UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of a doctoral thesis by

Rupesh Subhashchandra Shelar

and have found that it is complete and satisfactory in all aspects,
and that any and all revisions required by final
examining commitee have been made.

Professor Sachin S. Sapatnekar

Name of the Faculty Advisor

Signature of the Faculty Advisor

Date

GRADUATE SCHOOL

SYNTHESIS FOR NANOMETER TECHNOLOGIES

A THESIS

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

RUPESH SUBHASHCHANDRA SHELAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY IN ELECTRICAL ENGINEERING

Sachin S. Sapatnekar, Advisor

MAY 2004

ACKNOWLEDGEMENTS

I am beholden to my advisor, Prof. Sachin Sapatnekar, for his support, encouragement, and advice for this thesis. I owe him, apart from Prof. Madhav Desai and Prof. H. Narayanan at Indian Institute of Technology, Mumbai, who nurtured my passion for research, my development into a researcher. He patiently helped during initial faltering steps and pointed out even the most minute mistakes that I was making. His insisting on being meticulous and complete in whatever I do has helped me and will continue to help in the future.

I am grateful to Prof. Kiarash Bazargan, Prof. Gerald Sobelman, and Prof. Victor Reiner for serving on my Ph. D. committee. Attending their classes was a sheer pleasure and having them on the committee a privilege. I acknowledge the role of funding agencies for this research: Semiconductor Research Consortium (SRC) supported under contracts 99-TJ-692 and 2002-TJ-1092, while National Science Foundation (NSF) provided support under contract NSF CCR-0098117.

Part of the work on congestion-aware synthesis presented in this thesis was carried out at Intel Labs. (CAD Research) during the summers of 2002 and 2003, which provided me an unique opportunity to view the research and its applicability through an industrial perspective. Xinning Wang and Prashant Saxena, who were my mentors at Intel, were supportive during my learning process; if it were not for them, the thesis would have been different. Apart from them, other researchers at Intel Labs., namely Steve Burns, Pasquale Cocchini, Darshan Patra, Mike Kishinevsky, and Timothy Kam, who commented on the work or raised appropriate questions, during various stages have contributed indirectly. Members of Intel's physical design tools team helped in defining the design flow for experiments. Contemporary interns at Intel Labs., Kavel Buyuksahin,

Pankaj Chauhan, Chirayu Amin, Aseem Agarwaal, and Trevor Meyerowitz, made the labs. a “great place to work” - a reality about which I was skeptic, when representatives of Intel Human Resources used to mention about it during orientation.

I acknowledge valuable discussions with Prabhakar Kudva, Michel Berkelaar, and Prof. Andre Reis, whose suggestions helped me for the work on specific problems in the thesis. Numerous discussions with many other researchers, whom I met during various conferences, contributed indirectly towards the development of the thesis.

Former and current members of VLSI Electronic Design Automation (VEDA) and Reconfigurable Computing lab, who made my stay pleasant and maintained cheerful surrounding for the work include the following: Anup Sultania, Arvind Karandikar, Bing Lu, Cheng Wan, Cristinel Ababei, Haitian Hu, Haifeng Qian, Haihua Su, Hongliang Chang, Jaskirat Singh, Jiang Hu, Mahesh Ketkar, Tianpei Zhang, Venkat Rajappan, and Vidyasagar Nookala. I will miss their company in the future. A lot of friends supported, morally or otherwise, at various junctures during the last four years. Although names of all of them cannot be listed here, their support was invaluable. Some of these include Arvind, Chirayu, Cristinel, Mahesh, Ramona, Venkat, and Vidyasagar. Thanks are due to Chimai and Kevin for providing an excellent administrative support for the computing environment in VEDA Labs.

Finally, I acknowledge my mother, brother Tushar, and sister Sandhya: although we were physically thousand miles apart, I felt, we never were; they shared my all ups and downs over phone and email and stood by.

Dedicated to the memories of my father,

efforts of my mother, and love of my siblings

ABSTRACT

The challenges to be faced by Very Large Scale Integrated (VLSI) circuits in nanometer technologies include increasing power dissipation and interconnect dominance. The pass transistor logic (PTL) family is an excellent choice for low power designs, but its use has been limited due to the lack of design automation tools. The first part of the thesis addresses these design automation needs. The second part of the thesis deals with the aspects of interconnect dominance that manifest themselves in the form of routing congestion.

For performance-driven synthesis of PTL circuits, we propose a polynomial time algorithm based on the recursive bipartitioning of binary decision diagrams (BDD's). The algorithm can ensure logarithmic depth PTL implementations, while none of the previous synthesis heuristics guarantee such a lower bound on the depth of PTL circuits. Experimental results on ISCAS'85 benchmarks obtained employing the algorithm show that PTL can result in a significant improvement, up to 30% in area as well as delay, over static CMOS for *xor*-intensive circuits. For layout generation of PTL circuits, we propose a method that translates BDD's into a transistor-level placement, which minimizes the area by diffusion-sharing, linear tree arrangement of transistor clusters, and greedy row assignment of the BDD nodes. Apart from their utility for pure PTL implementations, layouts generated by our method can be used as macro cells in the context of static CMOS/PTL synthesis, since these layouts can fit into a standard cell methodology.

The remainder of the thesis considers the problem of routing congestion, which re-

sults in unroutable designs or detours of wires leading to timing violations. To address the routing congestion problem at the logic synthesis level, we propose congestion-aware technology mapping methods. These methods are guided by a predictive probabilistic congestion map, unlike previous approaches that rely on indirect metrics such as wirelength. The matching phase in technology mapping uses these predictive congestion maps to store congestion-aware choices. It employs a cost function that allows the selection of congestion-optimal matches in densely congested regions while permitting the choice of area- or delay-optimal matches in sparsely congested regions. Similar approaches based on this matching phase have been applied for area- and delay-oriented technology mapping. For area-oriented mapping, experimental results using an industrial circuit and ISCAS'85 benchmark circuits, proprietary placers and routers, and a cell library in high-performance microprocessor design in 90nm technology show, on an average, 37% improvement in routability, measured in terms of track overflows, at the cost of marginal increase in gate area. The results for delay-oriented mapping show 46% improvement in routability for approximately unchanged delays.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

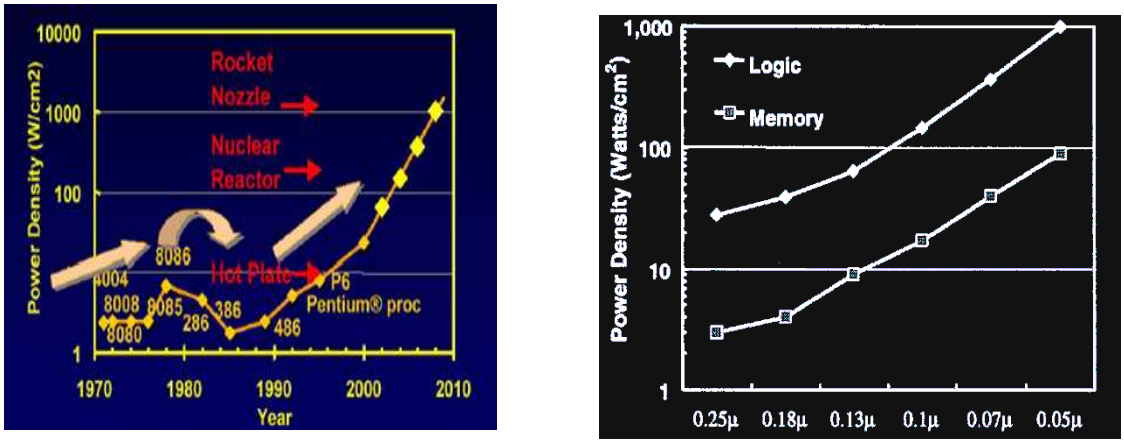
LIST OF ALGORITHMS

Chapter 1

Introduction

The design complexity of VLSI circuits is governed by device and interconnect architectures, the availability of device and interconnect resources, and consumer demands for increased functionality [ITR01a]. Over the last few decades, device sizes have been scaling according to Moore's law[1], which states that the number of transistors on a chip will double every twenty four months [Moo65]. The somewhat periodic process scaling and the consumer demands for new functionalities have led, eventually, to the arrival of the system-on-chip era, where designers are motivated to pack more functionality on the chip. In combination with the process scaling, this has created several new design challenges, noteworthy among which are those related to managing increasing power density and interconnect dominance.

[1]The law predicts that the number of transistors on a chip will double every year till 1975, while trends beyond 1975 are not shown in the article [Moo65]. However, Borkar's paper [Bor00], which considers trends from 1970's till 2000, shows that the number of transistors are doubling, actually increasing 1.96 times, to be accurate, every two years.



(a) (b)

Figure 1.1: Power density trends for microprocessors from [Bor00]: (a) Full-chip power density. (b) Power densities of logic and memory.

1.1 Power and Interconnect Challenges

Due to technology scaling, the number of on-chip transistors have been increasing exponentially, while die-sizes have been growing at much slower pace: the number of transistors double after every 2 years, while die-sizes double every 10 years [Bor00]. The increasing number of on-chip transistors results in a corresponding increase in power dissipation, and since die-sizes are expanding slowly, power densities have been rising rapidly. Figure 1.1(a) shows these trends for microprocessor designs, while Figure 1.1(b) shows the comparison of power densities for logic and memory for these microprocessors. It can be observed that the power densities for logic are almost one order higher than those in memory. This is partly because only small part of a memory block is active at any time, as opposed to a logic block, wherein many components may be active during every clock cycle. As technology scaling continues, leakage power will constitute a significant portion, up to 45%, of total power dissipation [Bor00]. Even in

such a scenario, these trends will continue to hold, as special circuit techniques to reduce the leakage power are more amenable to memories than to logic. Therefore, to reduce the power dissipation in the logic, the use of new circuit families must be explored and design automation support should be provided for the circuit styles that address power and performance challenges [ITR01a]. Traditionally, logic has been implemented using static CMOS standard cells that offer good performance, and have good tool and design methodology support. The pass transistor logic (PTL) family is a promising alternative, since it employs NMOS transistors that have small capacitance, which may reduce the power dissipation while offering similar performance as static CMOS. Few design automation solutions are available for PTL, and this has resulted in its limited usage. To address design automation needs for PTL, we have proposed synthesis and layout generation algorithms in this thesis; we will present an overview of these algorithms in the following section.

Another daunting issue that designs in sub-100nm technologies will face is that of interconnect dominance. High-performance designs are becoming wire-limited, i.e., the area of a design is not only determined by the area of the cells, but also the area required to route the wires. This, again, is a consequence of the increasing design complexity: the number of wires grows exponentially with the number of gates, according to Rent's rule [CS00]. Although the exponent is small, usually between 0.2 and 0.8 for real circuits, the base, i.e., the number of gates, has been increasing rapidly as the number of transistors increases. Even though a larger number of metal layers is available with the advances in technology, most of the upper metal layers are used for routing global signals such as clocks, and logic blocks are left with only a marginal increase in routing resources. This often results in the unavailability of a sufficient number of

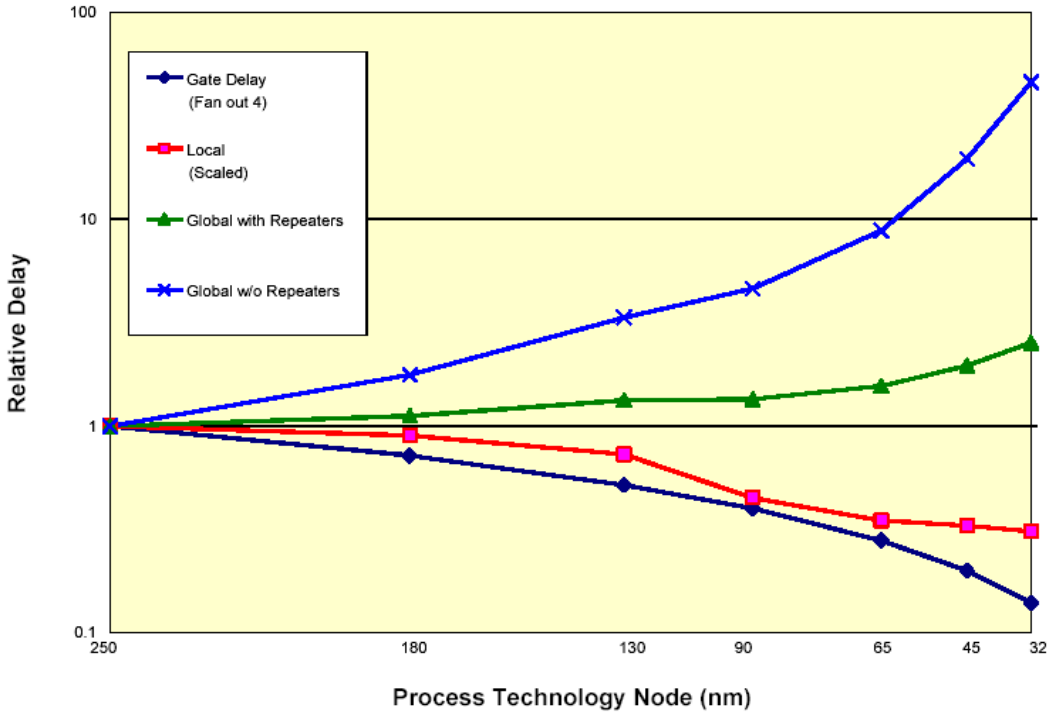


Figure 1.2: Interconnect delay trends from International Roadmap for Semiconductors, 2001 [ITR01b].

tracks to route the wires, a problem known as routing congestion. Apart from this problem, another issue is that interconnect delays have also started dominating gate delays, according to the trends from the 2001 International Technology Roadmap for Semiconductors [ITR01b], as shown in Figure 1.2. Together, these make the timing closure even more difficult: if the wires are detoured to avoid congested regions, they may violate the timing constraints. This thesis proposes congestion-aware mapping methods that consider the routing congestion early on in the design process, thereby facilitating rapid design closure.

1.2 Proposed Solutions: An Overview of the Thesis

In this thesis, synthesis solutions are provided to the problems of increasing power dissipation and interconnect dominance. The solutions to the former involve performance-driven synthesis algorithm for the low power circuit family pass transistor logic (PTL). This family offers an attractive alternative to static CMOS due to its potential for implementing circuits with a small number of transistor count and hence, small capacitance and power dissipation. Our simulations using predictive technology models [ptm] show that PTL will result in better implementations as compared to static CMOS in case of *xor*-dominated circuits[2]. Moreover, the use of PTL in ASIC libraries with feature sizes smaller than 130nm has also been on rise because of the performance gain that they offer over static CMOS [BHSA03]. However, very few design automation tools for logic synthesis and physical design for PTL are available for the following reasons:

1. Microprocessor designers, who are possibly the largest users of this family, employ a hand-crafted approach while designing PTL circuits.

2. ASIC designers rely on static CMOS standard cell libraries for which good tool support is available.

The unavailability of the design automation tools has resulted in the limited usage of PTL. Therefore, to fully exploit the potential of PTL, this work has developed synthesis algorithms targeting performance and power dissipation [SS01a, SS01b, SS02a]. These algorithms use a libraryless approach (using "fluid/liquid cells"), since library-based methods result in sub-optimal solutions in case of PTL. To translate these synthesized

[2] In such a scenario, PTL/Static CMOS mixed synthesis may be a viable approach and for this problem, design automation solutions will have to be developed.

circuits into completed designs, layout generation algorithms for libraryless PTL circuits are required. To address this need, we have developed a transistor-level placement algorithm [SS02b] for such circuits to optimize the area.

Logic families such as PTL are, realistically speaking, unlikely to completely displace static CMOS. As a result, the bulk of any design will consist of static CMOS gates, and the synthesis of high-performance circuits must address problems related to this circuit style. Prominent among these is the push towards unification of logical and physical design. While past research has addressed this problem partially, an important unsolved problem relates to handling routing congestion, which depends on the following factors:

- the connectivity of the network,
- the placement of the cells, and
- the routes taken by the wires.

Although the placement and routing stages offer great flexibilities to alleviate congestion, considering routing congestion only during these stages often results in design iterations as some parts of circuits are unroutable, or because the routability requires long detours of wires, leading to timing violations. Therefore, it is necessary to consider the routing congestion during synthesis stage, as it allows more freedom to address the problem. Technology mapping is a powerful transformation in the synthesis domain that allows absorption of long wires into complex logic gates or splitting of complex gates into smaller gates to lower the congestion. We have developed a congestion-aware technology mapper [SSSW04] that minimizes the area of a circuit while improving its routability; the algorithm is further extended for delay minimization in the presence of routing congestion.

1.3 Organization of the Thesis

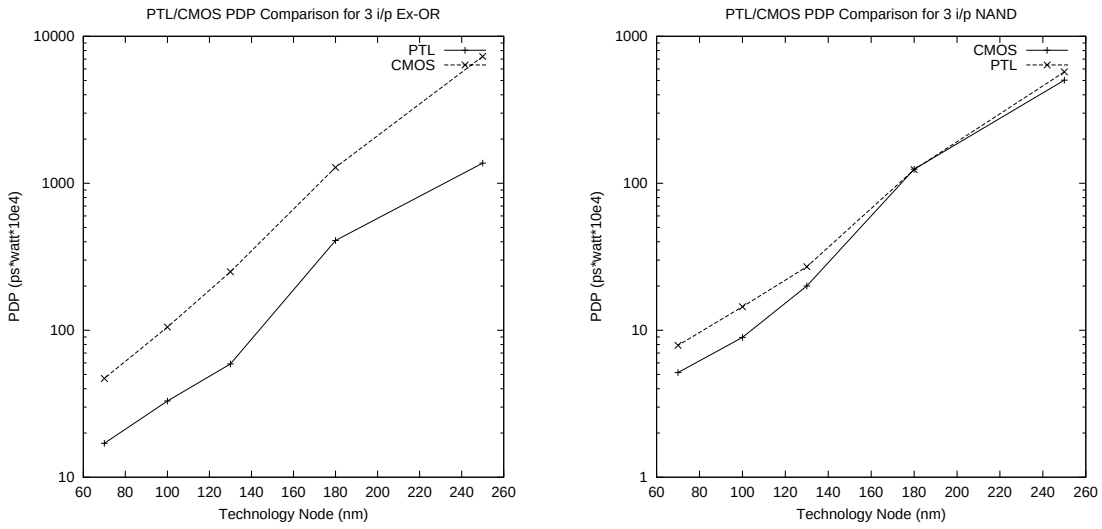
The rest of the thesis is organized as follows. Chapters 2 and 3 deal with pass transistor logic, while chapter 4 focuses on congestion-aware technology mapping. Chapter 2 presents a max-flow min-cut based exact polynomial time BDD decomposition algorithms for delay-oriented and power dissipation driven PTL synthesis. Chapter 3 provides the details of transistor-level placement algorithm, which produces layouts for PTL that fit in standard cell library methodology. Chapter 4 shows empirical evidence of congestion correlation and furnishes the particulars of congestion-aware technology mapping algorithms. We conclude the thesis in Chapter 5.

Chapter 2

Pass Transistor Logic Synthesis

2.1 Introduction

Static CMOS has been a favorite logic style of VLSI designers for the last two decades due to its advantageous noise immunity properties and good performance. However, due to technology scaling and the increasing number of transistors on chip, the performance of static CMOS circuits has been achieved at the expense of substantial area/power dissipation costs that may not be desirable, especially for portable appliances. Therefore, new logic families that address the power and performance challenges must be explored [ITR01a], and in this context, the logic styles such as domino and pass transistor logic (PTL) are attractive alternatives to static CMOS. However, with the increasing on-chip power densities becoming a concern, the use of domino logic circuits, known for good performance but high power dissipation, is limited only to critical parts of the design. PTL, on the other hand, has a potential for low power and good performance, but its potential remains unexplored due to lack of PTL-oriented CAD tools and method-



(a) (b)

Figure 2.1: Power-delay product (PDP) values for (a) a three-input XOR gate and (b) a three-input NAND gate. Both circuits are implemented in NMOS-only PTL and in static CMOS at various technology nodes, and the results of SPICE simulations using predictive technology models [ptm] are shown.

ologies. The primary benefits of PTL include the potential for a lower transistor count, lower capacitance, smaller delays and reduced power consumption.

PTL (or its variants) are known for better implementations as compared to static CMOS in case of arithmetic circuits, such as adders and multipliers that are *xor*-dominated [YYN+90, WE94, YSR96, Rab00]. This trend is likely to continue over several technology generations beyond 100nm as shown in Figure 2.1(a). The figure shows power-delay product trends, obtained by performing the simulations using predictive technology models [ptm], for 3-input XOR gate implemented in NMOS only PTL and static CMOS. It shows that the power-delay product for NMOS-only PTL will be, consistently and significantly, better than the corresponding product for the

static CMOS implementation over several technology nodes beyond 100nm. Most of these *xor*-dominated arithmetic circuits are designed manually for high-speed microprocessor designs. For ASIC designs, however, when these circuits are synthesized along with random logic using standard cell libraries, the structural properties of the network that are suitable for PTL may remain unexploited, resulting in possibly sub-optimal solutions. PTL elements are used for these designs even today: ASIC libraries typically contain PTL-like one-hot multiplexers and pass transistor gates because of the area/power/performance gains that they offer over static CMOS circuits, even though the latter have higher noise immunities [BHSA03]. In practice, most of these cells are used in an ad-hoc manner after verifying that the nets driving PTL cells are appropriately buffered. Thus, although the use of PTL is desirable, it remains underutilized, and more so because of the lack of good performance-driven synthesis algorithms and methodologies exploiting the properties of PTL circuits.

It is well known that PTL is not universally better than CMOS for all types of logic structures: for *nand*-intensive circuits, for example, static CMOS can result in better implementations than PTL. This is demonstrated in Figure 2.1(b) by our simulations for a three-input NAND gate in both logic styles at various technology nodes. Therefore, mixed static CMOS/PTL synthesis is likely to be an attractive alternative in the future. Even for such an approach, synthesis solutions targeting performance that exploit the properties of PTL circuits must be developed, and the work presented in this thesis may be considered as a step in that direction. To address these synthesis needs for PTL, this chapter presents delay-oriented and power dissipation driven synthesis algorithms. These algorithms are not tied to any particular PTL library[1], since we have observed that

[1]Ideally, the library should contain all possible functions, with different drive strengths, for a given number of inputs. For a large library size, characterization and maintenance becomes difficult. Limiting

conventional technology mapping based on 2-input NAND decomposition followed by covering using cells in a library results in sub-optimal PTL implementations, as PTL is suitable for *xor*-based (or Shannon co-factor based) circuits.

The rest of the chapter is organized as follows. Sections 2.2 - 2.6 deal with delay optimization in PTL circuits employing BDD decomposition, while sections 2.7 - 2.10 focus on a similar framework to reduce power dissipation. Section 2.2 reviews previous work in delay-oriented PTL synthesis and outlines our contributions. Section 2.3 explains PTL implementation using BDD decomposition, for which a recursive bipartitioning approach is proposed in Section 2.4. Section 2.5 describes delay modeling and analysis for PTL circuits, while Section 2.6 presents experimental results. Section 2.7 introduces power dissipation driven synthesis problem, and Section 2.8 describes a power model for PTL. A decomposition model for low power implementation is proposed in Section 2.9 followed by experimental results and conclusions in Section 2.10. Section 2.11 summarizes the chapter.

2.2 Delay-oriented Synthesis

2.2.1 Previous Work

Synthesis techniques for PTL circuits have been closely related to the binary decision diagram (BDD) representation of logic functions, for several reasons: firstly, BDD-based PTL circuits are guaranteed not to have any sneak paths [YSR96, BNNSV97], and secondly, the use of BDD-based methods can benefit from the plethora of efficient

the library size, however, affects the quality of design, and this has been observed for standard cell designs by numerous researchers, for instance [BF98, JSB98, CK00]. To overcome the limitation of the libraries, libraryless or "liquid cell" [CK00] synthesis flows are proposed, for instance, in [RRAR97, JSB98].

algorithms available for the construction of BDD's. The BDD representation of a logic function affects the PTL implementation, and BDD decomposition methods must be adapted to optimize cost functions that represent their PTL implementations.

The idea of decomposing logic functions, in general, and BDD decomposition, in particular, for optimizing specific objectives is not new, although there is little work on considering PTL-based cost functions during BDD decomposition. We review some of the representative work in the area of Boolean decomposition and BDD decomposition. In the area of decomposition of switching functions, Ashenhurst performed pioneering work with a theorem relating column multiplicities in a partition matrix, corresponding to a partition of variables into *bound set* and *free set* of variables, with the simple disjunctive decomposability of a switching function, and also proved relevant theorems on non-simple decompositions of the switching function [Ash57]. An excellent review on the development of theory of decomposition of switching functions in 1960's and 1970's is presented in [DDT78]. Recently, Pedram *et al.* have proposed an ordered BDD (OBDD) based function decomposition method that involves forming a cutset in the BDD, and then encoding the nodes in the cutset to yield disjunctive or non-disjunctive decompositions [LPP96]. This OBDD-based decomposition has been applied to the synthesis of field programmable gate arrays targeting area, measured in terms of the number of configurable logic blocks, with no depth constraints. In [YC99], a BDD-based logic synthesis system is developed in which transformations such as AND/OR decomposition based on 0/1 dominators, and XOR and functional MUX-based decompositions are proposed; synthesis for performance-driven PTL is not specifically targeted.

Several techniques for PTL synthesis have been suggested in the recent past. A loose upper bound of theoretical utility on the number of multiplexers required to implement

a given logic function is developed in [Sas00]. Buch *et al.* propose a greedy heuristic in [BNNSV97] to decompose larger BDD's into smaller BDD's whose sizes are kept under a specified threshold. For area-driven PTL synthesis, Chaudhry *et al.* [CLAB98] present a method similar to traditional multilevel logic optimizations, first invoking the iterative application of logic transformations, and then mapping the BDD representation on to a PTL cell library. A similar philosophy has been used for performance-driven synthesis in [LAB99]. Both [BNNSV97] and [LAB99] imply that multilevel BDD's are to be employed, but the limitation of these approaches is that they are unable to predict the performance gain beforehand; such a prediction is very helpful in directing the decomposition. Ferrandi *et al.* propose the use of PTL cell generation and subsequent binate covering of the nodes in the Boolean network utilizing a set of heuristically generated BDD's to minimize the cost [FMM$^+$98]. Becker *et al.* [SB00] report the application of multiplexer circuits for area and delay optimizations of PTL circuits. Unlike [BNNSV97], they allow the threshold size of the decomposed BDD's to be varied, and their cost function allows area and depth to be traded off.

2.2.2 Our Contributions

In this chapter, we present a novel approach to performing delay-oriented PTL synthesis through the decomposition of a monolithic BDD representing a circuit. Our contributions can be summarized as follows:

- We explicitly incorporate delay and area considerations simultaneously into a global technique for finding the decomposition.
- Our bipartitioning scheme applies the max-flow min-cut technique to roughly

halve the delay of a PTL implementation of a BDD with the least area overhead. The delay in a PTL circuit is well known to be linear in the number of input variables after buffer insertion, and our recursive bipartitioning approach can result in logarithmic depth reductions over the PTL implementation of the monolithic BDD. Although logarithmic depth reductions, in terms of transistors, may not translate to logarithmic delay reductions, the resulting delay reductions are still substantial. The area penalty is minimal, up to the accuracy in estimation, as the algorithm explicitly attempts to minimize this overhead by finding an optimal cut.

- Experimental results, obtained using the above techniques, on a set of ISCAS'85 benchmarks containing *xor*-dominated arithmetic circuits, such as multiplier and the circuits for error correcting codes, show that PTL outperforms static CMOS implementations with 31% improvement in delay and 30% improvement in area, on an average, for a 0.13μm technology. We found that in case of arithmetic logic unit (ALU) and control circuits, the improvements over static CMOS are small and inconsistent, although PTL (or its variant CPL) is known to yield cost effective implementations of adders, which are important components of ALU and control circuits. This anomaly may be attributed to the scripts in SIS [Sen92] that are used for preprocessing and also to the structure of control logic, which is usually *nand*-intensive, in these circuits. Employing our PTL synthesis algorithm in case of the designs that are inherently well suited for PTL, one may obtain performance that is close to custom designs while the use of static CMOS standard cell libraries to obtain the same performance may come at a very high area/power cost.

2.3 PTL Implementation using Decomposed BDD's

2.3.1 The Relationship between BDD's and PTL

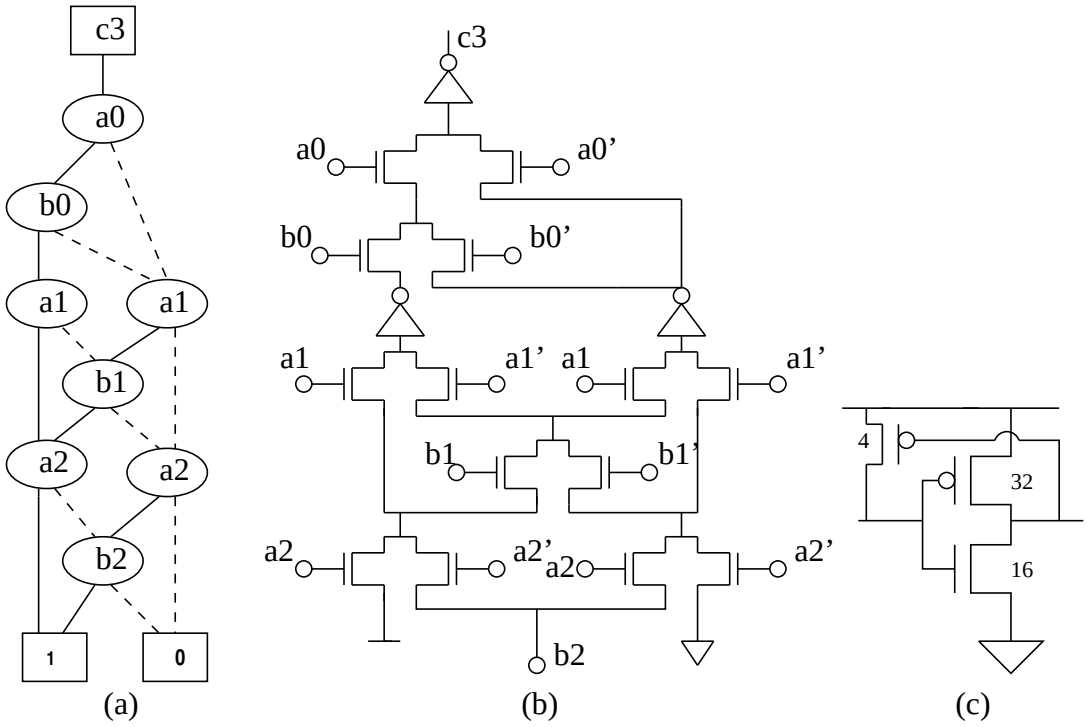


Figure 2.2: (a) The BDD for Carry function for 3-bit adder. (b) Its corresponding PTL implementation, using inverters with weak pull-ups, shown at the transistor level in (c).

A BDD can be mapped on to a PTL implementation as follows. Each node of the BDD implements a Shannon expansion about the variable x associated with the node, and can be expressed as $F = x \cdot F_x + x' \cdot F_{x'}$, where F_x and $F_{x'}$ are, respectively, the Shannon cofactors of the function F. This may be translated to a multiplexer that passes F_x when x is high, and $F_{x'}$ when x is low; the procedure can then be applied recursively to the functions F_x and $F_{x'}$. Therefore, for any logic function, the BDD

representation can be used to directly arrive at its PTL implementation, as shown in Figure 2.2. Moreover, mapping from a BDD on to a PTL circuit ensures a sneak-path-free implementation: this follows from the property of BDD's that for any assignment of inputs, only one path from the root node to a terminal node is active. For the purposes of this chapter, all BDD's are reduced ordered BDD's (ROBDD's), which implies that the order of variables on any path from an output node to a leaf node is identical. We also restrict ourselves to NMOS-only PTL, although the algorithms proposed in this thesis are applicable to other variants of PTL, such as transmission gate PTL, albeit with different area/delay trade-offs. For the NMOS-only PTL designs considered here, buffers with weak pull-ups are inserted after every k transistors in series to avoid long chains of pass transistors, and also to recover the voltage drops across pass transistors.

2.3.2 BDD Decomposition for Delay Optimization

Mapping a BDD directly to PTL can result in delays that are linear in the number of input variables, and BDD decomposition can be used to reduce these delays. We outline a general BDD decomposition technique with the help of the following example. Consider a carry function for a three-bit adder whose optimized BDD is shown in Figure 2.3(a). This BDD is built on six variables, a0, b0, a1, b1, a2 and b2, and one output, c3. We choose a cutset across the BDD that is indicated by the shaded nodes in Figure 2.3(a).

When this cut is used to separate the upper and lower parts of the BDD, dangling edges are created in the upper part, for instance, edges from nodes labeled a1 to nodes labeled a2. We introduce dummy nodes V_0, V_1, and V_2 that replace these shaded nodes, as shown in Figure 2.3(b). These dummy nodes can be assigned unique codes employing one-hot or minimum-bit encoding, as shown in Table 1. The two types of encoding lead

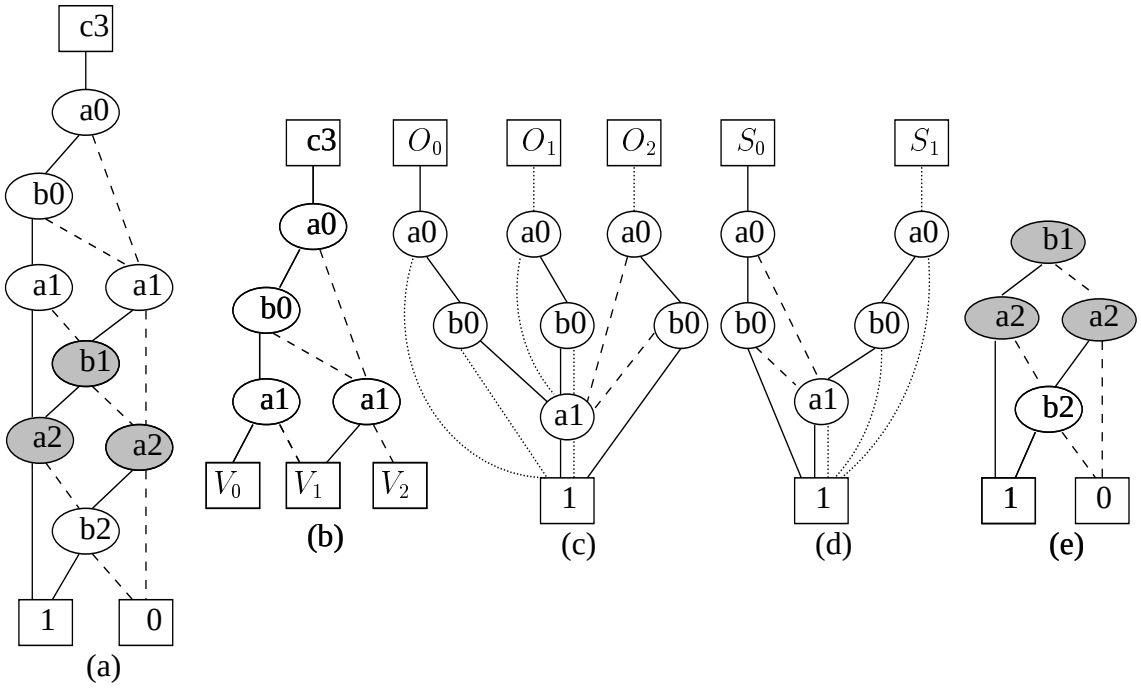


Figure 2.3: (a) The BDD for the carry function for a three-bit adder, where the shaded nodes form the cut that is employed to decompose the BDD. (b) The upper part of the cut with the dangling edges replaced by dummy nodes V_0, V_1, and V_2. (c) The select function under a one-hot encoding for $V_0V_1V_2$. (d) The select function under minimum-bit encoding for $V_0V_1V_2$. (e) The data function. In all of these pictures, the solid edges in the BDD denote the 1-cofactor, F_x, the dashed edges denote the 0-cofactor, $F_{x'}$, and the dotted edges denote the complemented 0-cofactor, $\overline{F_{x'}}$.

to two alternative PTL implementations with different area/delay trade-offs.

