping of N_v consists of LUT_v and a mapping of $N_v - LUT_v$, and optimal mapping of N_v selects the "best" LUT_v to minimize the delay of the optimal mapping of $N_v - LUT_v$, using dynamic programming. Usually, a *label* for each node in a topological order is assigned to lead the dynamic programming procedure

and decides LUT_v for each node v. Delay-oriented FPGA mapping algorithms can be classified into two classes. The first class of algorithms, such as Chortle-d [47], DAG-Map [38], and FlowMap [39], perform LUT mapping without logic resynthesis. Chortle-d guarantees depth-optimal technology mapping for simple-gate tree networks. It partitions the input into leaf-directed acyclic graphs (leaf-DAGs). Each leaf-DAG is mapped separately as a tree for area minimization using dynamic programming method by enumerating all possible LUT implementations of the root node. In addition, it minimizes area as a secondary objective by using area optimal node decomposition along noncritical paths and depth optimal node decomposition along critical paths, as well as predecessor packing. DAG-Map employed a classical labeling algorithm called *Lawler's labeling* [48]. Lawler's labeling is a monotonic labeling procedure where the labels along any path from a primary input (PI) to a primary output (PO) are nondecreasing with $l(v) = 0$ for any PI node v. DAG-Map has to deal with K-bound input network in order to find a feasible mapping solution and it does not guarantee delay optimality. FlowMap overcomes this drawback and guarantees depth-optimal LUT mapping for general K-bounded networks. It formulates the problem of finding a *minimum height K-feasible cut* $(X, \overline{X})$ of N_v, where the height $h(X, \overline{X})$ is defined to be the largest label of nodes in X. The key idea is to compute a minimum height K-feasible cut for every node v. FlowMap also uses $node - splitting$ transformation and *maximum flow computation*. It is applicable to not only K-bounded network but also to any K-mappable network. FlowMap-r [44] and CutMap [45] extend FlowMap to reduce the area while keeping depth optimality. FlowMap-d [49] and Edge-Map [50] minimize delay with a more accurate net delay model. Under dynamic delay models, the delay of a net is associated with its structure in the mapping solution, and different branches of a multi-fanout node will interact. Consequently, the optimal mapping of v depends not only on the optimal mapping of nodes in N_v, but also on that of nodes outside N_v. So dynamic programming cannot be used with this class of delay models. It was shown by Cong and Ding in [49] that the delay optimal mapping problem is NP-hard for $K \geq 6$. This also shows that the duplication-free mapping on general networks for dynamic nominal delay minimization

is NP-hard for $K \geq 6$. A heuristic was presented in [49] to incrementally adjust the static delay based on the dynamic nominal delay as the mapping process proceeds.

The second class of LUT mapping algorithms, such as MIS-pga-delay [51], TechMap-D [52], and FlowSyn [53] collapse critical paths followed by delay-oriented logic resynthesis. MIS-pga-delay is an extension of the UC Berkeley MIS-II logic synthesis system [54] to FPGA synthesis. It tries to collapse each critical node into its critical fanouts in a topological order. If such a collapse is K-feasible, or can be made K-feasible by decomposition with increasing the total level, it is accepted. This operation is repeated until no more collapse is possible. The best result of all applicable approaches is selected for the final mapping solution. TechMap-D is a combined area and depth minimization algorithm. It uses a cost function that represents the tradeoff of depth, area, and input size. It also has a placement phase, but is performed separately after mapping without resynthesis. FlowSyn improves FlowMap from another angle. It targets at further enhancement of the depth minimization by incorporating logic optimization into the technology mapping procedure. In general, this class of algorithms could achieve mapping solutions with smaller delay the optimal depth obtained with FlowMap due to resynthesis. But they suffer from longer computation times.

2.3.2 Technology Mapping for Area Minimization

Unlike delay minimization mapping, area minimization cannot be carried out independently in each subnetwork N_v, since the LUT sharing among overlapped subnetworks must be taken into account. It was shown by Levin and Pinter [55] that for K=4 the problem of determining whether to duplicate a node of multiple fanouts or implement it as a LUT for area-optimal mapping is NP-hard. This result was further generalized by Farrahi and Sarrafzadeh [33] to $K \geq 5$. This implies that duplication-free mapping alone will not achieve optimality. There are two main dimensions in the solution space: to select a subset of nodes to be implemented by LUTs, or to select a K-feasible cone to be covered by its LUT implementation. We further classify the mapping techniques for area minimization as:

- *Node selection based enumeration*: Node selection based enumeration generates all node subsets for LUT implementation, and for each chosen subset determines the LUT implementation of each node. Techniques proposed in [45] [49] [56] fall into this class.

- *Node covering based enumeration*: Node covering based enumeration first generates all LUT implementations of the nodes, and then selects a subset of them to implement. This was proposed by Murgai et. al. in [57]. For each node v, it creates all possible LUT implementations (called *supernodes*) of v. Then, it selects a subset of the supernodes under the condition that if one supernode is selected each of its inputs must be a primary input or generated by another selected supernode.

- *Integer linear programming*: Integer line programming formulation was proposed by Chowdhary and Hayes [58] for area minimization. Each node v is associated with a variable $e(v) \in \{0, 1\}$, where $e(v) = 1$ indicates v is visible in the mapping solution. The objective is to minimize $e(v)$ which is the total number of LUTs in the mapping solution, under a set of linear constraints that specify the LUT size constraints, and LUT size evaluation with consideration of reconvergent paths in the network.

- *Node selection based heuristics*: Genetic algorithm is used in node selection. A node subset is represented by a bit string where each bit represents a node. If the value of the bit is 1, it implies that this node is selected. This method was used in [59].

- *Node covering based heuristics*: This class of heuristics can be regarded as approximation to the node covering based enumeration. Instead of enumerating *all* possible supernodes of v, the heuristic only produce a subset of supernodes trying to pack as many nodes into each supernode as possible or sharing as many inputs among the supernodes as possible. The packing based approach was introduced in [60] known as $flowpack$. It was later improved in *cutmap* [45] which computes a *minimum cost* K-feasible cut. The cost of a cut $(X, \overline{X})$ is defined as the sum of the costs of nodes in $input(\overline{X})$. Cutmap assigns lower costs to nodes that are predicted to be implemented

by LUTs. Once a high-cost node is implemented by a LUT, its cost will be lowered. This algorithm is also able to generate delay optimal mapping solutions.

2.3.3 Technology Mapping for Routability and Low Power

Compared to mapping algorithms in area/delay optimization, very limited work has been reported on routability driven technology mapping. Schlag et. al. proposed an approach in [42] using a heuristic cost function to guide the mapper. Another algorithm reported by Togowa et. al. in [43] combines mapping with placement and routing.

Recently, with the increasing popularity of wireless devices and portable computers, reducing power consumption has become an important issue. However, relatively few work has been done on minimizing power consumption in LUT-based technology mapping. In one of the works on low-power technology mapping for LUT-based FPGAs, Farrahi and Sarrafzadeh [2] introduced a low-power driven mapping algorithm at the expense of increase in depth and the number of LUTs. Wang et. al. [1] presented another algorithm to reduce the power consumption by enumerating a predefined number of cuts and choosing the one with smallest power consumption as the final mapping. Li et. al. proposed several algorithms [61] [62] [63] to reduce the power consumption by computing low-power K-feasible cut. An extension of their work [63] also guarantees optimal delay while achieving power reduction.

2.4 FPGA Placement

In current deep sub-micron (DSM) regions, designs often have over a million logic components. Placement is one of the most persistent steps in design automation as it directly defines the on-chip interconnects which have been the bottleneck to determine the system performance. Therefore, due to the dominance of interconnect delay in DSM technology, placement has become the major factor to affect timing [64]. Essentially, in the process of placement, all logic units are placed in such a way that the design can be completely routed, while satisfying a certain number of constraints or optimization objectives. The

placement problem is in nature a very difficult problem. For example, the placement of blocks in order to minimize the total wirelength is an NP-complete problem [65]. Over the past years, a large number of heuristic algorithms have been developed for solving the placement problem with a set of optimization objectives. Generally, the optimization objectives of placement algorithms include: minimizing the wiring (wirelength-driven) [66] [67] [68], balancing the wire density (routability-driven) [69] [70], and maximizing the circuit speed (timing-driven) [71] [72] [73] [74].

2.4.1 Optimization Objectives of Placement

Layout design consists of *placement* followed by *routing*. Therefore, an acceptable placement first has to be routable within the given layout area. Since the routing information will not be available until the placement is done, *estimation* is used during placement. The speed of estimation has an important effect on the performance of the placement algorithm. It must be as quick as possible and the estimation error must be the same for all nets, i.e., it cannot be skewed. Several commonly used techniques for estimation of wirelength are given below.

2.4.1.1 Estimation of Wirelength

The most common assumption in estimating the total wirelength is that routing uses Manhattan geometry which means routing tracks can only be either horizontal or vertical. For a two pin net connecting block i and block j, the Manhattan distance of this net is $r_{ij} + c_{ij}$, where r_{ij} and c_{ij} are the number of rows and columns separating these two blocks. For multi-pin nets, various estimation techniques are developed.

- Half-perimeter: This is the most widely used estimation method and it is efficient. It first finds the smallest bounding rectangle that encloses all the nets to be connected. The estimated wirelength is half the perimeter of this bounding box. For circuits having only two and three pin nets and assuming no detours in actual routing, this

method provides the best estimation. For heavily congested chips, however, this scheme tends to under-estimate the actual wirelength.

- Complete Graph: Each n-pin net is represented as a complete graph which has $\frac{n(n-1)}{2}$ edges. Since a tree has $(n-1)$ edges which is $\frac{2}{n}$ times the number of edges in the graph, the estimated wirelength is computed using Equation 2.1:

$$Wirelength = \frac{2}{n} \sum_{\forall\ pair \in net} (pair\ distance) \tag{2.1}$$

- Steiner Tree Approximation: A *Steiner tree* is the shortest route for connecting a set of pins. A wire can branch from any point along its length to connect to other pins of the net. The problem of finding the minimum Steiner tree is NP-complete. So approximation algorithm such as Lee algorithm [75] is used to find an approximate steiner tree by propagating a wave for the entire net.

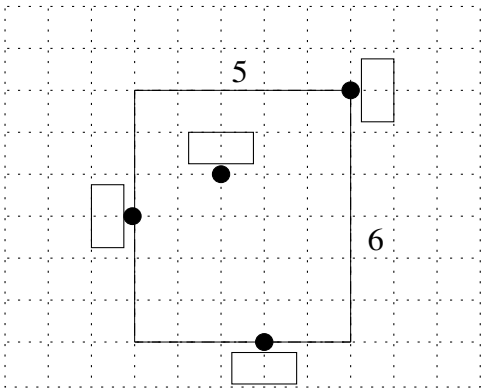
- Minimum Spanning Tree: In a minimum spanning tree, branching is only allowed at the pin locations. There are algorithms to find the minimum spanning tree in polynomial time such as Kruskal's algorithm and Prim's algorithm [76].

In Figure 2.9, we show some examples to illustrate the wirelength estimation techniques discussed above.

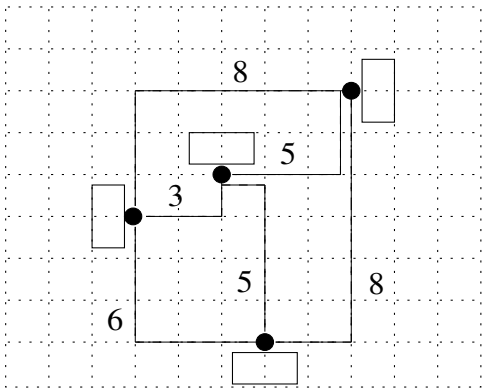
2.4.1.2 Minimize Total Wirelength

To place a circuit which is routable, the area used by the routing wires should be minimized. One common approach to achieve this is to place strongly connected nets close to each other. The corresponding objective function to minimize the total weighted wirelength over all nets, $L(P)$, is computed as shown in Equation 2.2.

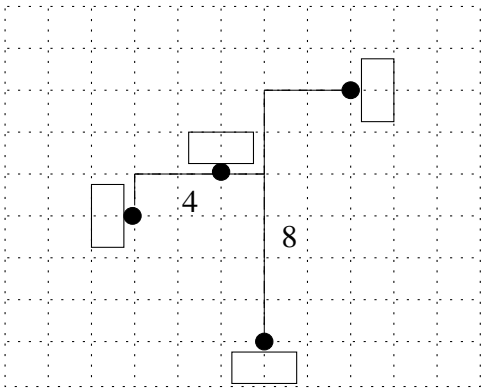
$$L(P) = \sum_{n \in N} w_n \cdot d_n \tag{2.2}$$



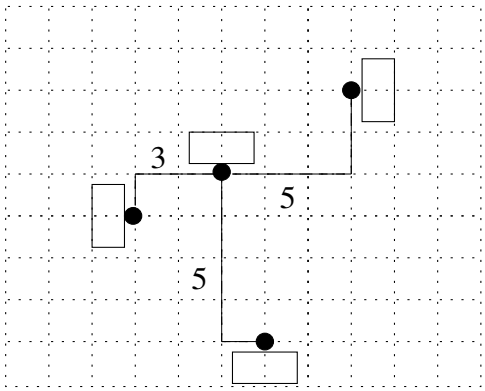
(a) Half-perimeter length=11



(b) Complete graph length 2/n = 17.5



(c) Steiner tree length=12



(d) Minimum spanning tree length=13

Figure 2.9. Different Techniques for Wirelength Estimation.

where, d_n is the estimated wirelength of net n, w_n is the weight of net n, and N is the total number of nets. Using this method, the length of each net is computed independently. Hence it is not very accurate.

2.4.1.3 Minimize Maximum Density

The routability of a placement can be estimated by the *density*, denoted as $D(P)$. For each routing channel, there exists a maximum number of wires that can pass through this channel. We call this number $c(e_i)$ as the *routing capacity* for this channel. Given a placement P, let $n(e_i)$ denote the estimated number of nets passing through edge e_i of a channel. Then the density of edge e_i is defined as:

$$d(e_i) = \frac{n(e_i)}{c(e_i)} \tag{2.3}$$

To have a routable placement, $d(e_i)$ cannot exceed 1. The routability of the placement is then given in Equation 2.4.

$$D(P) = \max_i \left[d(e_i)\right] \tag{2.4}$$

where the maximum value of $d(e_i)$ is among all edges e_i in the layout area.

2.4.2 Placement Approaches

As we have mentioned previously, the placement problem is NP-complete. This is not affected once cost functions and constraints are taken into consideration. For current large digital system designs, the solution space is so large that enumerative techniques are practically prohibitive. Therefore, developing heuristic methods which have short execution time and generate *good* solution is the most appropriate choice for researchers and CAD tool engineers. A heuristic algorithm is either *constructive* or *iterative*. As the name implies, a constructive placement approach constructs a solution by placing one block at a time. At the end of each step, we have a *partial placement* and there are two decisions to be made: (i) which unplaced block should be selected and added to the partial placement? and (ii)

where should this block be placed? Various heuristics are adopted to make the decisions. Typically, constructive placement technique is *greedy*. At each step, it makes the best possible selection. This method does not guarantee optimum solution because each decision is made in the *absence of complete information*.

The constructive placement algorithms in use today mainly use one of the following three approaches: min-cut (partitioning-based), analytic-based (numerical-based), and simulated annealing based. In the next sub-sections, we will provide more details on these approaches.

2.4.2.1 Partitioning-based Placement Algorithms

Partitioned-based placement algorithms repeatedly partition the circuit into two circuits. Meanwhile, at each level of partitioning, the available layout area is divided into horizontal and vertical subsections alternately. Each of the circuits is assigned to a subsection. This process is carried out until every circuit consists of a single block and has a unique position on the layout area. During partitioning, the number of nets that are cut by the partition is usually minimized. Consider a rectangular layout space as shown in Figure 2.10. The vertical line, denoted as Cut_v, dividing the layout area into top region and bottom region is called a *vertical* cut. The horizontal line, denoted as Cut_h, dividing the layout area into left region and right region is named a *horizontal* cut. The *cut size* of a cut is defined as the number of nets that cross the cutline. For example, in Figure 2.10, Cut_h has a cut size of 3 and Cut_v has a cut size of 4. For a given placement P, let $H(P)$ denote the maximum cut size among all horizontal cuts, and let $V(P)$ denote the maximum cut size among all vertical cuts. By minimizing $H(P)$ and $V(P)$, the routability of a gate-array placement will be improved and hence improving the wirelength and timing.

Partitioning-based placement was first proposed by Breuer [77] [78]. Following work include [69] [79] [80] [81]. Several objective functions employed by these algorithms are given below:

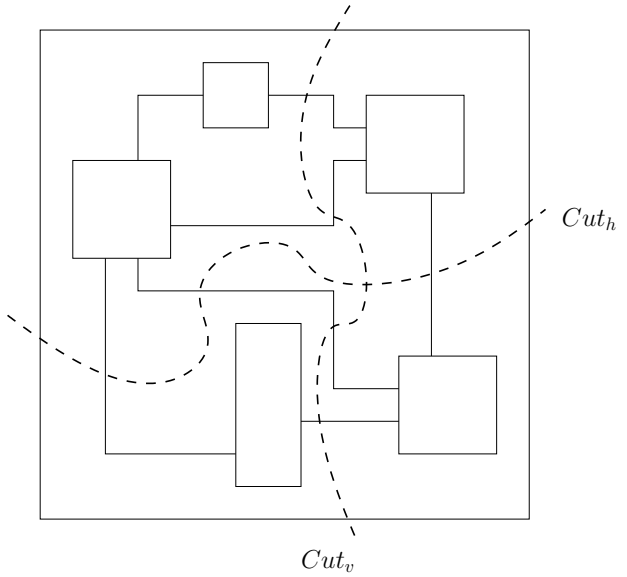


Figure 2.10. Vertical and Horizontal Cutlines for a Placement.

1. Total net-cut: All nets cut by the partitioning (horizontal and vertical cut lines) are considered. It is shown that minimizing the sum is equivalent to minimizing the half-perimeter wirelength [77] [78].

2. Min-max cut value: The objective is to minimize the number of nets cut by the cut line across the channel. This reduces the channel congestion which leads to smaller channel width and chip area. Some nets have to be routed with detours using less condensed channels.

3. After each partition, the number of net cuts is minimized. It is basically a greedy approach so it may not minimize the total number of nets cut.

In addition to the above objective functions, several placement procedures in which various sequences of cut lines are developed.

- Cut oriented min-cut placement: The circuit is first cut by a partition into two circuits such that the net cut is minimized. All the circuits are further partitioned by the second cut line and this procedure is carried out for all cut lines. This method is sequential and easy to implement. It is illustrated in Figure 2.11 (a).

- Quadrature placement: Each region is partitioned into four regions of equal sizes by applying horizontal and vertical cut lines alternatively as shown in Figure 2.11 (b). During each partitioning, the cutsize of the partition is minimized. This procedure reduces the routing density in the center of the layout area. It is the most popular sequence of cut lines for min-cut based algorithms.

- Bisection placement: The layout area is repeatedly bisected (partitioned into two equal parts) by horizontal cut lines until each subregion consists of only one row. Then, these rows are bisected with vertical cut lines till each subregion contains only one slot. This procedure is mainly used in standard cell placement and it is shown in Figure 2.11 (c).

- Slice bisection placement: A number of blocks are partitioned from the rest of the circuit using horizontal cut line and assigned to a row, called as a *slicing*. This is repeated until every block has been assigned to a row. Then the blocks are assigned to columns by bisecting with vertical cut lines. This method is most suitable for circuits having a high degree of interconnection at the periphery. Figure 2.11 (d) shows the sequence of cut lines for this procedure.

However, layouts derived by solely partitioning blocks and assigning them to regions are not good enough. The main reason is that this procedure does not take into account the location of external pins, the probable locations of blocks in the final placement and signals which enter a group of blocks. A method called *terminal propagation* was developed by Dunlop and Kernighan [79]. In this work, dummy terminals are generated such that external pin positions are taken into account while computing the min-cut.

2.4.2.2 Analytic-based Placement Algorithms

The placement problem can often be transformed into an analytic-based optimization problem. Then it is reduced to the problem of solving a set of simultaneous linear equations to determine the equilibrium positions (*ideal x-y coordinates*) for the logic blocks. This class

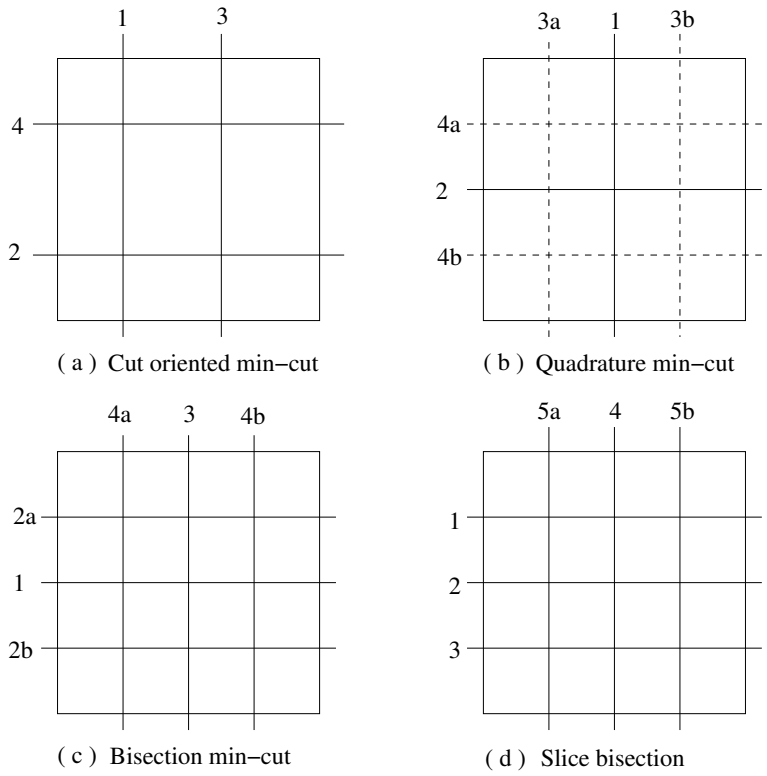


Figure 2.11. Different Sequences of Cut Lines for Min-cut Placement.

of algorithms are usually based on a *quadratic wirelength* objective. The first quadratic placement algorithm was proposed by Hall in [82]. The cost of connecting a pair of blocks B_i and B_j, denoted by c_{ij}, is given in a connectivity matrix $C = [c_{ij}]$. Then minimizing the quadratic wirelength is equivalent to minimizing the total wirelength of the circuit which is shown in Equation 2.5.

$$Wirelength = \frac{1}{2} \sum_{i,j=1}^{n} c_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2] \tag{2.5}$$

where (x_i, y_i) are the coordinates of the logic block i on the layout area. The quadratic wirelength objective function can be rewritten using matrix notation as in Equation 2.6.

$$Wirelength = \overrightarrow{x}^T B \overrightarrow{x} + \overrightarrow{y}^T B \overrightarrow{y} \tag{2.6}$$

where matrix $B = [b_{ij}]$ has entry b_{ij} as given in Equation 2.7:

$$B = [b_{ij}] = \begin{cases} -c_{ij} & i \neq j, \\ \sum_{j=1}^{n} c_{ij} & Otherwise \end{cases} \tag{2.7}$$

A trivial solution for the above objective function is: $x_i = y_i = 0$ for all i. Apparently, this is not what we want to have since every block will be placed at the same location. It was proved in [82] that a nontrivial solution is derived and the total wirelength is minimized if the smallest eigenvalues of the matrix B are chosen. The corresponding eigenvectors X and Y then give the coordinates of all the blocks. This technique is further improved in [66] [83] [84] [85].

