UNIVERSITY OF MINNESOTA

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

Min Zhao

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

Professor Sachin S. Sapatnekar

_____

Name of Faculty Advisor

_____

Signature of Faculty Advisor

_____

Date

GRADUATE SCHOOL

Analysis and Optimization Problems in High Speed Circuits with Special Reference to Domino Logic and Supply Network Design

A THESIS

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

MIN ZHAO

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Sachin S. Sapatnekar, Advisor

DECEMBER 1999

Abstract

The thesis consists of five parts, the first four of which are on domino logic synthesis and optimization, while the last related to fast power/ground analysis technique.

First, the problem of technology mapping for domino logic is addressed. The domino logic family is non-inverting, and has flexible gate configurations with a large NMOS pull-down network. In this work, several efficient algorithms aimed at these special features are presented.

The non-inverting property of domino logic requires logic duplication for generating both the negative and positive signal phases, which results in an area overhead. In addition, the area cost of the negative and positive polarity can be significantly different. This area overhead can be reduced by selecting an optimal output phase assignment. In second part of the thesis, a 0-1 integer programming formulation for the output phase assignment problem is presented.

Given a combinational network and its timing specification, it is a non-trivial problem to decide which part should be implemented with domino logic and which part with static logic. The third part of the thesis handles this static-domino partitioning problem under timing constraints. The algorithm is extended to develop a method for partitioning domino logic into two phases, with inverters permitted between the two phases.

Next, we address a timing verification and sizing optimization tool for circuits containing mixed domino and static logic. After sizing, a realistic charge sharing estimation and correction procedure is proposed to avert potential errors.

Finally, a fast power/ground network analysis technique is presented. To handle the large size of power distribution networks in high performance designs, a fast hierarchy analysis method is presented and a port admittance matrix sparsing algorithm is proposed to speed up the hierarchical analysis method.

Acknowledgment

I would like to express my deep gratitude to my advisor and mentor, Professor Sachin Sapatnekar. He offered me enthusiastic support and guidance, and provided valuable suggestions and encouragement. I also thank Professors R. Janardan, E. Shragowitz, G. E. Sobelman and B. Vinnakota for their helpful comments.

I would like to express my sincere appreciation to Dr. Liang-Fang Chao, now at Cadence Design Systems, for introducing me to the field of Electronic Design Automation. I would like to acknowledge many useful discussions with Dr. Andrew Tickle, Dr. Cyrus Bamji, Dr. Hakan Yalcin, Dr. Mohammad Mortazavi, Christina Chen at Cadence Design System, and Dr. David Blaauw, Dr. Rajendran Panda, Tim Edwards, Rajat Chaudhry and others at Motorola.

Many thanks to fellow graduate students for making my stay pleasant and fruitful: Jianfeng Shi, Naresh Maheshari, Songhua Xu, Wendong Wang, Huibo Hou, Yanbin Jiang, Kishore V. Kasamsetty, Jiang Hu, Kaushik Gala, Mahesh Ketkar, Haihua Su, Suresh Raman, Arvind Karandikar, Raza ul Mustafa.

I am grateful to National Science Foundation, Semiconductor Research Cooperation for funding parts of my research; IEEE and University of Minnesota for providing financial support to attend conferences; and Cadence Design System and Motorola for providing me with the opportunity to work as a summer intern.

Finally I would like to thank my parents for their encouragement and support throughout the years ever since I was a child. They have not only given me life, but also encouraged me to do my best. They have been always there to share my happiness of success as well as depression of failure.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Our Research Goal

With the exponential scaling of feature sizes in Very Large Scale Integrated (VLSI) circuits, it is expected that over half a billion transistors will be integrated on a single chip with an operating frequency of $2-3$ GHz in the 70 nm technology by the year 2009 [1]. Because of this increase in the complexity, circuit designs require sophisticated Electronic Design Automation(EDA) tools capable of handling large and high speed circuits. Due to the increase in complexity and reduced time to market, designers cannot rely on their intuition to optimally design circuits of this complexity. Thus circuit analysis and optimization tools are indispensable for designers. In particular, high speed circuits require more careful design and therefore more efforts need be focused on the development of good analysis and optimization tools to support high performance circuit design. This work addresses two aspects of analysis and optimization problems in high speed circuits, namely, (1) synthesis and optimization of domino logic, and (2) fast analysis techniques for supply/ground networks.