Once an encoding has been chosen, the original function can be realized using a decomposition based on this cut. The encoding bits (i.e., $O_0O_1O_2$ or S_0S_1) can be employed as *select* inputs to a multiplexer whose *data* lines correspond to the evaluated values of the BDD's rooted at the three shaded nodes as shown in Figure 2.3(e), depending on the value of the encoding. Therefore, each such *select* input corresponds to a BDD representation that sets the leaf nodes according to the chosen encoding. As an example, the select bit O_1 for the one-hot encoding corresponds to the combination V_0 = 0, V_1 = 1, V_2 = 0, and is used to select the BDD rooted at the shaded node b1. By substituting these values into the dummy terminals in Figure 2.3(b), we can obtain the BDD for the *select* input O_0. The BDD's for other *select* inputs such as O_1 and O_2 can be obtained similarly. The multioutput BDD for $O_0O_1O_2$ is illustrated in Figure 2.3(c).

One-hot Encoding		Minimum-bit Encoding	
Terminal Node	$O_0O_1O_2$	Terminal Node	S_0S_1
V_0	100	V_0	00
V_1	010	V_1	01
V_2	001	V_2	11

Table 2.1: One-hot and minimum-bit encoding schemes for the dummy terminal nodes introduced during decomposition.

If, instead, a minimum bit encoding is employed, a similar procedure may be used to derive the BDD for the *select* inputs S_0 and S_1; the corresponding multioutput BDD is depicted in Figure 2.3(d). We observe that depth of the BDD's for the *select* inputs is the same for one-hot encoding and for minimum-bit encoding. Note that in case of

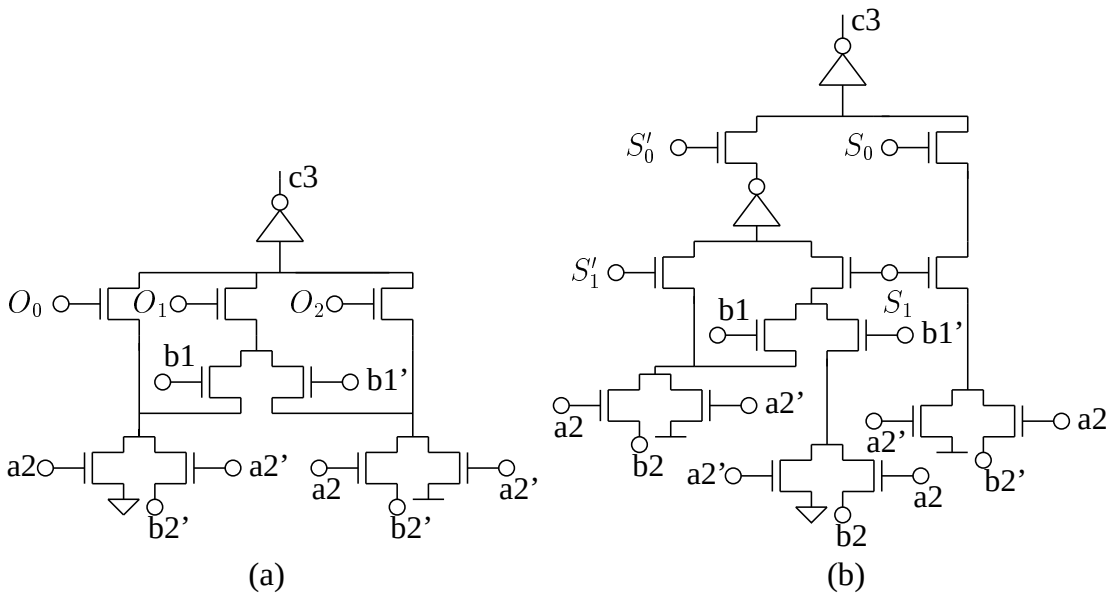


Figure 2.4: Alternative implementations of c3 (a) using a one-hot multiplexer and (b) using a regular multiplexer.

select functions obtained by one-hot encoding, for any assignment of a0, b0, and a1, only one of the *select* functions is true, and we can utilize a one-hot multiplexer circuit to implement c3. On the other hand, *select* functions that correspond to minimum-bit encoding are implemented using regular multiplexers. Two alternative implementations of c3 using a one-hot multiplexer and a regular multiplexer are shown in Figure 2.4(a) and in Figure 2.4(b), respectively. The *select* inputs are simply the PTL implementations of the BDD's shown in Figures 2.3(c) and 2.3(d).

Table 2.2 shows the active area and delay, obtained by circuit simulations under an excitation with a 50ps[2] transition time, for alternative implementations obtained by (1) directly mapping the BDD, and using decomposition based on (2) one-hot multiplexers and (3) regular multiplexers in 0.13 μm technology [ptm]. All of the pass transistors

[2]The transition time of 50ps is chosen, as it corresponds to a typical microprocessor clock period of 500ps corresponding to a 2GHz frequency.

Implementation	Active Area(μm^2)	Delay(ps)
Monolithic BDD	2.974	201
One-hot Multiplexer	3.532	103
Regular Multiplexer	3.768	174

Table 2.2: A comparison of alternative implementations of c3.

have widths of 14 λ and the inverters are sized as follows: all PMOS transistors have widths of 32λ, all NMOS transistors have widths of 16λ, and all weak pull-ups have widths of 4λ, as shown in Figure 2.2, where λ is the minimum feature size. Clearly, the one-hot multiplexer based implementation has the least delay, albeit with a slightly larger area than that obtained by directly mapping the BDD. We also observe that in the decomposed implementation using one-hot multiplexers, the depth of the circuit is halved as compared to the implementation obtained by a direct mapping of the BDD. Moreover, this procedure can be applied recursively, halving the depth every time to result in a logarithmic depth PTL implementation.

2.3.3 Trade-offs between the Choice of a One-hot or a Regular Multiplexer

Figures 2.5(a) and 2.5(b) show transistor-level implementations for 4:1 one-hot and regular multiplexers, respectively. In case of a one-hot multiplexer, four *select* inputs are required, of which only one can be high at a time. In contrast, the regular multiplexer has two *select* inputs, which are used to select among four *data* inputs. We observe that the depth of a one-hot multiplexer circuit, as measured by the maximum number

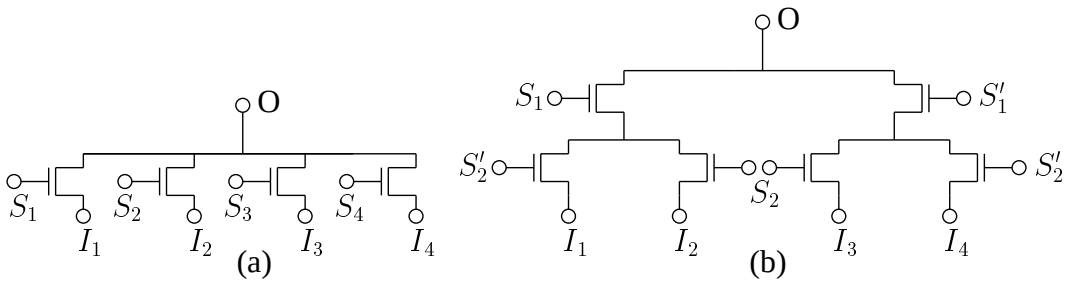


Figure 2.5: Transistor-level description of (a) a one-hot 4:1 multiplexer, (b) a regular 4:1 multiplexer.

of series transistors, is always one, irrespective of the number of *data* inputs. On the other hand, the depth of a regular multiplexer increases logarithmically with the number of *data* inputs. Apart from the delay advantage that can be obtained from this reduced depth, a one-hot multiplexer with n data inputs also employs fewer transistors than a regular multiplexer. Specifically, the number of transistors required to implement a one-hot multiplexer is n, while the corresponding number for a regular multiplexer is $2n-2$.

The complete picture, however, is more complex. The number of *select* inputs required for a one-hot multiplexer is the same as the number of *data* inputs, and therefore, such a multiplexer requires the generation of more *select* functions than a regular multiplexer. Moreover, although the number of levels for a one-hot multiplexer is always one, its delay is not constant but increases with the number of *data* inputs. This arises because an increase in the number of transistors connected to the output results in an increase in the load driven by the one-hot multiplexer, since additional drain capacitances, which contribute to the total output capacitance, are brought in by each data input. This is one of the reasons why the logarithmic depth reductions provided by our approach, which uses one-hot multiplexers, do not translate into logarithmic delay reductions. However,

this is not a significant limitation since the obtained delay reductions, as shown in Table 2.2, are nevertheless substantial for real circuit examples.

2.4 The BDD Decomposition Algorithm

The decomposition technique presented in the previous section can be thought of as a bipartitioning that halves the circuit depth and therefore, shortens the critical path and its delay. If we take a single cut across the BDD that halves the critical path, then we find that the delay in the PTL implementation using a one-hot multiplexer, which adds one extra series transistor, is approximately halved. We can apply this bipartitioning procedure recursively, such that on each application of the procedure, the critical path is roughly halved. The price being paid for this delay reduction is in terms of area, since the number of transistors required for implementation may increase as we recursively bipartition the BDD. BDD decomposition for delay reduction does not always result in an area penalty, since one-hot encoding of the select functions may result in simpler Boolean functions and hence, smaller BDD's. In such a case, bipartitioning should be performed so that it approximately halves the delay and also results in area-wise good implementation. In our algorithm, we perform this bipartitioning to aim for the minimum area penalty.

2.4.1 Recursive Bipartitioning for Performance

A key step during bipartitioning is that of identifying candidate nodes for the cut that will succeed in halving the circuit delay. Our delay estimator for the PTL implementation of a given BDD assumes the insertion of a buffer after at most three pass transistors in

series. Based on this assumption, each node in the BDD is assigned two delays:

Delay from Bottom (D_{bottom}): This is the delay of the PTL network rooted at a given BDD node.

Delay from Top (D_{top}): This is the maximum delay from a given BDD node to any of the outputs.

These delays can be evaluated employing the delay analysis procedure outlined in Section 2.5, which can be used to identify the critical path through the PTL network.

We define three types of nodes for delay-balanced bipartitioning:

Essential Nodes, for which D_{bottom} lies within a small range ($\pm\delta$) of half of the critical path delay ($D_{critical}$). In other words, essential nodes lie in the middle of critical path, with small tolerance δ.

Candidate Nodes, for which D_{top} and D_{Bottom} are both less than ($D_{critical}/2$–δ).

Non-candidate Nodes, which comprise all of the remaining nodes. These nodes are not considered for inclusion in the cut.

The optimum cut will halve the critical path, ensuring that no other path in the decomposed implementation has a delay of more than half the critical path delay. Therefore, all essential nodes must be in the cut, while we have the freedom to choose among the candidate nodes. We assign an area cost, explained in the next subsection, to the candidate nodes and then use the max-flow min-cut technique [CLR98] to find an optimum cut that halves the circuit delay with the smallest area cost.

Figure 2.6 shows an example of how the flow network is created. The procedure begins with a digraph corresponding to the given BDD, illustrated in Figure 2.6(a). In this

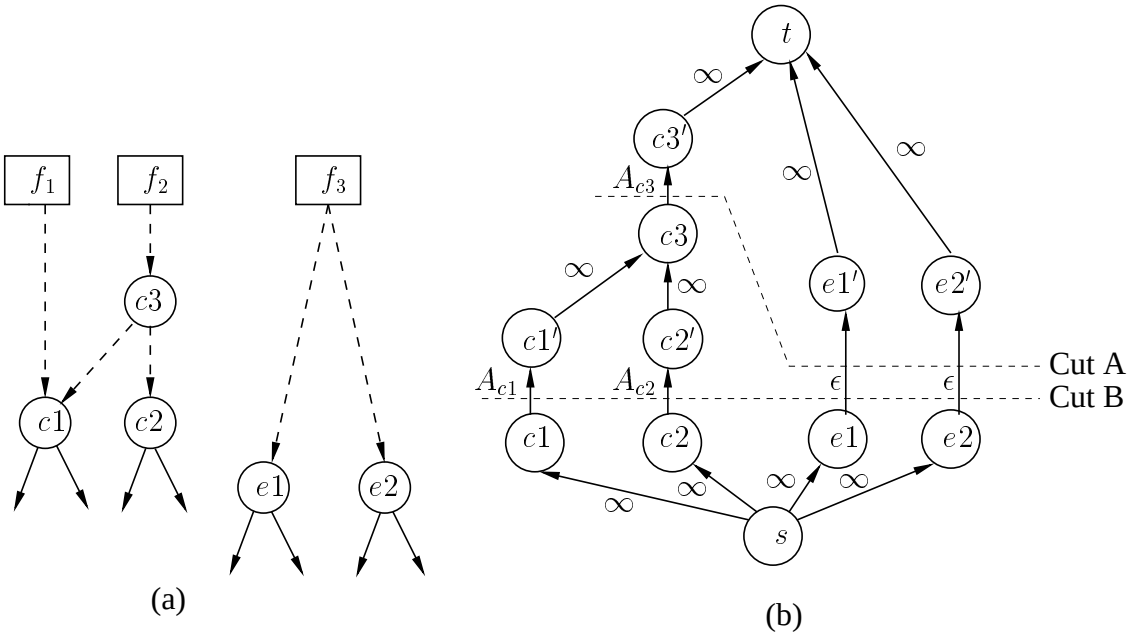


Figure 2.6: Creating a flow network: (a) A digraph corresponding to a BDD with essential and candidate nodes, and (b) the corresponding flow network.

example, let us assume that there are three nodes f_1, f_2, and f_3 corresponding to the three primary outputs, three candidate nodes $c1$, $c2$, and $c3$, and two essential nodes $e1$ and $e2$. The dashed edges in Figure 2.6(a) (for instance, an edge from f_1 to $c1$) indicate that there are directed paths between the corresponding nodes, but the nodes on these paths are not shown since none of them are essential nodes or candidate nodes. Figure 2.6(b) shows the corresponding flow network with one source node s and one destination node t. Each essential node in the digraph is split into two nodes: for instance, node $e1$ is represented by two nodes $e1$ and $e1'$ with an edge from $e1$ to $e1'$ that is assigned a small capacity[3], ϵ. Similarly, candidate nodes are also split into two nodes: for instance, node $c1$ in the digraph is represented by two nodes $c1$ and $c1'$, respectively. However, the edge capacity for these nodes is not ϵ, but is set to the area cost of the candidate nodes in the BDD. In this example, the edge from $c1$ to $c1'$ has an edge capacity of A_{c1}. The remaining edges in the flow network are assigned a capacity of ∞, and therefore will not appear in the cut. Thus, in this example two cuts are possible, Cut A and Cut B, corresponding to cutsets $A_{cutset} = \{e1, e2, c3\}$ and $B_{cutset} = \{e1, e2, c1, c2\}$ in the digraph corresponding to the given BDD. The application of the Ford-Fulkerson technique to find the minimum cut will result in one of these, depending on values of A_{c1}, A_{c2}, and A_{c3}. The pseudocode for the procedure is shown in Algorithm 2.4.1. Once the cut has been determined, the vertices in the cut are replaced by dummy terminal nodes, which can be assigned unique codes and implemented as PTL circuits, as illustrated in Section 2.3.

[3]The small capacity ϵ to the edges between split essential nodes ensures that all essential nodes are in the cut. A capacity of 0 cannot be associated with these edges because of the conventions in the max-flow min-cut algorithm; a capacity of 0 means that edge does not exist.

Algorithm 2.4.1 Find a delay-optimal cut that minimizes area penalty

Input: $G(V, E)$ = Graph underlying a given BDD

Output: S_{cut} = An optimal cut

1: DelayAnalysis(G)

2: $D_{critical} \leftarrow \max\{v.D_{Bottom} \forall v \in V\}$

3: $V_{Essential} \leftarrow \{v{:}v \in V$ and $D_{critical}/2 - \delta \leq D_{bottom} \leq D_{critical}/2 + \delta\}$

4: $V_{Candidate} \leftarrow \{v{:}v \in V$ and $D_{top}, D_{bottom} < D_{critical}/2 - \delta\ \}$

5: AreaCostEstimate($V_{Candidate}$)

6: $G_{Flow} \leftarrow$ CreateFlowNetwork(G,$V_{Essential}$,$V_{Candidate}$)

7: Ford-Fulkerson(G_{Flow}, G, S_{cut})

This bipartitioning procedure can be applied recursively till no further delay reduction can be achieved. If we define the depth of the implementation as the maximum total number of series transistors (discounting buffers) from any input to any output, then the resulting implementation has a depth that is logarithmic in number of inputs. In contrast, the original undecomposed BDD yields an implementation whose depth is linear in the number of variables. This is stated in the following theorem:

Theorem 2.4.1 *The recursive application of the method shown in Algorithm 2.4.1 to any BDD with the use of one-hot multiplexers results in an implementation that has a depth, in terms of the number of series transistors, of* $O(\log(Depth_M))$*, where* $Depth_M$ *is the depth of the PTL implementation obtained by directly mapping the BDD.*

Proof Since the PTL implementation obtained by directly mapping the BDD has depth $Depth_M$, the *select* and *data* functions obtained by the first level of bipartitioning each has a depth of at most $\lceil Depth_M/2 \rceil$. The use of the one-hot multiplexer adds a constant depth of one transistor to this. The bipartitioning procedure can be applied further to

these decomposed *select* and *data* functions, a process that can continue recursively. There can be at most $\lceil \log(Depth_M) \rceil$ such recursions, and after each recursion only a constant depth is added due to the one-hot multiplexer. Therefore, at the end of the recursion, the resulting implementation has the depth of $O(\log(Depth_M))$.

Unlike the regular multiplexer-based implementation for PTL circuits in [SB00] that obtains a logarithmic depth for only *xor* functions (for which the cutset size is always two), our use of one-hot multiplexers and recursive bipartitioning results in a logarithmic depth implementation for any circuit, irrespective of the cutset size. A logarithmic depth PTL implementation for any BDD is neither claimed nor proved in [SB00] whose approach relies on the use of regular multiplexers. It is clear that if, instead of one-hot multiplexers, regular multiplexers are employed during decomposition, the reduction in depth is somewhat lower: regular multiplexer based implementation has depth with a lower bound $\Omega(\log(Depth_M) \log(Min_{Cutsize}))$ and an upper bound $O(\log(Depth_M) \log(Max_{Cutsize}))$, where $Min_{Cutsize}$ and $Max_{Cutsize}$ denotes the minimum and the maximum of cardinalities of cuts at any bipartitioning stage, respectively.

2.4.2 Area Estimation

The flow network described above requires an estimate of the area cost for each candidate node in the BDD. To generate this estimate, we assume a BDD-mapped PTL implementation with pass transistors and buffers after every k transistors. The contribution of a node to the area cost is estimated as the sum of

- the area of the PTL implementation of the BDD rooted at a given node, and
- the area of the PTL network that terminates on the given node.

This area cost is computed in linear time by a postorder traversal of the network. At multi-fanout BDD nodes, the area cost is divided by the number of fanout edges. This heuristic is similar to that used in technology mappers for standard cell libraries such as [CP92, Sen92].

2.4.3 Complexity analysis

The computation time required to find essential and candidate nodes is linear in the size of the BDD network, as it involves a traversal, similar to the critical path method [SK92], of the the BDD. The time required for area cost estimation is also linear in the size of the network. The only computationally expensive procedure is the max-flow min-cut algorithm, which is employed to find an optimum cut with minimum area penalty. The time complexity of the Edmonds-Karp implementation of the Ford-Fulkerson algorithm for finding the max-flow and min-cut is $O(||V||||E||^2)$, where V (E) is the set of nodes (edges) in the flow network [CLR98]. While this seems expensive, in practice the time complexity of this algorithm is hardly reflected in the CPU times for the following reasons:

1. In our case, the size of flow network is very small as compared with the size of the BDD to be bipartitioned, since only a small fraction of all the BDD nodes qualify as either essential or candidate nodes.

2. Since the capacity assignment to the nodes is such that essential nodes are assigned very small capacity and are always in an optimum cut, a majority of flow augmentations are associated only with the part of the network that involves paths with candidate nodes. In other words, the flow network effectively contains only

the candidate nodes and related edges.

Since bipartitioning is applied recursively, the following recurrence equation describes the time complexity of the entire algorithm for a BDD containing n nodes.

$$T(n) = 2T(n/a) + f(n) \tag{2.1}$$

In the above equation, $a \leq 2$ to account for the possible increase in number of BDD nodes for the *select* functions, $f(n) = \Omega(n)$ due to linear time complexity of delay analysis and $f(n) = O(n^3)$ assuming the size of flow network to be same as that of the size of the BDD. Note that assumption that the size of the flow network is $O(n^3)$ is highly pessimistic for the two reasons mentioned before, but is useful enough to derive an loose upper bound on the time complexity. Employing the master theorem [CLR98], the above equation yields

$$T(n) = \Omega(n^{\log_a 2}), T(n) = O(n^3) \tag{2.2}$$

In practice, the algorithm requires CPU times that vary between super-linear to quadratic in number of BDD nodes, and in absolute terms, the run-times for the ISCAS'85 benchmarks are of the order of seconds.

2.5 Delay Modeling and Analysis

To identify essential and candidate nodes, it is important to perform delay analysis using a delay model that has good fidelity. The Elmore delay model [Elm48] satisfies such a requirement while being computationally inexpensive and has even been applied in the past for timing verification of complex microprocessor chips [NDH98]. We adapt this model for computing delays in PTL networks that are mapped directly from BDD's. It

is important to note that the Elmore delay model is utilized only for identifying essential and candidate nodes during the synthesis stage, while for the post-synthesis delay analysis of netlists for the PTL and static CMOS circuits, we employ the widely used non-linear delay model (NLDM) [WE94, CW97] that involves other factors, such as consideration of the slope of the input signal transition and load. In the following subsections, we describe the adaptation of Elmore delay model to PTL circuits, the corresponding delay analysis procedure and the post-synthesis delay model.

2.5.1 Delay Model for BDD-mapped PTL Circuits

The insertion of buffers that break up long transistor chains can result in short pass transistor segments such as that shown in Figure 2.7(a). Each pass transistor in such a segment can be modeled using an RC π model, where R denotes the resistance of the transistor and C denotes the drain/source capacitance. Pass transistors offer different resistance for rising and falling transitions: typically, for NMOS pass transistors, falling transitions are faster than the rising transitions. To account for this, two different values of resistance, one for the rising and one for the falling transition, are associated with each pass transistor. The value of the resistance is obtained by characterization of a pass transistor employing circuit simulator such as SPICE.

Figure 2.7(b) shows the corresponding RC network for the pass transistor segment in Figure 2.7(a). This is a special case where the pass transistor segment maps to an RC line. For more complex BDD's, it is likely that the pass transistor segment may be more complex, as illustrated in Figure 2.7(c). From this picture, it can be seen that BDD-mapped PTL networks, when modeled using an RC π model, look like an RC mesh rather than an RC line. In such a mesh, directions can be assigned to the resistive edges since transistors act as unidirectional switches; such methods have long been used

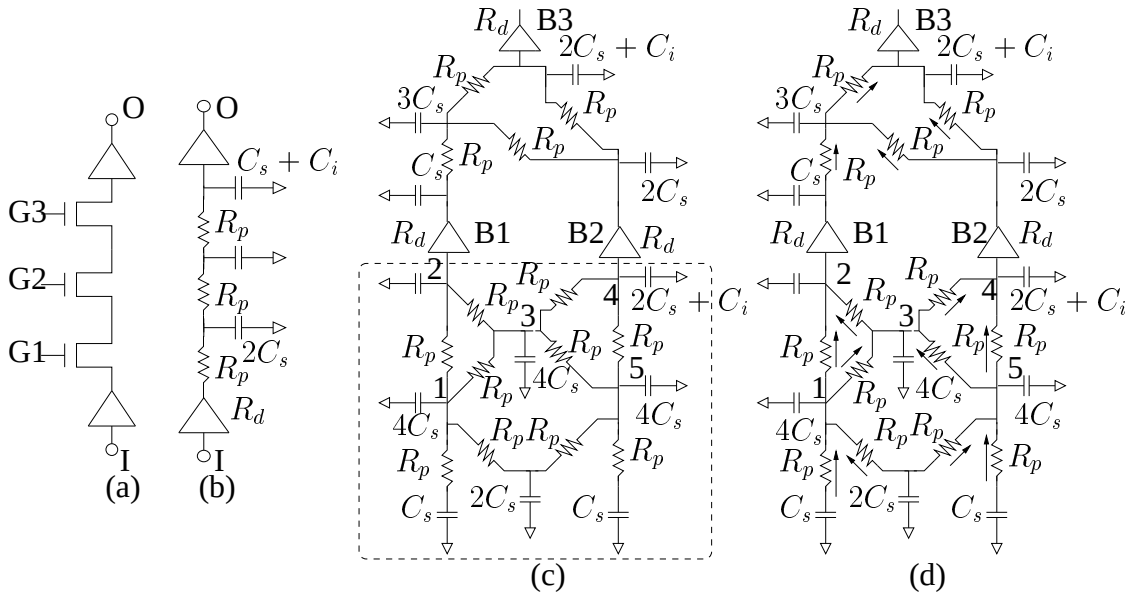


Figure 2.7: (a) A PTL circuit segment with three pass transistors in series. (b) The equivalent RC model for the PTL segment in (a). (c) The equivalent RC network corresponding to PTL implementation in Figure 2.2(b). (d) A set of assigned directions for the resistances. Here, R_p (R_d) corresponds to the pass transistor (driver) resistance, while C_s and C_i represent the source (as well as drain) capacitance and the inverter input capacitance, respectively.

in delay analysis tools and have even been employed in very old timing verifiers such as Crystal [Ous85]. Therefore, we can model a BDD-mapped PTL network, using RC π models, as a set of RC directed acyclic graphs (DAG's) between buffers, as shown in Figure 2.7(d).

Delay analysis for an RC tree can be performed using tree traversal in linear time in the size of a tree [SK92], while delay analysis for RC meshes using tree/link partitioning requires $O(nm^2)$ time, where n is the number of edges in a tree and m, the number of links, which when removed from an RC mesh results in an RC tree [CK90]. In our

case, however, the resulting RC structure is neither a tree nor a mesh, but an RC DAG – a more complex structure than trees and perhaps, simpler than meshes, from a graph theoretic perspective. The delay for an RC DAG can be defined as the maximum of the delay along any path and the Elmore delay along any path is defined similar to RC trees, as in [SK92]

$$D_{Elmore} = \sum_{i \in path} R_i \cdot C^i_{downstream} \tag{2.3}$$

This model of the network as an RC DAG does not take into account the logical dependencies between signals. If we consider these, we can see that depending on the input assignments, some of the resistances can be treated as open circuits, when the corresponding gate signals to the transistors are not high and may be removed from the RC network. The RC DAG's that model BDD-mapped PTL networks have a peculiar property that arises from a well known property of BDD's, namely, that for any assignment of inputs, only one path from a terminal node to a given node is active. The implication of this for RC DAG's is stated by the following observation.

Observation For any assignment of inputs, a given RC DAG must reduce to an RC forest.

Proof The proof proceeds by contradiction. Assume that for some assignment of inputs, a given RC DAG does not reduce to an RC forest. It implies that there is a cycle, which in turn implies that there exists a node which is driven by two different signals – a contradiction, since only one path to any node in a BDD is active.

For the part of the RC DAG covered by the dashed square in Figure 2.7(c), a number of different RC forests corresponding to all possible input assignments is shown in Figure 2.8. The Elmore delay can be computed for each of the trees.

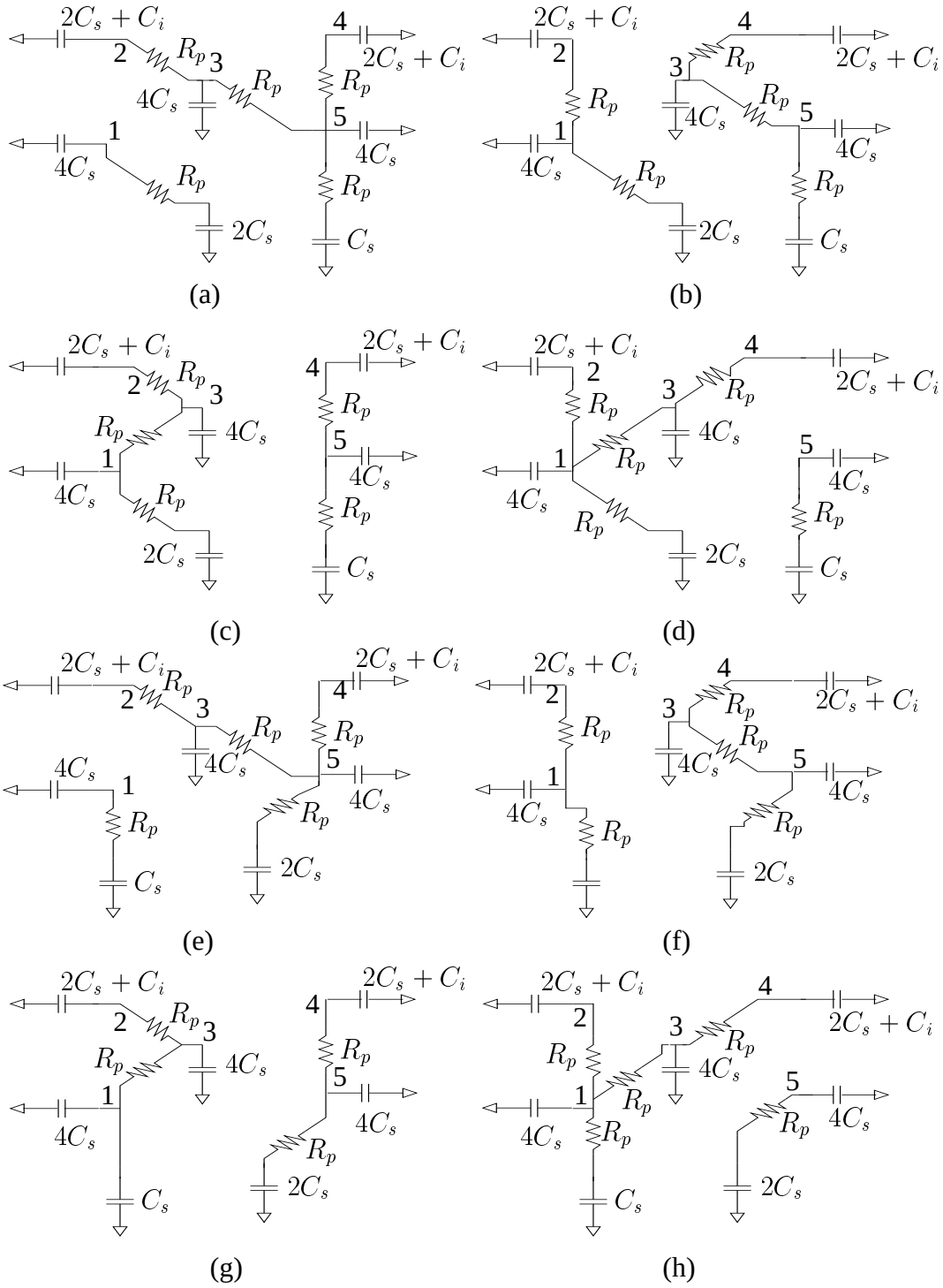


Figure 2.8: RC forests for the part of RC network covered by dashed square in Figure 2.7(c) corresponding to the assignments (a) 000, (b) 001, (c) 010, (d) 011, (e) 100, (f) 101, (g) 110, (h) 111 to the triplet $a_2 b_1 a_1$.

2.5.2 Delay Analysis for BDD-mapped PTL Circuits

To analyze RC DAG's, we may have to consider all possible input assignments; this number of such assignments is exponential in the number of inputs, assuming that the primary inputs are independent of each other. Fortunately, we can assume a reasonable PTL implementation from a given BDD that will allow us to perform delay analysis in linear time[4].

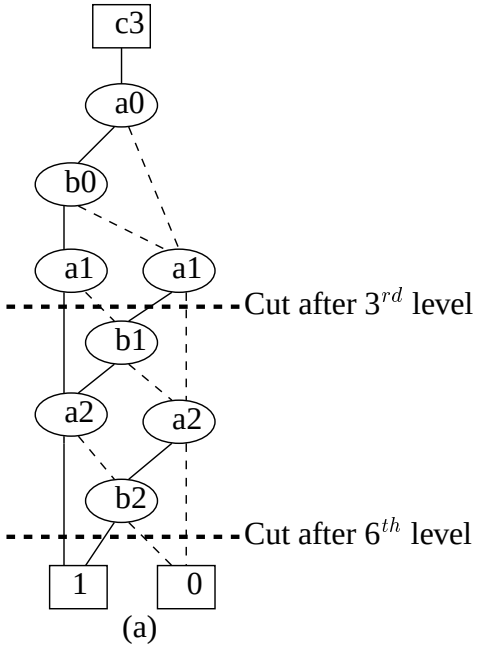


Figure 2.9: A pictorial illustration of the inverter insertion heuristic in [MBIS01]: inverters are used for the edges that are indicated as being cut. Under such an assumption, at most 2^k assignments must be considered for the part of the BDD between inverters.

One such implementation[5] is shown in Figure 2.9. This assumes that each BDD

[4]Performing the delay analysis in linear time is critical to keep the time complexity of our bipartitioning algorithm reasonable, since the delay analysis procedure is invoked during each bipartitioning call of our algorithm.

[5]Although our delay analysis procedure is targeted for this particular implementation, it can be extended to consider other PTL implementations that may use different heuristics for inverter insertion.

node is mapped on to a PTL multiplexer and the edges that cross every multiple of the k^{th} level (edges crossing the cuts in the figure, where $k = 3$) have inverters/buffers on them [MBIS01]. This buffer insertion heuristic ensures that there is an inverter after at most k transistors in series. In case of such a BDD-mapped PTL network that has buffers or inverters after at most k transistors in series, where k is bounded by a small constant, it is adequate to consider only 2^k different assignments for parts of the RC DAG that lies between successive buffer levels to find the maximum delay. Each edge will have to be traversed no more than 2^{k-1} times, as stated by the following observation.