2.4.2.3 Simulated Annealing Placement Algorithms

Simulated annealing is applied in placement as an iterative improvement algorithm and it is one of the most sophisticated placement techniques in use currently [28] [68] [71] [86] [87]. The place-and-route package named TimberWolf developed by Sechen [86] was the earliest package to use simulated annealing to the placement problem.

In simulated annealing, a series of random moves (move one block to another location, swap the locations of two blocks, etc.) are executed on an existing solution. Moves that result in a decrease in cost are accepted. Cost is often defined as the total wirelength or area required for the placement. Moves that result in an increase in cost will be accepted with a probability which reduces over the iterations. This is useful because it can jump out of *local minimal* which are very common in real designs. A parameter known as *temperature* T is introduced to control the probability. The probability of accepting a *bad* move decreases along with the decrease in temperature T. The acceptance probability is given by $e^{\frac{-\Delta C}{T}}$, where ΔC is the increase in cost. Simulated annealing based algorithms start with a very high temperature which gradually decreases so that moves that increase cost have lower probability of being accepted. The algorithm terminates once the temperature is below a

Algorithm: Simulated Annealing Based Placement

```
1  begin
2     P = RandomPlacement();
3     T = InitialTemperature();
4     while (ExitCriterion() == False) do
5        while (InnerLoopCriterion() == False) do
6           P' = NewPlacement(P);
7           Δ ← cost(P') − cost(P);
8           if Δ ≤ 0 then P ← P'
9           else if (Random(0,1) > e^(−Δ/T)) then P ← P'
10       end while;
11       T = CoolingSchedule(T);
12    end while;
13    return P;
End algorithm;
```

Figure 2.12. Outline of Simulated Annealing Placement Algorithm.

predefined threshold and it converges to an optimal or sub-optimal solution. The outline of simulated annealing based placement algorithm is illustrated in Figure 2.12.

The parameters and functions used in the simulated annealing algorithm determine the quality of the placement generated. For example, the cooling schedule controls the speed the temperature decreases. If the temperature decreases slowly, the quality of the placement will be generally better but this leads to longer running time. On the other hand, if the cooling schedule is designed in such a way that the algorithm converges quickly, the execution time can be saved but at the expense of possible worse placement. Therefore, various cooling schedules have been developed to achieve a better tradeoff. The advantage of simulated annealing based placement algorithms is the ease of adding new optimization objectives or constraints. But the drawback is often very undesirable which is the excessive running times needed.

2.4.2.4 Summary of Different Placement Algorithms

For large problems, analytical placement approach is preferred because it is fast. It usually uses a quadratic wirelength objective which can be minimized very efficiently. It is typically employed on a flat scheme in order to sustain the global view of the design. One drawback of analytical based placement algorithms is that it tends to create large amount of overlaps and the quadratic objective function is only a coarse indicator of the wirelength. For simulated annealing and partitioning based algorithms, a hierarchical methodology is generally utilized to reduce the problem size and hence speed up the resulting algorithms. Recently, Hu et. al. proposed a method to convert a flat design to a hierarchical approach by incorporating the fine granularity clustering technique [88].

In this dissertation, we focus on the FPGA placement problem for enhancing timing as well as netlength. Below, we will discuss the details of several placement algorithms proposed by previous researchers for optimizing timing.

Previous timing-driven placement algorithms try to minimize the longest path delay or maximize the minimum slack value of the circuit. Existing algorithms can be divided into two categories: *path-based* algorithms [71] [72] [83] and *net-based* algorithms [73] [85] [87]. Path-based algorithms generally minimize the longest path directly and keeps accurate timing information during optimization. Most of the approaches in this class are based on mathematical programming techniques. But they suffer from relatively high computational costs and cannot be integrated into certain placement suites. Net-based algorithms usually transform timing information into either delay or net weight constraints, and employ a weighted wirelength minimization. *Net weighting* is a commonly used approach in this class. Essentially, more timing critical nets are assigned higher weights. Net-based placement algorithms are more favorable currently because of: relatively low complexity, strong flexibility and easy implementation. With more and more complex timing constraints present in modern digital systems, such as multiple clock domains, multiple cycle paths, etc., these advantage make the net-based approach more attractive.

2.5 Conclusion

In this chapter, we discussed the details of modern VLSI design and computer-aided physical design automation. We gave special emphasis on FPGA-based technology mapping and placement. As the logic complexity and capacity on-chip have been increasing constantly, many problems in VLSI CAD domain are becoming more and more challenging. Multiple constraints (optimization objectives) in a single design phase are very common nowadays. For instance, optimizing timing, minimizing area, and maximizing routability are usually required to be met simultaneously in placement. Sometimes, these constraints may even conflict with each other (reducing area can hurt routability), which makes the problem more difficult. However, on the other hand, there still remains a large scope to be explored in VLSI CAD research. According to a recent work presented by Cong et. al. [89], the performances of the best existing algorithms in placement and partitioning have a significant gap to reach the optimum. This fact gives many researchers in VLSI design and CAD great inspiration and confidence to study and develop more advanced algorithms to attack the seemingly *tough* problems. Through all these efforts, we expect a lot of difficult problems in existence now to be solved or alleviated soon.

CHAPTER 3

LOW POWER TECHNOLOGY MAPPING FOR LUT-BASED FPGAS

In recent years, mobile computing and portable devices such as cellphone, laptop, PDA, etc., have been used more and more extensively. These devices are power sensitive and it makes power saving technology an important issue for both VLSI chip designers and CAD tool developers/researchers. In this chapter, we study the low power technology mapping for LUT-based FPGAs. As we have shown in Chapter 2, technology mapping for LUT-based FPGAs has been studied extensively for minimizing area, delay, and improving routability. However, relatively fewer work has been done in technology mapping for power saving. The problem has been proved to be NP-hard previously. Therefore, we present an efficient heuristic algorithm to generate low-power mapping solutions. The major contribution of our algorithm is that while generating a LUT, we look ahead at the impact of the mapping selection of this LUT on the power consumption of the remaining network. We choose the mapping that results in the least power consumption. The key idea is to generate and select low-power K-feasible cuts by an efficient incremental network flow computation method. Experimental results show that our algorithm reduces power consumption as well as area over the best algorithms reported in the literature. In addition, we extend this work to compute delay-optimal power saving mappings. We compute low-power K-feasible cuts to generate LUTs covering nodes on non-critical paths and compute min-height K-feasible cuts to generate LUTs covering critical nodes. Compared with Cutmap, a delay-optimal mapper with simultaneous area minimization, we achieve 14% power savings on the average without any delay penalty.

This chapter is organized as follows: In Section 3.1, we present the outline of our algorithm and briefly review two previous low power mapping algorithms. In Section 3.2,

we show the problem formulation. In Section 3.3, we give the power estimation model used in our work. In Section 3.4, we propose our power saving technology mapping algorithm. In addition, we extend this algorithm such that it guarantees the solution to be delay optimal while reducing power consumption. In Section 3.6, experimental results are given. In Section 3.7, we draw the conclusions for low power technology mapping.

3.1 Introduction

Power consumption has become a limiting factor for high performance designs and mobile computing. Figure 3.1 tracks the overall power dissipation of an FPGA chip with process technology [90]. The desired performance is to maximize the operating frequency

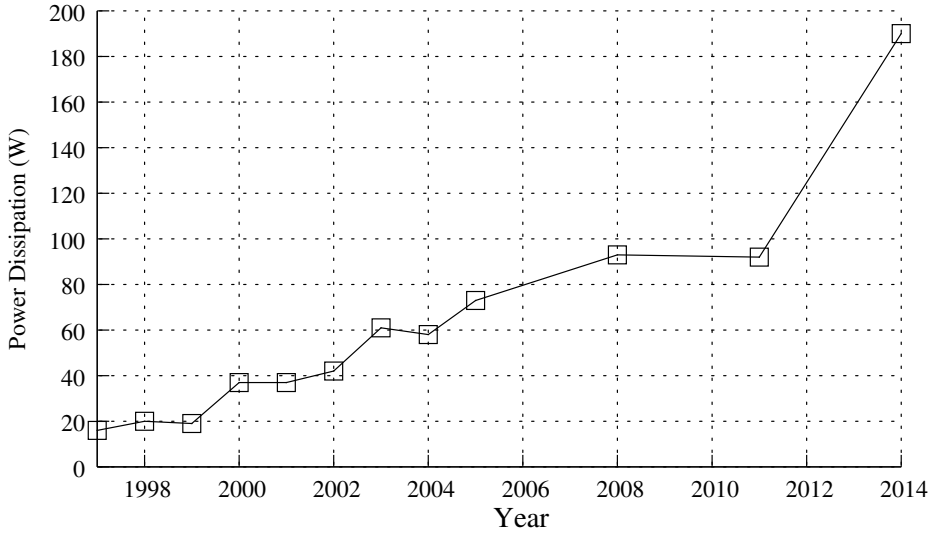


Figure 3.1. Power Dissipation of FPGAs.

satisfying power constraints. For FPGA based designs, minimizing power consumption is especially crucial, because FPGA chips are less power efficient than logically equivalent ASICs. This is due to fact that a large portion of FPGA area utilized by transistors realizing programmability. This drawback hinders FPGA devices from being used in low power applications. Therefore, it is of great importance to consider reducing power consumption in the phase of technology mapping.

The FPGA technology mapping problem for power minimization has been proved to be NP-complete [33]. Therefore, several heuristics have been proposed on LUT-based FPGA technology mapping for power reduction [1] [2] [91]. The main idea used is to hide nodes with high transition activity inside LUTs such that the overall power consumption can be minimized. In [2], Farrahi and Sarrafzadeh presented that the power consumed by a LUT depends on the transition density and the fanout number of the LUT. They also proposed an approach to estimate the total power consumption of a technology mapped circuit. One deficiency of their algorithm is that it fails to take the impact of the fanout number of LUTs on the total power consumption into account. And it occupies more area. In [1], Wang et. al. proposed another algorithm using a "cut enumeration" method to generate a predefined number of mapping solutions for the sub-circuit rooted at each node. However, it does not guarantee that if more number of mapping solutions are tested, the final mapped circuit will consume less power. All these algorithms pay expense at longer critical path delay. As part of this dissertation, we developed an algorithm known as *PowerMinMap* which not only reduces power dissipation compared to previous approaches, but also guarantees depth optimality.

The main idea is to compute low-power K-feasible cuts to generate LUTs to cover the given circuit. Experimental results show that our algorithm reduces both power and area over [1] and [2] significantly. An extension of this work is also implemented which computes delay-optimal mappings. This is done by computing min-height K-feasible cuts for nodes on critical paths and low-power K-feasible cuts for non-critical nodes. Our algorithm results in 14% power savings without any delay penalty compared to the delay-optimal Cutmap algorithm.

3.2 Problem Formulation

A general Boolean network N, can be represented as a directed acyclic graph (DAG) $N(V, E)$, where V is the set of nodes and E is the set of directed edges. Each node in the DAG represents a logic gate, and a directed edge (u, v) exists only when the output of node

u is an input of node v. A *primary input* (PI) node has no incoming edge and a *primary output* (PO) node has no outgoing edge. For a node v, $inputs(v)$ denotes the set of fanin nodes of node v. A Boolean network N is *K-bounded* if $|inputs(v)| \leq K$ for every node v in N. Given a subgraph H of $N(V, E)$, $inputs(H)$ denotes the set of distinct nodes that provide inputs to the nodes in H. A cone at node v is a subgraph C_v consisting of v and its predecessors such that any path connecting a node in C_v and v lies entirely within C_v. A cone C_v is *K-feasible* if the number of nodes feeding into C_v is no larger than K. The technology mapping problem can be also considered as partitioning the DAG representation of the circuit into K-feasible cones (equivalently, K-LUTs). Note that we allow these cones to overlap which means node duplication is allowed.

The *level* of a node v in a network is the length of the longest path between v and some PI node. The levels for all PI nodes are zero. The *depth* of a given network is the largest node level in the network. Given a K-bounded Boolean network, let N_v denote the subnetwork consisting of node v and all its predecessors. A cut $(X_v, \overline{X}_v)$ for node v is a partition of the nodes in N_v such that v is in $\overline{X_v}$. The *node cut-size* is defined as the number of nodes in X_v that are adjacent to some nodes in $\overline{X_v}$. If the cut size is no larger than K, it is called a *K-feasible* cut. If $(X_v, \overline{X_v})$ is a K-feasible cut, we can cover all nodes in $\overline{X_v}$ by a single K-input LUT. For example, Figure 3.2 shows a 3-feasible cut $(X_v, \overline{X}_v)$ for node v, and the node cut set is $\{g, h, i\}$.

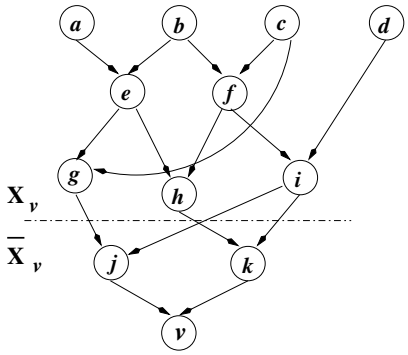


Figure 3.2. A 3-feasible Cut for Node v.

The delay of a circuit mapped into K-LUTs is determined by two factors: the delay due to the LUT and the delay due to the interconnections. The inputs to a LUT may have different signal arrival times, but once the input combination settles, the LUT takes constant time (i.e., access time of the SRAM) to produce the output independent of the function it implements. Since at the stage of mapping, the layout information is not available, we assume that each edge in the mapping solution has the same delay. Thus the total delay can be measured by the depth of the mapping solution. This unit delay model has been widely used in previous research work [38] [39] [44] [45].

In this dissertation, we study LUT-based technology mapping for power minimization. Given a K-bounded Boolean network, we would like to find a mapping solution consisting of only K-LUTs in such a way that the total power consumption of the mapped circuit is minimized. We also study this problem further with the constraint that the mapped circuit should have optimal depth.

3.3 Power Estimation Model

In this section, we review the power estimation model used in [1] [2] which estimates the power consumption of a mapped circuit consisting of LUTs only. The majority of power consumption in CMOS circuits is due to the dynamic power dissipation [12]. Dynamic power dissipation P_d occurs because of the switching activity (either 0→1 or 1→0 transition) of the circuit, which results in the charging/discharging of the load capacitance.

The *equilibrium probability* of a signal v, denoted as $p(v)$, is defined as the probability that signal v has the value 1. And the *transition density* of a signal v, denoted as $d(v)$, is defined as the number of times that signal v changes its value in unit time. The formal definitions are shown below:

Definition 1. *The equilibrium probability of $x(t)$, denoted as $p(x)$, is defined as:*

$$p(x) = \lim_{T \to \infty} \frac{1}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} x(t)\, d(t) \tag{3.1}$$

Definition 2. *The transition density (TD) of $x(t)$, denoted as $d(x)$, is defined as:*

$$d(x) = \lim_{T \to \infty} \frac{n_x(T)}{T} \tag{3.2}$$

where $n_x(T)$ is the number of transitions of $x(t)$ in the time interval $(-\frac{T}{2}, +\frac{T}{2}]$.

Let $y = f(x_1, x_2, \ldots, x_n)$ be a Boolean function. The *Boolean difference* of y with respect to x_i, denoted as $\frac{\partial y}{\partial x_i}$, is defined as:

$$\frac{\partial y}{\partial x_i} = y|_{x_i=0} \oplus y|_{x_i=1} \tag{3.3}$$

Let $\mathcal{M}$ be a logic module with inputs $x_1, \ldots, x_m$ and outputs $y_1, \ldots, y_n$. If there is no propagation delay associated with $\mathcal{M}$, $\mathcal{M}$ is known as a *zero-delay* logic module. We quote the theorem presented in [92] to show the relationship between $d(y_i)$ and $d(x_i)$. The transition density at the output of a gate can be calculated as follows:

$$d(y) = \sum_{i=1}^{n} p\left(\frac{\partial y}{\partial x_i}\right) d(x_i) \tag{3.4}$$

Using this theorem we can compute the transition density of any node in the network when the transition densities at the primary inputs are given. The proof of this theorem can be found in [92]. This is based on the assumption that the fanins of every node are independent. However, this may not be always true. Even though this holds for PIs, the existence of reconvergent paths may cause correlation among the the values of the fanins to a node. It is shown in [92] that if the modules are large enough such that tightly coupled nodes are kept inside the same module, then the independence assumption holds. The gate propagation delay and the temporal correlation on the nodes are ignored as in [1] and [2].

Once the transition density for every gate has been computed, the power consumption of a given circuit can be computed as:

$$P_d = \frac{1}{2} \sum_{i=1}^{n} C_i \cdot V_{dd}^2 \cdot d(i) \tag{3.5}$$

where C_i is the load capacitance of gate i, V_{dd} is the supply voltage, and $d(i)$ is the transition density of the output of gate i.

For LUT-based FPGAs, the above formula still holds except that the transitions happen only at the input/output of each LUT. Given a mapping solution, for a LUT rooted at node v and with input nodes $u_1, u_2, \ldots, u_k$, the power dissipation due to the LUT, $P(v, \{u_1, u_2, \ldots, u_k\})$, is given by:

$$P(v, \{u_1, u_2, \ldots, u_k\}) = K_p \cdot C_{out} \cdot d(v) + \sum_{i=1}^{k} K_p \cdot C_{in} \cdot d(u_i)$$

where $K_p = 0.5 \cdot V_{dd}^2$ is a constant, C_{out} is the output capacitance of the LUT, C_{in} is the input capacitance of the LUT.

Now if we are given a Boolean network, we estimate the minimum power dissipation when it is mapped into K-input LUTs as follows. Define a LUT-cover rooted at node v as a LUT that covers all the nodes in a K-feasible cone rooted at v. Let S_v denote the set of all possible LUT-covers rooted at node v. Let $inputs(L)$ denote the set of nodes that provide inputs to a LUT-cover L in S_v. We use the following recursive formula to estimate the minimum power consumption of a LUT-subnetwork covering N_v:

$$EP(v) = \min_{L \in S_v} \{P(v, inputs(L)) + \sum_{u \in inputs(L)} EP(u)\} \tag{3.6}$$

Note that $EP(v)$ is actually an upper bound on the minimum power consumption of any LUT-subnetwork covering N_v when there are reconvergent fanouts within N_v. For any PI node u, we assume that $EP(u)$ is equal to zero. For example, for the mapping solution shown in Figure 3.3,

$$EP(j) = K_p \cdot C_{out} \cdot d(j) + K_p \cdot C_{in} \cdot [d(a) + d(b) + d(c)] + EP(a) + EP(b) + EP(c)$$
$$= K_p \cdot C_{out} \cdot d(j) + K_p \cdot C_{in} \cdot [d(a) + d(b) + d(c)] \quad (\text{ as } EP(a) = EP(b) = EP(c) = 0\,)$$

and

$$EP(l) = K_p \cdot C_{out} \cdot d(l) + K_p \cdot C_{in} \cdot [d(j) + d(k)] + EP(j) + EP(k)$$

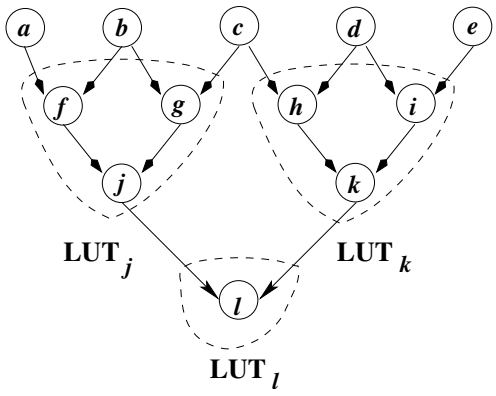


Figure 3.3. A Mapped Network into 3-LUTs Rooted at l (Nodes *a, b, c, d*, and *e* are PI nodes).

3.4 Power Minimization Algorithm

In this section, we first present our power minimization algorithm called *PowerMinMap* (PMM) and then discuss its extension named *PowerMinMap-d* (PMM-d) which optimizes both power and delay simultaneously. Further, we will show that PMM-d yields delay-optimal solutions.

The input to the algorithm is a K-bounded gate-level network. If the netlist is not K-bounded then we will first decompose the network to satisfy the K-bounded property. The equilibrium probabilities and transition densities of the PIs are given. The PMM algorithm generates a power-optimized mapping solution and it consists of the following three phases:

1. Transition density propagation.
2. Computation of $EP(v)$ for every node v.
3. Mapping generation.

In the first phase, we compute the transition density for all internal nodes as well as PO nodes in a topological order. Note that by doing so, we would have also propagated equilibrium probabilities. Below, we describe the details in phases two and three.

3.4.1 Phase II: Computation of EP(v)

The EP values of all nodes are computed in a topological order starting from the PIs. For each node v, a sequence of *T-bounded K-feasible* cuts is computed based on the EP values of v's ancestors. $EP(v)$ is computed based on the cut that yields the minimum power consumption. A *T-bounded K-feasible* cut is a cut whose input size is no larger than K such that the EP value on any input to the cut is no more than T. This term is further explained in detail below.