Domino logic is an effective circuit configuration for implementing high-speed

logic designs. It has made important contributions in the design of low cycle time microprocessors and other high performance circuits [2–8]. As its use becomes more widespread, there is a growing need for good EDA synthesis and optimization tools to support this circuit style. The non-inverting nature of this logic family, and the fact that its behavior is tightly coupled with the clock scheme, necessitate a more complex synthesis flow than that for static logic. In this thesis, we develop automation tools for domino logic design. Specifically, a domino synthesis flow is suggested and several problems along such a domino synthesis flow are addressed.

The second issue addressed in this thesis is related to fast analysis techniques for power distribution networks. A robust power distribution network is vital to meeting performance guarantees and ensuring reliable operation of high speed circuits. Higher device densities and faster switching frequencies cause large switching currents to flow in the power and ground networks, which degrades performance and reliability. Excessive voltage drops (IR drop) in the power grid reduce switching speeds and noise margins of circuits and inject noise, possibly leading to functional failures. High average current densities lead to undesirable wearing out of metal wires due to electromigration. Therefore, it is an important task to analyze the power distribution network to verify whether specified objectives on IR drop and electromigration have been satisfied. The challenge in power distribution network analysis problem is to be able to handle the large size of a power/ground network. Today the power network of microprocessor contains millions of nodes, and in the near future, this will increase to the order of tens of millions. Solving a network of such a size requires a long time and almost exhausts the memory available to a contemporary workstation. In this thesis, we provide a solution to this problem.

1.2 Contributions

The major contributions of this thesis are:

- Various novel approaches for technology mapping for domino logic are presented. We first provide a parameterized library mapping algorithm. The algorithm is optimal for tree-by-tree mapping and has a computation time that is polynomial in terms of constraint size. A mapping method using DAG covering that permits the implicit duplication of logic nodes is then incorporated into the framework of parameterized library mapping. We suggest a synthesis procedure that maps the complementary logic cones independently when AND/OR logic is to be implemented, and together using dual-monotonic gates in the case of XOR/XNOR logic. In our mapper, specific consideration has been given to practical domino circuit design techniques, such as wide dynamic AND/OR gates and multiple-output gates. The results show large improvements over existing approaches. Moreover, the area cost of our domino implementation is better than or close to the cost of a static implementation, even through the non-inverting property of domino logic may require the duplication of numerous nodes of input network.

  In addition, the parameterized library mapping algorithm is extended to static technology mapping. The application of the algorithm provides speed-ups by factors of over a hundred in CPU time over the static solution provided by SIS, and some improvement in the area.

- A novel 0-1 integer programming formulation is provided for the output phase assignment problem for domino logic. It considers the cost difference between two polarities and enables a standard linear programming package to be used to solve the problem. The 0-1 ILP algorithm provides the optimal solution to the problem and results show large improvement in area.

- An algorithm for timing-driven partitioning of domino and mixed static-domino circuits is presented. In most circuits, it is appropriate to use domino gates to speed up parts of the circuit, while the remainder is implemented in static CMOS. An important problem is, therefore, to partition a circuit to determine which parts should be implemented as static logic, and which parts as domino logic, in conformance with the specified clock scheme. We present an efficient timing driven static-domino partitioning algorithm. Our algorithm finds a partition of a logic network between static and domino implementations that minimizes the cost, subject to timing constraints specified by the clocking scheme. This algorithm is extended to develop a method for partitioning domino logic into two phases, with inverters permitted between the two phases. Our partitioning algorithms are then applied to the most common two-phase clocking strategy. Our results show that the area of original domino network is reduced significantly, while maintaining the circuit speed of a pure domino implementation.

- A timing verification and sizing optimization tool dealing with circuits containing mixed domino and static logic is presented. A timing analysis methodology is developed, permitting the application of PERT and allowing mixed static-domino circuits to be handled in a manner similar to static combinational circuits. The optimization procedure preserves the requirements of maintaining adequate noise margins by constraining the sizing procedure. Finally, after sizing, the circuit is evaluated for charge sharing using a procedure that is more accurate and less pessimistic than previously proposed methods, and the circuit is corrected to eliminate any charge-sharing problems.