Observation For a PTL network that has at most k transistors in series between buffers, the total number of edges in all of the forests corresponding to various input assignments is bounded by $2^{k-1} \cdot ||E||$, where $||E||$ is the number of edges in the RC DAG.

Proof Each edge in the BDD is associated with the true or complementary form of a variable in the BDD. Since inverters are inserted in such a way that at most k variables lie between two successive inverter insertion levels, only 2^k different assignments must be considered for a given part of the DAG. For exactly half of these assignments, a variable associated with the edge is true, and therefore, any edge can appear only in half of the forests. Since there are $||E||$ edges in the DAG corresponding to a BDD, the total number of edges in all the trees is bounded by $O(2^{k-1}||E||)$.

Remark In the Figure 2.8, $k = 3$ and $||E|| = 10$. It can be verified from the Figure 2.8 that the total number of edges, summed up over all of the RC trees, is $2^{k-1} \cdot ||E||$ = 40.

Different portions of the RC DAG can be successively considered to yield a linear time delay analysis procedure. We exploit this idea in the delay analysis algorithm. Algorithm 2.5.1 shows the pseudocode for the latter. The maximum downstream capacitance for a given node is computed by calling `GetDownStreamCapacitance()`

procedure for all possible $k-$bit Boolean assignments. This procedure traverses all the fanout edges that satisfy a specific $k-$bit Boolean assignment till the buffers are reached, adds the capacitances at the visited nodes, and stores the maximum downstream capacitance. Once the maximum downstream capacitance is computed, the delays at each node can be computed by sorting the nodes in topological order and computing the maximum arrival time at each node. The following theorem states the time complexity of the delay analysis algorithm.

Theorem 2.5.1 *The delay analysis using the Algorithm 2.5.1 takes no more than $O(\|E\|)$ time, where E is the set of edges in the BDD.*

Proof Using the arguments from the previous observation, it can be stated that the computation of downstream capacitance requires each edge to be visited at most 2^{k-1} times. The topological sort of the nodes requires linear time in the size of graph [CLR98]. Therefore, time required by delay analysis routine is $O(\|E\|)$, since k is a constant.

2.5.3 Post-synthesis Delay Models

2.5.4 Post-synthesis Delay Model for PTL

Figure 2.10 (a) shows timing arcs in PTL circuits, which correspond to two possible paths going through each transistor. Nonlinear delay models (NLDM's) are a popular way of representing the delays on the arcs of a timing graph. The widely used nonlinear Synopsys delay model for timing analysis involves the following equation [CW97]:

$$\tau = \alpha \cdot C + \beta \cdot \tau_r + \gamma \cdot C \cdot \tau_r + \delta \tag{2.4}$$

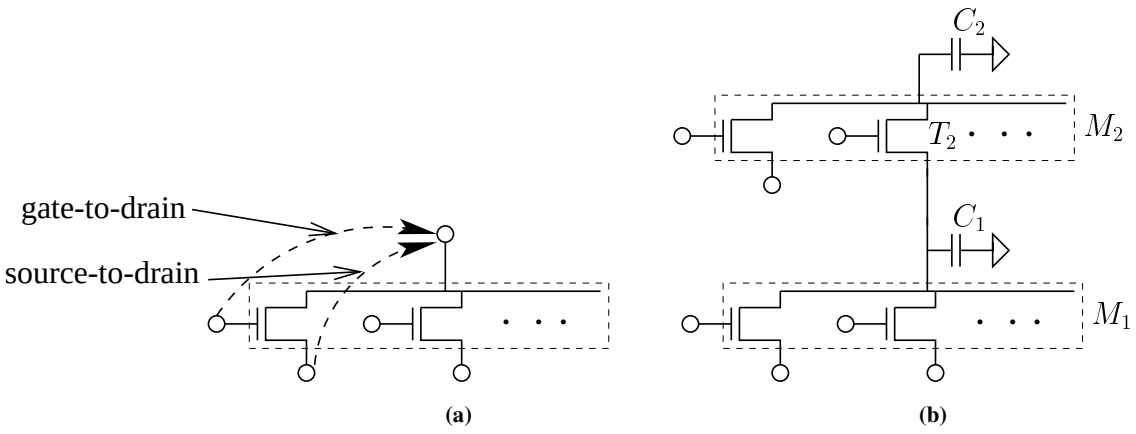


Figure 2.10: (a) Timing arcs for delay analysis of PTL circuits showing two timing arcs, gate-to-drain and source-to-drain, for a pass transistor. (b) If pass transistor T_2 in a multiplexer M_2 is ON, a capacitive load (C_{load}) seen by a driver at any source terminal for a multiplexer M_1 is related to C_1 and C_2 as follows: $C_{load} = C_1 + C_2$, assuming zero capacitances at the sources of M_1 and no shielding effect in pass transistors.

In the above equation, C stands for the load capacitance, τ_r the transition slope of input signal, τ the delay from input pin to output, while α, β, γ, and δ are the parameters obtained by characterization employing a circuit simulator such as SPICE. Each timing arc in the timing graph has four NLDM parameters associated with it, which are, typically, stored in lookup tables. The sizes of these lookup tables blow up when different supply voltages and temperatures are considered. To overcome this limitation, scalable polynomial delay models (SPDM) [spd] are used currently in commercial tools. However, NLDM is still accurate with a single supply voltage and a given temperature. Therefore, for comparison purposes at the logic synthesis level, where only a single supply voltage and a single temperature is considered, non-linear delay model of the form of Equation 2.4 is still valid and we use the same delay model for static CMOS and PTL

synthesis results.

Adapting NLDM to timing analysis for PTL circuits requires the computation of downstream capacitance that may be beyond the given PTL multiplexer, as shown in Figure 2.10 (b). We employ a DAG traversal similar that is similar to that of the procedure shown in Algorithm 2.5.1 to compute the downstream capacitance by traversing the downstream DAG. Unfortunately, the resulting pass transistor network does not possess a structure that will allow the incorporation of logical dependencies between signals during computation of downstream capacitance. This is because the consideration of these logical dependencies is at least as difficult as the NP-hard *dynamic path sensitization* problem, as pointed out in [DY96]. This differs from the problem of delay analysis used during recursive bipartitioning, where the gates of all of the multiplexers are controlled by primary inputs, which, however, is not true in case of the pass transistor network obtained after recursive bipartitioning. For this reason, this analysis ignores logical dependencies and computes the downstream capacitance by simply traversing the DAG until the buffers are reached. This may result in an overestimate of the downstream capacitance and hence, an overestimate of the delay. Apart from logical dependencies, another source of pessimism in the capacitance estimate is the shielding effect due to the (nonlinear) resistance of the pass transistors that are ON.

Once the downstream capacitance for all multiplexers is computed, the precharacterized NLDM parameters are used to compute the delays in an entire PTL circuit. After precharacterization, it was verified that the delay estimated by the model was indeed an overestimate, as compared to SPICE, as illustrated on several benchmark circuits in Table 2.3. All of these circuits were mapped directly on to a PTL network with inverters with weak pull-ups inserted after every three series-connected transistors. Because of

Example	#. Transistors	SPICE delay (ps)	Timing Analysis Delay (ps)
9sym	102	462.40	618.50
parity	100	999.90	1172.64
rd84	194	501.50	628.34
rd73	144	401.40	473.65
rd53	72	196.10	205.54

Table 2.3: Comparison of SPICE delays with the delays obtained from static timing analysis using NLDM on several combinational benchmark circuits.

the direct mapping, the critical paths in these circuits are long and contain at least as many number of transistors as there are primary inputs (besides the inverters with weak pull-ups). Technology parameters for 0.13μm technology [ptm] are used for SPICE simulations and all of the transistors have a length of 0.13μm, while the widths of the transistors are as follows. The widths of NMOS pass transistor are 0.91μm, the widths for transistors in inverters with weak pull-ups are $w_n = 1.04\mu$m, $w_p = 2.08\mu$m, and $w_{pull-up} = 0.26\mu$m, where w_p, w_n, and $w_{pull-up}$ are widths of PMOS, NMOS, and weak pull-up, respectively. The critical paths are determined using static timing analysis (STA) and simulated by applying appropriate stimulus. The transition time for rising as well as falling transition of input signal is 50ps. We can verify from the table that the delays estimated by static timing analysis are always overestimates. For long critical paths, overestimates tend to be high because of the cumulative effect.

2.5.5 Post-synthesis Delay Model for Static CMOS

We use the same Equation 2.4 for the static timing analysis of static CMOS standard cell circuits. For each cell, all input-to-output timing arcs are precharacterized for NLDM parameters employing SPICE. We note that in case of static CMOS circuits, there is no pessimism in capacitive load estimation, unlike PTL circuits, since all the inputs are always connected to the gates of the transistors.

2.6 Experimental Results

2.6.1 Experimental Setup

The algorithms described in the sections 2.4 and 2.5 are implemented in a C++ program called PTLS (Pass Transistor Logic Synthesizer). For all of our experiments, the BDD package CUDD [Som] is employed for generating BDD's, along with sifting [Rud93] for variable ordering. We use NMOS transistors as pass transistors and employ inverters with weak pull-ups after every three pass transistors in series. Inverters with weak pull-ups are also inserted to drive the gates of transistors in one-hot multiplexer for the implementations obtained by our recursive bipartitioning technique. We synthesize both the PTL and static CMOS circuits (which are used to compare the PTL circuits against) in a 0.13μm technology [ptm]. All transistor lengths are set to 0.13μm, and the following two sets of transistor sizes are employed for the PTL implementations:

- **Set 1**: All NMOS pass transistors have $w_n = 1.82\mu$m, and inverters have sizes w_p = 4.16μm and $w_n = 2.08\mu$m, while the weak pull-up transistor in each inverters are sized to a width of 0.52μm.

- **Set 2**: All NMOS pass transistors have $w_n = 0.91\mu$m, and inverters have sizes w_p = 2.08μm and $w_n = 1.04\mu$m, while the weak pull-ups in the inverters are sized to a width of 0.26μm. In other words, all the transistors in **Set 2** have half the widths of the corresponding transistors in **Set 1**.

For static CMOS circuits, we choose the lib2.genlib library in [Sen92] and add at least two strengths ($w_{effective} = 0.78\mu$m and $w_{effective} = 1.56\mu$m) for each of the cells in the library. Simpler gates such as inverters, NAND's (up to four inputs), and NOR's (up to four inputs), have up to four strengths ($w_{effective} = 0.78\mu$m, 1.56μm, 2.34μm, 3.12μm). All of these gates and pass transistors are characterized for falling and rising input transitions, with the input signal transition times varying from 50ps to 130ps in steps of 5ps, while the characterized loads vary from 1fF to 50fF in steps of 1fF. The supply voltage and temperature used are 1.3V and 25^{o}C.

Under this sizing scheme, we see that PTL circuits have uniform sizes for pass transistors and inverters, while the static CMOS implementations use better sizing, with each gate having several choices for the transistor sizes. In spite of this, we show that PTL results in implementations that have the same (or better) delay as that of static CMOS implementation with a significant area average in case of arithmetic, error correcting, and some control circuits in ISCAS'85 benchmark suite. If we allow a larger variety of transistor sizes for PTL circuits, it is likely that these results may improve even further in favor of PTL.

2.6.2 Synthesis Procedure

Static CMOS circuits are preprocessed by running *script.rugged* in SIS [Sen92] before performing technology mapping for optimizing delays. For PTL synthesis, we use the

same Boolean network obtained from *script.rugged*, and create a multilevel BDD representation. Our recursive bipartitioning procedure is then applied level-by-level on this multilevel BDD representation.

While creating this multilevel BDD representation, it is important to control the number of BDD nodes, since the number of transistors in the resulting implementation depends on this number. It has been shown in [YC99] that the use of traditional multilevel boolean network optimization, followed by the construction of BDD's for nodes in the network, results in reasonable BDD sizes. These sizes are comparable to those obtained by applying area-oriented pass transistor logic synthesis techniques such as [CLAB98]. We employ these multilevel BDD representations, which have reasonable sizes, for delay-oriented decomposition. Further improvements may be possible if the multilevel BDD's are preprocessed using algorithms such as *eliminate*, as in [CLAB98], and by applying better variable ordering heuristic such as symmetric sifting.

2.6.3 Analysis of Results on ISCAS'85 Benchmarks

Table 2.4 shows the area/delay comparison between PTL circuits and their corresponding static CMOS implementations for all of the ISCAS'85 benchmarks. In this comparison, the PTL circuits in Table 2.4 have transistor sizes from **Set 1** and **Set 2** described in Section 5.1. For the same table, Column 1 shows the name of the benchmark and its functionality, while columns 2 and 3 show the area and delay, respectively, for the static CMOS implementation. Columns 4 and 5 show the area and delay, respectively, for the PTL implementation with the transistor sizes from **Set 1**, while Column 6 shows the CPU time required for our PTL synthesis algorithm on a 400 MHz Sun Ultra-Sparc 60 machine. Columns 7 and 8 show the area and delay, respectively, for the PTL implementation with the transistor sizes from **Set 2**.

Example (Functionality)	Static CMOS		PTL **Set 1**			PTL **Set 2**	
	Area	Delay	Area	Delay	CPU	Area	Delay
	μm^2	ps	μm^2	ps	s	μm^2	ps
C1355 (Error correcting codes)	4886	962	3521(38%)	599(60%)	00.1	1760 (177%)	1009 (-4%)
C1908 (Error correcting codes)	5157	1205	3726(38%)	980(20%)	00.2	1863 (176%)	1432 (-15%)
C2670 (ALU and Control)	12307	1208	8781(40%)	1660(-27%)	02.7	4390 (180%)	2132 (-43%)
C3540 (ALU and Control)	34280	1796	15409 (122%)	1850 (-2%)	22.4	7704 (344%)	2454 (-26%)
C432 (Priority Decoder)	3370	1334	3981(-15%)	1278(4%)	00.2	1990 (69%)	1758 (-24%)
C499 (Error correcting codes)	4784	938	3259(46%)	630(48%)	00.1	1692 (182%)	929 (1%)
C5315 (ALU and Selector)	24042	1355	24612 (-2%)	2233 (-39%)	17.4	12306 (95%)	2668 (-49%)
C6288 (16-bit Multiplier)	29592	4799	25771(14%)	4152(15%)	17.7	12885 (129%)	5563 (-13%)
C7552 (ALU and Control)	32051	1323	16649(92%)	2112(-37%)	10.7	8324 (285%)	2528 (-47%)
C880 (ALU and Control)	5579	1080	4748 (17%)	952 (13%)	01.1	2374 (134%)	1430 (-24%)
Avg. Improvement			**39%**	**5%**		**177%**	**-24%**

Table 2.4: Area/Delay comparisons for static CMOS and PTL implementations of IS-CAS'85 benchmarks.

For the comparison between static CMOS implementations and PTL implementations with transistor sizes from **Set 1**, the following observations can be made from Table 2.4.

- For circuits that implement error correcting codes, namely, C1355, C1908, and C499, for the multiplier circuit, C6288, and for the arithmetic logic unit (ALU) and control circuit, C880, PTL implementations are superior in terms of area as well as delay. On average, the area advantage is 30% while the delay advantage is 31%.

- For ALU and Control circuits such as C2670, C7552, and C3540, the PTL im-

plementations are superior in terms of area but could not match the static CMOS delay. On average, the area advantage is 84%, while the delay disadvantage is 22%. With the area numbers strongly favoring PTL, it is likely that static CMOS delays may be matched with sizing for PTL circuits, perhaps even while retaining some area advantage.

- For the priority decoder circuit, C432, PTL has a marginal delay advantage of 4% at the cost of a 15% area increase.

- For the ALU and selector circuit, C5315, PTL produces inferior results both in terms of area as well as delay. The area disadvantage is minor, at 2%, while the delay disadvantage is 39%.

Note that the observations are made with a pessimistic delay model for PTL, as described in Section 4.3. From the above observations, the following conclusions can be arrived at:

- For the ALU and control circuit, C5315, static CMOS results in a superior implementation. In case of other ALU and control circuits such as C2670, C3540, and C7552, static CMOS yields a superior delay, but with a significant area cost. For the ALU and control circuit, C880, using even näive transistor sizing PTL results in a superior implementation as compared to static CMOS. All these circuits contain arithmetic components such as adders apart from control logic. Since these circuits contain *nand*-intensive control logic, scripts in SIS [Sen92], which are skewed towards control logic synthesis, perhaps destroy the structure of arithmetic components in these circuits and PTL implementation due to our PTL synthesis algorithm with näive transistor sizing is not able to match (or outperform)

the static CMOS delay consistently. PTL implementations still have a good area advantage that can be utilized by a sizer to match delays due to static CMOS implementations.

- The PTL implementation provides large area savings and improved delays in case of the purely arithmetic and error correcting circuits (C1355, C1908, C499, C6288) as compared to the static CMOS implementations. In these cases, SIS [Sen92], perhaps, is not able to destroy the structure as much, since most of these circuits are heavily *xor*-dominated circuits.

A comparison between PTL implementations with smaller transistor sizes and static CMOS implementations is also shown in Columns 7 and 8 in Table 2.4. For these columns, the PTL circuit implementations employ transistor sizes that are chosen according to **Set 2** in Section 2.6.1. With the smaller transistor sizes, the delays in PTL circuits are degraded, as expected. Although the transistor sizes are halved, the delay in static CMOS is still matched in case of C499, and is within 20% for the circuits C1355, C1908, C6288, but with a large average area advantage of 166%. Given such an area margin, it is likely that an intelligent sizer may be able to match the delays in static CMOS circuits, while maintaining an area that is less than that of the PTL circuits shown in Column 4 of Table 2.4, which use double the transistor sizes of this case. For the set of circuits that includes ALU and control circuits such as C2670, C7552, C432, C880, C3540, and C5315, the average area advantage is 184%, while delays degrade by 35%, on an average. It is likely that improved transistor sizing in these cases may improve the results in favor of PTL, since as shown in Table 2.4, simplistically doubling the sizes of all transistors results in a large improvement in delay, while maintaining significant area savings in most cases.

2.6.4 Comparison with Previous PTL Approaches

We now present a comparison of the number of transistors in our PTL implementations with the previous PTL synthesis approaches such as [BNNSV97, FMM$^+$98]. The delay comparison could not be performed because of the unavailability of parameters for the delay models. Moreover, these approaches do not specifically optimize for delay, and primarily target the area of the PTL implementation.

Table 2.5 shows the number of transistors required for the PTL implementation of the ISCAS'85 circuits as a result of applying the algorithm of [BNNSV97] in Column 2, the algorithm of [FMM$^+$98] in Column 3, and our approach in Column 4. For most of the benchmarks, the number of transistors due to our method is comparable with that of the other two approaches, with an average increase of 12% with respect to [FMM$^+$98] and an average reduction of 41% with respect to [BNNSV97].

2.6.5 Conclusions

Based on the experimental results, we can conclude the following.

1. Using the delay-driven synthesis algorithm, näive uniform transistor sizing for PTL circuits and a pessimistic delay model for PTL, we have shown that PTL can certainly match (or improve upon) the delays in static CMOS circuits, with a significant area advantage of an average of about 40% for arithmetic circuits, error correcting circuits, and some control circuits.

2. For control circuits, static CMOS may sometimes, but not always, result in superior implementations than PTL in terms of area as well as delay.

Example	Number of Transistors		
	[BNNSV97]	[FMM$^+$98]	Ours
C1355	1969	1013	1037
C1908	2116	1526	1145
C2670	3198	2674	2876
C3540	4997	4440	4757
C432	979	727	1110
C499	1947	1013	998
C5315	8277	4043	8221
C6288	10787	7073	7794
C7552	13268	6590	5347
C880	1622	1339	1467
Total	49160	30438	34752

Table 2.5: Comparison of the number of transistors resulting from our approach with previous PTL approaches [BNNSV97, FMM$^+$98].

3. Some control circuits, may be implemented well in PTL with slightly degraded delays as compared to static CMOS, but with large area savings. Allowing multiple transistor sizes for PTL may improve the results in favor of PTL, but this has not been explored in this thesis.

2.7 Power Dissipation Driven Synthesis

Power dissipation is becoming a critical problem in modern day deep sub-micron circuits, especially in case of circuits that are used in portable battery-operated devices. The problem of power optimization at various levels of abstraction has been addressed by numerous researchers. At the logic level, power optimizations include techniques such as gated clocks and precomputation; the latter involves the use of the observability *don't cares* to disable the clock signal at the input registers [AMD+94]. Employing a similar approach, Ruan *et al.* propose bipartitioned codec architecture, in which output values are encoded using the minimum number of bits and then decoded utilizing a decoder in the next clock cycle, or computed conditionally [RSL+99, RSLT01]. A limitation of the precomputation scheme [AMD+94] is the addition of extra logic to the circuit, while the bipartitioning codec approach may not be always optimal. For power optimization of PTL circuits, Lindgren *et al.* propose the use of sifting [Rud93], which reduces switching activity in BDD mapped PTL circuits [LKTD01]. Another approach by Tavares *et al.* comprises employing split cofactors based on the Shannon expansion at the root of BDD's, with the variable corresponding to the root node being used as a control input to disable the inverters [TB99]; the disabled inverters cannot make low-to-high transition, resulting in reduced switching activity. This approach differs from the method of independent of cofactors by Alidina *et al.* in disabling inverters rather than disabling registers and also in algorithm, as [AMD+94] utilizes area-efficient co-factors, while [TB99] employs the power-efficient cofactors.

We propose a BDD decomposition technique to minimize the power dissipation in combinational logic under the assumption that all primary inputs and primary outputs

are registered. Our contributions are summarized as follows.

- We apply the switching probability estimation technique proposed in [LKTD01] to estimate the switching probabilities in PTL circuits and also take into account the capacitance driven by each node in the PTL circuit, unlike [LKTD01] which uses a linear fanout model.

- Unlike the previous approaches [AMD+94, TB99], which use a single variable to disable the inputs of independent co-factors, we decompose the logic function using the max-flow min-cut technique to find a cut in the BDD that minimizes the power dissipation; the cut yields a subset of variables used as inputs to select logic that is used to disable the part of the circuit that does not perform useful computation in a given clock cycle.

- Our decomposition-based implementation model is more flexible than the bipartitioning codec architecture proposed in [RSLT01] and allows us to find optimum decomposition; optimality of decomposition is ensured due to the application of Ford-Fulkerson algorithm [CLR98] to find the min-cut.

2.8 Power Model

Figure 2.11(a) shows the BDD for the function f = ab+c(ab'+a'b), while Figure 2.11(b) shows the corresponding PTL implementation[6] in which every node in the BDD is translated into a 2-input multiplexer. Given the probabilities for the inputs, the switching probability of a function f can be expressed in terms of the probabilities of its co-

[6]In another implementation, transistors corresponding to a 2:1 multiplexer driven by c, c' can be replaced by just c'. The above implementation is used just to illustrate the capacitance estimation.

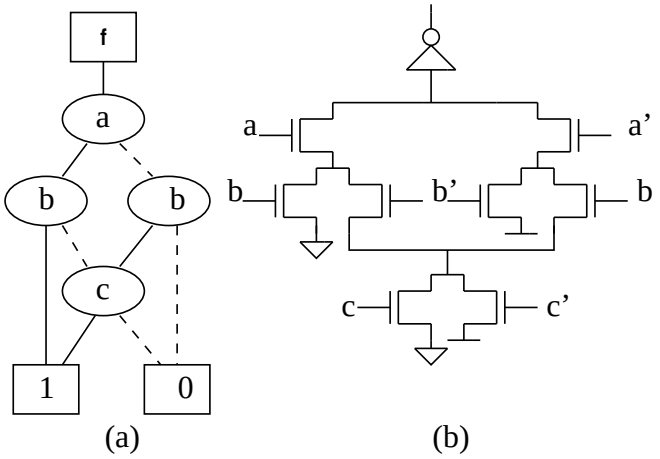


Figure 2.11: Correspondence between a BDD node and its PTL implementation: (a) The BDD for f=ab+c(ab'+a'b). (b) The corresponding PTL Implementation.

factors, f_x and $f_{\overline{x}}$, where x is an input. Equations (2.5) and (2.6) express the probability of f being 1 and 0, respectively, while switching probability is given by Equation (2.7).

$$p_{(f=1)} = p_{(x=1)} \times p_{(f_x=1)} + p_{(x=0)} \times p_{(f_{\overline{x}}=1)} \tag{2.5}$$

$$p_{(f=0)} = p_{(x=1)} \times p_{(f_x=0)} + p_{(x=0)} \times p_{(f_{\overline{x}}=0)} \tag{2.6}$$

$$p_{(f_{switching})} = 2 \times p_{(f=1)} \times p_{(f=0)} \tag{2.7}$$

The switching probabilities for the nodes in BDD shown in Figure 2.11 are computed assuming uniform input probabilities (i.e., $p(a=1) = p(b=1) = p(c=1) = 0.5$) and are shown next to the corresponding nodes in Figure 2.12(a). The triplet (p_{sw}, p_1, p_0) next to each node corresponds to the switching probability, the probability of the node being evaluated to 1, and the probability of node being evaluated to 0, respectively. For instance, the nodes in Figure 2.12(a) corresponding to the nodes labeled 'b' have the switching probability $p_{sw} = 3/8$, a probability of being evaluated to 1, p_1, of $3/4$,

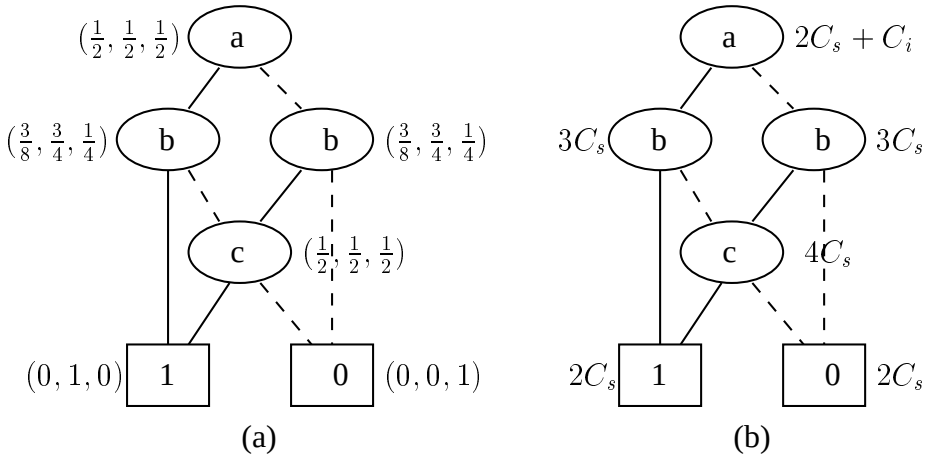


Figure 2.12: Power estimation in PTL circuits: (a) Switching probability estimation. (b) Capacitance estimation.

and a probability of being evaluated to 0, p_0, of $1/4$. The capacitances driven by each node can be computed by examining the PTL implementation. For instance, the node labeled 'c' drives four source capacitances (C_s) of NMOS transistors. The capacitance driven by each node is shown next to that node in Figure 2.12(b), where C_i is the input capacitance of an inverter. Once the switching probabilities and capacitances are known, the dynamic power can be obtained by employing the following formula, where, V_{dd} is supply voltage, f is the clock frequency, P_{sw} and C_{sw} are switching probabilities and capacitances, respectively.

$$Power = \sum_{\forall nodes} P_{sw} C_{sw} V_{dd}^2 f \tag{2.8}$$

2.9 Decomposition for Low Power

Figure 2.13 shows a general combinational logic circuit with registered inputs and outputs. Assuming PTL implementation of the combinational logic, we observe that switch-

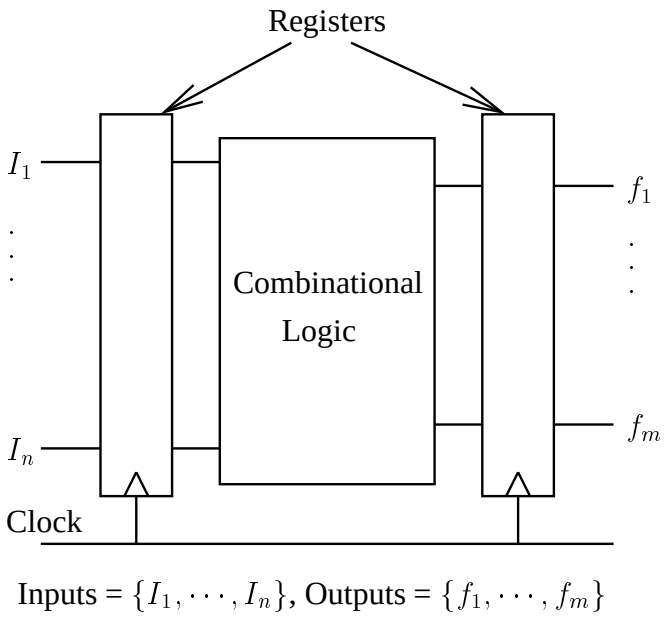
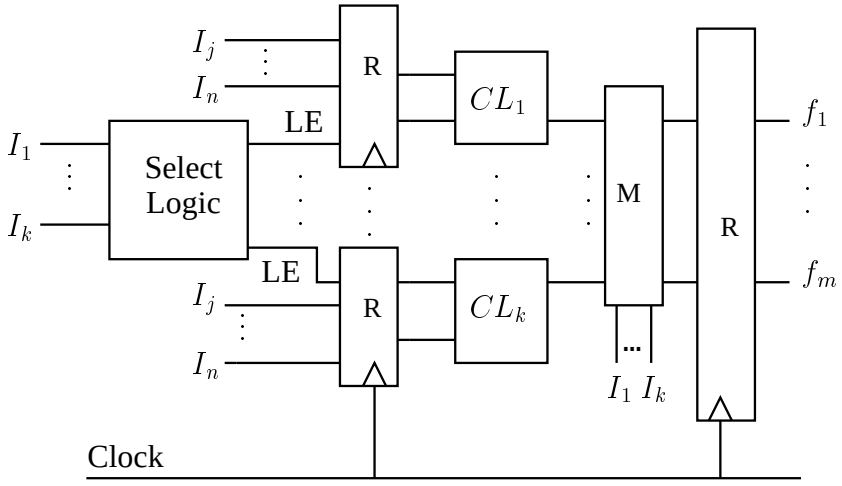


Figure 2.13: Combinational logic with registered inputs and outputs.

ing activity occurs in the entire PTL network during every clock cycle, although parts of the network may not perform useful computation. This can be observed from a property of BDD's that for any assignment of inputs, only one path from root to terminal node is active, so that the PTL implementation of this path performs useful computation for a given assignment, while the rest of the PTL circuit still dissipates power because of the switching of its inputs. Therefore, reduction in power dissipation can be achieved, if we disable the part of the PTL network that does not perform useful work. Figure 2.14 shows the decomposition-based implementation model in which a subset of inputs is used to generate latch enable signals for the input registers. The enable signals constitute the 'select logic' block, while the other combinational logic blocks, CL_1 through CL_k, are PTL implementations of logic derived from original BDD's, as explained in the following subsection. Multiplexers are used to select the outputs from the block



Inputs = $\{I_1, \cdots, I_n\}$, Outputs = $\{f_1, \cdots, f_m\}$, LE = Latch Enable
CL_i = Combinational logic block i, M = Multiplexer, R = Register

Figure 2.14: Decomposition model for the implementation of pipelined combinational logic.

that performs useful computation in a given clock cycle. In this case, we observe that only the select logic and multiplexers are active all the time, while other combinational blocks are not. Greater power reduction can be achieved if select logic and multiplexers dissipate small power and if the combinational blocks CL_1 through CL_k are active with small probabilities. In this case, the total power dissipation in the combinational logic is given by Equation (2.9), where p_i is the probability of combinational block CL_i being active and P_{CL_i} is the power dissipation[7] in combinational block CL_i.

$$P_{Decomp} = P_{SelectLogic} + P_{Muxes} + \Sigma_{i=0}^{k-1} p_i P_{CL_i} \tag{2.9}$$

[7]We consider only dynamic power dissipation. In the technologies beyond 100 nm, where leakage power becomes dominant, foot transistors, whose gates are driven by latch enable signals, can be used for the different combinational logic block, as in case of Boosted Gate MOS (BG MOS) [ITN+00].

While the above equation is similar to the equation presented in [RSLT01], it is more general in the sense that k is allowed to take any value, unlike [RSLT01], where k is restricted to 2, in the bipartitioning codec architecture.

2.9.1 Example

Consider the optimized BDD on 6 inputs for the carry output function for a 3-bit adder as shown in Figure 2.15(a); Figure 2.3 is duplicated here for readability purposes. If we map the BDD to PTL using the implementation model of Figure 2.13, then the entire PTL network will dissipate power in each clock cycle. On the other hand, if we take

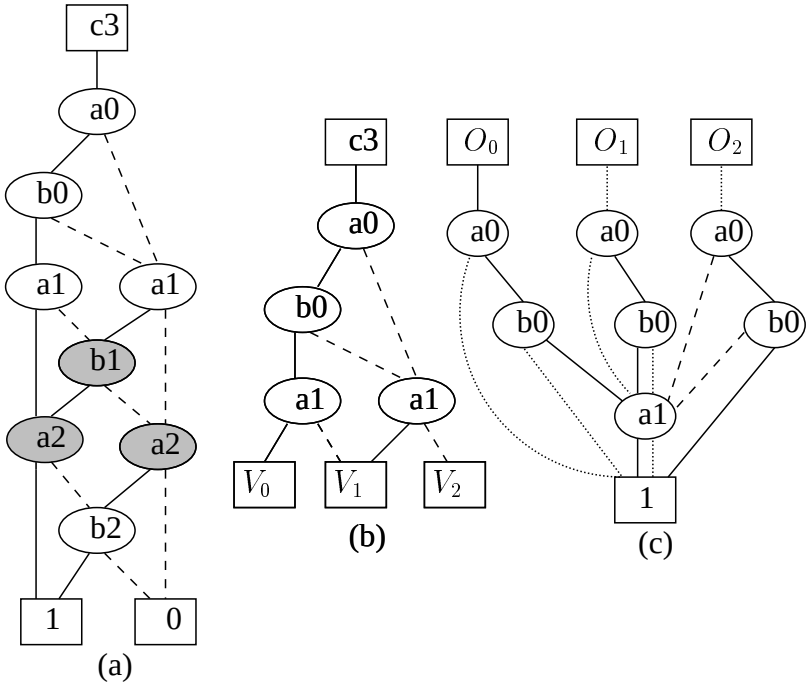


Figure 2.15: (a)BDD for carry function for 3-bit adder. (b) Introducing dummy nodes in the original BDD. (c) BDD's for select logic after one-hot encoding of dummy nodes.

a cut across the BDD containing the shaded nodes as shown in Figure 2.15(a), then

we can decompose the BDD into smaller BDD's and build the original function using multiplexers and PTL implementation of these BDD's.