3.4.1.1 T-Bounded K-Feasible Cut

While generating a LUT rooted at node v, we would like to minimize the sum of the power consumption of the fanins feeding into this LUT. However, it has been proved in [33] that the problem of finding a technology mapping solution with minimum power consumption is NP-hard. We propose an efficient method to compute a low-power K-feasible cut by computing a sequence of *T-bounded K-feasible* cuts using incremental network flow computation.

Definition 3. *A T-bounded K-feasible cut $(X_v, \overline{X_v})$ for node v is a K-feasible cut such that the maximum EP value of the nodes in X_v that provide inputs to the nodes in $\overline{X_v}$ is no larger than a threshold T.*

Definition 4. *A T-bounded K-feasible cut is a low-power K-feasible cut for node v, if it results in the smallest power consumption among the sequence of T-bounded K-feasible cuts computed for v.*

For a given non-PI node v, a low-power K-feasible cut is computed as follows: We start with the largest EP value in v's ancestors as the threshold T, and use a network flow based approach to check if there exists a T-bounded K-feasible cut. We convert subnetwork N_v into a node-capacitated flow network by assigning infinite flow capacity to those nodes having EP values larger than the threshold and unit flow capacity to other nodes. Then, we compute a max-flow in the constructed network. If the value of the max-flow computed is

no larger than K, we will repeat with the next largest EP value in v's ancestors as the new threshold value, so on and so forth. Each time we find a new K-feasible cut, we compute its total power consumption using Equation (3.6). We retain the cut that yields the smallest power consumption so far. When the value of the max-flow exceeds K, we terminate and the estimated power for node v, $EP(v)$, is set to the smallest power consumption recorded. This is so because we will not be able to find a K-feasible cut for any smaller threshold.

Illustrative Example:

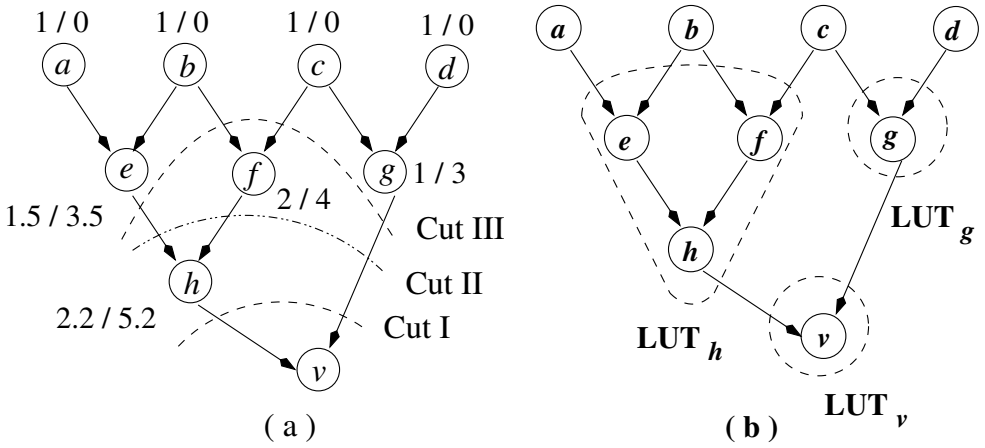


Figure 3.4. (a) Selecting a 3-feasible Cut for Node v. (b) The Mapping Solution.

Figure 3.4(a) illustrates how a sequence of T-bounded K-feasible cuts is determined for node v for K=3. Next to each node (except v), the transition density (d) and EP value are shown in d/EP format, which have already been computed before processing node v. For simplicity, we assume that $K_p \cdot C_{out}$ and $K_p \cdot C_{in}$ are equal to 1. The threshold value is first set to 5.2, which is the largest EP value (for node h) of the nodes in the set $N_v - \{v\}$. The corresponding min-cut computed is Cut I. According to Equation (3.6), the estimated power for Cut I is:

$$EP_{Cut\,I}(v) = [\,d(v) + d(g) + d(h)\,] + EP(g) + EP(h) = d(v) + 11.4$$

We store Cut I as the best cut at this point. Then threshold T is lowered to 4, the next highest EP value (for node f) in the set $N_v - \{v\}$, and Cut II is the min-cut computed. The estimated power for Cut II is:

$$EP_{Cut\,II}(v) = [\,d(v) + d(e) + d(f) + d(g)\,] + EP(e) + EP(f) + EP(g) = d(v) + 15$$

Since d(v)+15 is larger than d(v)+11.4, we move on to the next smaller EP value as the threshold. The threshold is lowered to 3.5 (for node e) and Cut III is the min-cut computed. Since the cut size is larger than 3, we will terminate. Hence, the mapping induced by Cut I is the most desirable choice for node v. Figure 3.4(b) shows the corresponding mapping solution to cover N_v.

3.4.1.2 Incremental Network Flow Computation

In this section, we will describe how we may use incremental network flow method to efficiently find the low-power K-feasible cut for a node v.

We first construct the flow network with the largest EP value in N_v as the threshold. Once we have found the first T-bounded K-feasible cut, we have to repeat with the next largest EP value as the threshold. However, there is no need to build a new flow network to compute the new cut. We only need to update the residual network for all node v such that $EP(v)$ equals to the old threshold value.

We use the same example as in Section 3.4.1.1. Suppose we want to compute a low-power 3-feasible cut for node v in Figure 3.4(a). The EP values of the ancestors of node v in decreasing order are 5.2, 4, 3.5, 3, and 0. First, the threshold is set to 5.2, therefore $\overline{X}_v$ can have any node as its input node. The required flow network is shown in Figure 3.5(a). Here we used a standard network transformation technique, known as *node-splitting*, that transforms a node-capacitated network into an edge-capacitated network so that any existing edge cut computation algorithm can be applied. A minimum cut of size 2 is computed by maximum flow computation, the residual network is shown in Figure 3.5(b). This cut

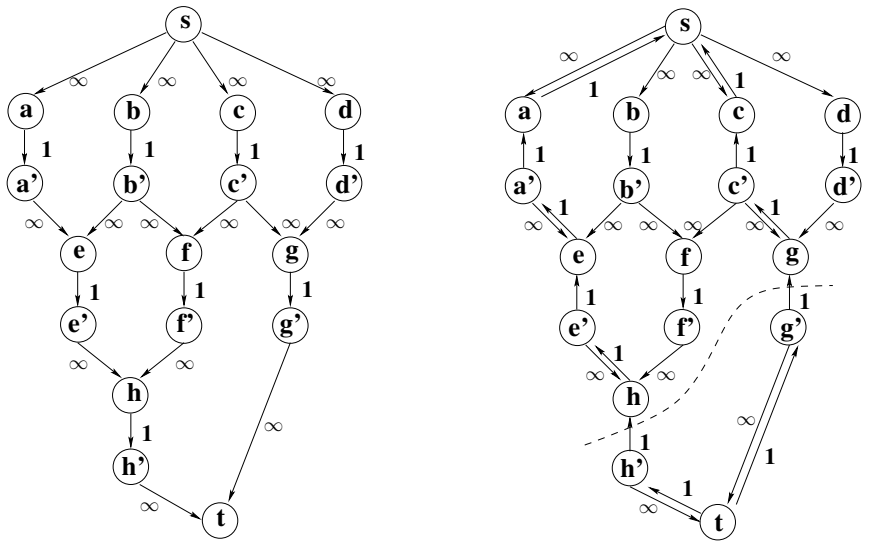
corresponds to Cut I in Figure 3.4(a). Since Cut I is 3-feasible, so we lower the threshold to 4 to compute a new cut $(X_v, \overline{X}_v)$ such that no node with EP value greater than 4 can be an input node to $\overline{X}_v$. To compute such a new cut, we may simply update the residual network by changing the flow capacity of node h whose EP value is 5.2 from unit to infinity. If there exists a new augmenting path in the updated residual network, it can be found efficiently.

We associate each node v with two flags: $FS(v)$ equals to TRUE indicates that v is reachable from source s, and $FT(v)$ equals to TRUE indicates that there exists a path from v to sink t in the residual network. Note that the initial values of $FS(v)$ and $FT(v)$ can be derived simultaneously while traversing the network to compute the min-cut of the original network. When we update the flow capacity of a node v, we also check if its flags have to be updated. For example, $FS(h)$ is TRUE and $FT(h)$ is FALSE in Figure 3.5(b). After node h's flow capacity is updated, $FT(h)$ needs to be updated to TRUE as in Figure 3.5(c). If $FS(v)$ and $FT(v)$ are both equal to TRUE, we know that there exists an augmenting path. In this case, we will augment the flow, traverse the network again to compute a new min-cut and the new FS and FT values for each node. Since we will terminate once the max-flow value exceeds K, we will traverse the network at most K times.

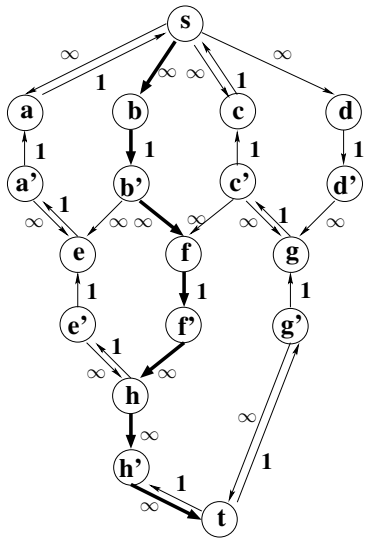
3.4.2 Phase III: Mapping Generation

Phase III is the mapping generation phase that generates a K-LUT mapping solution of the whole circuit. The mapping solution is generated in a bottom up manner starting from the PO nodes. If a LUT needs to be generated rooted at v, a K-feasible cut is computed for node v based on the cost values of its ancestors.

Initially, $cost(u)$ is set to $EP(u)$ for all node u. When we generate a LUT rooted at node v, the power dissipation of subnetwork N_u for all node u that provides input to LUT_v will be counted. So in order to avoid counting the power dissipation of subnetwork N_u again if we generate another LUT that also receives input from node u, we have to reset $cost(u)$ to 0 after the first time it is counted. When we compute the low-power K-feasible cut in this phase, we use the dynamically updated cost values.



(a) (b)



(c)

Figure 3.5. (a) Initial Flow Network (for T=5.2). (b) Residual Network after the Max-flow is Computed. (c) The Updated Residual Network with a New Augmented Path Shown in Bold Edges.

We also introduce a technique to further improve the quality of the mapping. If a low-power K-feasible cut found by the flow method has a size less than K, we will check whether it is possible to reduce the sum of the power consumption of the fanins by replacing one node in the node cut set with its fanin such that the cut size still does not exceed K. We call this *cut frontier refinement*. For example, assume Cut 1 shown in Figure 3.6 is a 4-feasible cut computed for node v. Since the cut size is only 3, we can increase the cut size to 4 and it is still 4-feasible. If $cost(l) + cost(m) < cost(i)$, we know that a LUT generated according to Cut 2 consumes less power. Similarly, we also check if choosing the node cut set $\{i, m, n, k\}$ or $\{i, j, n, o\}$ can save more power over Cut 2. We do this recursively until we cannot increase the cut size any more. If there exist cuts that use less power than the K-feasible cut computed, we will replace it with the one that reduces the most power.

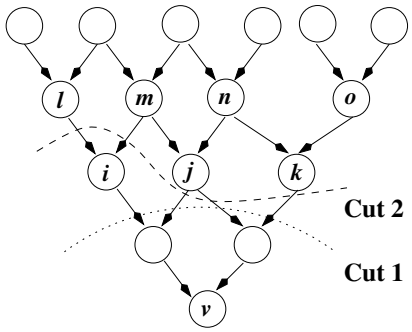


Figure 3.6. Cut Frontier Refinement for Network Rooted at v (Assuming K=4).

The complete algorithm of PowerMinMap is shown in Figure 3.7. Line 1 – 4 is the first phase which computes the transition density for each non-PI node in the DAG. Line 5 – 9 is the second phase that computes $EP(v)$, the estimated minimum power consumption of a LUT-subnetwork covering N_v, for every node v. Line 10 – 21 is the last phase of PMM algorithm. In this phase, the mapping solution is generated. Once a LUT is generated, we perform cut frontier refinement to further reduce power and update the cost values of its fanins.

Algorithm PowerMinMap
Input: General Boolean network N
Output: A mapping of the network into K**-LUTs**

```
                    /* phase 1 */
1   L ← list of non-PI nodes in topological order from PIs to POs;
2   for each node v ∈ L
3      compute d(v);
4   endfor
                    /* phase 2 */
5   for each node v ∈ L
6      compute a sequence of T-bounded K-feasible cuts for node v;
7      EP(v) ← power consumption corresponding to the low-power K-feasible cut;
8      cost(v) ← EP(v);
9   endfor
                    /* phase 3 */
10  Q ← a queue of all POs;
11  while Q ≠ ∅
12     v ← dequeue (Q);
13     compute a low-power K-feasible cut for node v;
14     perform cut frontier refinement;
15     generate a LUT rooted at v;
16     for each node u ∈ inputs(LUT_v)
17        if node u ∉ Q and node u ∉ PIs then
18           enqueue (u, Q);
19           cost(u) ← 0;
20     endfor
21  endwhile
End algorithm
```

Figure 3.7. Pseudocode of PowerMinMap Algorithm.

3.4.3 Computational Complexity of PowerMinMap

Theorem 1. *A low-power K-feasible cut in* N_v *can be computed in* $O(Km' + n' \log n')$ *time, where* n' *and* m' *are the number of nodes and edges in* N_v*, respectively.*

Proof 1. *It takes* $O(n' \log n')$ *time to sort the EP values of the nodes in* $N_v - \{v\}$*. If we use an incremental network flow computation approach to compute a set of T-bounded K-feasible cuts to find the low-power K-feasible cut in* N_v*, we need to traverse the network at most K times. Traversing the network once takes* $O(m' + n')$ *time. Hence, the total time needed to compute a low-power K-feasible cut in* N_v *is* $O(n' \log n') + O(Km' + Kn')$*, which*

is $O(Km' + n' \log n')$. When m' and n' are of the same order and K is fixed, the complexity then becomes $O(n' \log n')$.

Theorem 2. *Given a general Boolean network N, PowerMinMap generates a low-power mapping solution in $O(Kmn + n^2 \log n)$ time, where n and m are the number of nodes and edges in N, respectively.* Proof 2. *In phase one of PMM algorithm, it takes a total of $O(n)$ time to compute the output transition density $d(v)$ for all non-PI node v. According to Theorem 1, to calculate the low-power K-feasible cut for a subnetwork N_v takes $O(Km' + n' \log n')$ time, where m' and n' are the number of nodes and edges in N_v, respectively. Since m' and n' are bounded by m and n, phase two takes up to $O(n \cdot (Km + n \log n)) = O(Kmn + n^2 \log n)$ time to compute $EP(v)$ and $cost(v)$ for all node v. Phase three differs from phase two in that low-power K-feasible cut is recomputed for L nodes, where L is the number of LUTs generated in the mapping solution ($L \leq n$). And cut frontier refinement is also performed. The time needed for cut frontier refinement is $O(K)$. So phase three takes $O(n \cdot (Km + n \log n + K)) = O(Kmn + n^2 \log n)$ time. Thus, the overall runtime for PMM algorithm is $O(n) + O(Kmn + n^2 \log n) + O(Kmn + n^2 \log n)$, which is $O(Kmn + n^2 \log n)$. If m and n are of the same order and K is fixed, the total runtime is $O(n^2 \log n)$.*

3.5 PowerMinMap-d: Simultaneous Power and Delay Optimization

In this section, we continue to study the technology mapping problem for minimizing power consumption with the priority of optimal depth. We first review the Flowmap algorithm [39], which is known to be a delay-optimal LUT-based technology mapper. Then we present an extension of the PowerMinMap algorithm to achieve optimal delay as well as power savings. This extension is named as *PowerMinMap-d* (PMM-d), and it is very efficient and effective in computing a low-power delay-optimal mapping solution.

3.5.1 Review of Flowmap Algorithm

Flowmap [39] is a LUT-based FPGA technology mapping algorithm that generates delay-optimal K-LUT mapping solutions. Given a K-bounded Boolean network, let N_v denote the subnetwork consisting of node v and all its predecessors. The label of node v, denoted by $label(v)$, is the optimal depth for a K-LUT mapping of N_v. For any PI node v, $label(v)$ is zero. In the first phase of Flowmap, the nodes are processed in a topological order from PI nodes to PO nodes and the labels for all nodes are computed. Assume $\mathcal{L}$ is the maximum label of the predecessors of node v. If node u is a fanin of v, we define *collapse* as the operation of removing u from N_v and replacing every edge $(x, u) \in E(N_v)$ by (x, v). To compute $label(v)$, Flowmap first collapses all nodes u with $label(u) = \mathcal{L}$ into v and computes a min-cut in N_v. If the cut size is no larger than K, Flowmap sets $label(v)$ to $\mathcal{L}$ and stores the cut. Otherwise, Flowmap sets $label(v)$ to $\mathcal{L}$+1 and stores the cut $(N_v - \{v\}, \{v\})$. Note that $(N_v - \{v\}, \{v\})$ is guaranteed to be a K-feasible cut because the original network is K-bounded. In either case, the cut computed is called a *min-height K-feasible* cut for node v.

In the second phase, Flowmap generates the mapping solution. Let $(X_v, \overline{X}_v)$ be the cut stored for node v in the first phase where $v \in \overline{X}_v$. Q is a queue that contains all PO nodes initially. The following operation is repeated until Q contains only PI nodes: Remove the head node v from Q and generate a LUT to cover the nodes in $\overline{X}_v$ according to the min-height K-feasible cut $(X_v, \overline{X}_v)$ computed for node v; Insert into Q the nodes that provide inputs to the LUT just generated, if they have not been inserted yet.

It is shown that Flowmap guarantees to generate depth-optimal K-LUT mapping solutions for a given K-bounded network N.

3.5.2 PowerMinMap-d Algorithm

PMM-d algorithm consists of two phases. The first phase of PMM-d is similar to that of Flowmap. For each non-PI node, we compute its label and store its min-height K-feasible cut. In addition, for each node v, we also compute its output transition density $d(v)$, and

estimate the power consumption of a K-LUT mapping of subnetwork N_v rooted at v. Unlike the PMM algorithm, we estimate $EP(v)$ in PMM-d according to the min-height K-feasible cut computed.

The second phase of PMM-d differs from that of Flowmap: before generating a K-LUT rooted at v, we determine if the depth of this LUT can be relaxed from the value $label(v)$ without increasing the depth of the overall mapping solution.A similar idea was also used by Cong and Hwang in [45] for area minimization with optimal depth. If we can relax its depth, we will compute a low-power K-feasible cut for v and generate a K-LUT accordingly. Otherwise, we will generate a K-LUT using the min-height K-feasible cut stored in the first phase.

We can determine if the depth of a LUT rooted at node v can be relaxed by computing its slack value:

$$slack(v) = D_{opt} - label(v) - D_v \tag{3.7}$$

where D_{opt} is the optimal depth of a K-LUT mapping solution of the entire network, and D_v is the maximum number of LUTs on a path from some child of node v to some PO node.

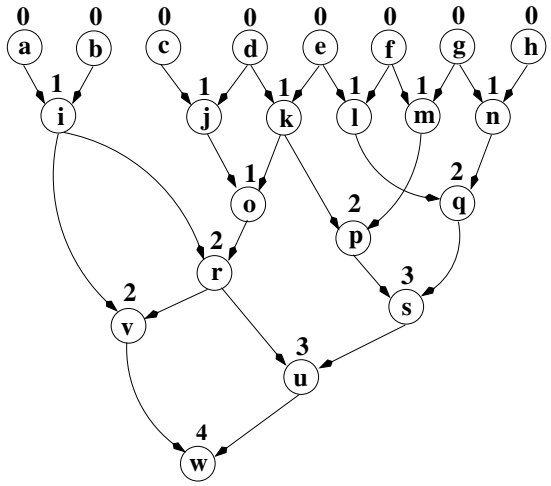


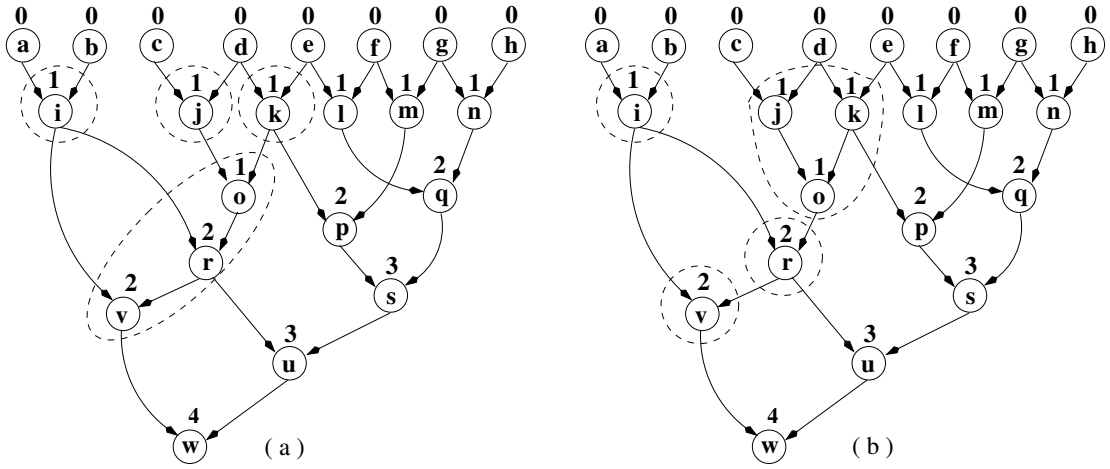
Figure 3.8. Labels Computed for a Boolean Network Assuming $K = 3$.

Note that D_{opt} is known after phase one since $D_{opt} = max\{\, label(u) : u \in set\, of\, PO\, nodes\}$, but D_v has to be determined dynamically when we generate the actual mapping in a bottom up manner. When we are generating a LUT rooted at node v in phase two, a partial mapping solution covering all the successors of v should have been generated. Thus, D_v is also known by this time. If $slack(v) > 0$, then the depth of a LUT rooted at v can be relaxed and node v is called a *non-critical* node. Otherwise, the depth of a LUT rooted at v cannot be relaxed and such a node is called a *critical* node. For example, consider computing the slack value for node u and v in Figure 3.8. The optimal depth D_{opt} of the network is 4, $label(u) = 3$, $label(v) = 2$, and $D_u = D_v = 1$. We can compute the slack for node u and v according to Equation 3.7: $slack(u) = 4 - 3 - 1 = 0$, and $slack(v) = 4 - 2 - 1 = 1$. So node u is a non-critical node but node v is a critical node.

Since node v is non-critical, our algorithm has the flexibility to generate a different mapping for N_v if power savings can be achieved. The mapping solution generated by Flowmap for N_v is shown in Figure 3.9(a). On the other hand, PMM-d may generate the mapping solution shown in Figure 3.9(b) to reduce power consumption. Note that the depth of node v is increased by one, but this will not increase the optimal depth of the entire network since node v was non-critical.

Our algorithm guarantees to generate optimal delay mapping solutions and the explanation is as follows. When we are generating a LUT rooted at a non-critical node v with $label(v) = L$, we know that its predecessors have slack values no less than $slack(v)$. Once a low-power K-feasible cut is computed, if a node u in the cut set has $label(u) = L$, we know that $slack(u)$ will be decreased by one. So after LUT_v is generated, the slack value of each node $u \in inputs\,(\mathrm{LUT}_v)$ will not become negative. If $slack(u)$ becomes 0, node u becomes critical. Then we will use the min-height K-feasible cut to cover node u afterwards. This justifies that our algorithm always generates delay optimal mapping solutions.

The complete algorithm of PMM-d is shown in Figure 3.10. Lines 1 – 10 account for the first phase where $d(v)$, $label(v)$ and $EP(v)$ are computed for each non-PI node v in a topological order. Lines 11 – 16 check if node v is critical or not and make sure that a LUT



(a) (b)

Figure 3.9. Different Mappings Assuming $K = 3$: (a) Using Flowmap and (b) Using PMM-d.

will be generated to cover each PO node. Lines 17 – 33 generate the final mapping. For a critical node, the min-height K-feasible cut computed in the first phase is used to generate the mapping. For a non-critical node, we compute a low-power K-feasible cut and generate a LUT accordingly. Then, we update the slack values of the nodes feeding into this LUT and update the cost value for these nodes.

3.5.3 Computational Complexity of PowerMinMap-d

Theorem 3. *Given a general Boolean network N, PowerMinMap-d generates a depth-optimal low-power mapping solution in $O(Kmn + n^2 \log n)$ time, where n and m are the number of nodes and edges in N, respectively.*

Proof 3. *According to [39], the labeling phase for all nodes in N can be done in $O(Kmn)$ time. Besides, it takes $O(1)$ time to compute the output transition density $d(v)$ and $EP(v)$ for each non-PI node v. So the time needed in the first phase of PowerMinMap-d is $O(Kmn)$. In the second phase, the number of LUTs generated is bounded by n. If v is a critical node, it takes $O(1)$ time to generate LUT_v using the min-height K-feasible cut computed in phase one. Otherwise, it takes $O(Km' + n' \log n')$ time to compute a low-power K-feasible*

Algorithm PowerMinMap-d
Input: General Boolean network N
Output: A mapping of the network into K**-LUTs**

```
                    /* phase 1 */
1  L ← list of nodes in topological order from PIs to POs;
2  for each node v ∈ L
3     if v ∈ PIs then label(v) ← 0;
4     else
5        compute the min-height K-feasible cut for node v and label(v);
6        compute d(v);
7        EP(v) ← power estimation for the computed cut;
8        cost(v) ← EP(v);
9  endfor
10 D ← max { label(v): v ∈ POs}
                    /* phase 2 */
11 for each node v ∉ PIs
12    D_v = 0;
13    if node v ∈ POs then
14       lut(v) ← TRUE, slack(v) ← D − label(v);
15    else lut(v) ← FALSE, slack(v) ← 0;
16 endfor
17 Q ← a queue of POs;
18 while Q ≠ ∅
19    v ← dequeue (Q);
20    if lut(v) = TRUE
21       if slack(v) = 0 then
22          generate LUT_v using the min-height K-feasible cut for node v computed;
23       else
24          compute a low-power K-feasible cut for node v;
25          generate LUT_v according to the low-power K-feasible cut computed;
26       for each node u ∈ inputs(LUT_v) and u ∉ PIs and u ∉ Q
27          lut(u) ← TRUE;
28          D_u ← max (D_u, D_v + 1);
29          slack(u) ← D − label(u) − D_u;
30          cost(u) ← 0;
31          enqueue (u, Q);
32       endfor
33 endwhile
End algorithm
```

Figure 3.10. Pseudocode of PowerMinMap-d Algorithm

cut to generate LUT_v, where m' and n' are the number of edges and nodes in N_v, respectively. Therefore, phase two takes $n_1 \cdot O(1) + n_2 \cdot O(Km + n \log n) < O(Kmn + n^2 \log n)$ time, where n_1 is the number of LUTs generated for covering critical nodes and n_2 is the number of LUTs generated for covering non-critical nodes ($n_1 + n_2 < n$). The total time needed by both phases is $O(Kmn) + O(Kmn + n^2 \log n)$, so the overall computational complexity of PowerMinMap-d is $O(Kmn + n^2 \log n)$. When m and n are of the same order and K is fixed, the total time complexity becomes $O(n^2 \log n)$.

3.6 Experimental Results of Low Power Technology Mapping Algorithms

We have implemented both the PMM and PMM-d algorithms using C language and experimented with a set of MCNC benchmark circuits. Given a general Boolean network, we first optimize it by running the general optimization script "rugged" within SIS. Then we decompose the circuit into 2-bounded network using the "DMIG" command. The mapping solution is computed on the decomposed network.

We assume that $V_{dd} = 5V$ and all PI nodes have equilibrium probability $p = 0.5$, and transition density $d = 10{,}000$. The capacitances C_{in} and C_{out} are set to 10 pF each. When we estimate the power consumption of a mapping solution, the external loading capacitance of PO nodes (which is not known *a priori*) is ignored since the power consumption due to a given external load is independent of the mapping. Note that the above assumptions have been used in [1] [2]. For comparison with the results reported in [1] and [2], we run our PMM algorithm for mapping into 5-input LUTs. The results of the PMM algorithm are shown in Table 3.1 in terms of power consumption and the number of LUTs for each circuit. For comparison, we quote the best result reported for each circuit in [1]. We also include the results from [2] because we notice that in some cases [2] outperforms [1]. On the average, our algorithm reduces the power consumption by 18.5% and 12.2% compared with [1] and [2], respectively. Besides, it uses 9.5% and 10.6% less LUTs than [1] and [2], respectively.

Table 3.1. Comparison of PowerMinMap with [1] and [2] (PWR: mW).

CKT	Wang et.al. [1]		Farrahi et.al. [2]		PMM		Improvement (%)			
							Vs. [1]		Vs. [2]	
	LUTs	PWR	LUTs	PWR	LUTs	PWR	LUTs	PWR	LUTs	PWR
5xp1	26	188	25	182	23	168	11.5	10.6	7.7	7.8
9sym	87	553	62	365	60	340	31	38.5	3.2	6.8
9symml	62	446	58	376	56	349	9.7	21.7	3.4	7.2
c499	98	1061	91	1076	74	983	24.5	7.4	18.7	8.6
c880	116	746	111	1060	106	718	8.6	3.8	4.5	32.2
alu2	128	874	146	836	124	815	3.1	6.8	15.1	2.5
apex6	192	1110	237	1404	183	1012	4.7	8.8	22.8	27.9
apex7	68	379	69	450	65	349	4.4	7.9	5.8	24.0
count	31	159	31	227	31	159	0	0	0	29.9
duke2	152	538	190	478	145	443	4.6	17.6	23.7	7.3
misex1	13	97	16	106	12	92	2.3	5.2	25	13.2
rd84	33	261	27	344	32	243	3	6.9	-18.5	29.4
rot	234	1306	238	1749	224	1203	4.3	7.9	5.9	31.2
vg2	32	182	25	60	26	158	18.7	13.2	-4	1.3
z4ml	7	56	5	80	5	41	28.6	26.8	28.6	48.8
Average							10.6	12.2	9.5	18.5

We note that the heuristic enumeration method of [1] does not guarantee that if more cuts are enumerated, the mapping solution will be better in terms of power consumption. One drawback of [2] is that it fails to take the impact of the fanout number of LUTs on the power consumption into account. It tries to generate LUTs rooted at nodes with smaller transition density and maximizes the number of inputs to each LUT. According to our experimentation, for some cases, we found that this approach resulted in larger total power consumption. Our algorithm improves this and generates better mapping solution because it directly considers the impact of generating a single LUT on the mapping of the entire network.

For our PMM-d algorithm, we compared it with Cutmap [45]. Cutmap is an extension of Flowmap that simultaneously minimizes the area and depth. In general, Cutmap also produces solutions with smaller power consumption than Flowmap because it minimizes the area. We have run Cutmap within SIS and estimated the power consumption on the

circuits mapped into 4-LUTs. The results are shown in Table 3.2 with respect to power consumption, area, and depth. The results generated by our algorithm have the same depths as those of Cutmap which are guaranteed to be optimal. On the average, PMM-d reduces the power consumption by 14.1% at the expense of using 9.2% more LUTs than Cutmap. We notice that for two small benchmark circuits (5xp1 and rd84), our algorithm performed slightly worse than Cutmap in terms of power consumption. But it outperformed Cutmap in power savings in all other cases.

Table 3.2. Comparison of PowerMinMap-d and Cutmap (Power: mW).

	Cutmap			**PowerMinMap-d**		
Circuit	4-LUTs	Power	depth	4-LUTs	Power	depth
5xp1	46	333	4	52	336	4
9sym	69	471	5	72	420	5
9symml	92	720	6	108	659	6
alu2	145	1404	13	146	936	13
alu4	326	3202	15	353	2110	15
c1355	74	2015	4	74	1463	4
c2670	280	2883	8	315	2527	8
c432	109	1114	10	113	951	10
c499	66	1836	4	74	1268	4
c5315	744	6954	9	831	6274	9
c7552	891	9592	8	984	7708	8
c880	132	968	10	144	912	10
dalu	683	4097	12	729	3761	12
duke2	185	958	8	195	719	8
i8	1383	9853	6	1647	8615	6
k2	1243	5358	6	1311	4368	6
rd84	53	425	9	67	450	9
rot	359	2450	8	362	2252	8
vg2	51	292	4	60	286	4
x1	207	1081	4	218	1074	4
Comparison	1	1	1	1.092	0.859	1

Previous works on minimizing dynamic power consumption did not handle the variation of transition density of the PI nodes. In practice, the PI nodes do not necessarily have uniform equilibrium probabilities. So, we also did some experiments using randomly generated

equilibrium activities for PI nodes. We include two sets of experimental results compared with Cutmap as shown in Table 3.3. On the average, the percentages of power and area savings of PMM-d are similar to the results shown in Table 3.2. Note that the mappings generated by Cutmap will not change while our algorithm generates different mappings according to the changing transition activities of the nodes in the network.

Table 3.3. Comparison of PMM-d and Cutmap with Randomly Generated Transition Densities for PI Nodes.

	Experiment #1				Experiment #2			
	CM		PMM-d		CM		PMM-d	
Circuit	LUTs	PWR	LUTs	PWR	LUTs	PWR	LUTs	PWR
5xp1	46	308	50	291	46	382	52	401
9sym	69	515	73	493	69	491	72	459
9symml	92	773	99	657	92	715	108	687
alu2	145	1438	149	1225	145	1350	148	1087
alu4	326	3381	350	2597	326	3422	348	2780
c1355	74	2132	75	1657	74	1985	74	1481
c2670	280	2912	313	2590	280	2685	308	2352
c432	109	983	79	557	109	625	78	587
c499	66	1242	80	1129	66	1422	81	1326
c5315	744	7275	830	6920	744	7416	821	6639
c7552	891	10428	980	8217	891	9964	975	8036
c880	132	1107	142	981	132	973	145	908
dalu	683	4251	726	3882	683	4129	713	3716
duke2	185	1057	193	877	185	972	195	816
i8	1383	9712	1588	8581	1383	10244	1639	9211
k2	1243	5409	1308	4872	1243	6140	1327	5327
rd84	53	478	65	490	53	510	64	524
rot	359	2792	362	2616	359	2507	263	2401
vg2	51	278	58	263	51	271	59	212
x1	207	1125	216	1082	207	1217	220	1202
Comparison	1	1	1.097	0.871	1	1	1.091	0.853

3.7 Conclusions on Low Power Technology Mapping

In this chapter, we studied the technology mapping problem to minimize power dissipation for LUT-based FPGAs. We presented an algorithm that minimized power as well

as area significantly. It uses an efficient incremental network flow computation approach to compute low-power K-feasible cuts minimizing the total power consumption of the generated K-LUTs. Through our experimentation, we notice that only minimizing the switching activity of nodes at the output of LUTs is not good enough. Hence our algorithm estimates the power consumption for a set of possible choices and choose the one that yields the best result. Experimental results show that our algorithm achieved both power and area savings over two previous power minimization mapping algorithms.

An extension of our work is also implemented to compute delay-optimal low-power mapping solutions. Compared with Cutmap, a delay-optimal mapper with simultaneous area minimization, our algorithm reduces power consumption without any delay penalty.

CHAPTER 4

PERFORMANCE-DRIVEN FORCE-DIRECTED PLACEMENT ALGORITHM FOR HIERARCHICAL FPGAS

As microprocessor with over a billion transistors around the corner, logic capacity and complexity have become extremely high. The technology feature size is in deep sub-micron (DSM) regime, e.g., the most advanced microprocessors from Intel and AMD all use 0.13 Micron technology. All these make physical design more important and difficult as well. As one of the phases in physical design, placement plays a crucial role as it indirectly determines the on-chip interconnects which nowadays dominates the overall system delay and hence is the bottleneck of enhancing system performance.

In this chapter, we study the performance-driven placement problem. We propose a net-based force-directed placement algorithm targetting hierarchical FPGAs. This method also works well for flat FPGA architectures. The input netlist is first transformed into a net dependency graph. Then we partition this graph into clusters and a net-cluster level floorplanning is derived by simulated annealing to optimize the wirelength. Force-directed net-level placement is performed to generate a coarse net placement. Next, a force-directed scheme is developed to compute the logic cell placement iteratively, where the forces on nets determine the positions of the cells. Finally, we assign I/O pins using a fast minimum-weight bipartite matching algorithm.

The main contribution of our work is that we develop a top-down design flow and apply force-directed method for hierarchical FPGAs to achieve satisfying performance as compared to previous CAD Tools. Compared with Xilinx Foundation tools, the experimental results show that our algorithm improves the post-layout delay (longest path delay) and average connection delay by an average of 10.2% and 19.3% respectively over a set of MCNC

combinational benchmarks. We also improve the maximum clock frequency by an average of 20.7% over a set of MCNC sequential circuits. We also compare our proposed with VPR [87], a state-of-the-art place and route suite for academic research, we achieve improvement on total netlength as well as critical path delay over the same set of benchmarks.

This chapter is organized as follows: Section 4.1 introduces several previously proposed placement algorithms for performance enhancement. In Section 4.2, we show the hierarchical FPGA architecture which we adopt in our research. In Section 4.3, we present the details of our proposed performance-driven force-directed placement approach. Experimental results are given in Section 4.4. We conclude in Section 4.5.

4.1 Introduction

Field-programmable gate arrays (FPGAs) are widely used for VLSI system design and rapid system prototyping due to its fast time-to-market and programmability. Recently, FPGA manufacturers have introduced hierarchical FPGAs such as Xilinx's Virtex family [10] and Altera's Apex family [93]. To use these million-gate devices more efficiently, computer-aided design for FPGAs has become very important. Placement has become one of the most persistent challenges in current digital system design as designs often contain over a million placement blocks. Moreover, due to the dominance of interconnect delay, placement is a major factor to timing closure results [64]. Therefore, it is of great value to develop high performance placement algorithm for FPGAs.

In recent years, many placement algorithms have been proposed to cope with the widely used objective of wirelength minimization. Among these algorithms, simulated annealing is one of the most well developed placement methods. It is used in placement as an iterative improvement algorithm. Given an original placement configuration, a change to the configuration is made by moving a component, swapping the locations of two blocks, or changing the aspect ratio of one block. Moves resulting in decreases in cost will be accepted. Cost can be wirelength, area, congestion, etc. Moves that result in an increase in cost are accepted with a probability which decreases over the iterations. This helps to

jump out of local optima. TimberWolf by Swartz and Sechen [71] is a well known simulated annealing based tool for timing-driven placement but its target is standard-cell based devices. Another simulated annealing based tool, Versatile Place and Route (VPR) by Betz and Rose [87] and Marquardt, Betz, and Rose [94] is considered as the best academic place and route suite for FPGAs. VPR is known for generating very tight placements. However, VPR handles primarily non-hierarchical island-style FPGA architectures. PROXI [95] is a simulated annealing based timing-driven placement algorithm for FPGAs. It performs simultaneous placement and routing by ripping-up and re-routing all disturbed nets after each perturbation. But it is very computationally expensive (appears to be $O(n^3)$ based on its experimental results) which makes it infeasible in real designs.

Analytical placement is known for generating fast placement solution using a quadratic wirelength objective function. Even though the quadratic objective is only an approximation of the wirelength, its main advantage is that it can be optimized very efficiently. Eisenmann and Johannes [85] presented a force-directed analytic placement technique which applies constant density-induced forces and uniformly distributes the cells. It works well in avoiding overlap but may unnecessarily result in timing penalty for sparse designs. Most recently, Alupoaei and Katkoori [96] proposed a net-based force-directed macrocell placement approach to optimize wirelength and their work showed remarkable improvement on wirelength. However, their work is limited to semi-custom ASIC devices.

Another commonly used placement approach is to partition the circuit into sub-circuits such that the interconnect between blocks is minimized. Hutton, Adibsamii, and Leaver [74] presented a k-way partitioning based timing-driven placement algorithm. It handles hierarchical architecture but only targets Altera's APEX devices.

As part of this dissertation, we developed a novel force-directed performance-driven placement algorithm for hierarchical and flat FPGA devices. Details will be provided in Section 4.3.

4.2 Hierarchical Xilinx FPGA Architecture

In our work, we mainly target the hierarchical Xilinx Virtex series FPGA chip. The top level of a Virtex chip consists of a two-dimensional array of configurable logic blocks (CLBs), vertical and horizontal routing channels, and input/output blocks. Figure 4.1 illustrates the top level architecture of an array-based FPGA. The configurable logic blocks, denoted as CLB in Figure 4.1, are customizable to implement the logic functions. The connection block, denoted as C in Figure 4.1, connect the CLB pins to the routing channels. A horizontal routing channel and a vertical routing channel are connected via a switch box denoted as S in Figure 4.1. A switch block is comprised by a number of programmable switches which account for the connections of FPGA routing. Usually, the switches have higher resistance and capacitance, and hence result in significant delays. The routing channels are segmented in order to balance the circuit performance and routability. Routing tracks consist of a set of wires with different lengths where longer wires are desirable for timing-critical nets and shorter wires are intended for short connections to save routing resources.

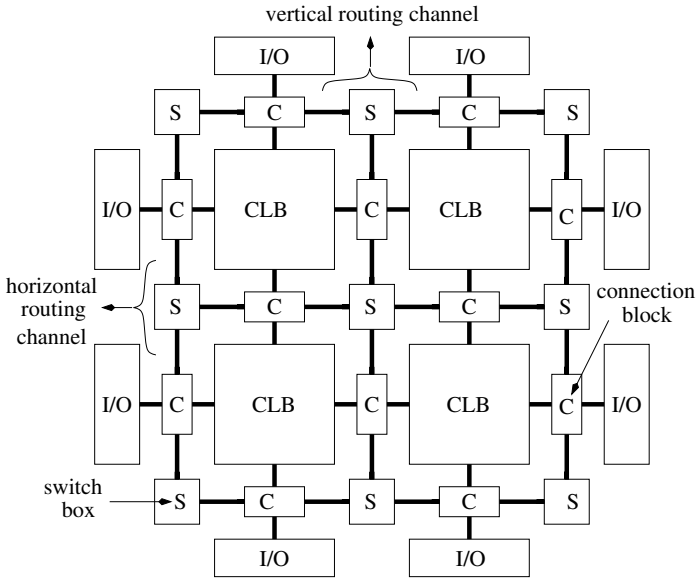


Figure 4.1. Top Level View of Xilinx Hierarchical FPGA.

Below, we describe the details of Virtex v800fg680 chip which is the main target device used in our experiments. The top level of this FPGA chip consists of an 84 by 56 array of CLBs. Each CLB consists of two slices where a slice is the basic logic cell at the bottom level. From now on, we use the terms *slice* and *cell* interchangeably throughout this chapter. In each slice, there are a pair of 4-input LUTs and a pair of D flip/flops, fast carry logic, three-state driver, and control logic. Figure 4.2 shows the simplified CLB structure.

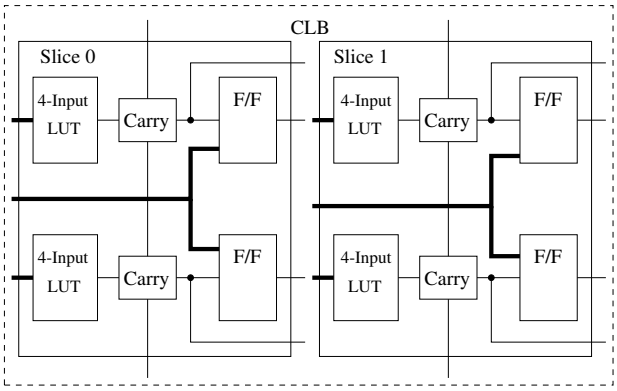


Figure 4.2. Simplified Architecture of an CLB.

Routing resources are available at both the top and the bottom levels. The local routing network within a CLB accounts for less delay than the interconnections between CLBs. On the periphery of the FPGA chip, there are 512 general-purpose I/O pads accessible to the users. In addition, there are four dedicated clock input pads, two at the top center and two at the bottom center. All I/O blocks are accessed via global routing resources. Other Virtex series FPGA chips differ from this example in the number of total CLBs, the number of slices in each CLB, and the amount of routing resources, etc.

4.3 Proposed Placement Algorithm

We propose our force-directed performance-driven placement algorithm in this section. This section is organized as follows: In Section 4.3.1, we outline this work. In Section 4.3.2, we discuss the net cluster level floorplanning. In Section 4.3.3, we present the net level

placement using force-directed method. In Section 4.3.4, we show the details of cell placement. In Section 4.3.5, we propose a revised bipartite matching based approach to place the IO pins. In Section 4.3.6, we summarize our algorithm.