The process of decomposition can be explained as follows. To generate the BDD's for select logic, we introduce dummy terminal nodes V_0, V_1, and V_2 as shown in Figure 2.15(b) and encode them using scheme such as minimum-bit encoding or one-hot encoding. This is similar to the BDD decomposition proposed in [SS01b] for performance-oriented PTL synthesis. Figure 2.15(c) shows the select functions obtained by one-hot encoding, i.e., to generate O_0, we set $V_0 = 1$, $V_1 = 0$, and $V_2 = 0$. Similarly, BDD's for O_1 and O_2 are obtained. Functions O_0, O_1, and O_2 are used as latch enables[8] for the registers and also as select inputs for the multiplexer to select among three combinational logic blocks whose BDD's are shown in Figure 2.16. As shown in Figure 2.16,

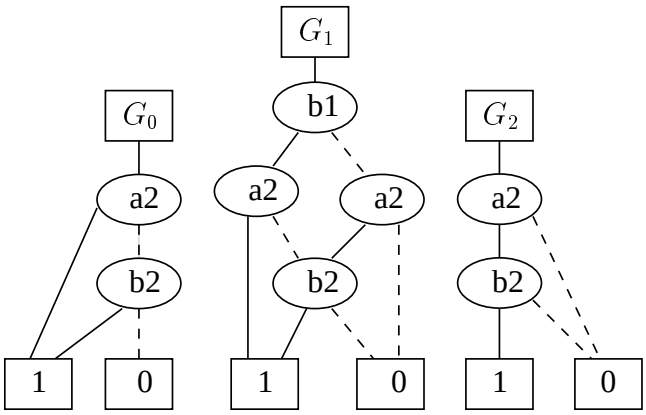


Figure 2.16: BDD's for functions in combinational logic blocks.

BDD's do not share the nodes with the same functionality; such nodes are duplicated, which may cause an area overhead. The decomposed implementation for the carry output function excluding 'select logic' is shown in Figure 2.17. In this case, total power

[8]These signals are latched on falling edge of clock to avoid hazards.

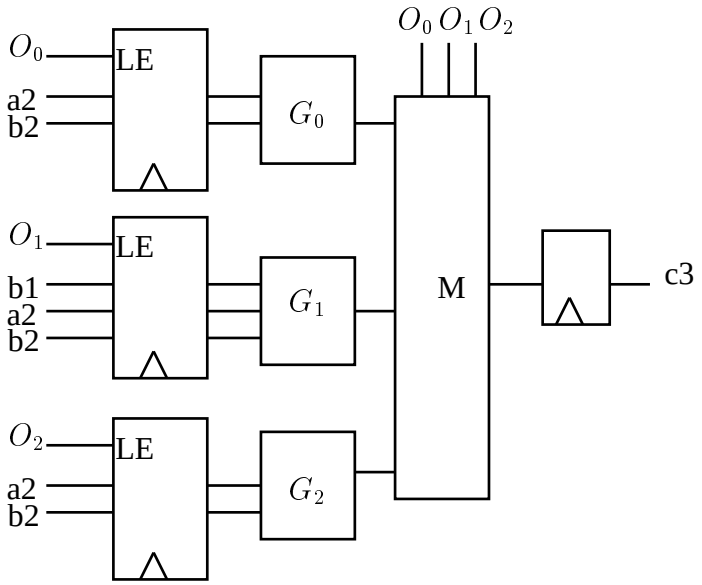


Figure 2.17: Decomposed implementation of the Carry function.

dissipation in combinational logic can be computed as follows: probabilities of O_0, O_1, and O_2 being evaluated to 1 are computed using uniform probability assumption at the primary inputs, which can be relaxed for the known input probabilities.

$$P_{c3} = P_{O_0,O_1,O_2} + P_{(3:1)mux} + p_{(O_0=1)}P_{CL_0} + p_{(O_1=1)}P_{CL_1} + p_{(O_2=1)}P_{CL_2} \quad (2.10)$$

$$P_{c3} = P_{O_0,O_1,O_2} + P_{(3:1)mux} + 0.125P_{G_0} + 0.5P_{G_1} + 0.375P_{G_2} \quad (2.11)$$

It is easily seen that the power dissipation in the combinational logic varies depending on the cut and that there are a large number of candidate cuts in a given BDD. Our objective is to find a cut such that power given by Equation (2.9) is minimized. We propose an algorithm to find an optimum cut in the following subsection.

2.9.2 Algorithm

We represent a BDD as a directed acyclic graph (DAG), where the nodes and edges are identical to the nodes and edges in the BDD, respectively, and are assigned a direction corresponding to variable ordering, from a lower indexed variable to a higher indexed variable. As explained in Section 2.8, the switching probability of a node depends on its cofactors and probability information of the input variable at that node. The power dissipation at each node can be computed employing the post-order traversal of a graph. The probability of node being selected as well as the power dissipation in select logic can be computed in a similar manner. The total power dissipation of PTL implementation of a function rooted at a given node is just the sum of power dissipation of its cofactors and power dissipation at that node[9].

The cost of each node is calculated as the sum of the power dissipation in select logic (P_{select}) and product of probability of the node being selected (p_{select}) and power dissipation at that node (P). The cost estimation for the BDD shown in Figure 2.11 (a) is shown in Figure 2.18; the probability and capacitance estimations have already been shown in Figures 2.12 (a) and 2.12 (b), respectively. In Figure 2.18, we use $C_i = K \times C_s$ and choose $K = 10$ for the sake of simplicity. The triplet ($p_{select}, P, P_{select}$) shown next to each node in Figure 2.18 corresponds to probability of the node being selected, power dissipation in the PTL network rooted at the node in terms of switching capacitance, and power dissipation in the select logic. As an example, Figure 2.18 lists the three node cuts[10], namely Cut A, Cut B, and Cut C containing one, two, and three nodes,

[9] In this analysis, signal correlations are ignored. While this may lead to some inaccuracies, it is generally considered as an acceptable approximation. This technique can be substituted by any other technique for probability computation that considers correlation.

[10] Note that cuts have been enumerated for illustrative purposes only and that our algorithm finds the minimum cut without any such enumeration.

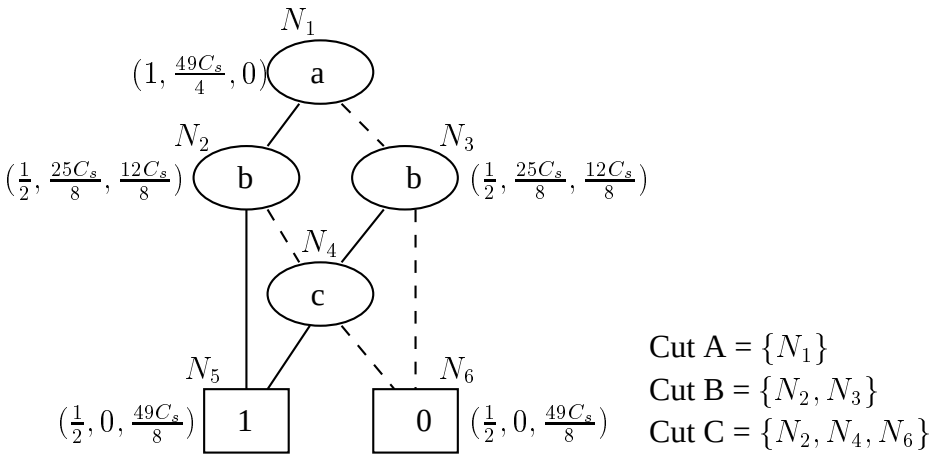


Figure 2.18: Estimating the cost of nodes.

respectively. The cost of a cut is simply the sum of the cost of each node in the cut. After evaluating the cost of each node, the DAG can be converted into a flow network. Ford-Fulkerson algorithm [CLR98] is then applied to find a minimum cut that corresponds to the implementation with the minimum power dissipation. In case of Figure 2.18, the minimum cut is Cut B with the cost $\frac{73C_s}{8}$, while the cost of Cut A and Cut C is $\frac{49C_s}{4}$ and $\frac{253C_s}{16}$, respectively. The pseudo-code of the overall procedure is as shown in Algorithm 2.9.1. Once the cut is determined, the vertices in the cut are replaced by dummy terminal nodes, which can be assigned unique codes to generate select logic, and the complete implementation can be produced, as illustrated in the previous section. The following proposition states the time complexity of our algorithm.

Proposition 2.9.1 *The procedure shown in Algorithm 2.9.1 takes $O(N^3)$ time to find an optimum cut, where N is the number of nodes in the original BDD.*

Proof 2.9.1 The Step 1 to Step 5 take $O(N)$ time since Step 1, 2, and 3 use post-order traversal that takes $O(\|V\| + \|E\|)$ time on a graph, where V is a set of nodes and E

is a set of edges, while Step 4 and Step 5 also require the linear time in the size of the graph. In case of BDD's, all nodes, except the terminal nodes, have two fanout edges, and therefore, $O(\|V\| + \|E\|) = O(N)$. The Step 6 takes $O(\|V\|\|E\|^2)$ for Edmond-Karp implementation of Ford-Fulkerson algorithm [CLR98]. Therefore, the algorithm takes $O(N^3)$ time to find a minimum cut in the worst case.

Comment: Although time complexity of our algorithm is $O(N^3)$, a tighter upper bound can be obtained, since we have observed that algorithm took less than a second in case of most of the MCNC benchmarks on Sun Ultra-60 machine and that the actual run times do not increase cubically. This is because the nodes that are not in the min-cut have higher costs, as we move away from the cut and therefore, number of flow augmentations performed by Edmonds-Karp implementation are far less than $O(N^2)$. Since the Ford-Fulkerson algorithm takes $O(N)$ time for each augmentation, this more accurately reflects the trend of run times for the circuits in the MCNC suite.

One can observe that each cut corresponds to implementation with power dissipation equal to the cost of cut, and since the Ford-Fulkerson algorithm cut results in a minimum cut, the decomposed implementation obtained by the procedure in Algorithm 2.9.1 is the implementation that has the minimum power dissipation under our approximations. Specifically, some of the inverters in the select logic can be relocated, and new inverters can be added or removed leading to inaccuracies in the capacitance estimation. However, these inaccuracies tend to be small since the area occupied by the select logic in the implementation is small as compared to the other combinational logic blocks. In this work, we have not taken into account area overhead due to node duplication and register duplication. This can be rectified, and a similar algorithmic framework can be applied to trade off area and power of the decomposed implementation.

2.10 Experimental Results

The above algorithm has been implemented as a C++ program. The BDD package CUDD [Som] is employed for generating BDD's, along with sifting [Rud93] for variable ordering for all our experiments. We assume the use of NMOS transistors as pass transistors and the insertion of inverters after every three pass transistors in series. The width and channel length for each transistor is assumed to be 0.5μm and 0.25μm, respectively. The capacitances of the transistors are measured employing parameters of TSMC 0.25μm CMOS process [Mos]. The primary inputs are assumed to have 50% probability of being at either 0 or 1 throughout the experiments, while the supply voltage and clock frequency is assumed to be 2.5V and 1GHz, respectively. We estimate the power dissipation in combinational logic for both the regular (undecomposed) and decomposed implementations of several MCNC benchmark circuits; the corresponding results are shown in Table 2.6. In the table, Columns 2 through 6 show the number of inputs/outputs, the power dissipation of a regular implementation, the power dissipation of decomposed implementation, the power reduction, and CPU time on Sun Ultra-60 machine, respectively. The CPU time includes the time for generation of BDD's, variable ordering, estimation, and decomposition. The various benchmarks used here include a variety of circuits, from arithmetic logic units to random logic. We observe significant power reductions in all of the cases, with an average reduction of 47.35%. We also observe that the power reduction is more significant in case of the circuits like ex4p, alu2, and 9symml, for which the number of outputs are relatively small as compared to the number of inputs. On the other hand, in case of 5xp1 and misex1 the reduction in power dissipation is relatively lower, and the number of outputs are relatively larger as

compared to the number of inputs. This correlation can be explained by observing that in case of circuits with a large number of outputs, the number of combinational logic blocks that may remain active in a given clock cycle is likely to be larger, resulting in a lower potential for power reduction.

Table 2.7 shows a comparison of our algorithms with previously proposed algorithms [LKTD01, TB99] for reducing power dissipation in PTL circuits. The experimental results reported in [LKTD01, TB99] are based on the same assumption of uniform probability of primary inputs, and both report switching activity reductions and not the actual power reductions. However, we assume that the switching activity reductions reported in [LKTD01, TB99] are translated to the same power reductions when the BDD's are mapped on to PTL. Column 2 in Table 2.7 shows the power reductions by our algorithm while Columns 3 and 4 show the power reductions obtained by the algorithms proposed in [LKTD01, TB99]. The '-' entry in Column 3 and Column 4 means that results are not available for the particular example in [LKTD01, TB99]. We observe that our algorithm performs better in all the cases. As compared to the average power reduction of 26.46% by algorithm in [TB99] over the first 8 benchmarks, our algorithm obtains 50.11% power reduction, on an average, over the same benchmarks. Over the last 3 benchmarks, our algorithm obtains power reduction of 33.6%, on an average, as compared to the average power reduction of 6.44% obtained by [LKTD01].

The results due to the algorithm in Section 2.9 are encouraging, since they show an average power reduction of 47.35% over a variety of MCNC benchmark circuits. The power reductions obtained by our algorithm, averaged over the MCNC benchmarks, are 23.65% and 27.16% higher than the power reductions obtained by previously proposed low power PTL synthesis algorithms [TB99] and [LKTD01], respectively. Therefore,

our algorithm can serve as a viable alternative for low power PTL synthesis. The same framework of the algorithm can be extended to consider area-power trade-offs in the decomposed implementations by considering the costs of duplication of nodes and registers.

2.11 Summary

We have presented efficient algorithms based on BDD decomposition for performance-driven and low power PTL synthesis. The BDD decomposition employs one-hot encoding of the BDD nodes in a cut, while the cut for the decomposition is determined by transforming bipartitioning of BDD's into the max-flow min-cut problem and then, finding the cut by employing Ford-Fulkerson algorithm. The algorithms result in optimal, up to the accuracy in estimation, solutions and have polynomial run-times, which, in practice, are in seconds for ISCAS'85 benchmarks.

In case of performance-driven synthesis, BDD decomposition is performed recursively till no delay improvements are possible. Using a simple delay metric such as the number of transistors in series for the recursive bipartitioning, our algorithm results in logarithmic depth PTL implementation; none of the previous synthesis heuristic guarantee this lower bound on PTL implementations. The experimental results on ISCAS'85 benchmarks show a 31% improvement in delay and 30% improvement in area, on an average, as compared to static CMOS implementations for *xor*-intensive circuits, while in case of arithmetic logic unit and control circuits that are *nand*-intensive, improvements over static CMOS are small and inconsistent. This points towards a static CMOS/PTL mixed synthesis; our performance-driven PTL synthesis algorithm can be thought of as

a step in that direction.

For low power PTL synthesis of combinational circuits whose inputs and outputs are registered, our algorithm finds an optimum cut that minimizes the power dissipation in the circuit using a similar algorithmic framework. The experimental results on a set of MCNC benchmarks show a significant improvement over the previous approaches.

Algorithm 2.5.1 Perform delay analysis on a given BDD.

Input: $G(V, E)$ = Graph underlying a given BDD, n = Number of variables in the BDD, k = Inverter interval, R = Pass transistor resistance

Output: $D_{Bottom} \forall v \in V$

1: **for** $level$ = 1 to $\lceil n/k \rceil$ **do**
2: $\forall v \in \{k \cdot level, k \cdot level - 1, ..., k \cdot level - (k-1)\}$, $\forall \overline{b} \in B^k$, $B = \{0, 1\}$
3: GetDownStreamCapacitance(v,$\overline{b}$);
4: In topological order, $\forall v \in V$
5: $v.D_{Bottom} \leftarrow$ max$\{v'.D_{Bottom}$ + R $\times$ v.DownstreamCap $\mid (v', v) \in E\}$
6: **end for**
7:
8: Procedure GetDownStreamCapacitance(v,$\overline{b}$)$\{$
9: **if** (AllFanoutEdgesBuffered(v)) **then**
10: v.DownstreamCap $\leftarrow$ v.Capacitance
11: **else**
12: $c \leftarrow 0$
13: **for** $\forall e(v, v')$ $\in$ fanout(v) **do**
14: **if** (e is not buffered && $\overline{b} \in$ BooleanExpression(e)) **then**
15: $c \leftarrow c$ + GetDownStreamCapacitance(v',$\overline{b}$);
16: **end if**
17: **end for**
18: $c \leftarrow c$ + v.Capacitance
19: **if** ($c > v$.DownStreamCap) **then**
20: v.DownStreamCap $\leftarrow$ c
21: **end if**
22: **end if**
23: $\}$

Algorithm 2.9.1 Find an optimum cut to reduce power dissipation

Input: $G(V, E)$ = The graph underlying the BDD

Output: S_{cut} = Optimum cut-set

1: Perform power estimation on G, i.e., compute P for $v \in V$

2: Estimate select probabilities, i.e., assign $p_{selelct}$ to $v \in V$.

3: Compute power dissipation in the select logic, i.e., assign P_{select} to $v \in V$.

4: **for** $v \in V$ **do**

5: v.Cost $\leftarrow v.P_{select}$+ $v.p_{select} \times v.P$

6: **end for**

7: $G_{Flow} \leftarrow$ CreateFlowNetwork($G(V, E)$)

8: Ford-Fulkerson(G_{Flow}, G, S_{cut})

Example	# of	Regular	Decomposed	Reduction	CPU time
	I/O	Power(mWatt)	Power(mWatt)	(%)	Seconds
5xp1	7/10	0.29	0.21	27.36	0.29
9symml	9/1	0.12	0.05	52.45	0.09
alu2	10/6	1.48	.447	69.9	5.3
alu4	14/8	4.14	3.48	15.86	45
c8	28/18	0.21	0.14	35.0	0.54
cm162a	14/5	0.15	0.07	51.26	0.1
comp	32/3	0.90	0.41	53.76	1.59
cordic	23/2	0.28	0.14	50.52	1.6
ex4p	128/28	3.72	1.23	66.78	19.28
f51m	8/8	0.35	0.18	47.86	0.48
i8	133/81	4.11	1.97	51.95	69.99
inc	7/9	0.39	0.18	52.63	0.42
misex1	8/7	0.16	0.1	35.52	0.11
parity	16/1	0.21	0.1	51.59	0.17
rd53	5/3	0.11	0.05	54.64	0.06
rd73	7/3	0.22	0.1	51.81	0.15
rd84	8/4	0.31	0.14	54.11	0.28
t481	16/1	0.16	0.09	38.12	0.7
z4ml	7/4	0.135	0.82	39.25	0.07
Average				47.35	

Table 2.6: Comparison of regular implementation with our decomposition-based implementation.

Example	Power Reduction(%)		
	Our	[LKTD01]	[TB99]
9symml	52.45	-	18.4
alu2	69.9	-	43.0
cm162a	51.26	-	39.4
cordic	50.5	-	44.6
f51m	47.85	-	32.6
parity	51.59	-	4.00
t481	38.13	-	10.2
z4ml	39.25	-	19.50
5xp1	27.36	6.25	-
duke2	20.83	13.08	-
inc	52.63	0	-
Average	45.61	6.44	26.46

Table 2.7: Comparison of our decomposition-based implementation with the methods of Tavares *et al.* [TB99] and Lindgren *et al.* [LKTD01].

Chapter 3

Transistor-level Layout Generation for Pass Transistor Logic Circuits

3.1 Introduction

In the previous chapter, we saw that pass transistor logic (PTL) can result in better implementations in case of *xor*-intensive circuits and also some control designs. In spite of the potential gains in performance due to PTL, its usage has been limited due to the unavailability of good PTL-specific layout tools. We address this problem by proposing an automatic layout generator for libraryless PTL circuits.

3.1.1 Previous Work

Although logic synthesis for PTL circuits targeting area, power, and delay optimization has been studied by numerous researchers [BNNSV97, FMM+98, SS01b], the layout generation problem for PTL circuits has received relatively less attention. In [MBM01],

Macii *et al.* propose a layout generator for PTL circuits that is suitable for standard cell layouts. The limitation of their approach is that they do not fully exploit diffusion-sharing between the PTL multiplexers, which can potentially save a large amount of area. The PTL layout work by Yano *et al.* [SRY98] does utilize diffusion-sharing using the idea of Eulerian trails in PTL multiplexers, but its applicability is limited to small cells, typically up to 4-5 inputs. The limitation to small cells is justified by the fact that buffers must be inserted after every few pass transistors, and that blocks of pure pass transistor logic that lies between buffers, which correspond to cells in their work, tend to have a small number of inputs. This approach corresponds to traditional cell-based method that generates layouts for individual PTL cells and assembles these by placing blocks of cells in rows in accordance with a standard cell layout methodology.

Due to the well known relationship between PTL and binary decision diagrams (BDD's), we use the BDD representation of a logic function as the input to our approach. Our approach consists of three steps: (1) assigning BDD nodes, which represent PTL multiplexers, to rows in the layout, through a max-flow min-cut based recursive bipartitioning technique followed by a greedy assignment, aimed at minimizing the number of wires going from one row to another, (2) forming diffusion-sharing clusters for PTL multiplexers in the same row using an Eulerian trail approach, allowing both horizontal and vertical placement of transistors to minimize the area, and (3) placing these diffusion-sharing clusters optimally within a row employing a linear tree placement algorithm [Yan85] to minimize the wiring overhead for connections within the same row. Thus, our approach minimizes the width of the layout by maximizing diffusion-sharing and the height by row assignment and by minimizing the routing overhead for inter-row and intra-row connections.

3.1.2 Our Contributions

Our layout generation algorithm, which takes flexible view of cell boundaries and tries to maximize the diffusion-sharing, has advantages over both [SRY98, MBM01] the previously published approaches. Our contributions can be summarized as follows.

- Unlike [SRY98], our approach takes a more flexible view of the boundaries between cells. While the layouts generated by our algorithm fit into the outline of a standard cell layout methodology, we do not limit individual cells to have rectangular boundaries, and permit a more fluid boundary between individual cells of pure pass transistor logic, and integrate the layout of inverting buffers into the layout of the pass transistor logic block. In doing so, we allow greater flexibility in layout and perform the layout for an entire block of PTL, including regenerating inverters, instead of working with one cell at a time with only a few inputs.

- Related work that also uses a flexible boundaries between the cells is proposed recently by Gopalakrishnan *et al.*, but in the context of static CMOS cells [GR01]. Our method differs from their approach in the application of linear tree placement, bipartitioning, and row assignment as well as in the cluster formations. We assign groups of transistors to rows and minimize the layout area by maximizing diffusion-sharing, minimizing the wiring area required to route the intra-row signals, and minimizing the number of inter-row signals.

The organization of the rest of the chapter is as follows. In Section 3.2, we illustrate a layout model for PTL circuits, followed by a description in Section 3.3 of diffusion-sharing in PTL circuits based on the adjacency relation between the BDD nodes. Section 3.4 provides a detailed description of various steps in the algorithm, and Section 3.5

presents the experimental results and conclusions, while Section 3.6 summarizes the chapter.

3.2 Layout Model

Figure 3.1(a) shows a PTL multiplexer represented by a BDD[1] node; a Boolean function f is represented by $f = Af_A + \overline{A}f_{\overline{A}}$, where f_A and $f_{\overline{A}}$ are the Shannon co-factors of the function f with respect to the variable A. As pointed out earlier in Section 2.3.1, this BDD node is well known to map on to a multiplexer, and Figure 3.1(b) shows a layout for this multiplexer that is made compact by allowing the drains of two multiplexer transistors share the diffusion. Empirically, it has been observed in [SRY98] that exploiting diffusion-sharing at the output of a multiplexer produces a good quality layouts for small cells, and also reduces the search space of a layout generator. Therefore, our work also employs the approach of maximizing diffusion-sharing. Unlike previous work on PTL layout that has largely required transistors to be laid out either all horizontally or all vertically, we allow a mix of the two and exploit this flexibility to maximize diffusion-sharing among transistors.

A typical pass transistor logic circuit contains a set of multiplexers, normally implemented using NMOS pass transistors, and buffers that are inserted after every three pass transistors in series to boost the signals. These buffers are usually implemented as inverters that are positioned so as to ensure the correct parity of signals within the network. Our layout model places the inverters and PTL multiplexers in rows of fixed height, as shown in Figure 3.2. The figure shows two rows of logic with an interven-

[1]For the purposes of this chapter, all BDD's are reduced ordered BDD's (ROBDD's).

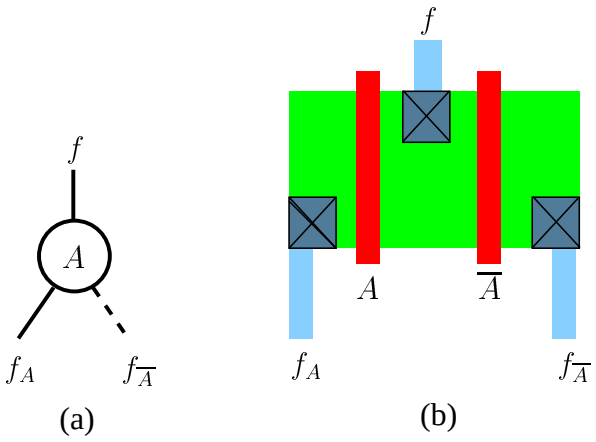


Figure 3.1: Layout of a multiplexer: (a) A BDD node. (b) Its corresponding layout.

ing channel for routing, with each row containing PTL multiplexers and inverters. The lower row shows a detailed view of the inside of some typical blocks, illustrating the layout of a 3-input NAND gate employing PTL multiplexers and an inverter. Each row has the same height, and rows are separated by a space for routing both intra-row and inter-row signals. We attempt to minimize this area by performing optimal linear tree placement

The proposed layout model can be extended easily to accommodate static CMOS cells due to the placement of power supply and ground lines for each row; however, the mixed synthesis of PTL and static CMOS is not addressed here.

3.3 Diffusion-sharing in PTL Circuits

The concept of Eulerian trails has often been used for diffusion-sharing optimizations in static CMOS standard cells. Static CMOS cells consist of a series-parallel network of transistors, and every Eulerian trail represents a transistor ordering that permits

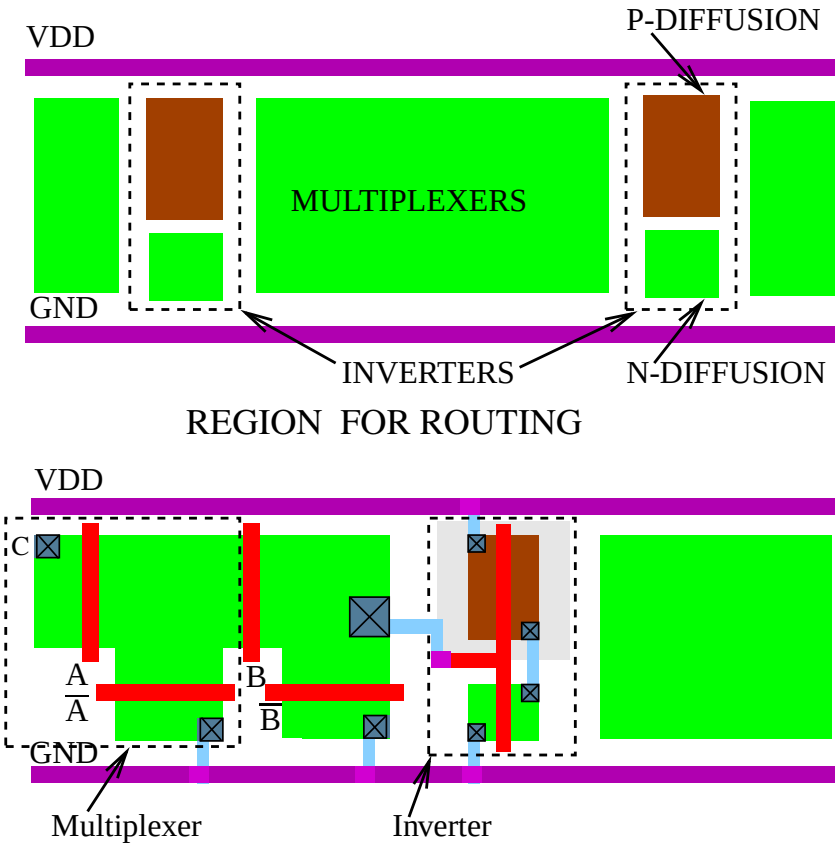
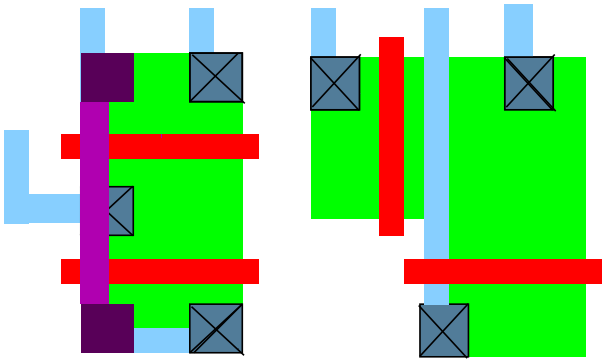


Figure 3.2: A row-based layout scheme for PTL.

diffusion-sharing between series-connected transistors. In our formulation, we use this notion of Eulerian trails, tailored specially for PTL layout.

The manner in which diffusion can be shared is dependent on the shapes of layouts and the sizes of the transistors. Shape level optimizations include making a choice between laying out the transistors in a multiplexers either horizontally as shown in Figure 3.1(b), or vertically, as shown in Figure 3.3(a), or using a hybrid of the two, as shown in Figure 3.3(b). In terms of sizing, we consider only uniformly sized transistors in this work. However, the layout of arbitrarily sized transistors can be handled using our framework by considering transistor folding in case of large sized transistors and

weighting the BDD nodes corresponding to such transistors with an appropriately large area cost during bipartitioning and row assignment. Subsequently, shape level optimizations and diffusion-sharing may be used to handle arbitrary transistor sizes. We do not, however, consider this issue in our current work.



(a) (b)

Figure 3.3: Different multiplexers layout schemes: (a) With vertical transistors. (b) With horizontal as well as vertical transistors.

In case of PTL circuits synthesized from BDD's, we observe that diffusion-sharing between any two nodes may be possible in only the following two cases:

Input diffusion-sharing Two nodes share the same co-factor, as shown in Figure 3.4(a).

Output diffusion-sharing One node is a co-factor of the other node, as shown in Figure 3.5(a).

The layouts for each case are shown in Figures 3.4(b) and 3.5(b), respectively. Note that in Figure 3.5(b), the transistor associated with input $\overline{B}$ is laid out vertically so that it shares the diffusion at the output of the multiplexer with the transistor associated with

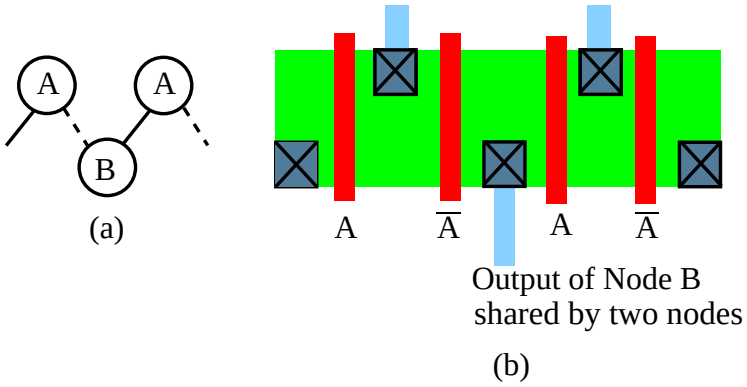


Figure 3.4: An example of input diffusion-sharing: (a) A BDD. (b) Its corresponding PTL implementation.

input B, and also with the transistor associated with input A of the adjacent multiplexer. If two nodes share both cofactors, as shown in Figure 3.6(a), then diffusion-sharing can be obtained as shown in Figure 3.6(b).

Our approach for maximizing diffusion-sharing between transistors involves finding Eulerian trails through a walk on the input or output edges of the nodes, as explained in the next section.

3.4 Algorithm for Layout Generation

Figure 3.7 outlines the overall algorithm for layout of PTL circuits. The algorithm begins with a multilevel BDD network[2] of the circuit and uses a recursive bipartitioning scheme to assign the BDD's corresponding to various regions to different parts of the layout. This is followed by greedy assignment of the rows to the BDD nodes such that

[2]A multilevel BDD network [YC99] is similar to a multilevel Boolean network. It uses a BDD to describe the functionality at each node in the network. It may be recalled that in the previous chapter, multilevel BDD's are employed for performance-driven synthesis.

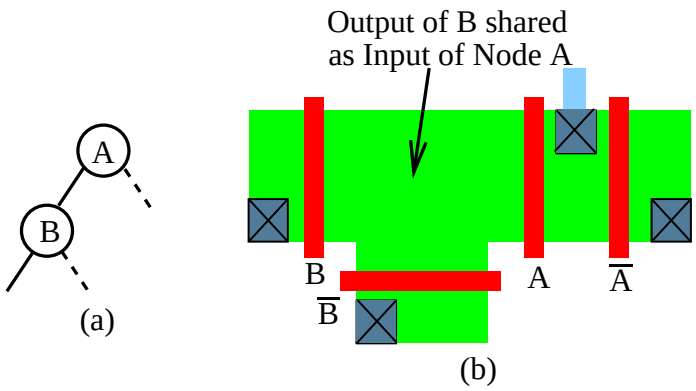


Figure 3.5: An example of output diffusion-sharing: (a) A BDD. (b) Its corresponding PTL implementation.

the number of signals across different rows is minimized, and each row has approximately the same number of nodes and inverters. This bipartitioning approach can be interpreted as performing a coarse layout region assignment, similar to the use of partitioning in placement problems, and can be formulated as a max-flow min-cut problem. The greedy procedure assigns nodes in each BDD in the multilevel representation to rows and takes into account possibilities of diffusion-sharing. The assignment is performed in such a way that the number of wires from any row to another row is minimized. After the row assignment is completed, the multiplexers have been assigned to rows, but their positions within a row have not been finalized. To do so, we first cluster nodes in each row using an Eulerian trail approach to maximize diffusion-sharing. Once the clusters have been formed, they are assigned specific positions in each row; this is performed employing a linear tree placement algorithm proposed by Yannakakis [Yan85]. Figure 3.8 summarizes the features of our approach for minimizing the area of the layout. We minimize the width of each row by diffusion-sharing using Eulerian trails; the height of the row is fixed. The space for routing is minimized by optimizing the num-

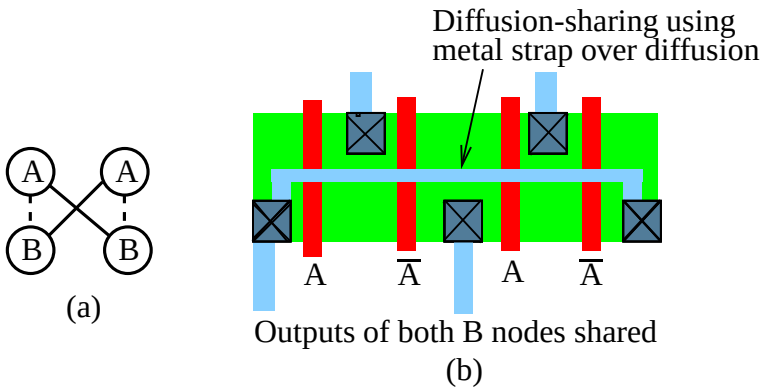


Figure 3.6: A diffusion-sharing scheme for the case when two cofactors are shared: (a) A BDD. (b) Its corresponding PTL implementation.

ber of inter-row signals; this is addressed using min-cut recursive bipartitioning for the signals between different BDD's and by greedy row assignment for the inter-row signals among the nodes of the same BDD. We minimize the number of rows for routing intra-row signals by employing linear tree placement for the diffusion-sharing clusters; Yannakakis's algorithm ensures optimal placement for trees.