4.3.1 Overview of the Algorithm

Our proposed FPGA placement algorithm has three levels in a top-down manner. The input netlist to the placer is obtained after technology mapping by Xilinx CAD Tool. We first check the number of connected components according to the netlist. If it has multiple connected components, we will process each component separately. Then, a net dependency graph is formed from the netlist. The top level of our algorithm partitions the nets into clusters and net-cluster level floorplanning is performed using simulated annealing to optimize the total netlength. The intermediate level of our algorithm computes a coarse net placement using a force-directed method. The bottom level of our approach achieves the detailed placement also using force-directed method where the forces on nets determine the positions of the cells in this phase. Finally, the I/O pins are placed with a revised minimum weighted bipartite matching scheme. The final placement is fed back to the Xilinx tool to perform a re-entrant routing. Figure 4.3 illustrates the design flow of our proposed algorithm.

4.3.2 Net Cluster Floorplanning

The input netlist to our placement algorithm is derived after technology mapping. We first check if the design has multiple connected components which can be done by running a depth-first search on the netlist. By doing so, we can process each component individually. This not only reduces the subproblem size, but also assures that the logic cells in different components will not be placed in the same region on the FPGA. Hence, routing traffic can be decreased and the performance of the placement will be increased. In the top level of our algorithm, a net dependency graph is constructed from the netlist and clique partitioning is carried out on this graph. A *net dependency graph* $G = (V, E)$ is a weighted undirected

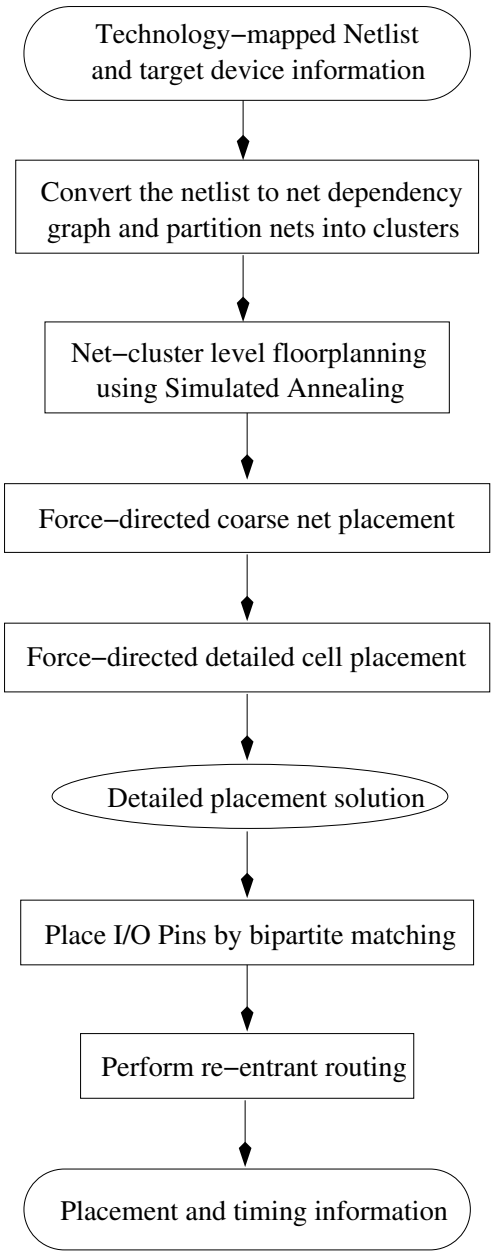


Figure 4.3. Design Flow of the Proposed Placement Algorithm.

graph which is constructed from the netlist. Each node $v \in V$ denotes a net, and an edge (u, v) exists if and only if the nets represented by u and v share at least one cell. The weight of each edge (u, v) is determined by the number of cells shared by net n_u and n_v. A *net clique*, also called a *net cluster*, is a complete subgraph of the net dependency graph. A net cluster indicates a set of strongly connected nets which preferably should be routed in the same region to optimize timing and the total netlength. The clique partitioning problem is NP-complete in nature, so we use the clique partitioning heuristic proposed by Tseng and Siewiorek [97]. We modify this method in such a way that we try to find the maximum weighted cliques. We first build a priority queue to store the nets according to the number of their neighboring nets in descending order. Then, we partition the net dependency graph starting from the head of the queue. We check if other nets can be added to the current clique according to their order in the priority queue. The clique partitioning procedure terminates when every net has been added to a net cluster. Figure 4.4(a) gives an example of an input netlist. Figure 4.4(b) shows the corresponding net dependency graph and the net clusters derived by using the heuristic. Each edge in the net dependency graph has unit weight except edge (n_1, n_6) and edge (n_4, n_6) which have a weight of 2.

The motivation for clustering nets rather than clustering cells is that we would like to group nets having high interconnections together such that they will be placed in nearby locations while performing net-level floorplanning. Consequently, logic cells within each cluster are to be placed close to each other in the final placement. As a result, these strongly connected nets can be routed as much as possible by using shorter routing segments. This also implies that lesser number of routing switches will be used which helps to improve timing. Longer routing segments can be saved to route global critical paths and the total wirelength will be reduced as well.

Once the net dependency graph is partitioned into net clusters, we will place the net clusters in such a way that the overall interconnect is minimized. Because there are slices connected by nets which belong to different clusters, we would like to minimize the distance

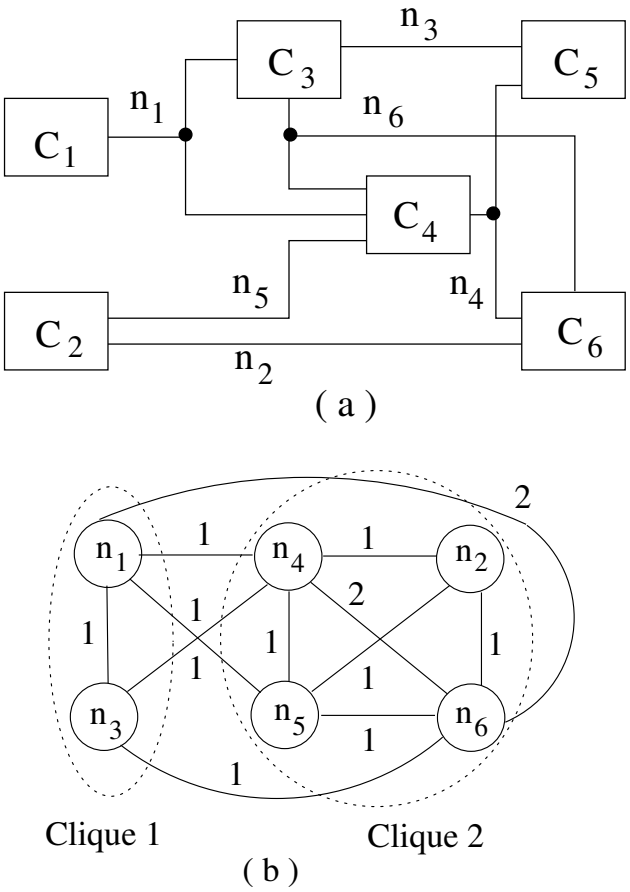


Figure 4.4. Example of Net Clustering: (a) Netlist (b) Net Dependency Graph.

between these clusters as well. The area of a cluster $\mathcal{C}$ is estimated as:

$$A(\mathcal{C}) = \sum_{s_i \in \mathcal{C}} A(s_i) \tag{4.1}$$

If slice s_i solely belongs to $\mathcal{C}$, $A(s_i) = 1$. Otherwise, $A(s_i) = \frac{1}{n}$, where n is the number of clusters that have slice s_i in common.

The net cluster level floorplan is obtained by simulated annealing based approach using the sequence-pair representation [98]. The floorplan is represented by two sequences of the clusters, $\mathcal{P}$ and $\mathcal{N}$, which denote the relative positions of the clusters. Since the number of net clusters is smaller than the number of slices, the time needed to find a net level floorplan is much less than the time required to find a floorplan at the slice level which is

very desirable. The initial seed to the simulated annealing approach is a randomly generated placement which conforms to the dimension constraints of the target FPGA chip. We allow the following moves: exchange two clusters in the $\mathcal{P}$ sequence, exchange two clusters in both the $\mathcal{P}$ and $\mathcal{N}$ sequences simultaneously, and change the aspect ratio of a cluster [98]. The placement problem for FPGAs differs from that of ASICs in that there exists dimension constraint. This is so because the numbers of rows and columns of CLBs are fixed for the target FPGA device. So a move is accepted if it does not violate the constraint. While changing the aspect ratio of a cluster, we change its width by a factor randomly generated within the range of -30% to +30%. Its height is modified accordingly in order to keep the area of this cluster unchanged.

The cost function of a floorplan, $\mathcal{F}$, is a weighted sum of the interconnections between clusters, and the interconnections between clusters and I/O pins on the boundaries of the target device:

$$Cost(\mathcal{F}) = \sum_{i \neq j} k_1 \cdot D(C_i, C_j) \ + \sum_{i} k_2 \cdot N_i \cdot I(C_i) \tag{4.2}$$

where $D(C_i, C_j)$ is the Manhattan distance between the geometrical centers of the bounding boxes of clusters C_i and C_j, $I(C_i)$ is the shortest distance between the geometrical center of C_i and the boundaries of the device, N_i is the number of slices in C_i which have connections with I/O pins. The factors k_1 and k_2 in Equation 4.2 account for the overhead for routing. They are obtained empirically and can be adjusted according to the routing resources availability of the FPGA chip. In our approach, k_1 and k_2 are set to 1 and 1.3, respectively. We assign k_2 a larger value than k_1 because we would like to prioritize those cells connected to external I/O pins. This indicates that I/O connected cells will be placed close to the boundaries to improve total wirelength and timing. Note that we compute $D(C_i, C_j)$ only when the two clusters share at least one slice, and compute $I(C_i)$ only when at least one slice in C_i has connection with an I/O pin.

The initial temperature of the simulated annealing procedure is computed after generating 20 random floorplans as in [99]:

$$InitTemp = -\sum_{i=1}^{20} Cost(\mathcal{F}_i) \Big/ \log \Delta \tag{4.3}$$

where the factor Δ is set to be 0.95 in our approach.

The temperature cooling schedule used is shown as follows:

$$T_{new} = \begin{cases} 0.95\ T_{old} & if\ T_{old} \in [0.3, 0.8] \cdot InitTemp \\ 0.80\ T_{old} & Otherwise \end{cases} \tag{4.4}$$

The temperature is lowered at a faster rate while it is very high ($>$ 0.8 InitTemp) or very low ($<$ 0.3 InitTemp), and it is lowered at a slower rate otherwise. Since generally simulated annealing is considered to be very time consuming, we use this faster cooling schedule to reduce the run time without penalizing the quality of the result. The simulated annealing process is terminated when the temperature value drops below 1.

4.3.3 Coarse Net-level Placement

The intermediate level of our algorithm is to perform a coarse net level placement using a force-directed scheme. Below, we first discuss the characteristics of various forces we used in our approach. Then, we describe how we use this method to place nets.

4.3.3.1 Attractive and Repulsive Forces

In our force-directed placement approach, we use two types of forces. The main force is the *attractive force* which obeys Hooke's law as in Equation 4.5, where k is a constant known as spring constant and Δx is the spring deformation.

$$F = -k \cdot \Delta x \tag{4.5}$$

The attractive force pulls nets connected in the net dependency graph to closer positions. Using attractive force on placement was first proposed in [100]. Since net clusters have different shapes and sizes, we also apply a *repulsive force* to avoid net cluster overlaps. We have to find the equilibrium positions for the objects (nets in intermediate level placement and logic cells in low level placement). This gives a potential placement that results in short net lengths. Since the repulsive force is *electrostatic* in nature, it has a circular symmetry characteristic. We model each net as a circle with radius R estimated as follows:

$$R = 0.4\ (h + w)/(\log n + 1) \tag{4.6}$$

where h and w are the height and width, respectively, of the net cluster containing this net, and n is the number of clusters this net is connected with.

In general, the attractive force on object i due to object j is computed as follows:

$$\overrightarrow{F}^{a}_{(i,j)} = -K^{a}_{(i,j)} \cdot (\overrightarrow{p}_i - \overrightarrow{p}_j) \tag{4.7}$$

where $K^{a}_{(i,j)}$ is an analogous attractive coefficient, $\overrightarrow{p}_i$ and $\overrightarrow{p}_j$ are the position vectors of objects i and j, respectively. In our case, the value of the force factor $K^{a}_{(i,j)}$ is set to 1 initially. It will be increased after each iteration using Equation 4.8 in order to attract connected objects even closer.

$$K^{a}_{(i,j)} = \alpha \cdot K^{a}_{(i,j)}\ (old) + (1 - \alpha)(|\overrightarrow{p}_i - \overrightarrow{p}_j|/D) \tag{4.8}$$

where α is a user defined constant between 0 and 1 (0.5 in our approach), and D is the maximum distance between objects i and j.

In order to prevent two objects from overlapping, we introduce some form of repulsive force. The repulsive force has an electrostatic-like characteristic, which is strong at close distances and weak at long distances. So, whenever two objects are getting too close, the repulsive force will be very strong to push them apart. The repulsive force on object i due

to object j is determined by the following equation:

$$\overrightarrow{F}^{r}_{(i,j)} = K^r \cdot (r/d)^2 \frac{\overrightarrow{p}_i - \overrightarrow{p}_j}{|\overrightarrow{p}_i - \overrightarrow{p}_j|} \tag{4.9}$$

where K^r is the repulsion constant which is set to 1 in our case, r denotes the sum of $R_i + R_j$ where R_i and R_j are the radii of objects i and j, respectively, and d is the distance between the two objects.

In order to find the equilibrium positions for a set of objects, we need to find a position for each object such that the total force on object i, $\overrightarrow{F}_{total}(i)$, is zero. $\overrightarrow{F}_{total}(i)$ is computed as shown in Equation 4.10:

$$\overrightarrow{F}_{total}(i) = \sum_{j=1, i \neq j}^{n} [\overrightarrow{F}^{a}_{(i,j)} + \overrightarrow{F}^{r}_{(i,j)}] \tag{4.10}$$

This problem can be solved using the Newton-Raphson's method [101]. The equilibrium position for object i, $\overrightarrow{p}_{(i,e)}$, can be found as follows:

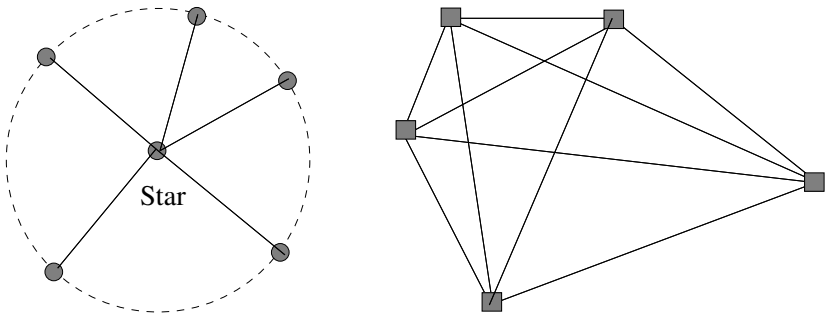
$$\overrightarrow{p}_{(i,e)} = \overrightarrow{p}_i - k_e \overrightarrow{F}_{total}(i) \Big/ \frac{\partial \overrightarrow{F}_{total}(i)}{\partial \overrightarrow{p}} \tag{4.11}$$

where k_e is a constant which in our approach is set to 0.5, and $\overrightarrow{p}_i$ is the original position of object i. We consider $\overrightarrow{p}_{(i,e)}$ as an equilibrium position when $|\overrightarrow{p}_{(i,e)} - \overrightarrow{p}_i| < \epsilon$, where ϵ is the maximum user chosen admissible tolerance.

We perform this computation on all objects one by one until the equilibrium positions of all objects are found.

4.3.3.2 Net Placement

After we have computed a net cluster floorplanning, we use the force-directed method to obtain a coarse net placement. In this process, an object refers to a net. In order to characterize the properties of nets, we use the interconnect model presented by Mo et al. [102], where each net is modeled as a star instead of a complete graph. Figure 4.5(a)



(a) (b)

Figure 4.5. (a) Star Model of a 5-pin Net. (b) Complete Graph Model of a 5-pin Net.

shows an example of a 5-pin net represented by the star model. For a k-pin net represented by this model, each net has a *star* node to which all k terminals of the net are connected. Hence, k wires and a star node suffice. On the other hand, the complete graph representation needs $k\,(k-1)/2$ wires. Figure 4.5(b) gives an example using this model for a 5-pin net. For nets with large number of pins, the complete graph model leads to redundant wires and overestimates the wirelength. The star model is appropriate for our force-directed method since the star node and the terminal attracts each other such that the wirelength will be optimized. In addition, it contains information of how nets may be routed by the router. But complete graph model does not possess any information about possible routing. In this paper, we only consider single star node nets. This model can be easily expanded to deal with multi-star nets. For example, we can model two-star net as an *elliptic*, where the two stars are the $foci$ of the ellipse.

The force on a net n_i due to a net cluster C_l is given by Equation 4.12.

$$\overrightarrow{F}(n_i) = \sum_{(n_i,n_j)\in E} \overrightarrow{F}^a(n_i,n_j) + \sum_{\substack{i\neq j \\ n_j\in C_l}} \overrightarrow{F}^r(n_i,n_j) + k_l\,\overrightarrow{F}^a(n_i,C_l) \tag{4.12}$$

The first term accounts for the attractive force between net n_i and all nets n_j which are adjacent to n_i in the net dependency graph G. The second term describes the repulsive force between nets in the same net cluster which is useful to avoid all nets being placed in

the same location. The last term is the force to maintain net n_i to be in the region allocated for the corresponding net cluster C_l. The factor k_l is the number of nets connected with n_i but do not belong to C_l. Initially, all nets in the same cluster are placed randomly in the designated area for this cluster. Then, the equilibrium positions of the nets are found using Equation 4.11. Note that the position of a net here is actually referring to the position of the star node of the net. This procedure works in a net by net fashion instead of processing all nets simultaneously.

4.3.4 Logic Cell Placement

In this phase, the detailed logic cell placement is generated again using the force-directed method. Each cell is also modeled as a circle and its radius is computed using the following formula:

$$R = (h + w)/4 \tag{4.13}$$

where h and w are the height and width of the cell, respectively.

At first, logic cells are placed randomly within the region determined in the net cluster placement phase. If a logic cell is in more than one net cluster, it will be placed inside the region of the first net cluster containing it. So each logic cell will be placed only once. An iterative force-directed scheme is carried out on the cells until the equilibrium positions are found. In order to save computing time, we will terminate this process once the number of iterations exceeds 200. The force on a slice s_i is computed as:

$$\overrightarrow{F}(s_i) = \sum_{n_j \in N_i} \overrightarrow{F}^a(s_i, n_j) + \gamma \sum_{I/O_j \in P_i} \overrightarrow{F}^a(s_i, I/O_j) + \delta \sum_{s_j \in S_i} \overrightarrow{F}^r(s_i, s_j) \tag{4.14}$$

The first term is an attraction force between slice s_i and the nets to which s_i is connected with. N_i denotes the set of nets connected with this slice. The second term is also an attraction force. If slice s_i is connected with I/O blocks, this force attracts s_i to the chip boundary. P_i denotes the set of I/O pins that are connected with s_i, and γ is a factor bigger than 1 (1.5 in our approach) to indicate the priority of interconnections with I/O

pins. The last term is a repulsive force between s_i and all slices that are adjacent to it where S_i denotes the set of slices connected with s_i. The factor δ is set to 0.5 in our algorithm. Unlike the placement problem in ASICs, it is unnecessary to specifically consider avoiding overlaps for FPGAs due to the fact that all logic will be placed into CLBs which are in discrete locations.

Once the positions of all the slices are stable, we will place them on the FPGA chip. Since the locations for CLBs are represented by integers, we should convert all slices' coordinates into integers at this time. For example, if slice s's coordinate is (4.3, 5.8), we would convert the new coordinate to (5, 6) using the *ceiling* function because the location for CLBs on the FPGA board starts from (1,1). This may cause a problem when the number of slices to be placed into the same CLB is greater than the numbers of slices available in a single CLB. This is the motivation why we have the third term in Equation 4.14. By enforcing repulsive forces on slices, their positions will not be too close to each other and this strategy effectively reduces the possibility that too many slices have to be placed into the same CLB. In case there still exists conflict, we will place those extra slices into nearby available CLBs. Generally, when we determine how to assign slices to available CLBs, we use the following criteria. First, the slices are placed according to the order obtained while reading the input netlist. Second, because routing within the same slice results in less delay, we will try to pair up slices with the largest number of connections between them and put them into the same CLB. They will be placed into $slice(0)$ and $slice(1)$, respectively. This will not only improve the timing between this slice pair but also reduce the demand for routing resources in the CLB level. Finally, we use the following scheme to *fine tune* the placement. After all slices have been placed, we will try to move slices around as long as the total netlength can be reduced. According to our experiments, in most cases this gives us slightly better results at negligible extra running time. Besides, if the number of I/O pins of the circuit is larger than 75% of the I/O blocks available on the chip, we would place the slices in the middle region of the chip. Otherwise, we place the slices in a region close to one corner of

the chip. The results generated using this simple scheme have better performance than the results without using it.

4.3.5 I/O Pin Matching

Once the slices have been placed, we need to place the I/O pins onto the I/O blocks on the FPGA chip. Essentially, this is a weighted bipartite matching problem which can be solved optimally by the Munkres' assignment algorithm [103]. A revised version of this algorithm was used in our previous work [104] in order to improve the critical path delay. In this paper, we modify the I/O matching algorithm we used previously to further reduce the running time. The computational complexity of the Munkres' algorithm is $O(n^3)$, where n is the number of I/O blocks available on the chip. The running time would be extremely high for I/O intensive circuits. We first perform a topological sort starting from the input slices (slices connected to input pins) of the circuit to compute the longest path delay for each output slice (slice connected to an output pin). We assume that for each slice connected with input pin, we can place the corresponding input pin at the nearest positions on the boundaries of the device. The delay is estimated using the Manhattan distance between slices because the routing information is not yet available at this time. By doing so, we can find the critical path for every output slice s_i and we also derive the corresponding input slice on this path. We call this input slice the *critical input slice* of output slice s_i. Then, starting from the output slice s_i with the largest delay, we find the nearest I/O block available on the FPGA chip to place the output pin for slice s_i. Then we place the input pin for the corresponding critical input slice in the same way if this input pin has not been placed yet. In case an input slice s_i is on the critical path of multiple output slices, s_i's input pin position is decided while processing the first output slice which has s_i as its critical input slice. Once all pins connected to output slices and critical input slices have been placed, we compute the minimum weighted bipartite matching for the I/O block matching of the remaining input pins to minimize the total interconnections. The advantages of this method include: (1) By placing the pins connected to output slices and critical input slices first,

we can improve the pin-to-pin delay of the circuit because basically we want to prioritize the critical paths for routing. (2) Since some I/O blocks have been placed, the problem size is decreased and thus the running time is reduced as well. Through our experiments, this scheme not only improves the post-layout delay but also reduces the computation time dramatically.

4.3.6 Summary of the Proposed Algorithm

We have introduced our force-directed performance-driven algorithm for hierarchical FPGAs. The overall design flow of our algorithm is shown in Figure 4.6. Lines 0 – 2 account for the top level of our proposed approach where a net-dependency graph is constructed on top of the input netlist derived after technology mapping and net-level floorplanning is performed using simulated annealing. Line 3 – 6 is the intermediate level of our algorithm. In this stage, we find a coarse net-level placement with a force-directed method. Lines 7 – 11 describe the phase that computes the cell-level placement. Lines 12 – 19 place the I/O pins using a minimum bipartite matching scheme. Since we have used a number of constants in our algorithm, we summarize these constants as shown in Table 4.1.

Table 4.1. List of Constants Used in Our Work.

Name	In Eqn.	Meaning	Value
k_1	(4.2)	routing overhead for CLBs not connected to I/O pins	1
k_2	(4.2)	routing overhead for CLBs connected to I/O pins	1.3
Δ	(4.3)	factor to reduce temperature during SA	0.95
α	(4.8)	factor to change the spring constant K_1	0.5
K^r	(4.9)	spring constant computing repulsive force	1
k_e	(4.11)	compute the equilibrium position	0.5
γ	(4.14)	weight on interconncetions with I/O pins during cell placement	1.5
δ	(4.14)	weight on interconnections between slices during cell placement	0.5

Force-directed Placement Algorithm
Input: Netlist obtained after technology mapping
Output: A performance-driven placement solution

```
                /* Net-cluster level floorplanning */
0  Construct net dependency graph G;
1  Perform clique partitioning on G;
2  Net-cluster-based floorplanning by simulated annealing;
                /* Coarse net placement  */
3  while nets are not stable do
4    Calculate forces on nets;
5    Find new positions for nets;
6  endwhile
                /* Slices placement  */
7  while slices are not stable do
8     Calculate forces on slices;
9    Find new positions for slices;
10 endwhile
11 Perform slices movement;
                /* I/O pins matching  */
12 Perform a topological sort to compute the longest path delays of the output slices;
13 Q ← a queue of the output slices according to their longest path delays in descending order;
14 while Q ≠ ∅ do
15    slice(i) ← dequeue(Q);
16    Place the output pin of slice(i) to the closet I/O block available;
17    Place the input pin of the corresponding critical input slice in the same way if it has not been placed;
18 endwhile
19 Place the rest of input pins by min-weight bipartite matching;
```

End-algorithm

Figure 4.6. Force-Directed Performance-Driven Placement Algorithm.

4.4 Experimental Results

We have implemented the force-directed placement algorithm using the C++ language and tested on a set of MCNC benchmarks on a Pentium 1.5 GHz Linux system with 256-MB RAM. We compare our force-directed placement algorithm with Xilinx Foundation 4.1i, a commercial FPGA CAD tool. We also modify the proposed algorithm in order to compare with VPR [87], a well-known place and route tool developed by the University of Toronto.

4.4.1 Comparison with Xilinx Foundation Tool

The input netlist is in VHDL format which can be derived using the blif2vhdl script. Firstly, we run Xilinx Foundation alone to map, place, and route each design. Secondly, we use the Xilinx tool for the mapping, and use our force-directed algorithm to place the mapped circuit and perform re-entrant routing using PAR of Xilinx Foundation tool. Our target device is Xilinx Virtex v800fg680 and we use the default settings while we run the Foundation tool. We used "par -k -rl 2" to perform the re-entrant routing on the placement generated by our algorithm and the routing level "-rl 2" was the same when we run the entire process solely using Foundation Tools. The detailed experimental flow is shown in Figure 4.7.

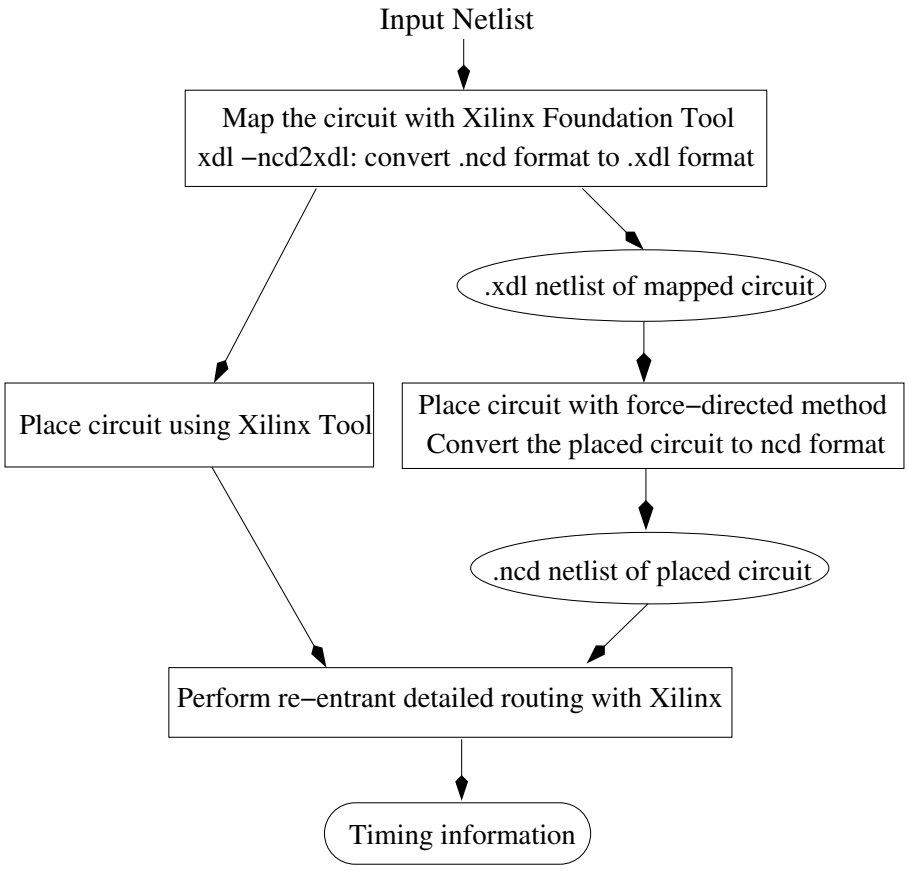


Figure 4.7. Experimental Flow of Our Algorithm.

We first conducted our experiments on a set of combinational circuits. Table 4.2 shows the characteristics of these circuits where the number of slices and nets are obtained after technology mapping.

Table 4.2. Characterics of Combinational Circuits.

Ckt	# Slices	# Nets
c2670	315	332
c3540	266	425
c5315	541	643
c6288	368	628
c7552	580	698
dalu	265	411
des	1159	1530
i10	885	1044
i8	414	529
k2	304	454
pair	554	644

The experimental results for these combinational circuits are shown in Table 4.3 where D1 refers to the maximum pin-to-pin delay from a PI to a PO, D2 is the average connection delay between slices, and T is the CPU time to place each design. Note that the CPU time for Xilinx is derived directly from the place and route report generated by Xilinx Foundation Tool. On average, we improve the maximum pin-to-pin delay and average connection delay by 10.2% and 19.3% respectively. We can see that our algorithm used more CPU time than Xilinx. However, the longest runtime (for "des") is around three minutes, which is not significantly inferior to that of Xilinx.

We also did experiments on a set of sequential circuits and the characteristics of these circuits are shown in Table 4.4. The experimental results for sequential circuits are shown in Table 4.5. The maximum improvement achieved is 75.7% for "bigkey" and the average improvement on the maximum clock speed is 20.7%. In general, our force-directed placement approach outperforms Xilinx's CAD tool. It not only results in better timing, but also reduces the wirelength dramatically which is also a good indication for better routability. In addition, as we do not have the information about how the PAR tool of Xilinx Foundation performs routing, sometimes we could not improve the longest path delay to its full potential.

Table 4.3. Comparison with Xilinx Foundation for Combinational Circuits.

	Xilinx			Ours			% Improvement	
Ckt	D1(ns)	D2(ns)	T(s)	D1(ns)	D2(ns)	T(s)	D1	D2
c2670	25.50	2.80	4	23.35	2.31	8.4	8.4	17.9
c3540	35.46	2.73	5	33.74	1.73	16.5	4.9	36.6
c5315	31.76	3.00	7	30.07	3.01	32.3	5.3	-0.003
c6288	58.17	2.90	6	56.67	1.96	36.9	2.6	32.4
c7552	37.26	3.20	8	30.07	2.88	40.9	19.3	10
dalu	33.08	2.27	3	26.91	1.60	13.9	18.7	29.5
des	37.69	3.05	63	37.07	3.25	188.2	1.6	-6.5
i10	44.68	3.92	35	43.23	3.26	89.6	3.2	16.8
i8	31.89	3.41	10	23.04	2.31	28.7	27.8	32.3
k2	29.21	2.98	3	25.02	1.89	19.1	14.3	36.6
pair	27.73	3.25	6	25.96	3.04	35.8	6.4	6.5
Avg.							10.2	19.3

Table 4.4. Characterics of Sequential Circuits.

Ckt	# Slices	# Nets
bigkey	1066	1375
clma	2502	4653
dsip	907	1149
planet	145	237
s13207	516	739
s38584	1954	3258
sbc	250	340
scf	254	360
styr	124	209
tbk	147	275

For example, according to our observation, the router sometimes uses unnecessary detours which increases the maximum delay. Thus, we feel that our algorithm can further improve its performance if we have better access to PAR which means we can guide the router to route the critical paths with higher priorities.

4.4.2 Comparison with VPR

We have also implemented the algorithm in such a manner that we can compare it with VPR. VPR does not target hierarchical FPGAs even though it has the functionality

Table 4.5. Comparison with Xilinx Foundation for Sequential Circuits.

	Xilinx		Ours		% Improvement
Ckt	f_{max} (MHz)	Time (s)	f_{max} (MHz)	Time (s)	Clock Speed
bigkey	26.04	25	45.77	235	75.7
clma	19.52	46	22.68	422	16.2
dsip	47.90	16	50.41	148	5.2
planet	69.34	3	74.06	6.6	6.8
s13207	88.11	6	98.39	86.9	11.7
s38584	57.04	28	69.03	534	21.0
sbc	69.22	4	81.37	12.4	17.6
scf	56.67	3	59.73	14.0	5.4
styr	62.37	2	68.76	4.6	10.2
tbk	47.32	3	64.69	8.1	36.7
Avg.					20.7

to cluster LUTs into the same CLB. Once the cell level placement is finished, we do not need to pair up CLBs as we can only place a single CLB in each location on the FPGA board. The original benchmarks are in ".blif" format. Each circuit is optimized in SIS using "script.rugged" and then is technology-mapped to 4-input look-up tables (LUTs) with Flowmap. Finally, VPACK program is used to pack the circuit into CLBs and the final netlist is in ".net" format which can be taken by VPR. We first run VPR to place and route each design using the default options. We set the size of the target FPGA to be 80 x 80 CLBs and we use the file "4lut_sanitized.arch" included in the VPR package as the architecture file. Note that the default algorithm used for VPR's placer is timing-driven. Next, we run our algorithm to generate the placement and then feed it into VPR for re-entrant routing. The experimental flow to compare with VPR is similiar to the one shown in Figure 4.7 except that we do not need to convert between different file formats and VPR handles the re-entrant routing instead of Xilinx. The experimental results are shown in Table 4.6 where D1 denotes the total net delay and D2 denotes the critical path delay. On average, compared with VPR, our algorithm reduces the total net delay and the critical path delay by 11.5% and 10.7%, respectively. As for the runtime, we can see that our algorithm is slightly faster than simulated annealing based VPR placer.

Table 4.6. Comparison with VPR.

	VPR			Ours			% Improvement	
Ckt	D1(ns)	D2(ns)	T(s)	D1(ns)	D2(ns)	T(s)	D1	D2
c2670	94.4	99.0	27.9	83.6	93.1	29.2	11.5	6.0
c3540	84.3	94.9	25.5	77.2	87.1	23.5	8.4	8.2
c5315	93.3	100	47.2	84.8	92.1	41.9	9.1	7.9
c6288	154	179	31.9	133.7	169.2	27.2	6.7	5.5
c7552	147	163	42.8	126	138	48.9	14.3	15.3
dalu	92.8	96.8	21.4	70.7	74.7	23.8	23.8	22.3
des	124	127	225.3	108.5	113.8	201.3	12.5	10.4
i10	167	185	118.6	142	161	111.5	14.9	13.0
i8	84.2	89.4	43.8	77.1	79.5	34.1	8.4	11.1
k2	66.8	74.1	24.2	61.3	67.5	16.5	8.2	8.9
pair	83.1	93.2	43.7	75.6	84.6	39.1	9.0	9.2
Avg.							11.5	10.7

4.5 Conclusions and Future Work

In this chapter, we presented a net-clustering based performance-driven placement scheme for hierarchical FPGAs. We developed a top-down design flow to generate the placement for FPGAs in several levels. The main contribution of our work is the force-directed placement scheme which is usually used in ASIC based designs. Our algorithm improves the post-place-and-route critical path delay and average net-length significantly over a commercial FPGA CAD tool from Xilinx. It also outperforms VPR, a well known place and route tool from the University of Toronto.

The improvement is achieved due to the following aspects:

• Net-level clustering and floorplanning give a very good entry point for the subsequent force-directed net placement and also save the computing time over a purely simulated annealing based cell level placement.

• The introduction of attraction and repulsive forces help to reduce the interconnection length effectively.

• Cell-level force-directed approach is very efficient to optimize net-length.

• The longest path delay oriented I/O matching scheme works very well in the sense of finding the best I/O block positions for input/output connected CLBs on critical paths. It also reduces the computing time by decreasing the problem size.

In future, we would like to extend this work on improving routability. When we determine the forces on nets and slices, we will keep track of the wire density in order to avoid high traffic. We can modify the formulation on calculating forces to accommodate this objective. Besides, the I/O matching scheme will be adjusted accordingly to distribute the I/O pads evenly in order to facilitate the routing.

CHAPTER 5

HIGH LEVEL SYNTHESIS FOR PERFORMANCE DRIVEN PLACEMENT

High level synthesis (HLS) is a process of transforming digital system from an abstract behavioral description to a structural specification. It generates a register-transfer level (RTL) structure to implement the behavior. *Behavior* is known as the way the system or its components interact with their environment (mapping from inputs to outputs). *Structure* refers to the set of interconnected components which make the system. Meanwhile, a set of optimization objectives are to be achieved. Usually, these objectives in HLS include: performance, area, cost, power, reliability, and testability. A large amount of scheduling algorithms have been proposed by previous researchers [15] [16] [18] [19]. Recently, with the increase of research interest in low power design, several approaches have been proposed for high level synthesis [105] [106] [107] [108].

As we can see, there are numerous algorithms proposed in scheduling and binding in high level synthesis. However, very few algorithms have been proposed to work in such a way that the high level synthesis tool is aware of the layout information and hence is able to generate new design accordingly to enhance the performance of succeeding physical design. Xu and Kurdahi proposed a layout-driven RTL binding technique for high-level synthesis using accurate estimators [109]. Later on, they proposed another layout-driven high level synthesis approach for FPGA architecture [110]. Their work was based on simple FPGA architecture, i.e., Xilinx XC3000/XC4000 series, and they do not have any research work to follow up in this area. Kim et. al. proposed an architectural high level synthesis approach which incorporates a performance-driven to guide the post-layout scheduling [111]. The target architecture they targeted is the distributed-register architecture which explicitly separates the long interconnect delays from logic delays. But this architecture is irregular

and hence easy to cause difficulty to estimate interconnect delay precisely. We propose a performance driven high level synthesis design flow for FPGAs. Our high level synthesis tool is able to iteratively enhance the system performance with the guidance information obtained from the physical design phase.

The AUDI (**AU**tomatic **D**esign **I**nstantiation) [112] system developed by the University of South Florida is a high level synthesis tool which is capable of automatically synthesizing datapaths. Given a behavioral description of a design, it generates a structural VHDL code. And the VHDL can be given to commercial CAD tools to perform logic synthesis followed by physical synthesis.

Currently, the complexity of most placement and routing algorithms are non-linear. With the amount of logic gates on a single FPGA chip exceeding one million, the placement and routing time could be enormously long. Hence, it is very important that in the process of finalizing the placement, we can predict and optimize the circuit performance before the routing is executed.

In this chapter, we study the performance driven placement problem with high level synthesis. The motivation is that our high level synthesis can produce different VHDL codes for the same design in a short time. So we can evaluate various options and select the one yields the best performance.

This chapter is organized as follows. In Section 5.1, we briefly introduce our high level synthesis system, AUDI. In Section 5.2, we propose our performance driven placement design flow with high level synthesis. In Section 5.3, we provide the experimental results. In Section 5.4, we conclude this chapter and discuss future research directions.

5.1 Automatic Design Instantiation System (AUDI)

AUtomatic Design Instantiation System [112] is a high level synthesis system capable of automatically generating RT-level designs from behavioral descriptions. It is developed at the University of South Florida by Dr. Katkoori's research group. Currently, VLSI chip designers are fabricating CMOS transistors at very small feature sizes (90 nm, as of today

in production). This has given rise to new challenges on the design automation front. The key paradigm shift in the deep-sub-micron (DSM) regime is the dominance of interconnect phenomena such as wire-delay, cross-talk, etc. AUDI is an interconnect-centric behavioral synthesis system, which is able to synthesize fully functional structural VHDL from a behavioral data flow graph (DFG) representation. At present, this system is capable of synthesizing datapath intensive designs. Various scheduling algorithms have been implemented in the system. They vary from simple algorithms such as as-soon-as-possible (ASAP) and as-late-as-possible (ALAP), to complex force directed algorithms such as force directed scheduling (FDS) proposed by Paulin and Knight [17] and the simultaneous scheduling, allocation, and mapping algorithm (SAM) proposed by Cloutier [113]. In addition, algorithms proposed by Gopalakrishnan and Katkoori [107] [108] have been integrated in AUDI to reduce leakage power.

Allocation and mapping are implemented using a clique partitioning heuristic proposed by Tseng and Siewiorek [97]. This approach tries to form a minimal set of maximal sized cliques which results in maximum sharing between components allocated. In the case of functional unit (FU) mapping, the input to the clique partitioning heuristic is the compatibility graph of the operations in the scheduled DFG. In the case of register mapping, a compatibility graph of the edges in the DFG is formed. Two edges are compatible if they have non-overlapping lifetimes. Sharing FUs and registers is implemented using multiplexers or buses. The datapath can be synthesized using either components from an available standard cell library or the multi-threshold CMOS (MTCMOS) component library which was developed at the University of Cincinnati [114]. The library consists of functional units (i.e., adders, subtractors, and multipliers), storage units (i.e., registers), and interconnect units (i.e., buses and multiplexers).

Figure 5.1 illustrates the RT-Level model synthesized by AUDI. The top level of the design instantiates a datapath and a controller. The datapath and the controller communicate with each other with flags and control signals. They are driven by the same clock signal. Essentially, the controller is a finite state Moore machine. The controller generates

control signals which enable the registers for writing and the select signals for the interconnect units (multiplexers). Designs synthesized by the AUDI system have been justified at the RT-Level using Cadence VHDL simulator. Layouts are generated using the LAGER IV Silicon Compiler [115] and verified using HSPICE.

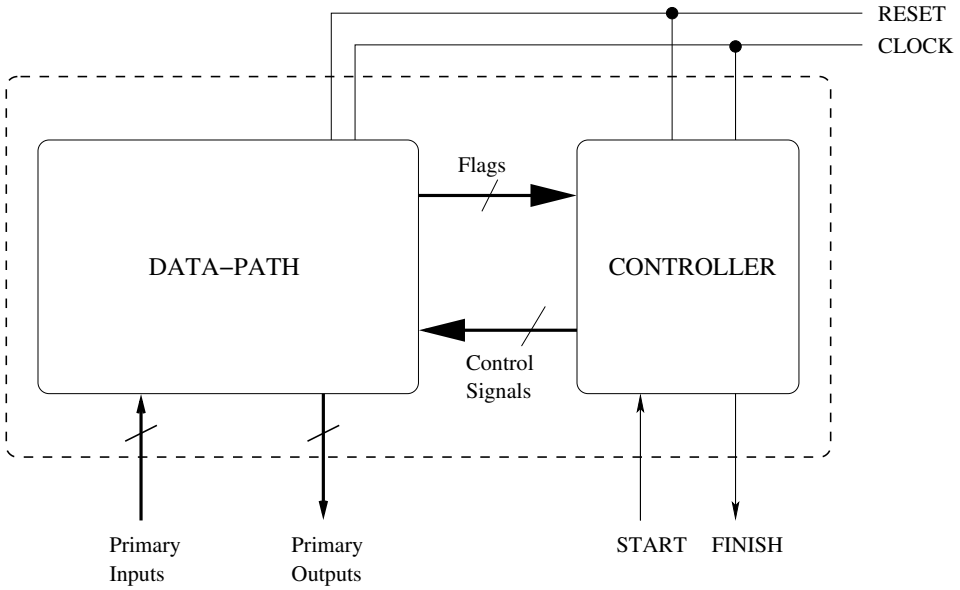


Figure 5.1. RT-Level Design Model of AUDI System.

AUDI system takes an AIF (audi intermediate format) file as its input. It describes the behavioral of the design in a straightforward way. An example design named "mx2" is given in Figure 5.2. Reserved word "inputs" denotes the primary inputs and "outputs" denotes the primary outputs of the system. Registers are declared with "regs" and operations are declared with "op". In the example "mx2", $a1$ is an input vector, *yout* is an output vector, $r1$ is a register. Note that they are all 8-bit wide. The data flow graph (DFG) representing this design is shown in Figure 5.3(a). A scheduling corresponding to this DFG is shown in Figure 5.3(b).