The time complexity of recursive bipartitioning using max-flow min-cut technique is $O(\|V\| \cdot \log \|V\| \cdot \|E\|^2)$ since the complexity of Ford-Fulkerson algorithm is $O(\|V\| \cdot \|E\|^2)$ [CLR98], where V and E are set of vertices and edges in the multi-level BDD network, respectively. The time complexity of the algorithm for row assignment is also polynomial in the number of nodes in individual BDD's. The time complexity of Eulerian trail algorithm is linear in number of edges of the nodes that are assigned the same row while the complexity of linear tree placement algorithm is $O(N \cdot \log N)$, where N is the number of vertices corresponding to diffusion-sharing clusters in a tree. Therefore, time complexity of overall algorithm is polynomial.

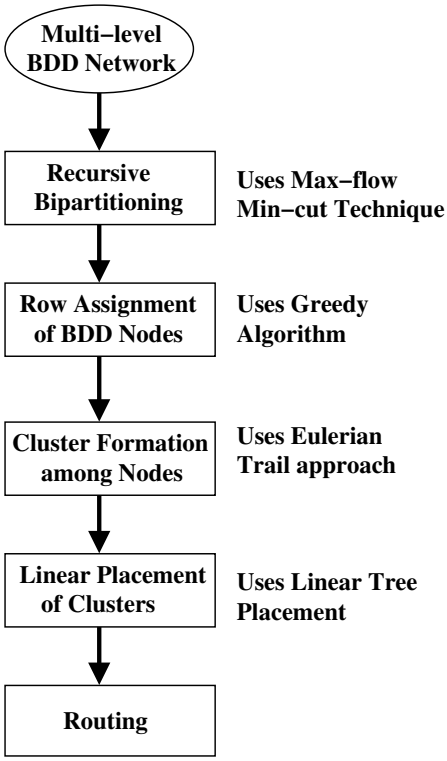


Figure 3.7: An overview of the algorithm.

3.4.1 Recursive Bipartitioning

The multilevel BDD network can be treated as a directed graph, where each vertex corresponds to a BDD and a directed edge means that output of a vertex represented by a BDD is an input to the BDD of another vertex; the edges are assigned directions leaving from primary inputs and pointing towards primary outputs. Our aim is to find a cut that partitions a given multilevel BDD network into approximately equal parts in terms of the area occupied by the transistors (both in multiplexers and in inverters), and has minimum connectivity among the partitions. To obtain such an area-balanced partition, we associate two costs with each node in the multilevel BDD network that are defined as follows.

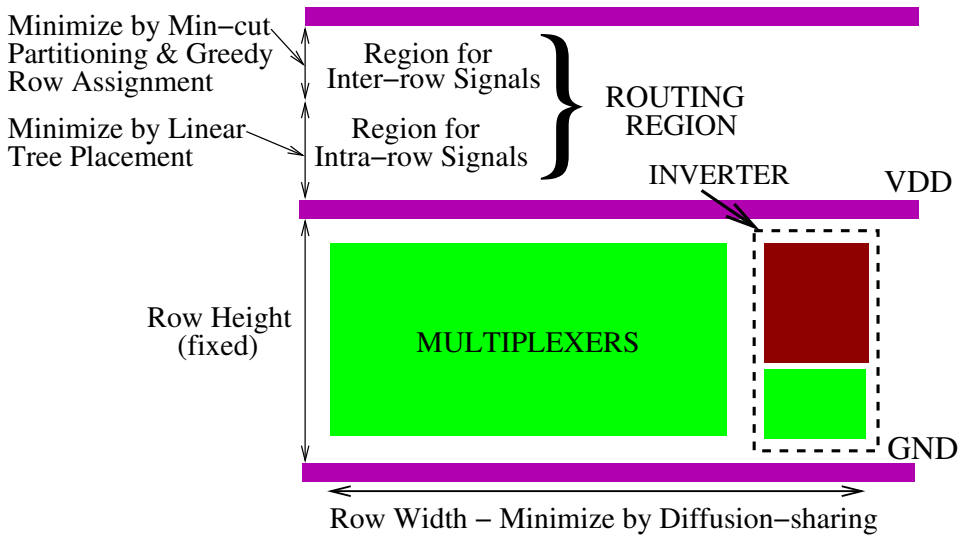


Figure 3.8: A pictorial view of area minimization strategies for the layout.

Lower Cost (A_{lower}) : This is the cost of a PTL implementation of the multilevel BDD network rooted at a given node.

Upper Cost (A_{upper}) : This is the cost of a PTL implementation of the multilevel BDD network rooted at a given node, assuming the directions of edges reversed.

The costs A_{lower} and A_{upper} can be obtained by a post-order traversal of the multilevel BDD network. The upper and lower costs reflect the contribution of a given node to the total area of a PTL implementation in the part of the multilevel BDD network towards the primary outputs and in the part of the multilevel BDD network towards primary inputs, respectively. A node that has almost the same upper and lower costs is a good candidate for inclusion in the cut since it will result in an area-balanced partition, and using all such nodes, we want to find a min-cut partition. We therefore define candidate nodes for the cut as follows.

Candidate Nodes are the nodes for which $|A_{lower} - A_{upper}| \leq \Delta$, where Δ indicates

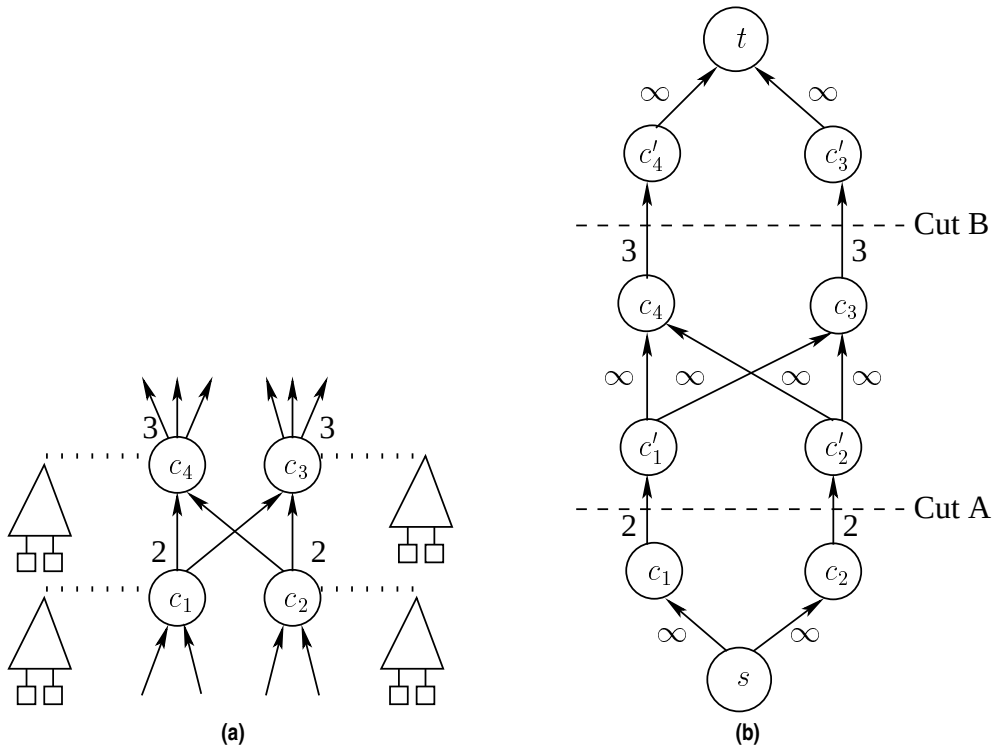


Figure 3.9: Recursive bipartitioning: (a) A multilevel BDD network. (b) Its corresponding flow network.

flexibility while choosing candidate nodes.

Although the idea of recursive bipartitioning proposed here is similar to that presented in the previous chapter, it differs from the latter in the objective as well as the application. The objective here is to generate area-balanced partitions that minimize the connectivity, while the method in the previous chapter aims to find delay-balanced partitions. Moreover, we apply recursive bipartitioning to a multilevel BDD network, instead of to individual BDD's.

We now try to find an optimal vertex cut across these candidate cut nodes with the intention of assigning approximately half of the multilevel BDD network, in terms of

area, to each side of the partition, while also ensuring that there is minimum connectivity between the partitions. To ensure the minimum connectivity, we transform the bipartitioning problem to the max-flow min-cut problem in which each candidate node is assigned a flow capacity equal to its number of fanouts. Figure 3.9 shows a part of multilevel BDD network containing candidate nodes c_1, c_2, c_3, and c_4, and its corresponding flow network with a source node s and a destination node t. To form a flow network for the minimum vertex cut, each candidate node is split into two nodes connected by an edge that is assigned a capacity equal to the number of fanouts. For instance, the capacity of the edge between c_1 and c_1' in Figure 3.9(b) is 2 because the number of fanouts of the node c_1 in the network of candidate nodes is 2 in Figure 3.9(a). Moreover, node c_1' is connected to node c_3 and c_4 in the flow network, since node c_1 is connected to the corresponding nodes in the network of candidate nodes. The edges that are assigned a capacity of ∞ are guaranteed not to appear in the cut. Figure 3.9(b) shows two possible cuts: Cut A with a capacity of 4, and Cut B with a capacity of 6. The application of the Ford-Fulkerson algorithm [CLR98] finds the min-cut, which in this case is Cut A. Therefore, the corresponding nodes that are chosen for partitioning are c_1 and c_2. In the resulting partition, nodes c_1, c_2, and their predecessors will be in one subset, while nodes c_3, c_4, and their descendents will be in another subset. Recursive bipartitioning of the multilevel BDD network on these subsets continues until a trivial network containing a single vertex corresponding to a single BDD of the multilevel network is obtained.

3.4.2 Greedy Heuristic for Row Assignment

After recursive bipartitioning of the multilevel BDD's, the BDD's are assigned an ordering for layout. To generate a compact layout for each BDD, we must assign the nodes in

each BDD to rows such that the diffusion-sharing among the nodes in the same row is maximized. We use a greedy method for this purpose, as described in Algorithm 3.4.1.

Algorithm 3.4.1 Perform row assignment for nodes in a BDD

Input: $G(V, E)$ = The graph underlying the BDD

Output: Mapping $f : V \rightarrow \{0, 1, \cdots, row_{max}\}$

1: $V_{left} \leftarrow V$
2: $row \leftarrow 0$
3: **while** $V_{left} \neq \Phi$ **do**
4: $\quad Area \leftarrow 0$
5: $\quad S_{row} \leftarrow \Phi$
6: $\quad$ **while** $Area < MaximumArea$ **do**
7: $\quad\quad v \leftarrow$ MostAdjacentNode(S_{Row},G,V_{left})
8: $\quad\quad S_{Row} \leftarrow S_{Row} \bigcup v$
9: $\quad\quad V_{left} \leftarrow V_{left}$ - v
10: $\quad\quad Area \leftarrow$ AreaEstimation(S_{Row})
11: $\quad$ **end while**
12: $\quad$ AssignRow(S_{Row}, row)
13: $\quad row \leftarrow row + 1$
14: **end while**

The set of multiplexer nodes assigned to each row is first initialized to the empty set. Subsequently, the algorithm successively adds nodes to the set as long as a user-defined area capacity of the row is not exceeded; this area capacity can be used to control the width of the layout. In each step, the node (found by the MostAdjacentNode() routine) that is greedily added to the set is the one that is adjacent to the maximum

number of nodes that are already in the set; any ties are broken in favor of a node with a lower variable index in the variable ordering for the ROBDD. This strategy serves two purposes:

1. It maximizes the possibility of diffusion-sharing among nodes in a row; more diffusion-sharing implies compact layouts and a likelihood of assigning more nodes to the same row.

2. It minimizes the number of edges going across different rows, and therefore, the routing area required for inter-row signals is also minimized.

This greedy algorithm implicitly minimizes the row area, since it considers the effects of diffusion-sharing. A node that is to be added to the set is chosen depending on the adjacency to the nodes already in the set, and adjacency implies possible diffusion-sharing. The following proposition states the complexity of the algorithm.

Proposition 3.4.1 *The procedure in Algorithm 3.4.1 terminates in* $O(N \cdot \|S_{Row}\| \cdot d_{max})$ *time, where* N *is the total number of nodes,* $\|S_{Row}\|$ *is the cardinality of the row and* d_{max} *is the maximum degree among all nodes.*

Proof 3.4.1 The FindMostAdjacentNode routine requires $O(\|S_{Row}\| \cdot d_{max})$ time. The routine AreaEstimation is completed in constant time, since it is incremental in nature. Therefore, innermost while loop takes $O(\|S_{Row}\|^2 \cdot d_{max})$ time, and since it is repeated $O(N/\|S_{Row}\|)$ times, the result follows.

3.4.3 Formation of Diffusion-sharing Clusters

The width of each row can be minimized by diffusion-sharing among different multiplexers placed in the same row. To form diffusion clusters, i.e., clusters of transistors that

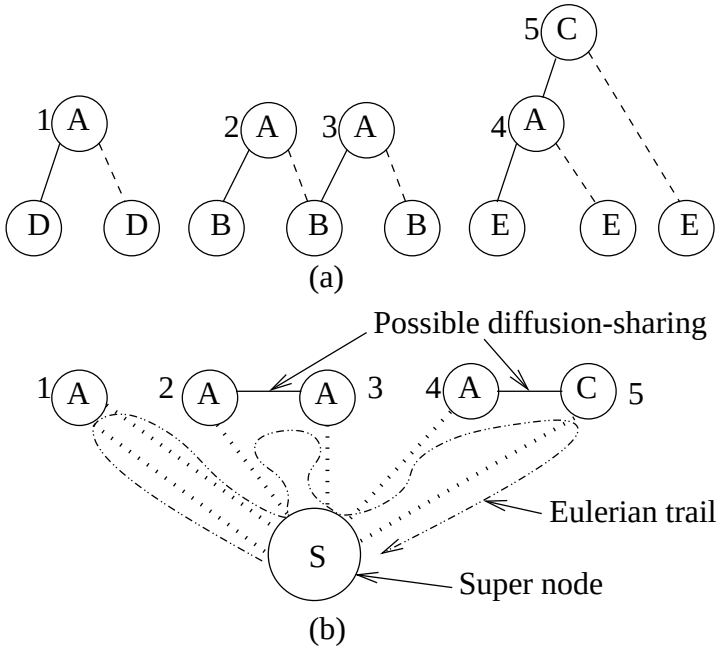


Figure 3.10: Cluster formation: (a) A group of BDD nodes to be placed. (b) The corresponding Eulerian graph. Dotted edges in (b) denote possible diffusion breaks.

can share diffusion, we use the Eulerian trail algorithm [PS98] after Eulerizing a given graph by adding a super node and connecting nodes with an odd[3] degree to it. This Eulerian trail algorithm is used in [RS99] for two-dimensional micro-cell placement in the context of CMOS cells. The nodes in our Eulerized graph for a row correspond to BDD nodes assigned to the row, and the edge set consists of two types of edges: ones that denote possible diffusion-sharing and the others that denote diffusion breaks. As explained in Section 3.3, in case of PTL, diffusion-sharing can occur in case of input sharing, illustrated in Figure 3.4, or output sharing, as shown in Figure 3.5. To capture this relationship, we introduce edges between the corresponding nodes in the Eulerized

[3]The nodes with 0 degree are also connected to the super node, and this does not affect the number of diffusion breaks.

graph. Dummy edges indicating diffusion breaks connect the odd degree nodes to a super node. Figure 3.10(a) shows an example with five BDD nodes, corresponding to variables A and C, to be placed in a row; we assume that nodes with labels B, D, and E are already placed in previous rows. The corresponding Eulerized graph is shown in Figure 3.10(b), and the cluster formations associated with the Eulerian trail in the figure are shown in Figure 3.11. Three clusters are identified: one with one multiplexer and two with two multiplexers each.

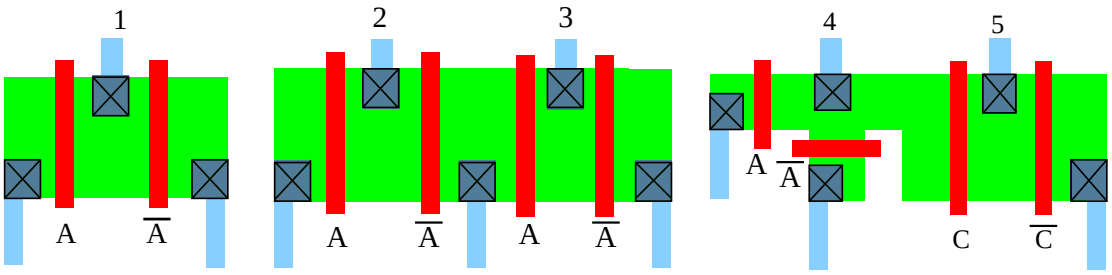


Figure 3.11: Diffusion-sharing clusters corresponding to Figure 3.10.

3.4.4 Linear Placement

Although there are potentially exponentially many Eulerian trails in the Eulerian graph, it is known that every trail yields a solution with the same minimum width corresponding to the same number of diffusion breaks. However, the routing cost varies for different trails, corresponding to different arrangements of the diffusion-sharing clusters. To optimize this cost, we perform linear tree placement of clusters while preserving the order of the multiplexers in each cluster. Since the multiplexer order in each cluster is preserved, altering the order of the clusters will not affect the row width. This step can also be thought of as a generating an Eulerian trail that has better routability than the previous

Eulerian trail. The objective here is to minimize the cost involved in routing the signals that connect different clusters in the same row.

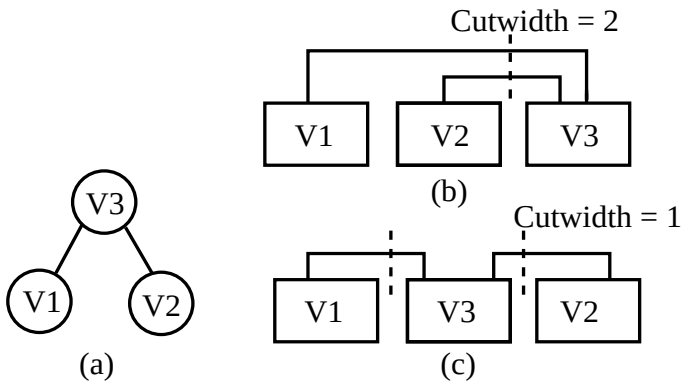


Figure 3.12: Linear placement for laying out the clusters: (a) A cluster tree. (b) A sub-optimal placement. (c) An optimal placement.

We define a graph, called a cluster graph, $G(V, E)$ on clusters, in which every vertex $v \in V$ corresponds to a cluster, and there is an edge between the vertices if there are signals connecting the corresponding clusters. If $G(V, E)$ is a tree, then application of Yannakakis' algorithm [Yan85] results in an optimal placement that minimizes the cut-width; here the cut-width corresponds to the number of rows required to route the signals between different clusters. Yannakakis' algorithm uses dynamic programming to build the solution in a bottom up manner and runs in $O(N \times \log N)$ time, where N is the number of vertices. If the cluster graph is not a tree, then we arbitrarily remove some of the edges to make it a tree; in this case, however, the optimality of solution is not ensured. In practice, it is seen that for BDD's, the cluster graph is typically close to a tree. To understand this, we observe that a cycle can appear in a cluster graph, if there are forward edges in the BDD that connect a node to its ancestor, and the node at either end of the the forward edge and the parent of the node are placed in the different clusters

of the same row. However, such a structure is likely to result in a great deal of diffusion-sharing, and this increases the probability that all such structures are placed within the same cluster. Therefore, cycles between clusters, while not impossible, are improbable. We verified the same from experiments on the benchmark examples. Figure 3.12(a) shows a cluster tree containing three clusters V1, V2, and V3, with connections between V1 and V3, and between V2 and V3. Figure 3.12(b) shows a suboptimal placement of these clusters that results in a cut-width of two, implying that number of rows required to route the signals will be two, if we use two metal layers. Figure 3.12(c) shows an optimal placement with cut-width of 1 that will be found by Yannakakis' algorithm, implying that one row suffices to route the signals between the clusters. This example shows the impact of linear tree placement and its contribution to minimizing the area.

3.5 Experimental Results

We have implemented the algorithms discussed in this chapter in a C++ program that generates layouts for PTL circuits from BDD's; the CUDD package [Som] is employed to construct multilevel BDD's. Inverters are inserted after every three pass transistors in series; we utilize a heuristic for inverter insertion, since the general problem of inverter insertion in PTL circuits is shown to be NP-hard [ZA98]. The technology that we use is scalable CMOS technology with one poly and three metal layers. Poly is employed for routing inputs; Metal1 is utilized for routing intra-row signals; and Metal2 and Metal3 are used for routing inter-row signals. Metal1 is allowed to route in the vertical as well as the horizontal directions, while Metal2 is employed for routing mostly in the vertical direction, and Metal3 is used to route only in the horizontal direction.

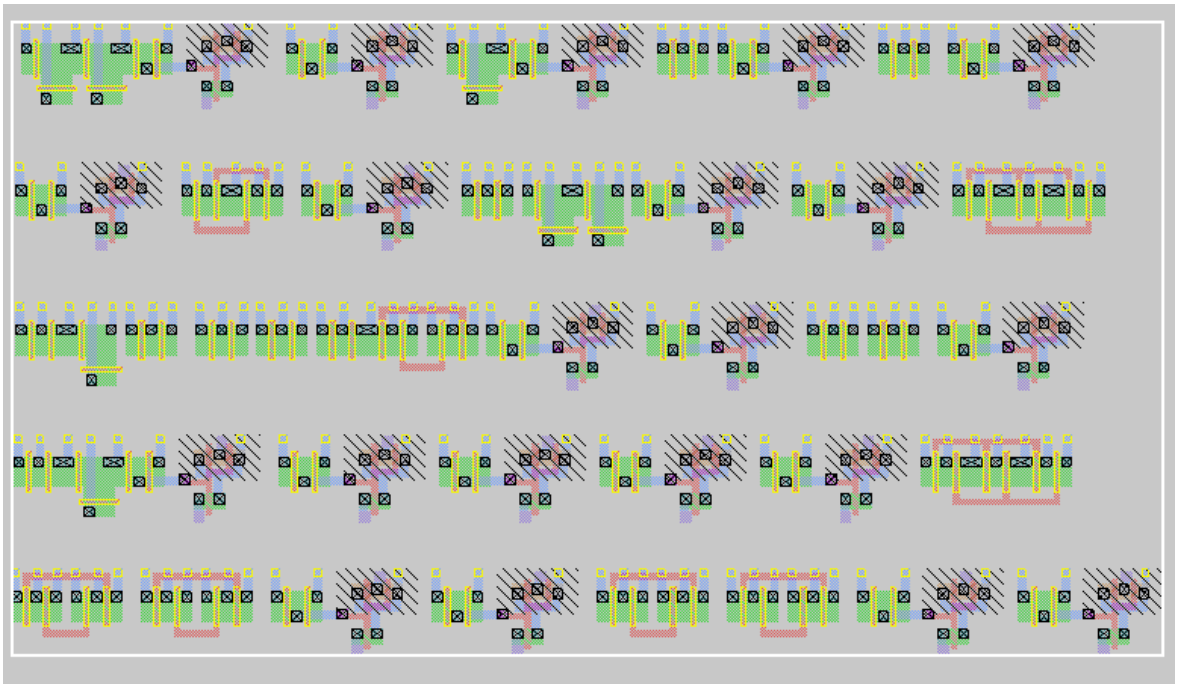


Figure 3.13: Effect of row assignment, clustering, and linear tree placement on rd84. The figure shows 122 pass transistors and 21 inverters.

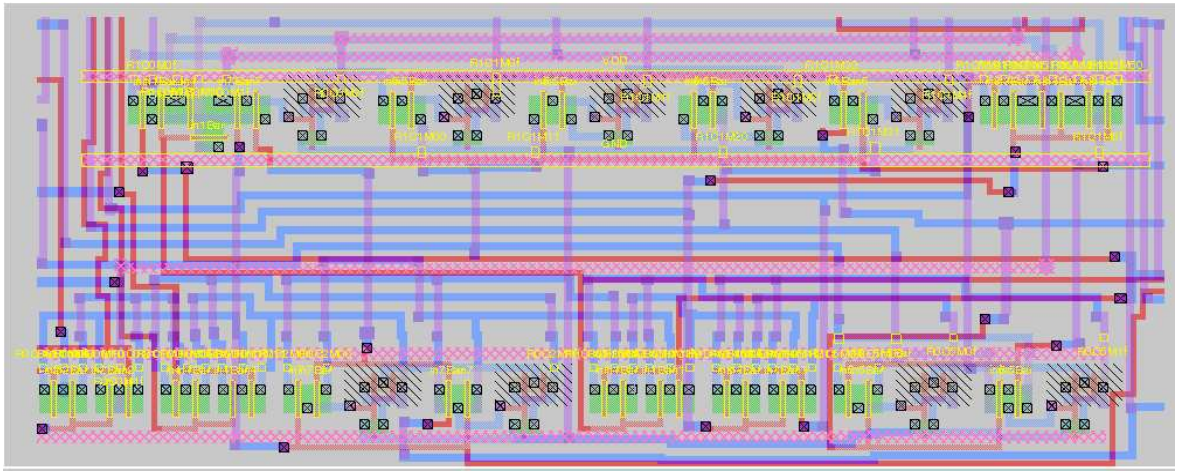


Figure 3.14: Intra- and inter-row routing for rows 1 and 2 in rd84 circuit.

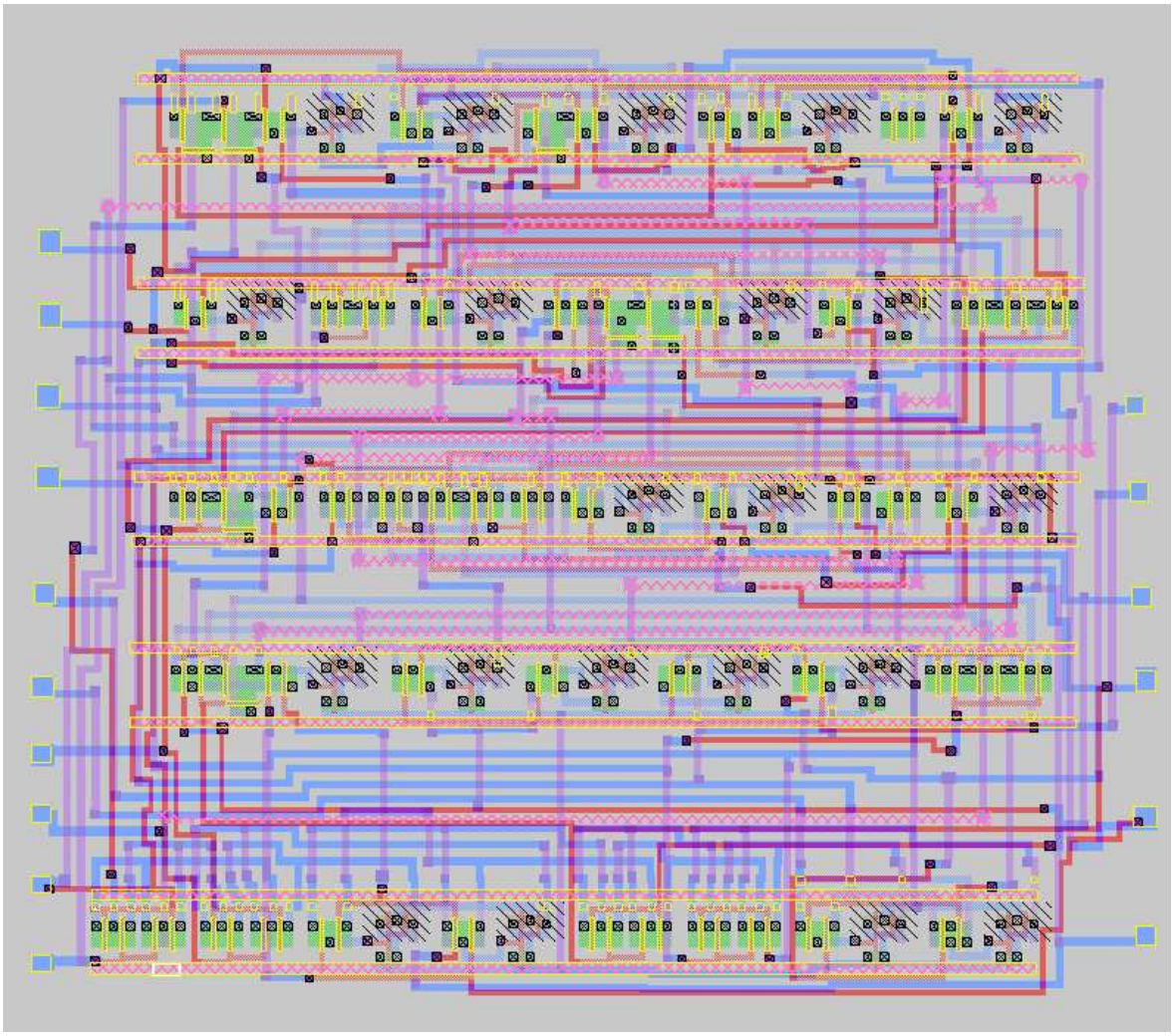


Figure 3.15: Post-routing layout for rd84 circuit.

Figure 3.13 shows the placement of multiplexer clusters and inverters in five rows for an MCNC benchmark rd84, while Figures 3.14 and 3.15 show the intra- and inter-row wires for the first two rows and a picture of the complete layout, respectively, after routing employing a router[4] in MAGIC [Ous]. The benchmark rd84 has 8 inputs and 4 outputs with a BDD representation that has a size of 63 nodes, while its transistor-level implementation includes 122 transistors apart from 21 inverters. Figure 3.13 clearly shows the effect of greedy row assignment, which results in area-balanced rows with high amount of diffusion-sharing among each row. Moreover, it is quite clear that the flexibility of placing transistors either horizontally or vertically serves to reduce the area and create a compact layout. All the five rows show the effect of diffusion-sharing and its contribution to the width minimization.

Table 3.1 shows the experimental results on ISCAS'85 benchmarks. Column 2 shows layout area for static CMOS circuits, while Column 3 shows the the layout area for PTL circuits due to our algorithm. The static CMOS circuits are obtained by technology mapping for area minimization employing lib2.genlib library in SIS [Sen92]; the library is characterized for 130nm technology [ptm]. The numbers in Column 2 correspond to the gate-area, and are, therefore, lower bounds on actual area. In case of PTL circuits, multi-level BDD's are directly mapped on to PTL employing inverter insertion heuristic proposed in [MBIS01], and transistors are laid out using our algorithm. To perform post-placement area comparison, we employ the left-edge algorithm for estimating the channels required to route the intra-row and inter-row signals. More sophisticated routing algorithms such as those employing dog-legs, or the application of a commercial router will, of course, result in smaller number of routing channels and therefore, greater

[4]Our tool, in its current form, does not perform routing, but it produces the netlists that can be used by publicly available tools such as MAGIC or commercial routers.

Example	Static CMOS	Ours
	Area (μm^2)	Area (μm^2)
C1355 (Error correcting codes)	4631	1825 (-60%)
C1908 (Error correcting codes)	4867	1924 (-60%)
C2670 (ALU and Control)	10930	11606 (+6%)
C3540 (ALU and Control)	11247	9040 (-19)
C432 (Priority Decoder)	3007	4303 (+43%)
C499 (Error correcting codes)	4611	1123 (-75%)
C5315 (ALU and Selector)	18065	20554 (+13%)
C6288 (16-bit Multiplier)	27947	13600 (-51%)
C7552 (ALU and Control)	28817	12655 (-56%)
C880 (ALU and Control)	3796	2729 (-28%)
Total	117918	79359 (-32%)

Table 3.1: Comparison of layout area for ISCAS'85 benchmark circuits.

area reduction. Our results with the left-edge algorithm for routing are, therefore, upper bounds on the area. We see that in case of *xor*-intensive circuits, such as C1355, C1908, C499, and C6288, PTL results in 61% reduction in area, while area improvements are not consistent for ALU and control circuits that are *nand*-intensive. On an average, area of PTL circuits that are laid out using our algorithm has been 32% smaller than the corresponding gate-areas of static CMOS circuits.

3.6 Summary

In this chapter, we have described an algorithm for transistor-level placement of PTL circuits. The algorithm is useful for translating the libraryless PTL circuits into layouts. The lack of any such tools has been one of the reasons for the limited usage of library-less PTL circuits despite their advantages. The layouts generated by the algorithm are amenable to static CMOS/PTL mixed synthesis, since they can fit in to standard cell methodology. It has advantages over previously published approaches, and comparison with the static CMOS circuits for ISCAS'85 benchmarks show a large improvement in area, especially in case of *xor*-intensive circuits.

Chapter 4

Congestion-aware Technology Mapping

Interconnect dominance is a daunting issue for sub-100nm VLSI designs. This is a consequence of the rising design complexity: following Moore's law [Moo65], the number of on-chip transistors are doubling every twenty four months, while, according to Rent's rule [LR71, CS00], the number of wires also grows exponentially with the number of gates. As a result, even today's designs have regions where the unavailability of sufficient number of tracks to route the wires causes the circuits either to be unroutable or to violate the timing constraints due to long wire detours. This is often referred to as the routing congestion problem. In this chapter, we focus on technology mapping, an important logic synthesis transformation, to alleviate the congestion.

4.1 Introduction

4.1.1 Motivation

Although exact routing congestion information is known only after global routing, a failure to address congestion prior to this point implies that the designer is left with few degrees of freedom. Moving one step back, to placement, provides greater flexibilities, but is still not enough and it is known that this does not remove the need for a number of design iterations. This is often due to the poor fidelity of congestion-unaware delay estimates, which cannot accurately estimate the effect of long wire detours required for congestion reduction, or due to the unroutability of some designs where there may not be enough tracks available for routing.

It is imperative to address congestion issues early in the design process to allow for more freedom to reduce congestion. Previous work on wire planning in logic synthesis [GNBSV98, GKSV01] at the technology-independent optimization stage is targeted towards wirelength estimation to consider the wire delays (as opposed to our work which targets routing congestion). At this phase of logic optimization, it is not entirely clear which wires will be in the logic netlist, as this is decided during the technology mapping step. Technology mapping provides powerful capabilities for absorbing long interconnect wires into internal connections within complex gates, or for splitting complex gates into simpler gates, thus helping to alter the overall distribution of wires in the layout. Thus, it is an ideal step where the routing congestion problem may be attacked with relatively more freedom (albeit relatively less information) than during placement and routing. Although several methods for integrating physical design with technology mapping have been proposed, there is little work on incorporating congestion consider-

ations. Existing methods for this purpose, which are based on indirect metrics such as wirelength, are unsatisfactory, and the work presented in this chapter is directed towards filling that void.

While congestion is an important consideration for technology mapping, the overriding objectives continue to be metrics such as area or delay or power. Therefore, it is more appropriate to use congestion as a constraint rather than as an objective. While optimizing for area and delay, it is desirable to ensure that the final netlist does not have congested spots, so that long detours are avoided and the netlist remains routable. Typically, very few places in the circuit (ideally, zero) should have congestion values that are greater than some threshold, and the final netlist should be well optimized from the area and/or delay perspectives.

4.1.2 Previous work

We review some of the previous works on congestion-aware technology mapping approaches in the literature. Stok *et al.* proposed a clustering of closely placed cells during technology mapping so that the matching choices covering distantly placed cells in the subject graph are ruled out [SK01]. This approach may result in long wires in the final netlist, and more importantly, may be so limiting as to leave a significant portion of the design space unexplored. Pandini *et al.* proposed wirelength as a metric to be minimized during technology mapping in order to reduce the congestion [PPS02]. Although large wirelength may be correlated with high congestion, the correlation is rather poor, and therefore, this may not result in an effective optimization. This observation has been borne out by recent work by the same authors [PPS03], who state that such a metric, when considered during technology mapping employing a traditional cost func-

tion ($K_1 \times$Area$+K_2 \times$Delay$+K_3 \times$Wirelength, where K_1, K_2, and K_3 are constants), may not result in decreased congestion. As pointed out by them, congestion is a local property that varies from bin to bin, and it is difficult to capture its effects using a global metric like wirelength. This inference led them to the conclusion that congestion can only be targeted using iterative placement and technology mapping. However, such a conclusion is valid only when the congestion optimization is performed using an indirect global metric in a traditional fashion, and is not generally true.

4.1.3 Our Contributions

We present a technique for performing congestion-aware technology mapping. Instead of trying to absorb the congestion information into a single metric, we work with information about the distribution of congestion over the entire layout. The contributions of our work can be summarized as follows.

- Using empirical data on several benchmarks, employing different script and libraries, we show the fidelity between the congestion maps for the subject graph and the mapped netlists, which is later exploited during the technology mapping.

- Instead of applying an indirect metric such as wirelength [PPS02, SIS99], we utilize probabilistic congestion estimates [LTKS02] to guide our technology mapping; these estimates are shown in [LTKS02] to have good fidelity with post routing congestion estimates and have even been implemented in a commercial tool.

- We define the congestion cost function in such a way that it allows the mapper to choose the area- or delay-optimal choices, when the corresponding wires are likely to pass through sparsely congested region. At the same time, the cost func-

tion allows the choice of congestion-optimal matches, when wires are likely to pass through densely congested region. Thus, different optimization modes are applied at different places in the circuit depending on the context. To the best of our knowledge, such selective optimization has not been applied before during technology mapping.

- Experimental results due to congestion-aware technology mapping algorithms on an industrial benchmark and ISCAS'85 circuits show an improvement of 37%, on an average, in track overflows as compared to conventional mapping, when area minimization is targeted, while, on an average, 46% improvement in track overflows is obtained over the regular method, when the circuit delay is an optimization objective. These improvements come at the cost of an 8% and a 2% gate-area penalty, respectively, for area and delay minimization.

The organization of the rest of the chapter is as follows. Section 4.2 introduces the terminology and problem definition, while Section 4.3 presents empirical and intuitive justifications for congestion fidelity for pre-mapped and mapped netlists. Sections 4.4 and 4.5 illustrate congestion-aware technology mapping algorithms targeting area and delay, respectively, while Section 4.6 discusses time complexity and the possible extensions to these algorithms. Section 4.7 presents experimental results and conclusions followed by the summary of the chapter in Section 4.8.

4.2 Preliminaries

4.2.1 Terminology

The following terminology is used throughout this chapter. A Boolean network is a directed acyclic graph (DAG), in which a node denotes a Boolean function, $f : B^n \rightarrow B$, where $B = \{0, 1\}$, and n is the number of inputs to the node. The traditional technology mapping is usually preceded by a decomposition of this abstract network into one that contains primitive gates, such as 2-input NAND's and inverters. The decomposed network is referred to as a subject graph or a premapped netlist. The subject graph is mapped on to a set of cells in the library during technology mapping; the resulting network is known as a mapped netlist, which is placed in a given block area and routed. The block area is divided into bins for congestion analysis purposes or for global routing. Each bin contains a limited number of horizontal and vertical tracks. The track overflow and congestion can be defined for every bin as follows.

Definition 4.2.1 *The horizontal (vertical) track overflow for a given bin is defined as the difference between the number of horizontal (vertical) tracks required to route the nets through the bin and the available number of horizontal (vertical) tracks.*

Definition 4.2.2 *The horizontal (vertical) congestion for a given bin is the ratio of number of horizontal (vertical) tracks required to route the nets through the bin to the number of horizontal (vertical) tracks available.*

A positive track overflow or a congestion more than 1.0 means that sufficient tracks are unavailable for the routing, while negative value of the overflow or the congestion smaller than 1.0 indicates the availability of tracks.

4.2.2 Problem Definition

Routing congestion depends on the following factors:

1. the connectivity of the network,

2. the placement of cells in the layout, and

3. the routing of interconnects between the placed cells.

Since there is relatively less freedom to attack routing congestion during the placement and routing stages, we concentrate on the first factor. The technology mapping step makes crucial decisions regarding the connectivity of the network, since the mapping of primitive gates to the library cells determines the set of wires that will be present in the circuit netlist. Traditionally, this has been carried out without any placement information. Although this has changed in recent physical synthesis vendor offerings, most approaches focus on the prediction of wirelength based on bounding box estimates that ignore congestion. The estimation of routing congestion without a placement for a network is, if not impossible, liable to be highly inaccurate, and one may have to rely on high level metrics such as adhesion (defined as a sum of min-cuts between all pairs of nodes in the network) [KSD02]. However, this is a very new metric and several open questions about it remain unanswered: for example, whether it can be measured in a computationally efficient manner, and whether its fidelity is valid for mapped netlists. On the other hand, probabilistic congestion estimation [LTKS02] used after the placement of a mapped network has been demonstrated to correlate well with the congestion map generated after the routing, on both academic and industrial benchmark circuits. The estimation method divides the layout into bins and computes the congestion for a

given bin under all possible routes for a given net. We use the same method to guide our technology mapping algorithm. However, even such a method is difficult to adapt, since only the premapped netlist is available prior to technology mapping, and the level of correlation between the probabilistic congestion maps of the premapped netlist and the mapped netlist has not been studied in the past. One contribution of this work is to perform such a study. From empirical evidence obtained employing different logic synthesis scripts and placement algorithms on a variety of benchmarks, we show a good congestion correlation between premapped and mapped netlists. Once we establish the congestion correlation between the premapped and mapped netlists, the problem of congestion-aware technology mapping can be defined as follows.

Problem definition 4.2.1 *Given a subject graph of a network and a library of gates, synthesize a network optimizing area or delay such that the maximum (horizontal/vertical) congestion over all of the bins is less than the given threshold.*

4.3 Congestion Fidelity

This section explores the level of fidelity between the congestion estimates before and after technology mapping for any given circuit. For a given circuit, a premapped netlist contains primitive gates such as 2-input NAND's, while a mapped netlist contains a set of cells from a given library. Intuitively, the premapped and mapped netlist for a circuit share the same global connectivity, since the mapper absorbs some wires of the subject graph into the internal nodes of library cells, leaving other wires untouched. This points towards the possibility of good fidelity between congestion maps for premapped and mapped netlists. However, congestion also depends on the placement of elements,

primitive gates or gates in the library, in the netlist. Placement algorithms employed by commercial tools and in academia are typically based either on recursive multi-level bisectioning or force-directed quadratic programming. It would be useful to understand, even empirically, whether these placement algorithms react to the same global connectivity and block area constraints in a similar way. If so, there may be a good congestion correlation between premapped and mapped netlist. We explore this issue by performing a set of experiments using a variety of placers, logic synthesis scripts, libraries, and benchmarks.

4.3.1 Experimental Setup

To verify the fidelity between congestion estimates before and after technology mapping, we placed several premapped netlists, and the corresponding mapped netlists using the same block area and the same placement of input/output terminals. Two different placement algorithms were employed – a recursive bisectioning based algorithm in a publicly available tool, Capo [CKM00], and a force-directed quadratic algorithm, Kraftwerk [EJ98], implemented in a proprietary industrial placer. Different scripts, such as *rugged*, *boolean*, *algebraic*, *espresso*, and *speedup* in SIS [Sen92] were applied for preprocessing the netlists before technology mapping employing different libraries in SIS as well as an industrial library used for high-performance microprocessor designs. The following options were used for mapping and placement.

- Mapping was performed in SIS using the *map -s -n 0 -AFG -p* command that performs area and fanout optimization. No layout information was utilized to guide this technology mapping.

- Placement using Capo [CKM00] was performed with default options to minimize the total wirelength based on half perimeter bounding box estimates, while
- As an alternative to Capo, Kraftwerk [EJ98] was employed to perform placement to minimize total wirelength as well as congestion.

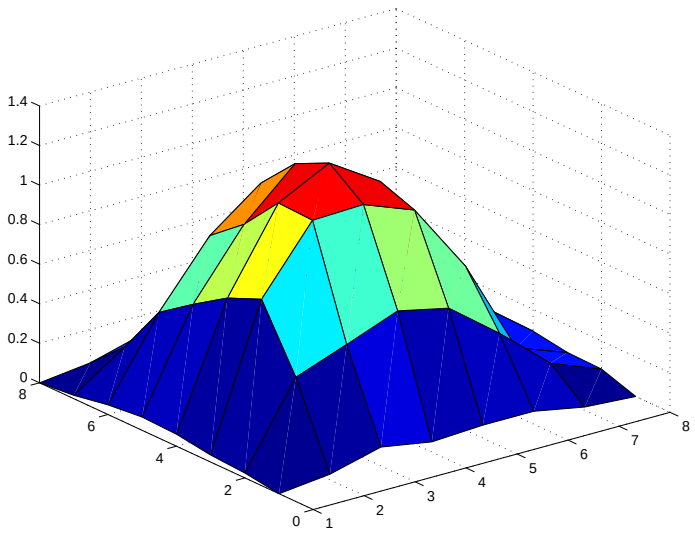
The premapped netlist is an abstract Boolean network containing primitive gates such as 2-input NAND's and inverters. For the placement of such a netlist, the primitive gates must be assigned areas. We assign the areas of the corresponding minimum-sized gates in the library to these primitive elements. Since the number of nodes in this netlist is large, the area of primitive gates must be scaled by a certain factor to present the same white space constraints as the mapped netlist for the placement. This factor is computed *a priori* as a ratio of the targeted gate area to the area of premapped network using the following equation.

$$\text{Scaling factor} = \frac{\text{Block area} - \text{White space area}}{\text{Area of premapped network}} \tag{4.1}$$

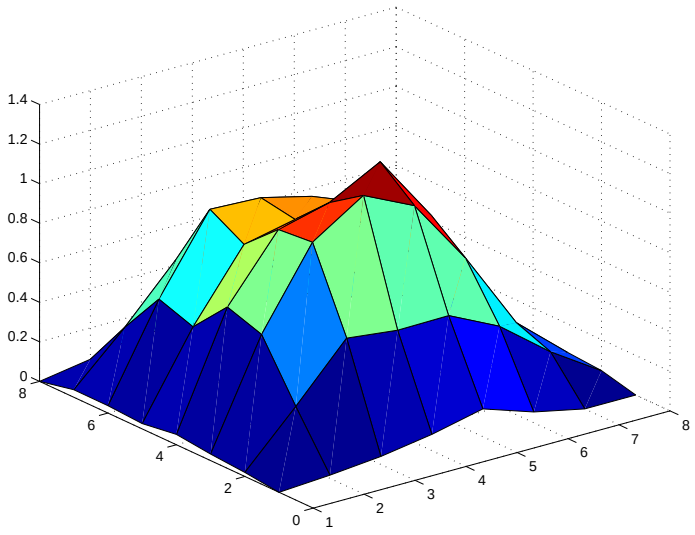
Note that this factor is readily available given the block area, the white space area, the premapped netlist, and the cell library, and does not require any testcase-specific tuning.

4.3.2 Experimental Results

We show results for a few representative benchmarks: C432, C6288, C7552, and an industrial circuit containing an instruction decoder (IDC) in a high-performance microprocessor. Apart from the vastly different functionalities, the sizes of these benchmarks also vary from a few hundred cells to a few thousand cells. Figures 4.1 (a) and (b) show congestion maps for the benchmark C432 for the mapped and premapped netlists, respectively. The placement of both the networks is performed using Capo. In these plots,



(a)



(b)

Figure 4.1: Horizontal congestion for C432 for (a) the area-oriented mapped netlist and (b) the premapped netlist. *script.rugged* is used for preprocessing the netlist and Capo [CKM00] is employed for placement.

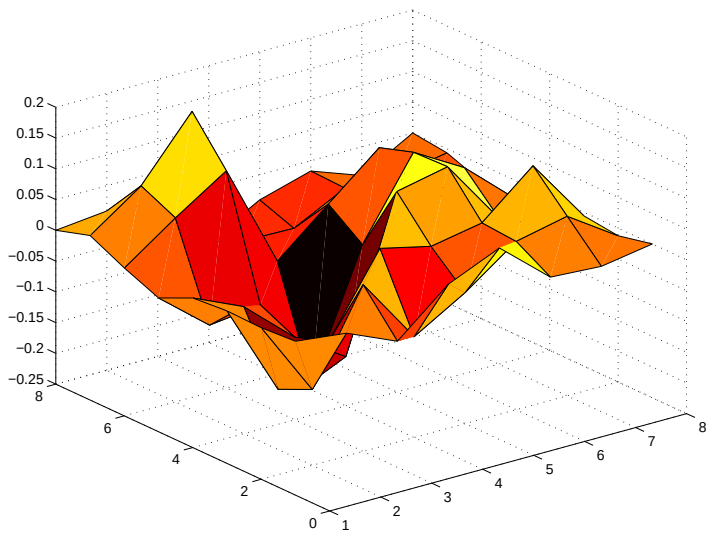
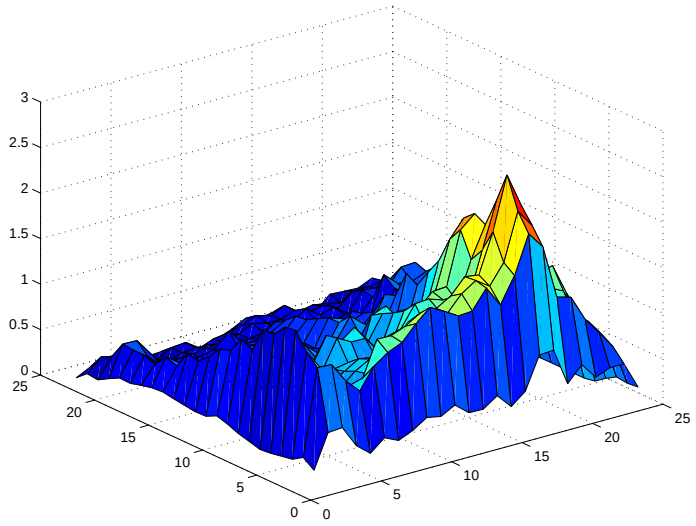


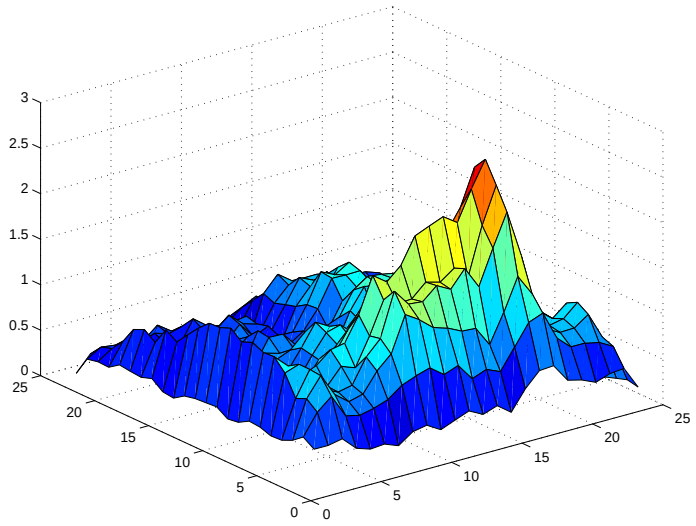
Figure 4.2: Bin-wise congestion difference between pre-mapped and mapped netlists corresponding to Figure 4.1(a) and 4.1(b), respectively, for C432.

the XY plane shows the two dimensions of the layout area, while the Z-axis depicts the congestion. Visually, one can conclude that the distribution shown in Figure 4.1(b) is similar in nature to the congestion map shown in Figure 4.1(a); the bin-wise difference in the congestion values is shown in Figure 4.2. One can observe that, for most of the bins, the difference between the congestion in the premapped and mapped netlists is less than 10%. Similarly, the congestion maps for the benchmark circuit C7552 and for the circuit IDC for the mapped and premapped netlists are shown in Figures 4.3 and 4.4, respectively; the netlists for these benchmarks are placed using Kraftwerk [EJ98]. The congestion maps for the premapped netlists for C7552 and IDC show characteristics similar to those of the corresponding mapped netlists.

Representative results for some ISCAS'85 benchmarks and the IDC circuit employing different scripts, libraries, and placers are shown in Table 4.1, while similar results on more extensive set of benchmarks are presented in tables 4.2 and 4.3. Columns

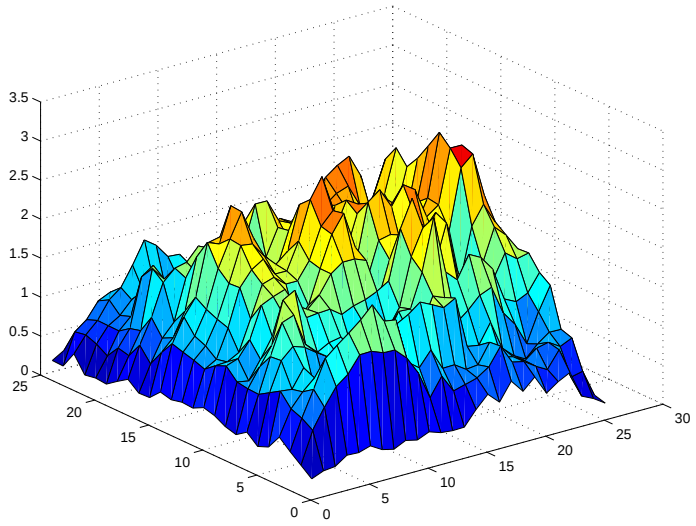


(a)

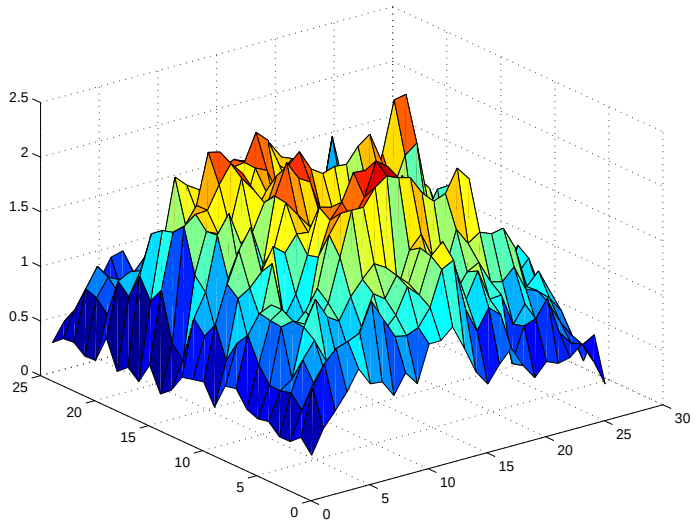


(b)

Figure 4.3: Horizontal congestion for C7552 for (a) the area-oriented mapped netlist and (b) the premapped netlist. *script.algebraic* is used for preprocessing the netlist and Kraftwerk [EJ98] is employed for placement.



(a)



(b)

Figure 4.4: Vertical congestion for IDC for (a) the mapped netlist and (b) the premapped netlist. *script.boolean* is used for preprocessing the netlist and Kraftwerk [EJ98] is employed for placement.

2, 3, and 4 show the scripts used, the number of cells in the mapped netlists, and placement tools used, respectively. Technology mapping in SIS [Sen92] is performed using the area and fanout optimization option, employing the lib2.genlib library in SIS and an industrial library. It is worth noting that the mapped netlist is fanout-optimized, which possibly restructures the network after the mapping and may affect the global connectivity adversely. Columns 5 (6) and 7 (8) in the table show the average and maximum horizontal (vertical) congestion, respectively, while columns 9 and 10 show the statistical correlation between the congestion in premapped and mapped netlist. The correlation is defined as $\frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$, where $E[]$ is the expectation, μ is the mean, σ is the standard deviation; in our case, X and Y correspond to the congestion in the premapped and mapped netlists, respectively. A correlation value closer to 1 (-1) means that two random variables are strongly positively (negatively) correlated, while a value close to 0 means that variables are weakly correlated [DS01].

4.3.3 Justification Based on Experimental Results

In spite of fanout optimization that may affect the global connectivity and hence congestion fidelity, the congestion correlation between subject graph and mapped netlist is always greater than 0.6 for all the netlists. One may deduce the following based on these experimental results.

- Across different libraries, scripts, benchmarks, fanout optimization, and placement algorithms, a good correlation exists between the congestion map for the subject graph and congestion map for a mapped netlist.

- The reasons for the congestion correlation are likely to be the similarities in the

global connectivity in the subject graph and the mapped netlist, the similar block area and I/O terminal constraints, and the way any reasonable placement algorithms react to such resemblances in global connectivity and the block area constraints.

4.4 Congestion-aware Area-oriented Mapping

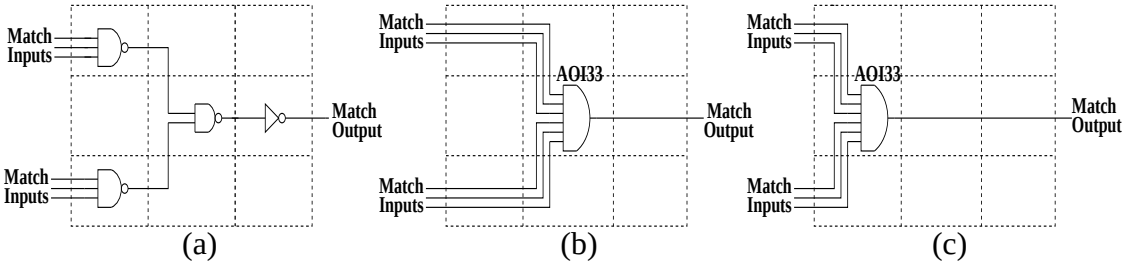


Figure 4.5: Mapping choices: (a) Sub-optimal area and track requirement = 12. (b) Area-optimal and track requirement = 20. (c) Area-optimal and track requirement = 15.

In this section, we focus on area optimization as an objective for technology mapping. For the purposes of congestion-aware mapping, the sparsely congested and densely congested regions must be identified. From the experiments in the previous section, which demonstrate the congestion correlation between a subject graph and its mapped netlist, we can conclude that the former netlist is accurate enough for this purpose. Since the primary objective of our congestion-aware technology mapper is area minimization, we employ a variation of a widely used dynamic programming-based technology mapping algorithm [Keu87]. The technology mapping procedure involves the matching and covering phases: the former comprises storing the set of optimal matches at each node,

while the latter involves constructing the network by selecting from the matches stored during the matching.

4.4.1 Example

A pure area/delay minimization objective during technology mapping can result in poor congestion, and Figure 4.5 illustrates a case where suboptimal matches may reduce congestion. Assume that all of the bins, shown as dashed squares in the figure, are congested and a match for the AOI33 function is considered. The inputs to the match enter through top and bottom bins on the left, while the output leaves from the middle bin on the right. Figure 4.5(a) shows one possible match containing two three-input NAND's, a two-input NAND, and an inverter, while Figure 4.5(b) and Figure 4.5(c) show an alternative match, an AOI33, under two different placements. To simplify the computations, if we use the number of bin-boundary crossings as the congestion metric, instead of the probabilistic congestion metric, then the cost for the match in Figure 4.5(a) is 12, while that for the AOI33 matches in Figures 4.5(b) and (c) are 20 and 15, respectively. The latter also happens to be the minimum over all placements for the area-optimal AOI33 match. It is clear that the match in Figure 4.5(a) distributes the logic and therefore, creates lower congestion. This example also highlights limitations of the placement in alleviating congestion, when area-optimal matches are chosen ignoring the costs of wires associated with them.

The cost of wires depends on the context: wires are inexpensive in sparsely congested regions, but are expensive in densely congested regions due to possible detours and hampered routability. One way to reduce this cost in densely congested zones without penalizing the design excessively is to account for their congestion contribu-

tions only in those zones. Our congestion-aware mapping heuristic serves this purpose well: in densely congested spots, it considers probabilistic routes based on the center-of-gravity locations for all possible matches and chooses the match that minimizes the congestion, while in sparsely congested spots, it chooses area-optimal matches.

4.4.2 Congestion Cost Computation

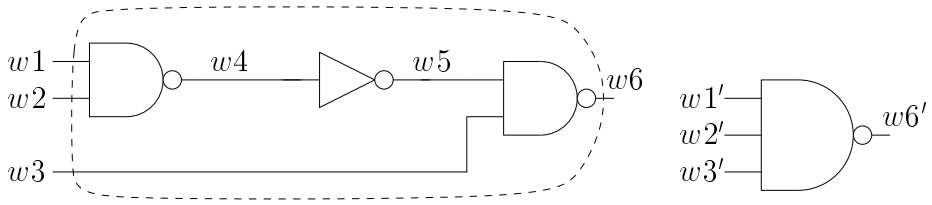


Figure 4.6: Computing the congestion cost of a match: (a) An example subject graph. (b) One possible match.

The congestion-aware mapping heuristic requires the assignment of a congestion cost, along with an area cost, to each match. The congestion cost depends on the total congestion caused due to the nets subsumed by a match, its fanin nets and its fanout nets. Specifically, it is given by,

$$cost^{C}_{Match} = \Sigma\, cost^{C}_{net\_created} - \Sigma\, cost^{C}_{net\_subsumed} \tag{4.2}$$

where, $cost^{C}_{Match}$ is the congestion cost of the match, $cost^{C}_{net\_created}$ ($cost^{C}_{net\_subsumed}$) is the congestion cost of the nets created (subsumed) by the match. For example, for a 3-input NAND match shown in Figure 4.6(b) corresponding to the subject graph shown in Figure 4.6(a), the congestion cost is as follows:

$$cost^{C}_{Nand3} = C_{w1'} + C_{w2'} + C_{w3'} + C_{w6'} - (C_{w1} + C_{w2} + C_{w3} + C_{w4} + C_{w5} + C_{w6}) \tag{4.3}$$

The nets $w1'$, $w2'$, $w3'$, and $w6'$ correspond to the new location of the match and the fanins and fanouts of the match; we compute the new location of a match as the center of gravity of the locations of its fanin and fanout gates. Multi-terminal nets are modeled using cliques for the congestion computation, and congestion contribution of each edge is scaled by a factor of $2/n$, where n is the number of edges.

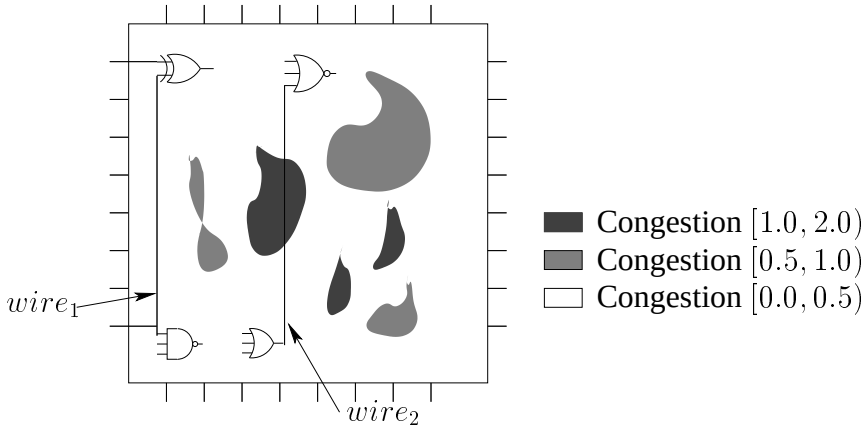


Figure 4.7: Context-dependent congestion cost for the wires.

The congestion cost of a wire depends on the route and the congestion in the bins that the route passes through. Probabilistically, all of the routes in the bounding box of the net are assumed to be equally possible[1] [LTKS02]. If a congestion (say 0.4) in a bin in the bounding box of the net is small as compared to the threshold congestion (say 1.0, for instance), then the congestion contribution of that net for that bin is assumed to be 0. This is because a small value of the congestion metric corresponds to the availability of numerous tracks, and the routability of the net through the bin is unaffected. However, if the bin is congested, then the probabilistic congestion contribution of the net to that

[1]This assumption may not always be true. Typically, routers try to minimize vias and therefore, for two terminal nets only L and Z routes are considered. Such information can be taken into account while generating the congestion map.

bin must be considered as its routability is hampered. In case of Figure 4.7, $wire_1$ and $wire_2$ will have different congestion costs even though the shortest routes in both the cases may have the same length; the congestion cost of the former will be zero, while that of the latter will have a positive value as its bounding box contains congested bins. The following equation captures this causality relation between routability and congestion while computing the congestion cost of a net, $cost^C_{net}$,

$$cost^C_{net} = \Sigma_{\{Bin \in BoundingBox(net): C(Bin) > C_{max}\}} C^{Bin}_{net} \tag{4.4}$$

where $C(Bin)$ is the congestion in a bin, C_{max} is the threshold congestion, and C^{Bin}_{net} is the congestion due to the specific net within the bin. It is easily seen that this definition filters out the contributions of uncongested bins from the congestion cost.

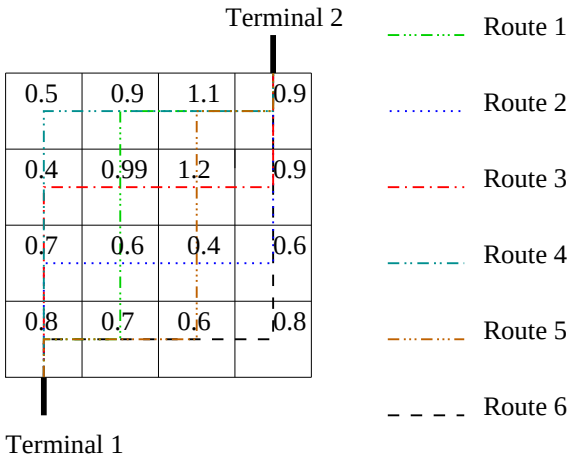


Figure 4.8: Computing the congestion cost of a wire probabilistically as in [LTKS02].

The bounding box for a two-terminal net is shown in Figure 4.8. It contains 16 bins, and the congestion value associated with each bin is shown in the figure. For the net connecting terminals 1 and 2, six possible L- and Z-shaped routes are shown for the purpose of illustration[2]. To compute the congestion cost, if the threshold value of

[2]In practice, we use probabilistic congestion estimates that consider river routes as well.

congestion (C_{max}) is set to 1.0, then we consider only the congested bins for which congestion value is greater than 1.0, i.e., bins for which the congestion metric is 1.1 and 1.2. Three routes (route 1, 4, and 5) pass through the bin with congestion 1.1, while two routes (route 3 and 5) pass through the bin with congestion 1.2. Assuming all the routes to be equally possible, the demand (the ratio of number of paths passing through the bin to the total number of paths) for tracks in the latter bin is $\frac{2}{6}$. Similarly, the demand for tracks in the former bin is $\frac{3}{6}$. Using the definition 2.1, congestion contribution of the net for these bins can be computed by dividing the demands by the number of available tracks (N_{Tracks}). Employing Equation 3, the congestion cost of the net is given by

$$cost^{C}_{net} = \frac{1}{N_{Tracks}} \times (\frac{2}{6} + \frac{3}{6}) \tag{4.5}$$

The congestion cost for a match can be calculated from that of its incident nets. A positive cost implies that it may increase the congestion beyond the threshold value in some bins, while a negative cost implies that it may decrease the congestion in some of the bins where congestion exceeds the threshold value.

4.4.3 Algorithm for Congestion-aware Area-Oriented Mapping

Algorithm 4.4.1 shows the pseudo-code for choosing the best match at a node during the matching phase of the technology mapping. The triplet (C_i, A_i, D_i) associated with a match M_i denotes the congestion cost, area cost, and delay cost associated with the match. The algorithm is called for every match at a node during the matching phase to decide the best one to be stored at the node. The congestion cost is given priority over the area and delay only in congested regions, and area-optimal matches will be chosen for the nodes in the sparsely congested regions, as stated by the following proposition.

Algorithm 4.4.1 Select the best match considering the congestion

Input: Match $M_1(C_1, A_1, D_1)$ and match $M_2(C_2, A_2, D_2)$

Output: The best match between the M_1 and M_2

1: **if** (C_1 == C_2) **then**
2: **if** $(A_1 < A_2)||((A_1 == A_2)\&\&(D_1 < D_2))$ **then**
3: return M_1;
4: **else**
5: return M_2;
6: **end if**
7: **end if**
8: **if** ($C_1 < C_2$) **then**
9: return M_1;
10: **else**
11: return M_2;
12: **end if**

Proposition 4.4.1 *If bins in bounding boxes of all of the nets, corresponding to all of the matches at a node, have congestion values that are smaller than the threshold congestion, then an area-optimal match will be stored as the best match at that node.*

Proof 4.4.1 This is a direct consequence from the fact that the congestion cost for all nets corresponding to all of the matches for such a case is zero from Equation (4.4), and the pseudocode shows that under this scenario, the area-optimal match is always chosen.

Remark 4.4.1 The above result is important for congestion-aware mapping, since previous work in [PPS03] has shown that the traditional way of considering the cost,

($K_1 \times$ Area $+ K_2 \times$ Wirelength) during technology mapping requires different values of K_2 in the different regions in the circuit as a single value of K_2 fails to capture the importance of congestion in different regions. Choosing a single value of K_2 may correspond to the case in which entire circuit is uniformly congested with a single congestion value. In reality, the congestion in the circuit varies continuously from 0 to 1, or is even $>$1, while the routability changes in a discrete manner: in case of a bin with congestion value $>$ 1, at least, some nets are detoured, or are unroutable, while routability of all the nets is unaffected when the congestion for the bin is $<$1. Assigning the congestion cost to the nets in the congested bins accounts for this discrete nature of routability and also allows the mapper to select area-optimal matches in the sparsely congested regions. Both of these purposes are critical and are served by our algorithm, while previous approaches [SK01, PPS02] have not addressed these.