While generating the structural VHDL netlist for the datapath, a self-explanatory header pertaining to the binding of operations to functional units and registers is also generated. The header information of the datapath corresponding to the example shown above is

```
inputs a1 8 y1 8 a2 8 y2 8
outputs yout 8
regs r1 8 r2 8
op1 MULT 8 a1 a2 r1
op2 MULT 8 y1 y2 r2
op3 ADD 8 r1 r2 yout
end
```

Figure 5.2. Behavioral Description of Design "mx2".

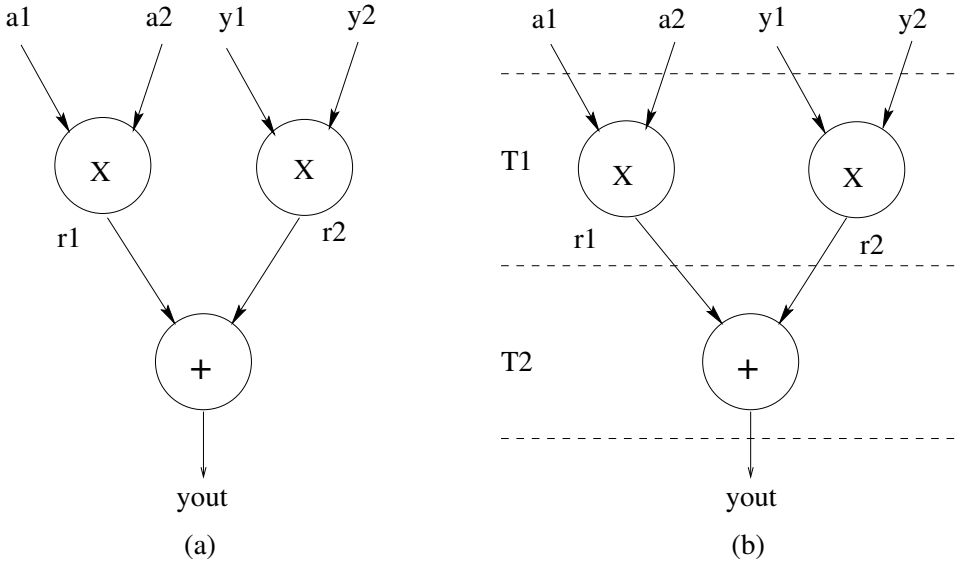


Figure 5.3. (a) DFG Representation of "mx2". (b) A Scheduling for "mx2".

given in Figure 5.4. It consists of two multipliers and one adder. The interconnections are implemented by multiplexers.

5.2 Performance Driven Placement with High Level Synthesis

In this section, we present the details of our performance driven placement approach with high level synthesis. An RT-Level VHDL description is first generated with AUDI system. The design is synthesized using a commercial CAD tool. We estimate its timing without going through the time-consuming routing process. Iteratively, we evaluate the design performance and provide guidance information to the high level synthesis tool to

– Type: Datapath
– CDFG statistics:
– * Number of PI's: 4
– * Number of PO's: 1
– * Number of internal edges: 2
– * Number of Operations: 3
– * Types of Operations:
–
– Design Flow/Algorithm Information:
– * Scheduling: ASAP
– * Allocation: User's Choice
– * Binding: Automatic
– Interconnect style: Mulitplexor-based
–
– Design Information:
– * Registers: 7
– * Functional units: 3
– Resource Id=0 type = MULT :
– Index = 0 type= MULT width = 8
– Mapped Ops = op1
– Index = 1 type= MULT width = 8
– Mapped Ops = op2
– Resource Id=1 type = ADD :
– Index = 0 type= ADD width = 8
– Mapped Ops = op3
–
– * Register Optimization Information:
– Register #0 (width = 8) a1
– Register #1 (width = 8) y1
– Register #2 (width = 8) a2
– Register #3 (width = 8) y2
– Register #4 (width = 8) yout
– Register #5 (width = 8) r1
– Register #6 (width = 8) r2
–
– Controller:
– * Type: Moore
– * Number of states: 2
– * Number of control bits: 7

Figure 5.4. Datapath Information.

improve performance. New VHDL code is produced as design entry for the CAD tool. Finally, we run our net clustering placement algorithm proposed in Chapter 4 to place and perform re-entrant routing on the final design. The layout information (pin-to-pin delay, wirelength, etc.) is also available for comparison.

5.2.1 Overview of the Proposed Design Flow

High level synthesis tool has the advantage of generating various RT-Level netlists quickly for functionally equivalent designs. This motivates our research interest in studying the performance driven placement problem with high level synthesis. Our proposed design flow is shown in Figure 5.5.

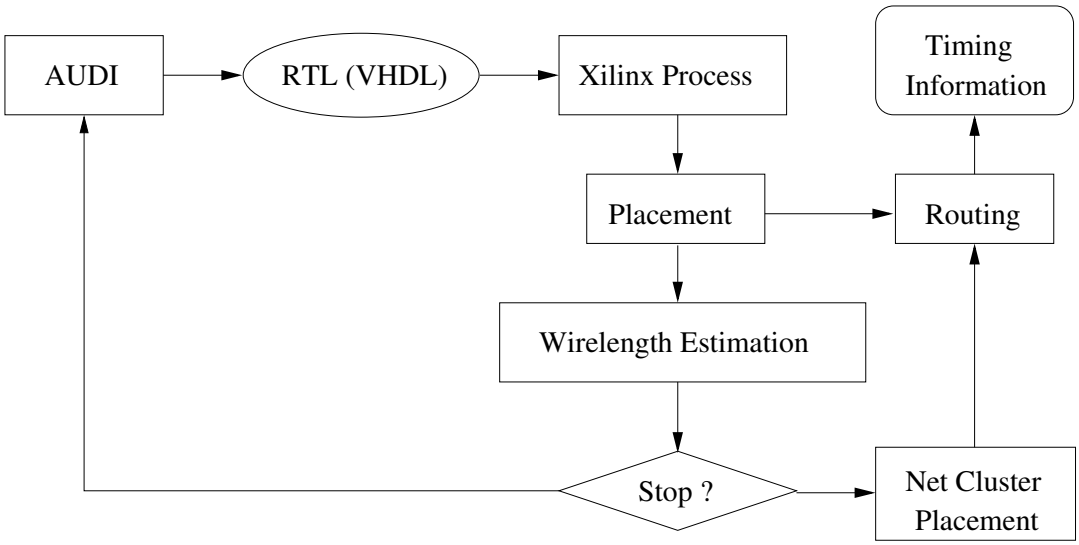


Figure 5.5. Overview of Design Flow.

First, we generate the initial RTL netlist (VHDL) with AUDI system. Here, we do not choose any specific scheduling algorithm and do not selectively allocate resources. Then we feed this VHDL description to Xilinx's CAD tool to perform logic synthesis. Next, a placement is performed on which we estimate the wirelength of this design. An approximation of the circuit performance is obtained. Based on this approximation, we provide the high level synthesis tool guidance to generate a new RTL design. AUDI chooses different scheduling algorithms as well as allocate various number of resources according to the guidance. This

process is repeated until no further performance improvement is possible or the optimization has been achieved. Finally, we place the design with our own net clustering based placement algorithm. Re-entrant routing is executed to route the design and generate the layout information.

5.2.2 Estimation of the Design Performance

Once a design is logic synthesized and placed, we can obtain a coarse performance estimation. This means time-consuming routing will not be performed in this phase. Different VHDL designs are derived by AUDI for the same behavioral description. After logic synthesized by Xilinx CAD tool, they should contain different number of logic blocks and interconnections. A fast and dependable performance estimation algorithm is necessary to estimate the current design (area, routing congestion, timing, etc.) and provide guidance for the high level synthesis tool to generate a better design over the iterations. Basically, we use Rent's rule [116] to predict interconnection. Rent's rule gives an empirical relationship between the number of pins and the number of logic blocks in a design, which tends to form a straight line in log-log plot. The relationship is shown in Equation 5.1:

$$N_p = K_p \; {N_g}^p \tag{5.1}$$

Here, N_p is the number of pins or the number of external signals connecting to the logic blocks, K_p is a proportionality constant which is the average number of interconnections per block, N_g is the number of logic blocks in a logic design, and $0 < p < 1$ is the *Rent's exponent* [117]. Researchers found that complex architectures are typically characterized by a Rent's exponent range $0.5 < p < 0.8$. In addition, systems with regular architecture have low Rent's exponent value. Therefore, we set p to 0.5 in our work due to the regular architecture of FPGAs. Several works have been proposed based on Rent's rule to predict interconnection and routability. Van Marck et. al. [118] proposed a technique to describe local variations in interconnection complexity. It fits well for algorithm such as VPR [87]

which uses a linear wirelength as cost function. Li and Banerji [119] derived a statistical model for predicting routability for hierarchical FPGAs prior to placement. Brown et. al. [120] proposed a stochastic model for estimating the channel densities for FPGA routing architecture. It predicts routing resource requirements in the post placement stage of design. In our approach, we adopt the method proposed by Buyuksahin and Najm [121] to estimate the average wirelength. It was shown in [121] that the average interconnect length estimation error is 14.4%, which is acceptable in predicting the actual delay. By integrating Rent's rule in our estimation approach, we are able to predict the interconnection delay consistently. This helps us to provide meaningful and instructive guidance for our high level synthesis tool. Hence, AUDI is able to generating new VHDL netlist which is highly possible to improve the system performance.

5.2.3 Iterative Design Space Search

Once a placement is computed, we do not actually route it before we are confident that the design performance will meet our design objective. This is mainly due to the fact that for CAD tools the majority of the design time is spent on performing routing. Since high level synthesis tool is capable of generating various RTL netlists for the same design very efficiently, we would like to find the *best* netlist in a certain solution space before we finally place and route it. The *best* netlist indicates the one that can be physically synthesized to yield the best performance among a set of netlists.

During high level synthesis, we can choose different scheduling algorithms, assign various number of resources such as adders, multipliers, multiplexers, etc. This leads to a very large solution space. Once we start searching in this space, we would like to find the correct search direction quickly which means that the algorithm converges fast. While probing the design space, our approach works incrementally. For example, assume we are trying to determine the number of one particular type of resource to be allocated. We first assign the minimal, the maximal, and the median number of resources to various designs. With the feedback derived from the interconnect estimation program, we would know in which

range the number of resource allocation is preferred. Then the high level synthesis tool will try to increase/decrease the resources allocated if it keeps on yielding better estimation. To avoid locally optimal solution, we also allow searching in opposite direction at a smaller rate. For instance, after searching in one direction for 10 times, we would like to try the other direction once. This can be done by keeping a record of the start point of the current searching process. Therefore, our design space search approach is non-greedy in nature. Meanwhile, we store the best design generated so far. Note that it does not necessarily mean the new design generated will always outperform the best design as the overall performance is affected by other factors as well, i.e., number of other resources allocated, scheduling algorithm selected, etc. But through this training process, we are able to guide the high level synthesis tool to search towards the better direction overall. This iterative process is terminated once it falls into one of the following situations:

- The estimation shows that the best synthesized design could satisfy the original system design objective;
- There is no further improvement gained after a certain number of consecutive searches which implies that it is unlikely to achieve further performance gain any more.
- A predefined number of iterations have been tested. This reduces the running time effectively.

5.3 Experimental Results

We have developed our proposed high level synthesis flow for performance driven placement. Firstly, our high level synthesis suite (AUDI) takes a behavioral data flow graph (DFG) representation as its input. An RTL design in VHDL format is generated without specifically selecting functional resources. This design is fed into Xilinx ISE CAD tool and logic optimization and physical synthesize are carried with this tool. Once the circuit is placed, we estimate its performance in terms of timing, routing congestion, and area. Based on the performance of the current design, a guidance is generated for AUDI system

to generate the next design. The objective is to search the design space towards a better direction. Basically, different number of resources (adders, subtractors, multiplexers, etc.) can be allocated and different scheduling algorithms (ASAP, ALAP, FDS, etc.) are used. This process is repeated until a predefined design target is met or there is no possible performance improvement. Once this iterative process is over, the finalized design is again given to Xilinx ISE for mapping. We run our own net clustering algorithm to place the mapped circuit and finally a re-entrant routing is carried using Xilinx ISE to generate timing information.

We have tested our proposed design flow over a set of behavioral benchmarks which can be taken by AUDI system. The characteristics of the benchmarks used are shown in Table 5.1. In this table, the number of primary inputs (PIs), the number of primary outputs (POs), the number of internal registers used, and the total number of operations of the designs are depicted. Operations include MULT, ADD, SUB, etc. Note that all PIs and POs are all vectors and their widths are given in the corresponding column "Width".

Table 5.1. Description of Behavioral Benchmarks for AUDI System.

Design	# PIs	# POs	# Registers	# Operations	Width
iir	10	1	8	9	8
ewf	9	7	26	33	4
fir	10	1	8	9	8
fft4	6	4	6	10	8
latt	8	3	10	13	8
dct8ip	24	8	40	48	4

The target device we chose is Xilinx Virtex v800fg680 and we used the default setting to run the Xilinx ISE tool to mapped, place, and route the initial designs. Once we have our own placed circuits, we use Xilinx ISE again to run re-entrant routing with the same set of default setting. The experimental results are given in Table 5.2, where *Wire* denotes the average wire delay, and *Delay* denotes the maximum pin-to-pin delay. Designs generated after going through the proposed design flow outperform the corresponding initial designs significantly. On average, we achieve 24.7% reduction in maximum delay. Meanwhile, we are

able to reduce the average wire delay by 11.5%. The biggest gain is for the design "dct8ip" where we successfully reduce the critical delay by 49.7% and reduce the average wire delay by 38.1%. Note that for two designs (ewf and fft4), we increase the average wire delay but we still get reduction in the longest path delay. We also note that for the largest benchmarks, we get the highest percentage of performance improvement. This makes us to believe with great confidence that for bigger designs, which have a larger solution space accordingly, our proposed approach has more room to make improvement. Therefore, the combination of high level synthesis and physical synthesis has excellent potential to outperform a purely physical synthesis design scheme.

Table 5.2. High Level Synthesis for Performance Driven Placement.

	Initial Design		Final Design		Improvement (%)	
Design	Wire (ns)	Delay (ns)	Wire (ns)	Delay (ns)	Wire	Delay
iir	2.873	9.318	2.465	7.443	14.2	20.1
ewf	2.552	9.384	2.737	6.971	-7.2	25.7
fir	2.664	7.508	2.341	5.873	12.1	21.8
fft4	2.414	9.929	2.477	8.107	-2.6	18.4
latt	2.661	8.369	2.270	7.254	14.6	13.3
dct8ip	2.743	8.982	1.699	4.464	38.1	49.7
Average					11.5	24.8

In order to justify the accuracy of our delay estimation scheme, we also run a series of experiments on a particular design "latt". The procedure is depicted as follows. After a placement is generated and the estimated delay is computed, we will proceed to route the design to get the actual delay information. But the feedback given to AUDI is still generated based on the estimated delay information. If the entire process runs for N iterations, we will finally have N estimated delay values and N actual delay values. Note that we route the design in every iteration only in order to compare the accuracy estimation. This is not the case while we actually run our experimental flow. In fact, we will route the design only one time at the end, when we are satisfied with the performance of final design. We show the estimated and actual delay values for design "latt" in Table 5.3. On average, the estimation error of average interconnect delay is 11.5%. This accuracy is acceptable and

hence it helps us to predict the performance of the RTL netlist generated by AUDI. We also plot the data in Table 5.3 to show that our space search approach converges quickly. We can also see that our proposed approach is non-greedy in nature as in some cases we actually generate a netlist which results in worse performance. This is useful in that our algorithm can jump out of local optima.

Due to the fact that we have to run the entire flow in a cross-platform fashion, i.e., running Xilinx in Windows system and AUDI in Linux, we did not report the runtime for the benchmarks. However, excluding the time taken by Xilinx Tools to parse, logic synthesize, and place each design, it usually takes less than 30 seconds to generate a new RTL netlist in each iteration.

Table 5.3. Delay Estimation for "latt".

Iteration	Estimated Delay(ns)	Actual Delay(ns)	Error (%)
1	3.46	4.06	17.3
2	3.38	3.85	13.9
3	3.14	3.42	8.9
4	3.02	2.75	9.8
5	2.61	2.89	10.7
6	2.57	2.82	9.7
7	2.58	2.31	11.7
8	2.48	2.27	9.2
Average			11.5

5.4 Conclusions and Future Work

In this chapter, we have proposed a performance driven placement design flow with high level synthesis. It is an iterative searching approach which converges very quickly. Given a behavioral description of a design, our high level synthesis tool (i.e., AUDI) generates an RTL netlist which is given as input to Xilinx logic and physical design tool. Once the placement is computed, a performance estimation algorithm is developed to evaluate the performance (area, routing congestion, timing, etc.) of this particular design. Feedback is given to AUDI system for generating the next design towards a better direction. With this

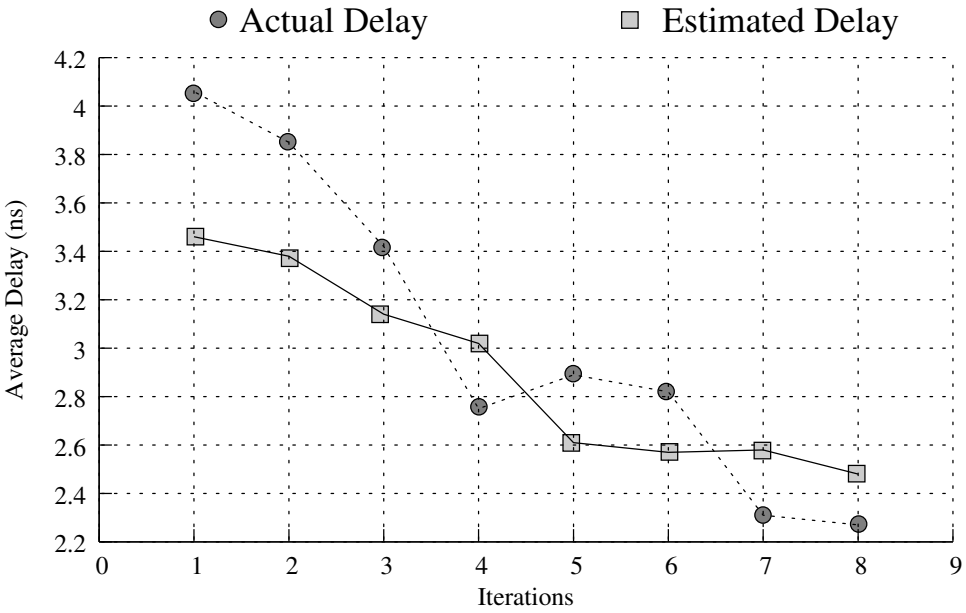


Figure 5.6. Delay Estimation and Cost Convergence for "latt".

guidance, the high level synthesis tool converges quickly and ends up with a design capable of improving the overall performance after placement and routing. Compared with Xilinx CAD tool, our approach improved the critical pin-to-pin delay significantly. This gives us confidence that the combination of high level synthesis tool and physical design tool has good potential to enhance the performance of modern VLSI designs.

The major contributions of our research include the following:

- The interaction between high level synthesis and physical synthesis is unique. It gathers physical level information and directs the high level synthesis tool for generating new designs very effectively.

- Our performance estimation tool is able to evaluate the performance of the current design with consistent accuracy. Hence, it provides instructive information to the HLS tool to search in the correct direction.

- Our net clustering placement algorithm is used effectively to further improve the performance by generating timing-driven placement solutions.

In our future research work, we would like to extend this in the following directions:

- Currently, we do not have much access to the router. Hence to gain more knowledge of the router will be very beneficial to predict the timing more accurately.
- Add functionalities to our high level synthesis tool so that it could have handy layout information once a RTL design is generated. This can be done to communicate with libraries developed for different target devices. And the physical level netlist can be available after the RTL netlist is generated. This can expedite the synthesis process.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this dissertation, we study logic synthesis and physical synthesis for FPGAs. As power consumption has become a major concern in VLSI design, precautions should be taken in early design phase. Several low power driven technology mapping algorithms are proposed in this work for FPGAs.

In Chapter 3, we present network flow based approaches to reduce dynamic power consumption. Switching activity at the output of a LUT affects the power consumed by this LUT, hence intuitively we would like to minimize the total switching activity of the mapped circuit. However, this strategy does not always work out. Our work distinguishes from other research work in that while computing a subnetwork LUT covering, we consider its effect on overall mapping solution. We choose the one that yields the least power consumption. In addition, we develop a very efficient incremental network flow computation method which expedites the process in finding the LUT covering. We achieve dramatic power savings as well as area reduction over previous LUT-based FPGA low power technology mapping algorithms. This work is further extended to handle delay-optimal low power mapping. Without increasing the optimal delay of the network, we map LUTs on non-critical path as long as power deduction is obtained. Compared with a well-known simultaneous delay and area optimization mapping algorithm, our approach achieves significant power reduction without increasing the optimal delay of the mapping solution.

Due to the fact that interconnect dominates the delay in current deep sub-micron regime, improve timing has become a necessity in every phase of physical design. Placement is of great importance because it defines the on-chip interconnects which have become the

bottleneck in enhancing the overall timing. We propose a net clustering based placement algorithm which works very well for net concentric modern designs.

In Chapter 4, a force-directed net cluster level placement approach is presented. Each net cluster denotes a set of strongly connected nets which should be routed in the same region. A net cluster floorplanning is carried out using Simulated Annealing to optimize the total wirelength. Next, a coarse net placement is computed using force-directed method. Then we find the detailed cell level placement again using force-directed approach. Finally, a novel maximum weighted bipartite matching method is developed to match the I/O pins. This method is very effective because it gives priority to cells on critical path which reduces the critical path delay. In addition, it reduces the bipartite matching problem size which in turn saves the running time dramatically. Our proposed algorithm leads to excellent post-layout-delay improvement.

In Chapter 5, we present a design flow with a combination of high level synthesis and net clustering placement for FPGAs. High level synthesis is capable of generating various behavioral descriptions (e.g., VHDL) for the same design. Once we have it mapped and placed with a CAD tool, we estimate the total wirelength which gives an approximation of the timing without actually route the design. If we see improvement over the iteration, we will guide the high level synthesis tool to generate another design accordingly. Finally, we place the design with our own net clustering placement algorithm. Experimental results show significant critical pin-to-pin delay improvement over a set of behavioral benchmarks.