4.5 Congestion-aware Delay-oriented Mapping

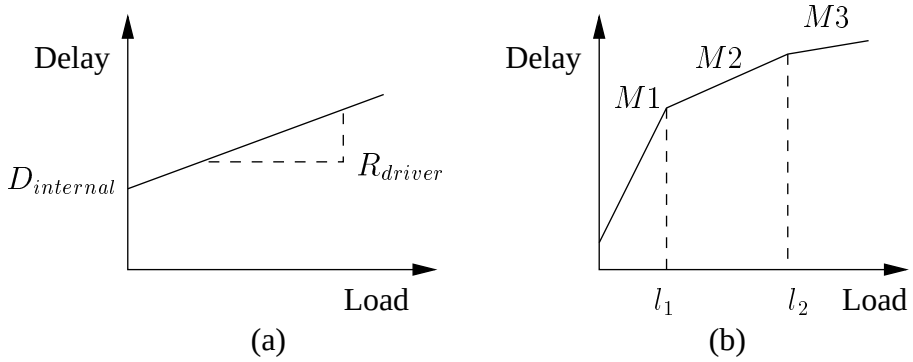


Figure 4.9: (a) A load-based delay model for a typical standard cell, such as an inverter. (b) A typical load-delay curve stored during matching.

The congestion-aware area-oriented mapping framework presented in the previous section can be extended to delay-oriented technology mapping. This typically employs one of the following two classes of delay models: load- or gain-based. In this section, we focus only on delay-oriented mapping based on the former, since an extension based on the latter is similar. The load-based delay model is shown in Figure 4.9(a) for a typical standard cell: it shows a straight line with the internal delay of the gate, $D_{internal}$, as an intercept on delay axis, while the slope of the line indicates the effective driver resistance [3]. Technology mapping targeting delays involves storing piece-wise linear load-delay curves, $\{(l_1, D_1), (l_2, D_2), \cdots\}$, during the matching phase, where l_i and D_i denote load and delay co-ordinates, respectively, of an end-point of a piece-wise linear segment. At each node, a set of matches that are delay-optimal for certain load ranges are stored on these curves; one such curve is shown in Figure 4.9(b) with three different matches $M1$, $M2$, and $M3$, where $M1$ is optimal for the load range $[0, l_1]$, $M2$ for the range $(l_1, l_2]$, and $M3$ for larger load values. During the covering phase, when loads are known, delay-optimal matches are chosen from the curves. SIS [Sen92] contains an implementation of a delay-oriented mapper based on this scheme, but the wire-delays that are ignored in this mapper may lead to suboptimal results. To perform delay-oriented mapping, it is necessary to consider wire-delays; delay computation considering the effects of wires is explained in the following subsection.

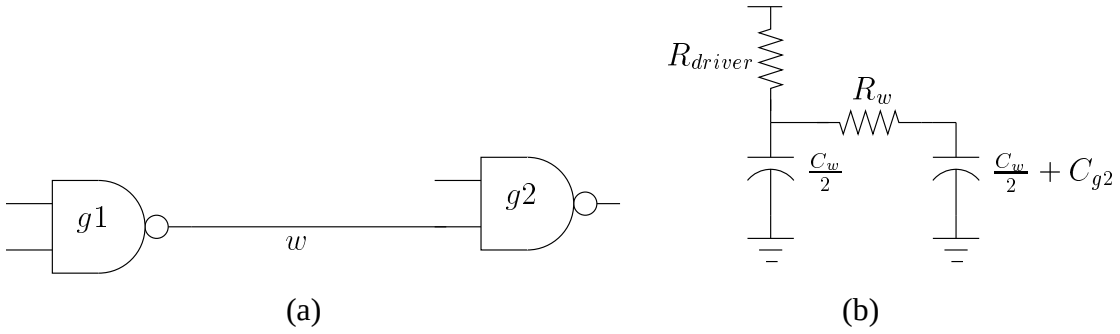


Figure 4.10: (a) Wire driven by a gate. (b) The corresponding RC model.

4.5.1 Delay Computation Considering Wires

The delay computation involves accounting for wire-loads as well as wire-delays by modeling the wires using RC π model as shown in Figure 4.10. In Figure 4.10(a), gate $g1$ drives gate $g2$ through wire w; the corresponding RC model is shown in Figure 4.10(b). The delay from the inputs of the gate g_1 to the input of g_2 through the wire shown the figure is given by the following equation

$$D = D_{g1} + D_w \tag{4.6}$$

where D_{g1} and D_w are the delays of the gate g_1 and the wire w, respectively. Employing the Elmore delay model[4] [Elm48], the gate delay D_{g1} is given by

$$D_{g1} = D_{internal} + R_{driver} \times (C_w + C_{g2}) \tag{4.7}$$

where $D_{internal}$ is the internal delay of the gate, R_{driver} is the effective resistance of the gate, C_w is the capacitance of the wire w, and C_{g2} is the input capacitance of the gate

[3]The delay of a cell also depends on the slope of the input signal transitions, which are considered during precise timing analysis, but are often ignored in the delay models at the technology mapping stage.

[4]More accurate delay models, such as asymptotic waveform evaluation (AWE) [PR90], can be employed while keeping the rest of the algorithmic framework intact.

g_2. Similarly, the wire delay D_w is given by

$$D_w = R_w \times (\frac{C_w}{2} + C_{g2}) \tag{4.8}$$

where R_w is the resistance of the wire.

In general, the resistance (and capacitance) of a wire is a function of the distance and a choice of metal layers. Since the resistivity of the upper metal layers is smaller because of the higher width and thickness as compared to lower metal layers, these metal layers are used to route the long wires. For the short wires, lower metal layers are utilized, since reliability and resistance of the vias along with subsequent congestion does not justify the use of upper metal layers. The range of wirelengths and choice of metal layers can be determined empirically for a given process technology and used to compute wire delays during technology mapping. We employ this scheme to account for the wire-loads and delays during congestion-aware delay-oriented mapping.

4.5.2 Congestion Cost Penalty Heuristic

To store congestion-aware choices during the matching phase, solutions that increase the congestion should be penalized. It can be achieved by computing the congestion cost of a match and adding the corresponding delay penalty while storing the match on the load-delay curve. This heuristic is a natural extension to congestion-aware area-oriented mapping presented in the section 4.4, where a match that minimizes the congestion is stored as the best match for a given node. In case of delay-oriented mapping, multiple choices are stored on the load-delay curve, each choice being optimal for a certain load range. The congestion cost for a match depends on the corresponding cost of fanin and fanout nets, and nets that are subsumed by the match, as given by Equation 4.2, which

is reproduced below for the sake of readability

$$cost^C_{Match} = \Sigma\, cost^C_{net\_created} - \Sigma\, cost^C_{net\_subsumed} \tag{4.9}$$

where, $cost^C_{Match}$ is the congestion cost of the match, $cost^C_{net\_created}$ ($cost^C_{net\_subsumed}$) is the congestion cost of the nets created (subsumed) by the match. To penalize the matches that cause congestion and to favor those that reduce it, a penalty is added to the delay due to a match before storing it on load-delay curve. The penalty corresponding to the congestion cost of a match is given by the following equation

$$D_{penalty} = D_{wire}(cost^C_{Match} \times bin_{dimension}) \tag{4.10}$$

where, $bin_{dimension}$ represents either the width or the height of a bin, while $D_{wire}(s)$ denotes the delay of a wire of a length s. The heuristic is explained using the following example.

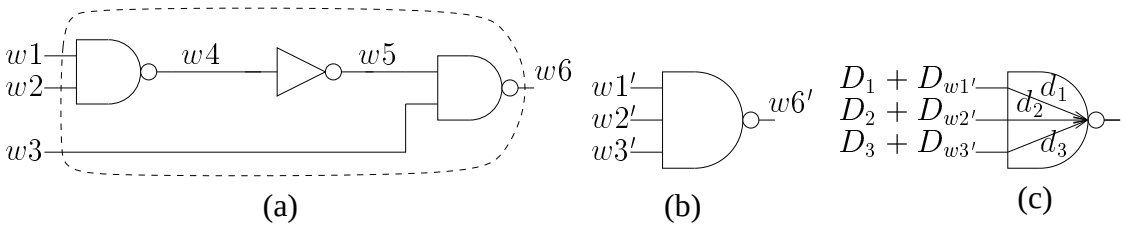


Figure 4.11: Delay computation for a match: (a) An example subject graph. (b) A match of 3-input NAND. (c) Delay computation.

Consider an example of a match of 3-input NAND, which is same as the example in Section 4.4.2, shown in Figure 4.11. It subsumes wires $w1$, $w2$, $w3$, $w4$, $w5$, and $w6$, as shown in Figure 4.11(a), while it creates wires $w1'$, $w2'$, $w3'$, and $w6'$, as shown in Figure 4.11(b); we assume that the match is placed at the center of gravity of its

fanins and fanouts, as in case of area-oriented mapping. Figure 4.11(c) shows the delay computation, where $D_i + D_{wi'}$, $i = 1, 2, 3$, are arrival times, including the corresponding wire delays, at the inputs of the match, while d_i are internal delays for the corresponding pins. The delay of the match is given by the following equation

$$D = max(D_1 + D_{w1'} + d_1, D_2 + D_{w2'} + d_2, D_3 + D_{w3'} + d_3) \tag{4.11}$$

The congestion cost of the match, repeated from Equation 4.3, is given by the following equation

$$cost^C_{Nand3} = C_{w1'} + C_{w2'} + C_{w3'} + C_{w6'} - (C_{w1} + C_{w2} + C_{w3} + C_{w4} + C_{w5} + C_{w6}) \tag{4.12}$$

To make the match congestion-aware, delay penalty proportional to the above cost is added to the delay. Therefore, the delay of the match is now given by

$$D_{congestion-aware} = D + D_{penalty}(cost^C_{Nand3} \times bin_{dimension}) \tag{4.13}$$

It is obvious that matches with positive congestion cost are penalized, while those with the negative congestion cost are favored. Note that, in sparsely congested regions no delay penalty is added and therefore, delay-optimal matches are still chosen in those regions.

4.5.3 Algorithm for Congestion-aware Delay-oriented Mapping

During the matching phase, all nodes are processed in topological order, and their load-delay curves are computed. Algorithm 4.5.1 shows the pseudo-code for computation of the load-delay curve. For each match m from the set M, the delay, D, is computed considering the arrival times of the inputs D_i, wire delays D_{wi}, and internal delay, d_i, of the gate corresponding to the match. The congestion cost of the match,

Algorithm 4.5.1 Compute load-delay curve for a match

Input: n = A node, M = A set of matches at the node

Output: Load-delay curve for n

1: **for** $m \in M$ **do**
2: **for** $i \leftarrow 1, \cdots, |inputs_m|$ **do**
3: $D \leftarrow max(D_i + D_{wi} + d_i)$
4: **end for**
5: $Cost_m^C \leftarrow$ ComputeCongestionCostOfMatch(m)
6: $D \leftarrow D+$ WireDelay($Cost_m^C \times bin_{dimension}$)
7: UpdateLoadDelayCurve(n, m, D)
8: **end for**

$Cost_m^C$, is then computed considering the corresponding costs of subsumed and created wires, and an appropriate penalty employing Equation 4.13 is added to the delay of the match. The load-delay curve is then updated to store the match, if it is optimal for some range of loads. The covering phase proceeds in a traditional manner to choose matches that are optimal for given loads.

4.6 Complexity, Limitations, and Extensions to the Algorithms

The time complexity of our congestion-aware technology mapping algorithms is almost unchanged from that of a conventional technology mapping. The congestion cost computation of a match takes $O(|Nets_{Match}| \times N_{Bins})$, where $|Nets_{Match}|$ is the number of nets associated with a match and N_{Bins} is the number of bins over entire layout;

$N_{Bins}$is a constant for a given layout, although it may be large as compared to other constants subsumed by $O()$. Therefore, congestion cost computation takes $O(|Nets_{Match}|)$ time, which is same as that of structural matching used in the mapper [Sen92].

Since this technology mapping procedure is applied to tree structures after the initial subject graph generation and the decomposition of DAG's into trees, the algorithm does not have any control over high fanout nets, or over the fanout nets created due to matches at the roots[5] of the trees. The congestion due to these high fanout nets is controlled by the structure of initial network and fanout optimization. The effectiveness of the congestion-aware mapper proposed here is influenced by the scripts used for technology independent optimization, technology decomposition, and fanout optimization after technology mapping.

Pre-routed blockages in the design can be incorporated into our congestion cost by reducing the appropriate number of tracks in the corresponding bins. Most placers are adequate at handling blockages. Therefore, subject graph nodes or mapped cells are not placed in blocked areas. While long wires may require repeaters that are not visible in the subject graph, observe that these buffers do not change the congestion cost.

In the current implementation, we do not update the congestion map dynamically during technology mapping. However, this update can be carried out during the covering phase, thus allowing a more accurate selection of the best match stored at a node. In case of area-oriented mapping, multiple congestion-aware choices must be stored during the matching phase in addition to the area-optimal one, in order to enable the selection of a good congestion-aware solution with the updated congestion map available during covering.

[5]All of the nodes in the tree have a fanout of 1 but the root.

4.7 Experimental Results

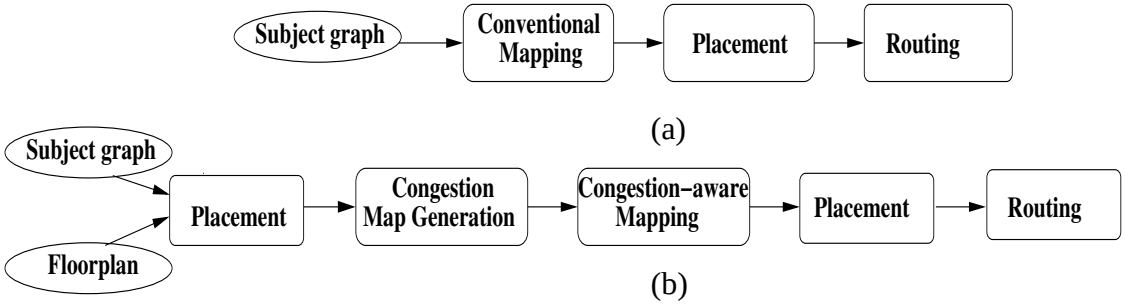


Figure 4.12: Design flows for (a) conventional and (b) congestion-aware mapping.

The probabilistic congestion estimation algorithm from [LTKS02] and the congestion-aware technology mapper were implemented in C/C++ and incorporated in SIS [Sen92]. The subject graphs were created by running *script.rugged* followed by *tech_decomp -o 2* in SIS [Sen92]. For area-oriented technology mapping, we present a set of experimental results obtained using a force-directed quadratic placer, Kraftwerk [EJ98], and a proprietary industrial maze router, while for delay-oriented mapping, we show results generated employing a recursive bipartitioning placer Capo [CKM00] and a global router [HS02]. The experimental flow used in our experiments is as shown in Figure 4.12. For congestion-aware mapping, a subject graph was first created. It was placed using Kraftwerk for area-oriented mapping, while Capo was employed for the placement in case of delay-oriented mapping. The congestion map for the subject graph was then generated and used in our congestion-aware mapper. After area-oriented technology mapping, the circuits were placed using Kraftwerk followed by global routing using a proprietary industrial router for area-oriented mapping. For delay-oriented mapping, Capo and a global router in [HS02] were employed, respectively, for placement

and routing. In all of our experiments, a bin-size of $4.8 \times 4.8 \mu \text{m}^2$ was used. We present the results due to area-oriented mapping followed by the same for delay-oriented mapping.

4.7.1 Results due to Area-oriented Mapping

Table 4.4 shows the post-routing results obtained using the Kraftwerk placer and proprietary maze router for conventionally mapped and congestion-aware netlists. Technology mapping is performed employing a proprietary industrial cell library used in high-performance microprocessor designs. Our experiments employ a 90nm technology and allow the router to use 4 metal layers[6]: metal 1 with no preferred direction, metals 2 and 4 for the horizontal direction, and metal 3 for the vertical direction. The entries of the form 'a / b' in the Columns 3 through 7 mean 'a' ('b') corresponds to conventionally (congestion-aware) mapped netlist. The block area shown in Column 2 is used for both of these netlists for the benchmarks shown in Column 1. Since the same block area is used for both the netlists, there is no area penalty. Columns 3, 4, and 5 show the average row utilization, the total track overflow over all the bins after global routing, and the number of bins with congestion more than 1.0, respectively, while Columns 6 and 7 show the maximum and average congestion, respectively. For small benchmarks such as C1355, C432, and C880, a few number of bins are congested in the conventionally mapped netlists while none of the bins is congested in the congestion-aware mapped netlists. This shows that congestion problem for a small number of bins can be easily resolved by congestion-aware mapped netlist without any area penalty. C499 and C1908

[6]While 90 nm and subsequent process generations have a large number of metal layers, the upper layers are usually reserved for global signals, clock and power distributions, leaving block synthesis to operate in the lower layers [SMCK03].

show zero routing track overflows, while other small benchmarks have only a few congested bins, indicating that routing congestion is not an important issue for designs up to a few hundred cells. As the design size grows beyond a thousand cells, routing congestion starts becoming a critical problem, as indicated by increased track overflows for benchmarks such as IDC, C6288, and C7552. In these cases, the congestion-aware mapped netlists have been able to reduce the track overflows by 87%, 43%, and 29% while the number of congested bins has decreased by 81%, 65%, and 25%, respectively. Based on the increase in average congestion for all of the benchmarks, accompanied by a reduction in the number of congested bins and the number of track overflows, we see that congestion-aware mapping tends to map the logic so as to distribute the congestion from densely congested regions to the sparsely congested regions. The improvement in congestion comes at the cost of an increase in gate-area, which is reflected in higher row utilization in case of congestion-aware netlist for all the benchmarks.

4.7.2 Results due to Delay-oriented Mapping

Table 4.5 show post routing results for delay-oriented mapping obtained using recursive bisectioning based placer Capo [CKM00] and a global router [HS02]. Technology mapping was performed employing lib2.genlib library in SIS [Sen92]. Up to 4 different strengths were added for each cell in the library, which was then characterized for 130nm technology [ptm]. Column 1 in the table shows the benchmark circuit, while column 2 shows the block area. Columns 3, 4, 5, and 6 show the average row utilization percentage, the circuit delay in ps, the maximum congestion, and the total overflow, respectively. The entries of the form ‘a / b’ in these columns have the same meaning as before. Congestion-aware netlists tend to have, on an average, larger gate-area, which

is reflected in overall 2% increase in average row utilization percentage. The delays in the congestion-aware netlists have remained almost unchanged from those in the corresponding conventional netlists. This indicates that for 130nm technology, interconnects are not sufficiently resistive to dominate the gate delays by large margin, especially for benchmarks up to few hundred cells. Track overflows have improved consistently for all the benchmarks due to congestion-aware mapping; the average improvement over conventional mapping is 46%. This shows that our heuristic is effective in alleviating the routing congestion without penalizing the delay values, which are, in fact, improved in most of the cases. The maximum congestion is improved in all cases but C880, in which case, however, track overflows and delays have improved. The overall improvement in maximum congestion has been 9%.

4.7.3 Wirelength and Detour Distributions

For large benchmarks, the wiring distributions obtained after global routing showed significant improvements as a result of our congestion-aware area-oriented technology mapping flow. The improvement in the wiring distribution is best exemplified by a reduction in the incidence of detours on the routes, where we define the detour of a route as the difference between its actual length and the total size of its minimum spanning tree (MST[7]).

Figure 4.13 shows plot of the number of nets vs. detour for the benchmark IDC. Similar wire distribution plots were obtained for other benchmarks. In the figure, the log-scale Y-axis shows the number of nets, while the X-axis shows the detour, in μm, for all the nets on a linear scale. The height of a brown (purple) bar in the figure rep-

[7]Because of the canonicity of MST's, MST estimates are used to compute the detours even though they tend to be overestimates as compared to minimum Steiner estimates.

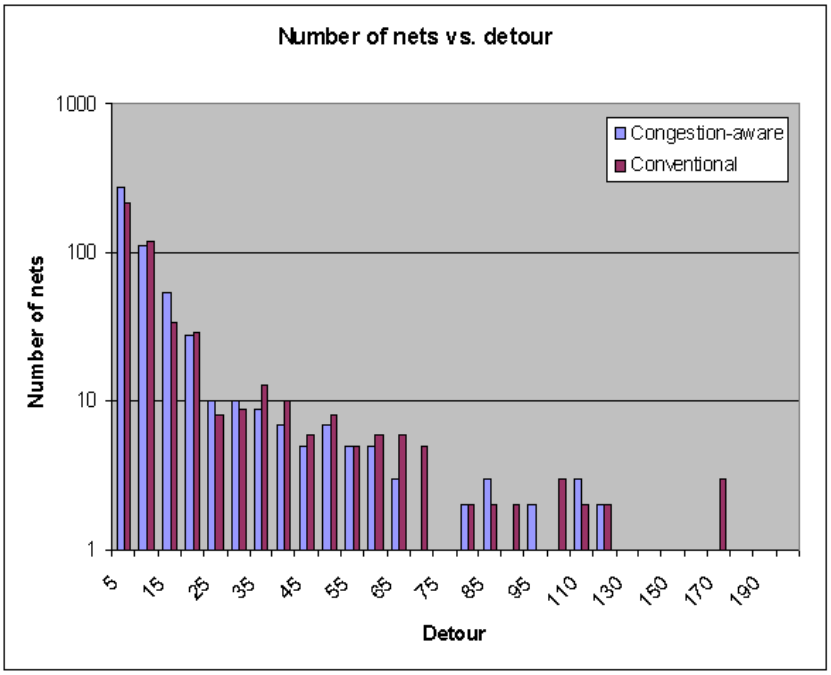


Figure 4.13: Number of nets vs. detour length (μm) for the IDC circuit. The placement of conventionally mapped netlist and that of premapped as well as mapped netlist, in case of congestion-aware mapping, is performed using Kraftwerk.

resents the number of nets in the conventional (congestion-aware) netlist for a given detour range. It can be observed that for shorter detour ranges, the number of nets in the congestion-aware netlist dominates their conventional counterpart, while as the detour length increases, the number of nets from the conventional netlist dominates that in congestion-aware netlist. Although the total number of wires increases in the congestion-aware case, most of this increase occurs at short wire lengths, as seen from the figure.

Figure 4.14 shows plot of net-length vs. detour length for all the nets in congestion-aware and conventionally mapped netlist for IDC. In the figure, the symbols '+' and '$\times$' indicate the actual length, in μm, of a net belonging to the corresponding detour range,

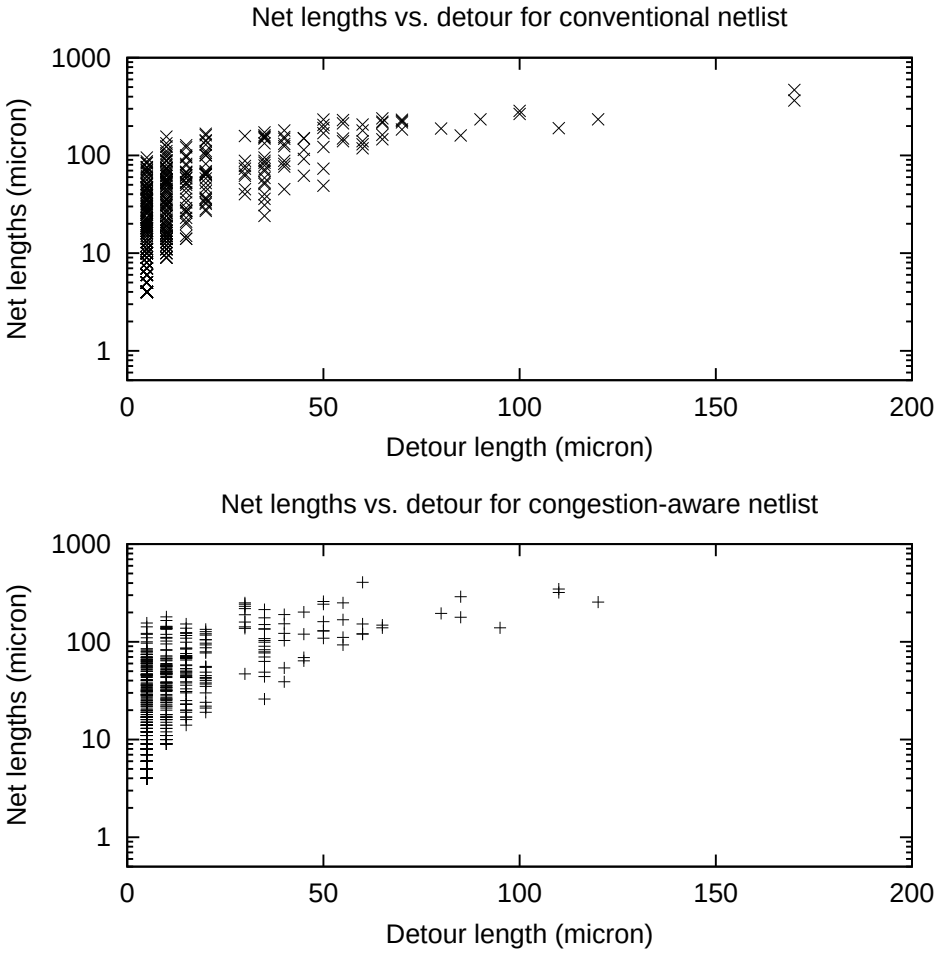


Figure 4.14: Scatter plots of net-lengths vs. detour length (μm) for the IDC circuit. In these plots, 'x' and '+' denote a net in conventional and congestion-aware netlist, respectively.

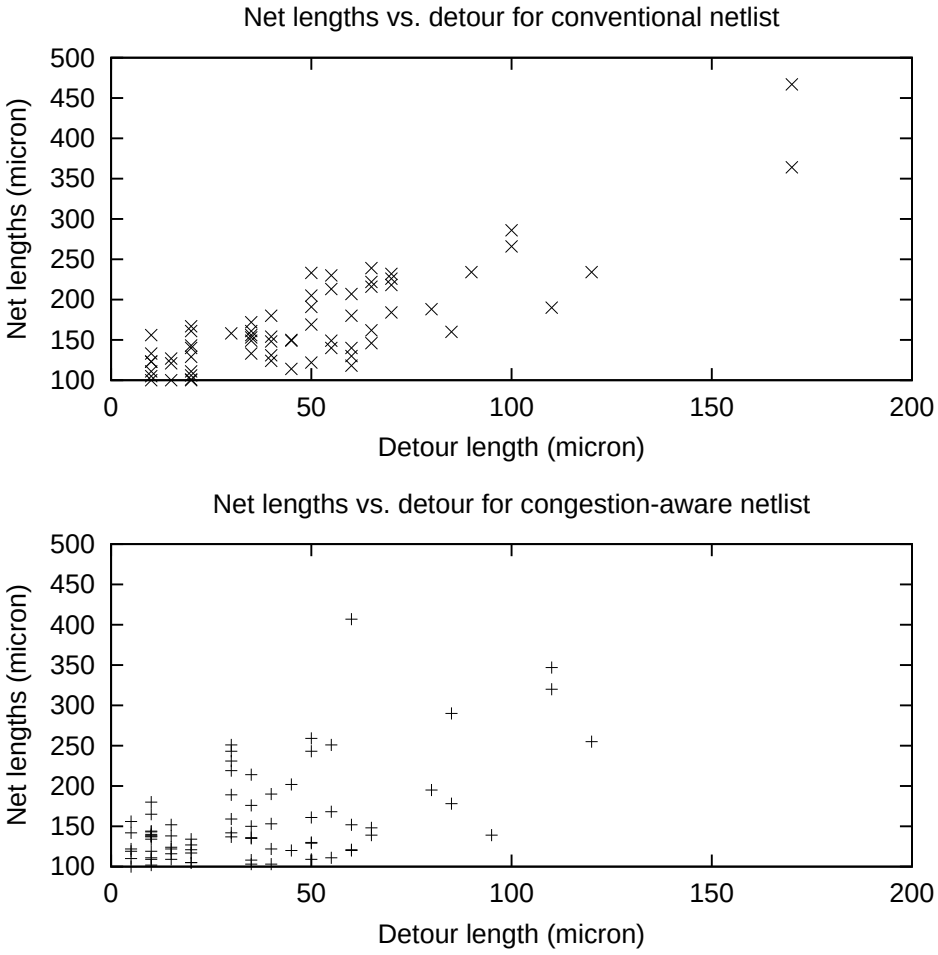


Figure 4.15: Scatter plots of net-length vs. detour length for long (> 100 μm) nets in the IDC circuit. In these plots, 'x' and '+' denote a net in conventional and congestion-aware netlist, respectively.

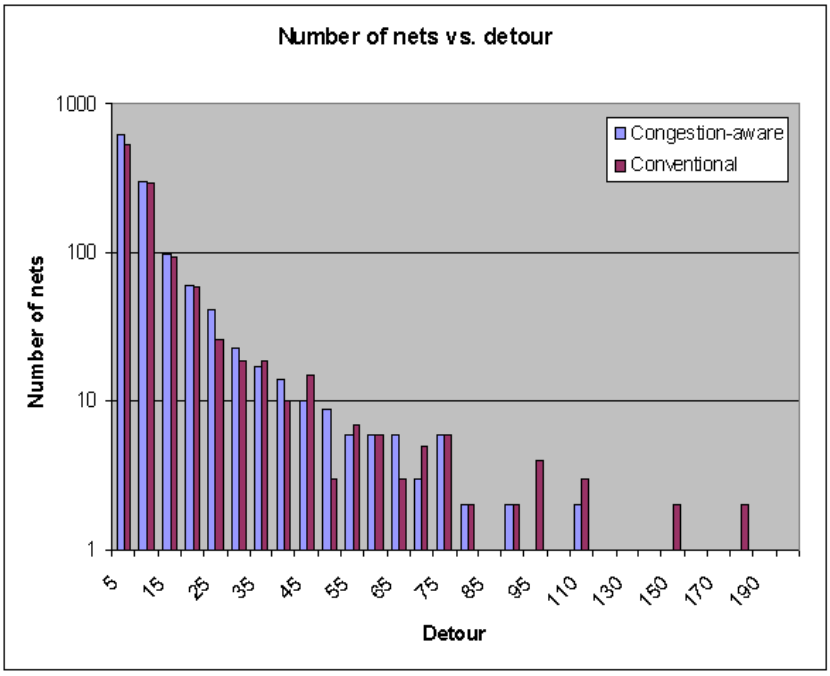


Figure 4.16: Number of nets vs. detour length (μm) for C7552: For congestion-aware mapping, the placement of premapped netlist is carried out using Kraftwerk. An in-house library for a 90nm technology is employed for congestion-aware as well as conventional mapping. The mapped netlists are placed employing Kraftwerk and routed using proprietary router.

in μm, specified on the X-axis, for the congestion-aware and conventionally mapped netlist, respectively. In the figure, a '$\times$' corresponding to 230 μm on the Y-axis and in the column for 70 μm on the X-axis implies that there is a net of length 230 μm whose detour length lies between 67.5 to 72.5 μm in the conventional netlist. It can be seen from the figure that the conventional netlist tends to have longer detours than the congestion-aware netlist, especially on its longer wires. The congestion-aware technology mapping not only tends to reduce the length of the long wires, but also tends to route them with smaller detours (hence, making them more predictable prior to the routing).

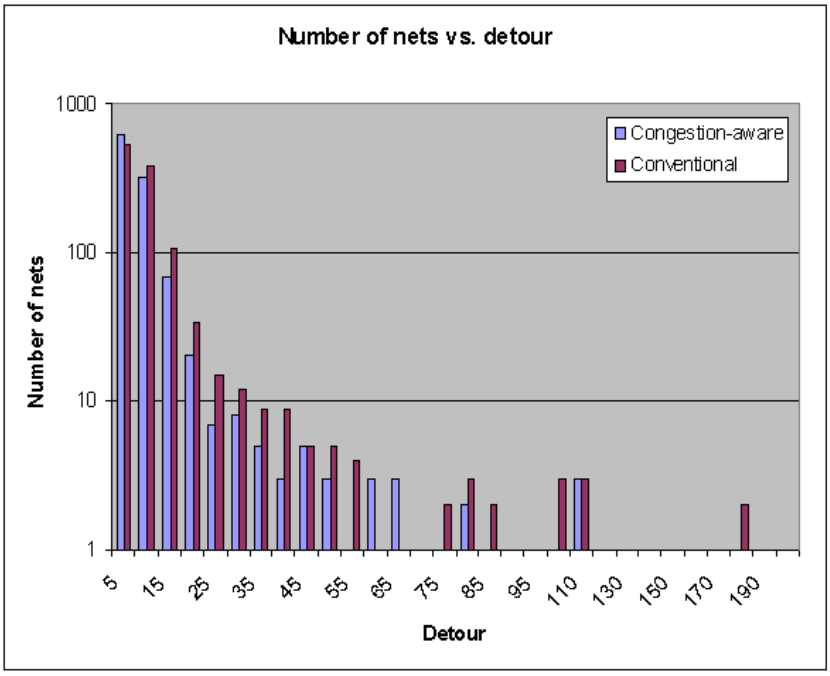


Figure 4.17: Number of nets vs. detour length (μm) for C6288: For congestion-aware mapping, the placement of premapped netlist is performed using Kraftwerk. An in-house library for a 90nm technology is employed for congestion-aware as well as conventional mapping. The mapped netlists are placed employing Kraftwerk and routed using proprietary router.

Figure 4.15 shows the nets whose length is greater than 100 μm, since these are the nets that are usually responsible for the routing problems; '+' and '$\times$' have the same meaning as in Figure 4.14. Congestion-aware mapping tends to reduce the length of the longest wires, as is apparent from a larger population of '$\times$' as compared to '+' in the figure. This is achieved by allowing the shorter wires to have slightly longer detours as compared to conventional mapping. However, since the predictability of the short wires is usually not a problem, the increased detours of the short wires do not impact the design convergence adversely. Furthermore, the reduction in the detours of the wires

under congestion-aware mapping also improves the predictability of their length, delay, load, and repeater requirements prior to routing.

The wirelength distribution trends for large benchmarks C7552 and C6288 are shown in Figure 4.16 and 4.17, respectively. The heights of brown and purple bars in these plots have the same meanings as before. The sets of congestion-aware mapped netlists are generated for these benchmarks using Kraftwerk for the placement of the premapped netlists. Technology mapping, conventional as well as congestion-aware, is performed using a 90nm technology library used in high-performance microprocessor designs. The final placement of mapped netlists, conventional and congestion-aware, is carried out employing Kraftwerk, while routing is performed using a proprietary maze router. We can see from Figure 4.16 and 4.17 that as detour length increases, the number of wires in conventional netlist starts dominating the corresponding numbers in the congestion-aware netlist. The length of the longest wires in conventional netlist is also large as compared to that of the longest wire in congestion-aware netlist. The number of nets in congestion-aware netlist dominate their counterparts in conventional netlists only for the short detours. This is due to increase in the number short wires in case of congestion-aware netlist. Thus, congestion-aware mapping has been able to improve the wirelength distribution by trading off the detours of long wires with smaller wires. In these cases, congestion-aware mapping has not only improved the total wirelength but the length of the longest wire also.

4.7.4 Conclusions

The following conclusions may be drawn from the experimental results.

1. The congestion-aware algorithms for area and delay-oriented technology map-

ping show consistent improvements in track overflows over conventional mapping methods. The improvement is significant: 37% in case of area-oriented mapping and 46% in case of delay-oriented mapping. These results indicate that technology mapping is indeed effective in handling routing congestion.

2. The consistency in the results also indicate that our heuristics are effective. More importantly, it also validates a point that there exists a strong congestion correlation between premapped and mapped netlists and justifies the use of congestion map prediction employing premapped netlists to guide the mapping process.

3. As compared to conventionally mapped netlists, congestion-aware netlists tend to have better wirelength distribution: typically, the length of the longest wire is shorter and the number of nets with long detours are smaller.

4.8 Summary

In this chapter, we have proposed technology mapping algorithms for alleviating the routing congestion. These algorithms employ a predictive congestion map based on the premapped netlists. Using empirical data, we have shown that there exists a strong correlation between the predictive congestion map based on a premapped netlist and the congestion map of the corresponding mapped netlist. This empirical evidence is utilized to justify the use of predictive congestion maps to guide the technology mapping algorithms. These algorithms employ congestion cost functions such that in sparsely congested regions, area- or delay-optimal matches are always chosen and hence, the corresponding penalty is minimized. Experimental results due to these algorithms show average improvements of 37% and 46% in track overflows, respectively, for area and

delay-oriented mappings, over conventional methods with marginal gate area and delay penalty. Moreover, congestion-aware netlists tend to have better wirelength distributions as compared to their conventional counterparts.

Example	script/mapping	# Cells	Placer	congestion after/before mapping				Correlation	
				Max. H	Max. V	Ave. H	Ave. V	H	V
C432	rugged/area	257	Capo	1.27/1.45	1.46/1.99	0.41/0.47	0.48/0.65	0.91	0.90
C432	rugged/delay	328	Capo	1.17/1.45	1.51/1.99	0.39/0.47	0.46/0.65	0.96	0.95
C432	algebraic/area	237	Capo	1.22/1.15	1.38/1.6	0.37/0.35	0.44/0.51	0.97	0.96
C432	algebraic/delay	279	Capo	1.06/1.15	1.21/1.6	0.35/0.35	0.40/0.51	0.93	0.93
C432	boolean/area	375	Capo	1.04/1.55	1.42/1.68	0.43/0.45	0.51/0.67	0.95	0.94
C432	boolean/delay	501	Capo	1.47/1.41	1.46/1.51	0.54/0.50	0.63/0.70	0.93	0.93
C432	speedup/area	265	Capo	1.08/1.22	1.25/1.5	0.34/0.41	0.40/0.55	0.92	0.91
C432	speedup/delay	314	Capo	1.03/1.29	1.27/1.81	0.37/0.52	0.44/0.67	0.93	0.94
C6288	rugged/area	2311	Capo	1.73/1.34	1.88/2.00	0.69/0.57	0.81/0.82	0.85	0.86
C6288	rugged/delay	2383	Capo	1.45/1.34	1.75/2.00	0.61/0.57	0.71/0.82	0.86	0.87
C6288	algebraic/area	2275	Capo	1.37/1.79	1.55/2.20	0.50/0.73	0.60/0.98	0.76	0.78
C6288	algebraic/delay	2620	Capo	1.38/1.05	1.59/1.31	0.48/0.52	0.58/0.73	0.83	0.79
C6288	boolean/area	2329	Capo	0.89/0.85	1.05/1.32	0.40/0.40	0.48/0.66	0.75	0.71
C6288	boolean/delay	2605	Capo	1.38/1.23	1.53/1.72	0.47/0.48	0.56/0.70	0.79	0.79
C6288	speedup/area	4182	Capo	1.11/1.10	1.34/1.39	0.41/0.48	0.51/0.66	0.78	0.81
C6288	speedup/delay	4395	Capo	1.19/1.20	1.47/1.58	0.48/0.51	0.58/0.63	0.86	0.82
C7552	algebraic/area	1521	Kraftwerk	2.60/2.70	2.70/2.40	0.61/0.71	0.66/0.71	0.81	0.76
C7552	rugged/area	2060	Kraftwerk	2.04/2.05	2.27/2.26	0.65/0.69	0.71/0.79	0.64	0.68
C7552	boolean/area	1582	Kraftwerk	2.23/2.50	2.50/2.00	0.61/0.74	0.66/0.71	0.82	0.83
C7552	espresso/area	1457	Kraftwerk	1.68/2.10	1.85/2.20	0.64/0.69	0.69/0.79	0.73	0.65
C6288	algebraic/area	2528	Kraftwerk	1.60/1.48	1.05/1.35	0.52/0.61	0.58/0.64	0.77	0.76
C6288	rugged/area	2391	Kraftwerk	1.50/2.00	2.00/2.00	0.53/0.62	0.58/0.63	0.63	0.62
C6288	boolean/area	2583	Kraftwerk	1.49/1.79	1.61/1.82	0.47/0.54	0.53/0.57	0.64	0.70
C6288	espresso/area	2549	Kraftwerk	1.76/1.79	2.06/2.09	0.52/0.62	0.59/0.66	0.61	0.64
IDC	rugged/area	972	Kraftwerk	1.25/1.30	1.13/1.47	0.65/0.60	0.60/0.65	0.67	0.68
IDC	algebraic/area	800	Kraftwerk	2.09/1.67	2.06/1.80	0.50/0.47	0.53/0.45	0.70	0.61
IDC	boolean/area	1622	Kraftwerk	1.75/1.78	1.52/1.23	0.57/0.59	0.64/0.65	0.67	0.66
IDC	espresso/area	2233	Kraftwerk	1.89/1.93	2.17/2.24	0.51/0.55	0.56/0.55	0.75	0.74

Table 4.1: Congestion comparison for the netlists before and after technology mapping. Max. (Ave.) corresponds to maximum (average), while H (V) corresponds to horizontal (vertical).

Example	script/mapping	# Cells	Placer	congestion after/before mapping				Correlation	
				Max. H	Max. V	Ave. H	Ave. V	H	V
9sym	rugged/area	314	Capo	1.40/1.28	1.68/1.29	0.29/0.32	0.32/0.34	0.79	0.82
9sym	rugged/delay	422	Capo	1.20/1.46	1.30/1.50	0.32/0.34	0.33/0.36	0.84	0.87
9sym	algebraic/area	283	Capo	1.05/1.02	0.99/.92	0.24/0.23	0.25/0.21	0.83	0.88
9sym	algebraic/delay	341	Capo	1.31/1.12	1.16/1.02	0.25/0.21	0.26/0.19	0.81	0.84
9sym	boolean/area	284	Capo	1.23/1.12	1.59/1.16	0.27/0.24	0.29/0.27	0.83	0.88
9sym	boolean/delay	397	Capo	1.46/1.16	1.62/1.30	0.31/0.29	0.29/0.29	0.78	0.80
rd84	rugged/area	406	Capo	1.23/1.14	1.11/1.07	0.29/0.26	0.31/0.25	0.76	0.82
rd84	rugged/delay	459	Capo	1.23/1.11	1.39/1.21	0.29/0.23	0.31/0.22	0.81	0.84
rd84	algebraic/area	672	Capo	1.39/1.19	1.46/1.25	0.38/0.28	0.41/0.37	0.79	0.86
rd84	algebraic/delay	680	Capo	1.62/1.51	1.81/1.59	0.45/0.40	0.41/0.41	0.79	0.78
rd84	boolean/area	559	Capo	1.36/1.27	1.38/1.28	0.37/0.36	0.40/0.35	0.81	0.81
rd84	boolean/delay	629	Capo	1.14/1.07	1.51/1.28	0.35/0.26	0.32/0.26	0.75	0.73
alu2	rugged/area	353	Capo	1.09/1.05	1.27/1.17	0.29/0.30	0.31/0.38	0.78	0.84
alu2	rugged/delay	454	Capo	1.32/1.11	1.34/1.28	0.32/0.34	0.34/0.35	0.76	0.75
alu2	algebraic/area	405	Capo	1.17/1.02	1.18/1.10	0.30/0.26	0.33/0.34	0.83	0.89
alu2	algebraic/delay	441	Capo	1.24/1.11	1.54/1.34	0.34/0.34	0.32/0.33	0.76	0.79
alu2	boolean/area	457	Capo	1.39/1.29	1.23/1.16	0.30/0.26	0.33/0.34	0.68	0.70
alu2	boolean/delay	579	Capo	1.43/1.33	1.49/1.41	0.32/0.32	0.34/0.31	0.71	0.67
C1355	rugged/area	356	Capo	1.23/1.32	1.58/1.39	0.50/0.37	0.55/0.38	0.82	0.86
C1355	rugged/delay	422	Capo	2.24/2.06	2.24/2.26	0.51/0.49	0.55/0.46	0.70	0.72
C1355	algebraic/area	602	Capo	1.49/1.28	1.70/1.32	0.42/0.37	0.45/0.36	0.76	0.78
C1355	algebraic/delay	638	Capo	1.32/1.03	1.23/1.21	0.46/0.41	0.48/0.41	0.79	0.85
C1355	boolean/area	601	Capo	1.04/1.21	1.09/1.00	0.40/0.36	0.44/0.37	0.76	0.74
C1355	boolean/delay	654	Capo	1.32/1.21	1.47/1.16	0.47/0.40	0.50/0.46	0.74	0.83

Table 4.2: Congestion comparison for the netlists before and after technology mapping. Max. (Ave.) corresponds to maximum (average), while H (V) corresponds to horizontal (vertical). For netlists of circuits from ISCAS'85 and MCNC benchmark suite, obtained using different scripts and mapping options, this table shows congestion correlation between mapped and corresponding premapped netlists. Similar results are shown in Table 4.3 for different benchmarks.

Example	script/mapping	# Cells	Placer	congestion after/before mapping				Correlation	
				Max. H	Max. V	Ave. H	Ave. V	H	V
C1908	rugged/area	386	Capo	0.98/1.10	1.10/1.33	0.37/41	0.43/43	0.83	0.84
C1908	rugged/delay	479	Capo	1.57/1.46	1.27/1.68	0.35/46	0.32/46	0.79	0.80
C1908	algebraic/area	342	Capo	1.25/1.15	1.55/1.30	0.42/0.40	0.45/0.41	0.88	0.90
C1908	algebraic/delay	580	Capo	1.57/1.08	1.48/1.16	0.47/0.42	0.51/0.46	0.81	0.82
C1908	boolean/area	569	Capo	1.54/1.37	2.03/1.55	0.45/0.39	0.49/0.39	0.82	0.84
C1908	boolean/delay	620	Capo	1.46/1.62	1.81/1.75	0.50/0.49	0.53/0.49	0.82	0.82
C499	rugged/area	391	Capo	1.28/1.28	1.26/1.60	0.43/0.40	0.46/0.42	0.80	0.78
C499	rugged/delay	402	Capo	1.70/1.50	1.92/1.51	0.51/0.49	0.54/0.50	0.63	0.64
C499	algebraic/area	593	Capo	1.01/1.15	1.13/1.24	0.40/0.31	0.42/0.33	0.74	0.82
C499	algebraic/delay	641	Capo	1.29/1.26	1.26/1.30	0.43/0.36	0.45/0.36	0.76	0.79
C499	boolean/area	611	Capo	1.09/1.21	1.20/1.22	0.35/0.33	0.37/0.33	0.77	0.77
C499	boolean/delay	642	Capo	1.75/1.62	1.26/1.30	0.42/0.36	0.44/0.36	0.69	0.69
C880	rugged/area	328	Capo	1.42/1.27	1.57/1.38	0.59/0.45	0.66/0.46	0.85	0.85
C880	rugged/delay	557	Capo	1.98/1.77	1.87/1.59	0.51/0.48	0.48/0.48	0.68	0.66
C880	algebraic/area	409	Capo	1.37/1.25	1.38/1.54	0.48/0.43	0.51/0.43	0.82	0.85
C880	algebraic/delay	410	Capo	1.49/1.13	2.03/1.65	0.43/0.37	0.51/0.48	0.81	0.82
C880	boolean/area	448	Capo	1.83/1.66	1.39/1.44	0.41/0.34	0.43/0.35	0.79	0.74
C880	boolean/delay	597	Capo	1.46/1.34	1.67/1.35	0.51/0.48	0.55/0.49	0.72	0.72
C7552	rugged/area	1930	Capo	1.84/1.39	1.48/1.41	0.55/0.39	0.50/0.37	0.66	0.64
C7552	rugged/delay	2688	Capo	1.61/1.42	1.31/1.36	0.46/0.47	0.48/0.50	0.63	0.66
C7552	algebraic/area	2378	Capo	2.03/2.17	2.14/2.32	0.76/0.68	0.74/0.71	0.77	0.78
C7552	algebraic/delay	2279	Capo	2.73/2.58	2.51/2.26	0.80/0.82	0.79/0.77	0.68	0.68
C7552	boolean/area	2321	Capo	2.48/2.17	2.98/2.32	0.81/0.82	0.86/0.77	0.71	0.70
C7552	boolean/delay	2735	Capo	2.59/2.48	2.67/2.49	0.86/0.83	0.78/0.74	0.64	0.65

Table 4.3: Congestion comparison for the netlists before and after technology mapping. Max. (Ave.) corresponds to maximum (average), while H (V) corresponds to horizontal (vertical). For netlists of circuits from ISCAS'85 suite, obtained using different scripts and mapping options, this table shows congestion correlation between mapped and corresponding premapped netlists.

Circuit	Area μm^2	Row utilization %	Overflow	Congested bins #	Congestion	
					Maximum	Average
C1355	2380	68 / 79	2 / 0	1 / 0	1.3 / 0.9	0.35 / 0.43
C1908	2457	68 / 78	0 / 0	0 / 0	0.8 / 0.9	0.34 / 0.40
C432	1728	66 / 69	1 / 0	1 / 0	1.1 / 0.9	0.35 / 0.37
C499	2618	64 / 73	0 / 0	0 / 0	0.9 / 1.0	0.34 / 0.40
C6288	16920	61 / 68	32 / 18	20 / 7	1.3 / 1.3	0.49 / 0.52
C7552	17633	61 / 67	655 / 461	258 / 193	1.3 / 1.3	0.65 / 0.69
C880	2534	71 / 82	4 / 1	2 / 1	1.3 / 1.2	0.42 / 0.48
IDC	6919	63 / 70	83 / 10	32 / 6	1.3 / 1.2	0.53 / 0.60
Average		65 / 73	97 / 61	39 / 25	1.16 / 1.08	0.43 / 0.48

Table 4.4: Comparison of conventional area-oriented mapping with congestion-aware area-oriented mapping. Placement and routing is performed using an in-house force-directed placer and a proprietary router, respectively, for a 90nm technology.

Example	Area	Row utilization	Delay	Maximum congestion	Overflow
	μ^2	%	ps		
C1355	6237	80 / 83	1061 / 1135	1.60 / 1.30	39 /18
C1908	7568	80 / 80	1388 / 1440	1.40 / 1.20	18 / 2
C432	2912	90 / 94	1222 / 1180	1.20 / 1.10	5 / 1
C499	6318	80 / 79	1040 / 1040	1.40 / 1.40	14 / 12
C6288	44099	80 / 80	7305 / 7078	1.50 / 1.20	16 / 8
C7552	47520	75 / 77	1755 / 1750	1.64 / 1.42	83 / 74
C880	9504	80 / 76	1276 / 1270	1.20 / 1.30	7 / 5
Avg. Improvement		2	0.42	-9	-46

Table 4.5: Comparison of conventional delay-oriented mapping and congestion-aware delay-oriented mapping. Placement and routing is performed employing a publicly available placer Capo [CKM00] and a router [HS02], respectively, for 130nm technology [ptm].

Chapter 5

Conclusions

In this thesis, we have proposed synthesis solutions related to the prominent challenges in nanometer technologies, namely, power density and interconnect dominance. For the former, synthesis and layout generation algorithms for low power pass transistor logic (PTL) are presented, while for the latter, congestion-aware technology mapping methods are proposed.

For performance-driven PTL synthesis, our recursive bipartitioning algorithm can result in a logarithmic depth implementation, while none of the previous synthesis heuristics guarantee such a lower bound on the depth of PTL implementation. By using max-flow min-cut formulation, we also ensure minimum area penalty, up to the accuracy in estimation, and the method can further be extended to consider other cost functions, such as power. A similar bipartitioning technique is presented to minimize the total power dissipation in pipelined combinational logic, when a low power PTL implementation is sought. As is well known, no logic style is a panacea and PTL is no exception. Our results on performance-driven PTL synthesis confirm that PTL results

in a significant improvement, up to 30% in area as well as delay, over static CMOS for *xor*-intensive circuits, such as those that implement multipliers or error correcting codes. In these cases, even under a näive uniform transistor sizing scheme, PTL outperforms static CMOS implementation with a more sophisticated sizing optimization. For circuits that are *nand*-intensive, static CMOS may result in better implementation than PTL. In such a scenario, static CMOS/PTL mixed synthesis is a viable approach. For that approach, our synthesis algorithms may still be useful for efficiently implementing the PTL part of the circuitry.

For layout generation of PTL circuits, we have proposed a method that translates binary decision diagrams (BDD's) into a transistor-level placed circuit that minimizes the area by diffusion-sharing, linear tree placement of transistor clusters and greedy row assignment of the BDD nodes. One of the advantages of this algorithm is that it is not tied to any particular PTL library, and it exploits the notion of librarylessness to form multiplexer clusters that are amenable to layout. Apart from their utility for pure PTL implementations, layouts generated by our algorithm can be used as PTL macro cells in the context of static CMOS/PTL synthesis, since these layouts can fit into a standard cell methodology.

For the routing congestion problem for static CMOS circuits, we present congestion-aware technology mapping methods. These methods are guided by a predictive probabilistic congestion map, unlike previous approaches that rely on indirect metrics such as wirelength. Using extensive experiments employing different benchmarks, libraries, and scripts, we have shown that there exists a correlation between the congestion maps of a pre-mapped netlist and a corresponding mapped netlist. The correlation is exploited to overcome the "chicken and egg" problem between the mapping and placement stages.

The matching phase in technology mapping uses these predictive congestion-maps to store congestion-aware choices. It employs a cost function that selects congestion-optimal matches in densely congested regions, while permitting the choice of area- or delay-optimal matches in sparsely congested regions. Similar approaches, based on this matching phase, have been applied for area and delay-oriented technology mapping algorithms. For area-oriented technology mapping, the experimental results using an industrial circuit and publicly available ISCAS'85 benchmark circuits, proprietary placers and routers, and a cell library in high-performance microprocessor design in 90nm technology show, on an average, 37% improvement in routability, measured in terms of track overflows, at the cost of marginal gate area increase. The results for delay-oriented mapping show 46% improvement in routability for approximately unchanged delays.

5.1 Future Directions

The cost effectiveness of PTL should be exploited for the implementation of *xor*-intensive functions, such as circuits for error correcting codes and arithmetic circuits. ASIC designers have realized this, and the use of PTL cells in the libraries has been on rise [BHSA03]. However, current approaches are ad-hoc, and algorithms for mixed static CMOS and PTL, which use the best of both the logic styles, are required. PTL synthesis algorithms presented in this thesis may be employed inside the inner loop of those algorithms to yield, at least locally, optimum PTL implementation. The layout generation method for PTL presented in this thesis may also be used to guide these algorithms, since it can provide accurate, up to the placement level, area and delay estimates for PTL choices. The layouts may also be generated automatically for these

mixed circuits, since layouts for standard cells are available and those for PTL can be generated using our algorithm.

There is scope for improvement in case of congestion-aware mapping algorithms: in the current variants, the matching phase is congestion-aware, while the covering phase uses a conventional method and does not try to reduce the congestion actively. In the covering phase, wire planning for non-critical nets may be employed for congestion alleviation. For interconnect-aware technology independent logic synthesis, our result on congestion correlation is useful, since it will allow an extension of the probabilistic metric to even higher level of abstraction. Logic synthesis transforms, such as *collapse* that remove a node from the fanin, may then be made congestion-aware; a collection of such congestion-aware transforms may lead eventually to a synthesis package that is interconnect-aware. Such a synthesis system is desired, as interconnect dominance increases with the technology scaling, since today's synthesis algorithms that minimize interconnect-unaware metrics such as literals and levels of logic are inadequate and may, in fact, lead to design points that do not satisfy timing constraints and are difficult to route.

Another sub-100nm technology effect related to interconnects is that of repeaters. Traditionally, repeaters have been ignored during synthesis and considered only during post routing optimizations to speed-up the interconnects. In technologies beyond 100 nm, where interconnect delays dominate, ignoring the area and power cost of repeaters during logic synthesis will inherently lead to sub-optimal netlists requiring design iterations. As technology progresses, the power dissipated by global interconnects also increases, even with copper interconnects [KCS02]. This is because, the resistivity of copper interconnect increases due to surface scattering effect and copper bar-

rier effect, even with the barrier deposition techniques like Atomic Layer Deposition (ALD) [KCS02], which is one of the most effective techniques for lowering the resistivity. To overcome this, wider wires and/or repeaters are required to speed-up the interconnects as the technology progresses, which results in an increased power dissipation. It is estimated that at the 50 nm node, for a typical microprocessor chip, the power dissipation due to buffers for copper interconnects will be 60 W [KCS02], while the entire chip will dissipate about 170 W. In such a scenario, producing a netlist that minimizes the number of buffers required in a synthesizable block is equivalent to, a certain extent, minimizing power dissipation. Interconnects (and hence, repeaters) are strongly affected by the logic synthesis optimizations, but logic synthesis tools still rely on the traditional metrics such as gate area, fanout load, etc. and do not consider interconnects and repeaters [SMCK03]. Logic synthesis transformations affect wires, and hence, repeaters. For instance, the *eliminate* transform [Sen92] that removes a node with a value that is less than certain threshold from the network is equivalent to eliminating a wire; the value is measured in terms of literal gains, and interconnect delay and overheads due to repeaters are ignored. Making such transforms aware of the effects that they produce will lead to synthesis algorithms that are interconnect-aware and are able to cope up with nanometer technology challenges.

BIBLIOGRAPHY

[AMD+94] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou. Precomputation-Based Sequential Logic Optimization for Low Power. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 74–81, November 1994.

[Ash57] R. L. Ashenhurst. The decomposition of switching functions. In *Proceedings of the International Symposium of Theory of Switching*, volume 1, pages 74–116, April 1957.

[BF98] J. L. Burns and J. A. Feldman. C5M–A control-logic layout synthesis system for high-performance microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated circuits and Systems*, 17(1):14–23, January 1998.

[BHSA03] C. Bittlestone, A. Hill, V. Singhal, and N. V. Arvind. Architecting ASIC libraries and flows in nanometer era. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 776–781, June 2003.

[BNNSV97] P. Buch, A. Narayan, A. R. Newton, and A. Sangiovanni-Vincentelli. Logic synthesis for large pass transistor circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 663–670, November 1997.

[Bor00] S. Borkar. Obeying Moore's law beyond 0.18 micron. In *Proceedings of the IEEE International ASIC/SOC Conference*, pages 26–31, September 2000.

[CK90] P. K. Chan and K. Karplus. Computing signal delay in general RC networks by tree/link partitioning. *IEEE Transactions on Computer-Aided Design of Integrated circuits and Systems*, 9(8):898–902, June 1990.

[CK00] D. G. Chinnery and K. Keutzer. Closing the gap between ASIC and Custom: An ASIC perspective. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 637–642, June 2000.

[CKM00] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Can recursive bisection alone produce routable placements? In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 477–482, June 2000.

[CLAB98] R. Chaudhry, T. Liu, A. Aziz, and J. Burns. Area oriented synthesis for pass transistor logic. In *Proceedings of IEEE International Conference on Computer Design*, pages 160–167, October 1998.

[CLR98] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. Prentice-Hall India, New Delhi, India, 1998.

[CP92] K. Chaudhary and M. Pedram. A near optimal algorithm for technology mapping minimizing area under delay constraints. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 492–498, June 1992.

[CS00] P. Christie and D. Stroobandt. The interpretation and application of Rent's rule. *IEEE Transactions on Very Large Scale Integrated Systems*, 8(6):639–648, December 2000.

[CW97] J. F. Croix and D. F. Wong. A fast and accurate technique to optimize characterization tables for logic synthesis. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 337–340, June 1997.

[DDT78] M. Davio, J-P. Deschamps, and A. Thayse. *Discrete and Switching Functions*. McGraw-Hill, Berkshire, UK, 1978.

[DS01] M. H. DeGroot and M. J. Schervish. *Probability and Statistics*. Addison Wesley, Boston, MA, 3rd edition, 2001.

[DY96] M. P. Desai and Y. T. Yen. A systematic technique for verifying critical path delays in a 300 MHz Alpha CPU design using circuit simulation. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 125–130, June 1996.

[EJ98] H. Eisenmann and F. M. Johannes. Generic global placement and floorplanning. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 269–274, June 1998.

[Elm48] W. C. Elmore. The transient response of damped linear networks with particular regard to wideband amplifiers. *Journal of Applied Physics*, 19(2):55–63, January 1948.

[FMM+98] F. Ferrandi, A. Macii, E. Macii, M. Poncino, R. Scarsi, and F. Somenzi. Symbolic algorithms for layout oriented synthesis of pass transistor logic circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 235–241, November 1998.

[GKSV01] W. Gosti, S. R. Khatri, and A. L. Sangiovanni-Vincentelli. Addressing timing closure problem by integrating logic optimization and placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 224–231, November 2001.

[GNBSV98] W. Gosti, A. Narayan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Wireplanning in logic synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 26–33, November 1998.

[GR01] P. Gopalakrishnan and R. Rutenbar. Direct transistor-level layout for digital blocks. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 577–584, November 2001.

[HS02] J. Hu and S. Sapatnekar. A timing-constrained simultaneous global routing algorithm. *IEEE Transactions on Computer-Aided Design of Integrated circuits and Systems*, 21(9):1025–1036, September 2002.

[ITN+00] T. Inukai, M. Takamiya, K. Nose, H. Kawaguchi, T. Hiramoto, and T. Sakurai. Boosted gate mos (BGMOS): Device/circuit cooperation scheme to achieve leakage-free giga-scale integration. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 409–412, May 2000.

[ITR01a] International technology roadmap for semiconductors, 2001 edition: Design. `http://public.itrs.net/Files/2001ITRS/Design.pdf`, 2001.

[ITR01b] International technology roadmap for semiconductors, 2001 edition: Interconnect. `http://public.itrs.net/Files/2001ITRS/Interconnect.pdf`, 2001.

[JSB98] Y. Jiang, S. S. Sapatnekar, and C. Bamji. A fast global gate collapsing technique for high performance designs using static CMOS and pass transistor logic. In *Proceedings of the IEEE International Conference on Computer Design*, pages 276–281, October 1998.

[KCS02] P. Kapur, G. Chandra, and K. C. Saraswat. Power estimation in global interconnects and its reduction using a novel repeater insertion methodology. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 461–466, June 2002.

[Keu87] K. Keutzer. DAGON: Technology Binding and Local Optimization by DAG Matching. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 341–347, June 1987.

[KSD02] P. Kudva, A. Sullivan, and W. Dougherty. Metrics for structural logic synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 551–556, November 2002.

[LAB99] T. Liu, A. Aziz, and J. Burns. Performance driven synthesis for pass transistor logic. In *Proceedings of the VLSI Design Conference*, pages 372–377, January 1999.

[LKTD01] P. Lindgren, M. Kerttu, M. Thornton, and R. Drechsler. Low Power Optimization Technqiue for BDD Mapped Circuits. In *Proceedings of the*

Asia South Pacific Design Automation Conference, pages 615–621, January 2001.

[LPP96] Y-T. Lai, K-R. R. Pan, and M. Pedram. OBDD-based function decomposition: algorithms and implementation. *IEEE Transactions on Computer-Aided Design of Integrated circuits and Systems*, 15(8):977–990, August 1996.

[LR71] B. S. Landman and R. L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Transactions on Computers*, C-20:1469–1479, 1971.

[LTKS02] J. Lou, S. Thakur, S. Krishnamoorthy, and H. S. Sheng. Estimating routing congestion using probabilistic analysis. *IEEE Transactions on Computer-Aided Design of Integrated circuits and Systems*, 21(1):32–41, January 2002.

[MBIS01] M. Munteanu, I. Bogdan, P. Ivey, and L. Seed. Single-ended pass transistor loic (SPL) - A design handbook. `http://www.shef.ac.uk/eee/esg/lowpower/pdf-papers/d4.5.pdf`, 2001.

[MBM01] L. Macchiarulo, L. Benini, and E. Macii. On-the-fly layout generation for PTL macrocells. In *Proceedings of Design Automation and Test in Europe*, pages 546–551, March 2001.

[Moo65] G. E. Moore. Cramming more components onto integrated circuits. In *Electronics Magazine*, volume 38, pages 114–117, April 1965.

[Mos] MOSIS Parametric Test Results for TSMC 0.25μ CMOS Runs. http://www.mosis.org/cgi-bin/cgiwrap/umosis/swp/params/ tsmc-025/t04r-params.txt.

[NDH98] N. Nassif, M. P. Desai, and D. H. Hall. Robust elmore delay models suitable for full chip timing verification of a 600 MHz CMOS microprocessor. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 230–235, June 1998.

[Ous] J. Ousterhout. MAGIC: An interactive layout editor. `http: bwrc.eecs.berkeley.edu/Classes/IcBook/magic/ index.html`.

[Ous85] J. K. Ousterhout. A switch-level timing verifier for digital MOS VLSI. *IEEE Transactions on Computer-Aided Design of Integrated circuits and Systems*, 4(3):336–349, July 1985.

[PPS02] D. Pandini, L. T. Pileggi, and A. J. Strojwas. Congestion aware logic synthesis. In *Proceedings of Design Automation and Test in Europe*, pages 664–671, March 2002.

[PPS03] D. Pandini, L. T. Pileggi, and A. J. Strojwas. Global and local congestion optimization in technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits aand Systems*, 22(4):498–505, April 2003.

[PR90] L. T. Pillage and R. A. Rohrer. Asymptotic waveform evaluation for timing analysis. *IEEE Transactions on Computer-Aided Design of Integrated circuits and Systems*, 9(4):352–366, April 1990.

[PS98] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, New York, NY, 1998.

[ptm] Berkeley predictive technology model. `http://www-device.eecs.berkeley.edu/~ptm/download.html`.

[Rab00] J. M. Rabey. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall India, New Delhi, India, Second edition, September 2000.

[RRAR97] A. Reis, R. Reis, D. Auvregne, and M. Robert. The library free technology mapping problem. In *Workshop notes of the IEEE/ACM International Workshop on Logic and Synthesis*, May 1997.

[RS99] M. A. Riepe and K. A. Sakallah. Transistor level micro-placement and routing for two-dimensional digital vlsi cell synthesis. In *Proceedings of the ACM International Symposium on Physical Design*, pages 74–81, April 1999.

[RSL+99] S. Ruan, R. Shang, F. Lai, S. Chen, and X. Huang. A Bipartition-Codec Architecture to Reduce Power in Pipelined Circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 84–90, November 1999.

[RSLT01] S. Ruan, R. Shang, F. Lai, and K. Tsai. A Bipartition-Codec Architecture to Reduce Power in Pipelined Circuits. *IEEE Transactions on Computer-*

Aided Design of Integrated circuits and Systems, 20(2):343–349, February 2001.

[Rud93] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 42–47, June 1993.

[Sas00] T. Sasao. *Switching Theory for Logic Synthesis*. Kluwer Academic Publishers, Boston, MA, 2000.

[SB00] C. Scholl and B. Becker. On the generation of multplexer circuits for pass transistor logic. In *Proceedings of Design Automation and Test in Europe*, pages 372–378, March 2000.

[Sen92] E. M. Sentovich. SIS: A system for sequential circuit synthesis. Memorandum No. UCB/ERL M92/41, May 1992.

[SIS99] L. Stok, M. A. Iyer, and A.J. Sullivan. Wavefront technology mapping. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 531–536, November 1999.

[SK92] S. S. Sapatnekar and S. M. Kang. *Design Automation of Timing-Driven Layout Synthesis*. Kluwer Academic Publishers, Boston, MA, 1992.

[SK01] L. Stok and T. Kutzschebauch. Congestion aware layout driven logic synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 216–223, November 2001.

[SMCK03] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick. The scaling challenge: Can correct-by-construction design help? In *Proceedings of the ACM International Symposium on Physical Design*, pages 51–58, April 2003.

[Som] F. Somenzi. CUDD: CU Decision Diagram package, release 2.3.0. `http://vlsi.colorado.edu/~fabio/CUDD/`.

[spd] Scalable polynomial delay model. `http://www.synopsys.com/products/library/lib_comp_spdm.html`.

[SRY98] Y. Sasaki, K. Rikino, and K. Yano. ALPS: An automatic layouter for pass-transistor cell synthesis. In *Proceedings of the Asia South Pacific Design Automation Conference*, pages 227–232, February 1998.

[SS01a] R. S. Shelar and S. S. Sapatnekar. BDD decomposition for the synthesis of high performance PTL circuits. In *Workshop notes of the IEEE/ACM International Workshop on Logic and Synthesis*, pages 298–303, June 2001.

[SS01b] R. S. Shelar and S. S. Sapatnekar. Recursive bipartitioning of BDD's for performance driven pass transistor logic synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 449–452, November 2001.

[SS02a] R. S. Shelar and S. S. Sapatnekar. An efficient algorithm for low power pass transistor logic synthesis. In *Proceedings of the Asia South Pacific Design Automation Conference*, pages 87–92, January 2002.

[SS02b] R. S. Shelar and S. S. Sapatnekar. Efficient layout synthesis algorithm for pass transistor logic circuits. In *Workshop notes of the IEEE/ACM International Workshop on Logic and Synthesis*, pages 209–214, June 2002.

[SSSW04] R. S. Shelar, S. S. Sapatnekar, P. Saxena, and X. Wang. A predictive distributed congestion metric and its application to technology mapping. In *Proceedings of the ACM International Symposium on Physical Design*, pages 210–217, April 2004.

[TB99] R. Tavares and M. Berkelaar. Reducing switching activity in pass transistor circuits. In *Workshop notes of the IEEE/ACM International Workshop on Logic and Synthesis*, June 1999.

[WE94] N. H. E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design: A Systems Perspective*. Addison-Wesley, New York, NY, October 1994.

[Yan85] M. Yannakakis. A polynomial algorithm for the min-cut linear arrangement of trees. *Journal of the Association for Computing Machinery*, 32(4):950–988, October 1985.

[YC99] C. Yang and M. Ciesielski. BDD decomposition for efficient logic synthesis. In *Proceedings of the IEEE International Conference on Computer Design*, pages 626–631, October 1999.

[YSR96] K. Yano, Y. Sasaki, and K. Rikino. Top-down pass-transistor logic design. *IEEE Journal of Solid-State Circuits*, 31(6):792–803, June 1996.

[YYN+90] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi, and A. Shimizu. A 3.8ns CMOS 16 x 16 multiplier using complementary

pass transistor logic. *IEEE Journal of Solid State Circuits*, 25(2):388–395, April 1990.

[ZA98] H. Zhou and A. Aziz. Buffer Minimization in Pass Transistor Logic. In *Workshop notes of the IEEE/ACM International Workshop on Logic and Synthesis*, pages 105–110, May 1998.