The work presented in this dissertation can be further improved or continued in the following areas:

- Since multiple power supply has become common in state-of-the-art FPGA chips, low power technology mapping problem should be studied considering the effect of different voltages. Take dual-voltage FPGA for example, for logic components with high switching activities, they can be powered using the lower supply voltage in order to save power. When we take delay into account, logic components on critical path are

preferably to be powered up using the higher power supply. For non-critical components, the lower power supply can be used without penalizing the overall performance.

- With the design complexity has reached multi-million blocks on a single chip, the compiling time has become a very important factor in evaluating the performance of the CAD tool. It will be of great interest to develop a ultra-fast placement algorithm without deteriorating the performance too much. Quadratic programming based approach is the most promising way to handle this problem.

- High level synthesis tool has not become popular in EDA industry currently mainly because it fails to provide dependable layout prediction. Due to its capability of probing the solution space more efficiently and quickly, high level synthesis is very important for modern large scale designs. Hence, it should draw large amount of research interest in exploring the precise correlation between high level synthesis and layout estimation in the years to come. A future CAD tool should contain both of high level synthesis tool and physical synthesis tool and they are expected to work jointly.

REFERENCES

[1] Z.-H. Wang, E.-C. Liu, J. Lai, and T.-C. Wang. "Power Minimization in LUT-Based FPGA Technology Mapping". In *Proceedings of ASP-DAC*, pages 635–640, 2001.

[2] A. H. Farrahi and M. Sarrafzadeh. "FPGA Technology Mapping for Power Minimization". In *4th International Workshop on Field Programmable Logic and Applications*, pages 66–77, September 1994.

[3] G. E. Moore. "Cramming more components onto integrated circuits". *Electronics*, 38(8):114–117, April 1965.

[4] Naveed Sherwani. *"Algorithms for VLSI Physical Design Automation"*. Kluwer Academic Publishers, 1999.

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

[6] Xilinx Inc. *"The Programmable Logic Data Book"*. San Jose, CA, 1999.

[7] D. Hill. "A CAD System for the Design of Field Programmable Gate Arrays". In *Proceedings ACM/IEEE Design Automation Conference*, pages 187–192, 1991.

[8] Actel Corporation. *"Accelerator Series FPGAs - ACT3 Family"*, 1997.

[9] Actel Corporation. *"SX Family of High Performance FPGAs"*, 2001.

[10] Xilinx Inc. *"Virtex 2.5 V Field Programmable Gate Arrays"*, 1998.

[11] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic. *"Field Programmable Gate Arrays"*. Kluwer Academic Publishers, 1995.

[12] N. Weste and K. Eshraghian. *"Principles of CMOS VLSI Design: A System Perspective"*. Addison Wesley Reading, 1993.

[13] M. C. McFarland, A. C. Parker, and R. Camposano. "Tutorial on High-Level Synthesis". In *Proceedings 25th ACM/IEEE Design Automation Conference*, pages 330–336, 1988.

[14] R. Composano and W. Wolf. *"High-Level VLSI Synthesis"*. Kluwer Academic Publishers, 1991.

[15] S. Hayanal and F. Brewer. "Automata-based symbolic scheduling for looping DFGs. *IEEE Transactions on Computers*, 50(3):250–267, March 2001.

[16] R. Camposano. "Path-based scheduling for synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(1):85–93, January 1991.

[17] P. G. Paulin and J. P. Knight. "Algorithms for high-level synthesis. *IEEE Design and Test of Computers*, 6(6):18–31, December 1999.

[18] E. Musoll and J. Cortadella. "Scheduling and resource binding for low power. In *Proceedings 8th International Symposium on System Synthesis*, pages 104–109, 1995.

[19] S. P. Mohanty and N. Ranganathan. "A framework for energy and transient power reduction during behavioral synthesis. *IEEE Transactions on VLSI Systems*, 12(6):562–572, June 2004.

[20] S. P. Mohanty, N. Ranganathan, and S. K. Chappidi. "ILP models for energy and transient power minimization during behavioral synthesis". In *Proceedings 17th International Conference on VLSI Design*, pages 745–748, January 2004.

[21] C. Park. *Task Scheduling in High Level Synthesis.* PhD thesis, University of Illinois at Urbana-Champaign, 1996.

[22] Altera Corp., San Jose, CA. *"Programmable Logic Devices Data Book and Design Guide"*, 1994.

[23] AT&T Microelectronics. *"AT&T Field-Programmable Gate Arrays Data Book"*. AT&T Corp., Berkeley Heights, NJ, 1995.

[24] Actel Corporation. *"FPGA Data Book and Design Guide"*, 1994.

[25] B. W. Kernighan and S. Lin. "An efficient heuristic procedure for partitioning graphs". *Bell Systems Technical Journal*, 49:291–307, 1970.

[26] C. M. Fiduccia and R. M. Mattheyses. "A linear-time heuristics for imporving network partitions". In *Proceedings 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[27] L. A. Sanchis. "Multiple-way network partitioning". *IEEE Transactions on Computers*, pages 62–81, 1989.

[28] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by simulated annealing". *Science*, 220(4598):671–680, May 1983.

[29] H. B. Bakoglu. *"Circuits, interconnections, and packaging for VLSI"*. Addison Wesley, 1990.

[30] Y.-T. Lai, K.-R. R. Pan, and M. Pedram. "FPGA synthesis using function decomposition". In *Proceedings IEEE International Conference on Computer Design*, pages 30–35, 1994.

[31] B. Wurth, K. Eckl, and K. Antreich. "Functional multiple-output decomposition". In *Proceedings ACM/IEEE Design Automation Conference*, pages 54–59, 1995.

[32] C. Bhat and N. N. Chiplunkar. "Routability-Driven Technology Mapping for LookUp Table-Based FPGAs". In *Proceedings 12th International Conference on VLSI Design*, pages 390–393, 1999.

[33] A. H. Farrahi and M. Sarrafzadeh. "Complexity of the Lookup-Table Minimization Problem for FPGA Technology Mapping". *IEEE Transactions on Computer-Aided Design*, pages 1319–1332, November 1994.

[34] R. J. Francis, J. Rose, and K. Chung. "Chortle: A Technology Mapping for Lookup Table-Based Field Programmable Gate Arrays". In *Proceedings 27th ACM/IEEE Design Automation Conference*, pages 613–619, June 1990.

[35] R. J. Francis, J. Rose, and Z. Vranesic. "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs". In *Proceedings 28th ACM/IEEE Design Automation Conference*, pages 227–233, June 1991.

[36] K. Karplus. "Xmap: a Technology Mapper Table-lookup Field-Programmable Gate Arrays". In *Proceedings 28th ACM/IEEE Design Automation Conference*, pages 240–243, June 1991.

[37] Y.-T. Lai, M. Pedram, and S. Sastry. "BDD based decomposition of logic functions with application to FPGA synthesis". In *Proceedings 30th ACM/IEEE Design Automation Conference*, pages 230–235, June 1993.

[38] K.-C. Chen, J. Cong, Y. Ding, A. Kahng, and P. Trajmar. "DAG-Map: Graph-based FPGA Technology Mapping for Delay Optimization". In *IEEE Design and Test of Computers*, pages 7–20, 1992.

[39] J. Cong and Y. Ding. "An Optimal Technology Mapping Algorithm for Delay Optimization in Look-up Table Based FPGA Designs". In *International Conference on Computer Aided Design*, pages 213–218, November 1992.

[40] R. Murgai, N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. "Improved Logic Synthesis Algorithms for Table Look Up Architectures". In *Proceedings IEEE International Conference on Computer Aided Design*, pages 564–567, November 1991.

[41] P. Sawkar and D. E. Thomas. "Technology Mapping for Table-Look-Up Based Field Programmable Gate Arrays". In *ACM/SIGDA Workshop on Field Programmable Gate Arrays*, pages 83–88, Feburary 1992.

[42] M. Schlag, J. Kong, and P. K. Chan. "Routability-Driven Technology Mapping for LookUp Table-Based FPGAs". In *IEEE International Conference on Computer Design*, pages 86–90, October 1992.

[43] N. Togawa, M. Sato, and T. Ohtsuki. "Maple: A Simultaneous Technology Mapping, Placement and Global Routing Algorithm for Field-Programmable Gate Arrays". In *Proceedings International Conference on Computer Aided Design*, pages 155–163, 1994.

[44] J. Cong and Y. Ding. "On Area/Depth Trade-Off in LUT-based FPGA Technology Mapping". *IEEE Transactions on VLSI Systems*, 2(2):137–148, June 1994.

[45] J. Cong and Y.Y. Hwang. "Simultaneous Depth and Area Minimization in LUT-Based FPGA Mapping". In *Proceedings International Symposium on Field Programmable Gate Arrays*, pages 68–74, 1995.

[46] J.-D. Huang, J.-Y. Jou, and W.-Z. Shen. "An Iterative Area/Performance Trade-off Algorithm for LUT-Based FPGA Technology Mapping". In *Proceedings International Conference on Computer-Aided Design*, pages 13–17, 1996.

[47] R. J. Francis, J. Rose, and Z. Vranesic. "Technology mapping of lookup table-based FPGAs for performance". In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 568–571, November 1991.

[48] E. L. Lawler, K. N. Levitt, and Module J. Turner. "Clustering to minimize delay in digital networks". *IEEE Transactions on Computers*, 18:47–57, 1969.

[49] J. Cong and Y. Ding. "On nominal delay minimization in LUT-based FPGA technology mapping". *Integration – the VLSI Journal*, 18:73–94, 1994.

[50] H. Yang and D. Wong. "Edge-Map: Optimal performace driven technology mapping for iterative LUT based FPGA designs". In *Proceedings IEEE/ACM International Conference on Computer-Aided Design*, pages 150–155, 1994.

[51] R. Murgai, N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. "Performance directed Synthesis for Table Look Up programmable gate arrays". In *Proceedings IEEE International Conference on Computer Aided Design*, pages 572–575, November 1991.

[52] P. Sawkar and D. E. Thomas. "Performance directed technology mapping for look-up table based FPGAs". In *Proceedings ACM/IEEE Design Automation Conference*, pages 208–212, June 1993.

[53] J. Cong and Y. Ding. "Beyond the combinational limit in depth minimization for LUT-based FPGA designs". In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 110–114, 1993.

[54] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. "MIS: A Multiple-Level Logic Optimization System". *IEEE Transactions on Computer-Aided Design*, 6(6):1062–1081, 1987.

[55] I. Levin and R. Y. Pinter. "Realizing expression graphs using table-lookup FPGAs". In *Proceedings European Design Automation Conference*, pages 306–311, September 1993.

[56] N. S. Woo. "A heuristic method for FPGA technology based on the edge visibility". In *Proceedings ACM/IEEE Design Automation Conference*, pages 248–251, June 1991.

[57] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. "Logic synthesis algorithms for programmable gate arrays". In *Proceedings ACM/IEEE Design Automation Conference*, pages 620–625, June 1990.

[58] A. Chowdhary and J. P. Hayes. "Technology mapping for field-programmable gate arrays using integer programming". In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 361–367, November 1995.

[59] V. Kommu and I. Pomeranz. "GAFPGA: Genetic algorithm for FPGA technology mapping". In *Proceedings European Design Automation Conference*, pages 300–305, September 1993.

[60] J. Cong and Y. Ding. "An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs". In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 48–53, 1992.

[61] H. Li, W.K. Mak, and S. Katkoori. "LUT-Based FPGA Technology Mapping for Power Minimization with Optimal Depth". In *Proceedings IEEE-CS Workshop on VLSI*, pages 123–128, 2001.

[62] H. Li, W.K. Mak, and S. Katkoori. "An Efficient LUT-Based FPGA Technology Mapping Algorithm for Power Minimization". In *Proceedings Asia and South Pacific Design Automation Conference*, pages 353–358, 2003.

[63] H. Li, S. Katkoori, and W.K. Mak. "Power Minimization Algorithms for LUT based FPGA Technology Mapping". *ACM Transactions on Design Automaton of Electronic Systems (TODAES)*, 9(1):33–51, January 2004.

[64] P. Villarrubia. "Important placement considerations for modern VLSI chips". In *Proceedings International Symposium on Physical Design*, page 6, 2003.

[65] S. M. Sait and H. Youssef. *"VLSI Physical Design Automation: Theory and Practice"*. World Scientific Publishing, 1999.

[66] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich. "Gordian: VLSI placement by quadratic programming and slicing optimization". *IEEE Transactions on Computer-Aided Design*, pages 356–365, March 1991.

[67] M. Wang, X. Yang, and M. Sarrafzadeh. " Dragon2000: Standard-cell placement tool for large industry circuits". In *Proceedings ACM/IEEE International Conference on Computer-Aided Design*, pages 260–263, 2002.

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

[69] A. E. Caldwell, A. B. Kahng, and I. L.Markov. "Can Recursive Bisection Alone Produce Routable Placements?". In *Proceedings ACM/IEEE Design Automation Conference*, pages 477–482, 2000.

[70] C.C-Chang, J. Cong, Z. Pan, and X. Yuan. "Physical hierarchy generation with routing congestion control". In *Proceedings International. Symposium Physical Design*, pages 36–41, 2002.

[71] W. Swartz and C. Sechen. "Timing driven placement for large standard cell circuits". In *Proceedings ACM/IEEE Design Automation Conference*, pages 211–215, 1995.

[72] T. Hamada, C.K. Cheng, and P.M. Chau. "Prime: A timing-driven placement tool using a peicewise linear resistive network approach". In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 531–536, June 1993.

[73] T. Kong. "A novel net weighting algorithm for timing-driven placement". In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 172–176, 2002.

[74] M. Hutton, K. Adibsamii, and A. Leaver. "Timing-driven placement for hierarchical programmable logic devices". In *ACM Symposium on FPGAs*, pages 3–11, 2001.

[75] C. Y. Lee. "An algorithm for path connection and its application". *IRE Transactions on Electronic Computers*, 1961. EC-10.

[76] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *"Introduction to algorithms, second edition"*. The MIT Press, 2001.

[77] M. A. Breuer. "A class of min-cut placement algorithms". In *Proceedings Design Automation Conference*, pages 284–290, 1977.

[78] M. A. Breuer. "Min-cut placement". *Journal Design Automation and Fault-tolerant Computing*, pages 343–382, October 1977.

[79] A. Dunlop and B. W. Kernighan. "A procedure for placement of standard-cell VLSI circuits". *IEEE Transactions on Computer Aided Design*, pages 92–98, 1985.

[80] D. Huang and A. Kahng. "Partitioning-based standard-cell global placement with an exact objective". In *ACM Symposium on Physical Design*, pages 18–25, 1997.

[81] T. Chan, J. Cong, T. Kong, and J. Shinnerl. "Multilevel optimization for large-scale circuit placement". In *Proceedings IEEE/ACM International Conference on Computer-Aided Design*, pages 171–176, 2000.

[82] K. M. Hall. "An r-dimensional quadratic placement algorithm". *Management Science*, 17:219–229, 1970.

[83] A. Srinivasan, K. Chaudhary, and E. S. Kuh. "RITUAL: Performance driven placement algorithm of small-cell ICs". In *Proceedings ACM/IEEE International. Conference on Computer Aided Design*, pages 48–51, 1991.

[84] J. Vygen. "Algorithms for large-scale flat placement". In *Proceedings ACM/IEEE Design Automation Conference*, pages 746–751, 1997.

[85] H. Eisenmann and F. M. Johannes. "Generic global placement and floorplanning". In *Proceedings ACM/IEEE Design Automation conference*, pages 269–274, June 1998.

[86] C. Sechen and A.L. Sangiovanni-Vincentelli. "Timberwolf 3.2: A new standard cell placement and global routing package". In *Proceedings ACM/IEEE Design Automation Conference*, pages 432–439, 1986.

[87] V. Betz and J. Rose. "VPR: A new packing, placement and routing tool for FPGA research". In *Proceedings International. Workshop on Field-Programmable Logic and Applicatons*, pages 213–222, 1997.

[88] B. Hu and M. Marek-Sadowksa. "Fine granularity clustering for large scale placement problems". In *Proceedings International Symposium on Physical Design*, pages 67–74, 2003.

[89] J. Cong, M. Romesis, and M. Xie. "Optimality, scalability and stability study of partitioning and placement algorithms". In *International Symposium on Physical Design*, pages 88–94, 2003.

[90] V. George and J. M. Rabaey. *"Low-Energy FPGAs"*. Kluwer Academic Publishers, 2001.

[91] C-C. Wang and C-P. Kwan. "Low Power Technology Mapping by Hiding High-transition Paths in Invisible Edges for LUT-based FPGAs". In *IEEE International Symposium On Circuits and Systems*, pages 1536–1539, 1997.

[92] F. Najm. "Transition Density: A New Measure of Activity in Digital Circuits". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 310–323, Feburary 1993.

[93] Altera Corp. *"Device Data Book"*, 1999.

[94] A. Marquardt, V. Betz, and J. Rose. "Timing-driven placement for FPGAs". In *Proceedings ACM Symposium on FPGAs*, pages 203–213, 2002.

[95] S. Nag and R. Rutenbar. "Performance-driven simultaneous place and route for row-based FPGAs". In *International Conference on Computer-Aided Design*, 1995.

[96] S. Alupoaei and S. Katkoori. "Net-based force-directed macrocell placement for wire-length optimization". *Journal of VLSI Signal Processing Systems*, pages 151–163, May 2004.

[97] C. Tseng and D. P. Siewiorek. "Facet: a procedure for the automated synthesis of digital systems". In *Proceedings ACM/IEEE Design Automation Conference*, pages 490–496, 1988.

[98] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. "VLSI module placement based on rectangle-packing by the sequence-pair". *IEEE Transactions on Computer Aided Design*, pages 1518–1524, December 1996.

[99] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli. "An Efficient General Cooling Schedule for Simulated Annealing". In *Proceedings International Conference Computer-Aided Design*, pages 381–384, 1986.

[100] N. R. Quinn and M. A. Breuer. "A forced directed component placement procedure for printed circuits boards". *IEEE Transactions on Circuits Systems*, pages 377–388, 1979.

[101] R. Kress. *"Numerical Analysis"*. Springer-Verlag New York Inc., 1998.

[102] F. Mo, A. Tabbara, and R. K. Brayton. "A force-directed macrocell placer". In *Proceedings International Conference on Computer Aided Design*, pages 177–180, 2000.

[103] H. W. Kuhn. *"The Hungarian method for the assignment problem"*. Naval Research Logistics Quarterly, 2(1):83-97, 1955.

[104] H. Li, S. Katkoori, and W.K. Mak. "Force-directed performance driven placement algorithm for FPGAs". In *Proceedings ISVLSI 2004*, pages 193–198, 2004.

[105] A. K. Murugavel and N. Ranganathan. "Gate Sizing and Buffer Insertion using Economic Models for Power Optimization". In *International Conference on VLSI Design*, pages 195–200, 2004.

[106] S. P. Mohanty, N. Ranganathan, and S. K. Chappidi. "Power Fluctuation Minimization During Behavioral Synthesis using ILP-Based Datapath Scheduling". In *International Conference on Computer Design*, pages 441–443, 2003.

[107] C. Gopalakrishnan and S. Katkoori. "Tabu Search Based Behavioral Synthesis of Low Leakage Datapaths". In *IEEE Symposium on VLSI*, pages 260–261, 2004.

[108] C. Gopalakrishnan and S. Katkoori. "KnapBind: An Area-Efficient Binding Algorithm for Low-leakage Datapaths". In *International Conference on Computer Design*, pages 430–435, 2003.

[109] M. Xu and F. J. Kurdahi. "Layout-driven RTL binding techniques for high-level synthesis using accurate estimators". *ACM Transactions on Design Automation of Electronic Systems*, 2(4):312–343, 1997.

[110] M. Xu and F. J. Kurdahi. "Layout-Driven High Level Synthesis for FPGA Based Architectures". In *Proceedings Design Automation and Test in Europe*, pages 446–450, 1998.

[111] D. Kim, J. Jung, S. Lee, J. Jeon, and K. Choi. "Behavior-to-placed RTL synthesis with performance-driven placement". In *Proceedings IEEE/ACM International Conference on Computer-Aided Design*, pages 320–325, 2001.

[112] University of South Florida. "AUDI - AUtomatic Design Instantiation". `http://vcapp.csee.usf.edu/~katkoori/kkweb/audi.html`.

[113] R. J. Cloutier and D. E. Thomas. "The combination of scheduling, allocation, and mapping in a single algorithm". In *Proceedings 27th ACM/IEEE Design Automation Conference*, pages 71–76, 1990.

[114] S. Katkoori. *"Behavioral profiling based high level power estimation methodologies for VLSI ASIC and FPGA synthesis"*. PhD thesis, University of Cincinnati, 1996.

[115] University of California Berkeley. *"Lager IV Release 4.0"*, 1991.

[116] B. S. Landman and R. L. Russo. "On pin versus block relationship for partitions of logic circuits". *IEEE Transactions on Computers*, 20:1469–1479, 1971.

[117] H. B. Bakoglu. *"Circuits, Interconnections, and Packaging for VLSI"*. Addison-Wesley, 1990.

[118] H. Van Marck, D. Stroobandt, and J. Van Campenhout. "toward an extension of rent's rule for describing local variations in interconnection complexity". In *4th International Conference for Young Computer Scients*, pages 136–141, 1995.

[119] W. Li and D. K. Banerji. "Routability prediction for hierarchical FPGAs". In *Great Lakes Symposium on VLSI*, pages 256–259, 1999.

[120] S. Brown, J. Rose, and Z. G. Vranesic. "A stochastic model to predict the routability of field programmable gate-arrays". *IEEE Transactions on Compuer-Aided Design of Integrated Circuits and Systems*, pages 1827–1838, December 1993.

[121] K. M. Buyuksahin and F. N. Najm. "High-level power estimation with interconnect effects". In *IEEE International Symposium on Low Power Electronics and Design*, pages 197–202, 2000.

ABOUT THE AUTHOR

Hao Li was born in Beijing, China in 1972. He received the Bachelor of Engineering degree in Telecommunications Engineering in 1995 from Beijing University of Posts and Telecommunications, Beijing, China. He received the Master of Science degree in Electrical Engineering from Beijing University of Posts and Telecommunications, Beijing, China in 1999. He then joined the Ph.D. program in the Department of Computer Science and Engineering at the University of South Florida, USA. His research interests include VLSI design, physical design for FPGAs, electronic design automation, High-Level synthesis for FPGAs. He has published a number of research papers in areas of VLSI Design and CAD. His paper was nominated for the best paper award at the Asian and South Pacific Design Automation Conference in 2004. He is a member of IEEE and ACM-SIGDA.