- A solution that permits fast analysis of power distribution networks is provided. With the increasing number of transistors on a chip, the size of power network analysis problem makes it difficult to solve in reasonable time without the use

of hierarchy. Moreover, the memory requirements may exceed the memory limits available for a contemporary workstation using a flat method. This work suggests a solution to this problem, where a hierarchical power network analysis method is presented, using macromodels to accurately capture the behavior of the partitioned blocks. A port admittance matrix sparsing method using a 0-1 integer linear programming formulation is suggested to maintain the sparsity of the macromodels, while controlling the error. The experiments on power networks with millions of nodes show significant reduction in the run time, and the memory requirements for power network analysis. The experimental results on matrix sparsing show the speed-up of the global solving of macromodel with negligible error.

Parts of this research have been published in [9–15].

1.3 Organization of this Thesis

This thesis is organized as follows.

Overview of domino logic In Chapter 2, we briefly describe the configurations, properties and general clocking schemes of domino circuits. In addition, a literature survey of recent research that relate to domino logic is included and a synthesis flow for domino logic is proposed.

Technology mapping algorithms for domino logic In Chapter 3, we study the problem of technology mapping for domino logic. An effective parameterized library (library-free) mapping algorithm for domino logic is presented. The mapping approaches for the different domino styles, such as dual-monotonic (dual-rail) gates, wide AND/OR gates and multiple-output gates are discussed. A static mapper based on the parameterized library algorithm is described.

0-1 programming output phase assignment problem The removal of intermediate inverters requires logic duplication for generating both the negative and positive signal phases due to the inherent non-inverting property of domino logic. The output phase assignment can reduce the logic duplication, and can also reduce the cost overhead due to differences in the implementation cost of the positive and negative phases. In Chapter 4, we consider the output phase assignment problem for domino logic and a 0-1 integer programming formulation for the output phase assignment problem is presented.

Timing-driven partitioning of domino and mixed static-domino circuits In Chapter 5, we discuss the issues of domino circuits partitioning, which includes static-domino partitioning problem and two-way domino partitioning problem. The static-domino partitioning problem is to decide which part of a given segment of combinational logic is to be implemented with static logic and which part to be implemented with domino logic so that the timing constraints are satisfied. The two-way domino partitioning problem is to partition a network implemented purely as domino logic into two phases with inverter permitted between two phases.

Timing verification and sizing optimization of mixed static-domino circuits In Chapter 6, we address the problem of timing verification and sizing optimization in circuits containing mixed domino and static logic. A charge sharing evaluation procedure and a method for correcting charge sharing problems are proposed.

Hierarchical analysis of power network distribution Chapter 7 describes the research on fast analysis of power distribution network. A hierarchical power network analysis method is presented. An algorithm that sparsifies the port admittance matrix using a 0-1 integer linear programming formulation is presented

to enhance the performance of hierarchical analysis method.

Conclusion In Chapter 8 we conclude this thesis and present a number of open problems on domino logic synthesis and optimization, as well as on fast power/ground analysis. We also present some ideas and thoughts on these problems. The problems related to domino logic include noise estimation and correction, low power domino logic design, clock network generation for domino logic circuits, domino logic cell synthesis, delay balanced partitioning and asynchronous domino techniques. The future work on power network analysis includes automatic partitioning techniques, port collapsing and hierarchical analysis for RLC networks.

Chapter 2

Overview of Domino Logic

In this chapter, we first briefly introduce the configurations, properties and clocking strategies of domino logic. Next, a survey of recent research on domino logic is described and finally, a special synthesis flow for domino logic is suggested.

2.1 Configurations and Properties

Domino logic is an effective circuit configuration for implementing high-speed logic designs. It is widely used in microprocessor and other high performance designs [2–8].

A representative domino gate configuration is shown in Figure 2.1. It consists of two clock controlled transistors, one pull-down NMOS network and one inverting static gate, which traditionally is an inverter. In some situation, the NMOS clock controlled transistor can be removed to obtain faster gate switch speed; this configuration is referred to as footless domino. When the clock input is low, the gate recharges the dynamic node d to logic 1. In the next half-cycle, when the clock goes high, the domino gate evaluates, i.e., the dynamic node either discharges or retains the precharged state, depending on the values of the input signals. The two-step mode

of operation with a precharge and an evaluate phase causes the timing relationships in domino logic to be more complex than those for static logic.

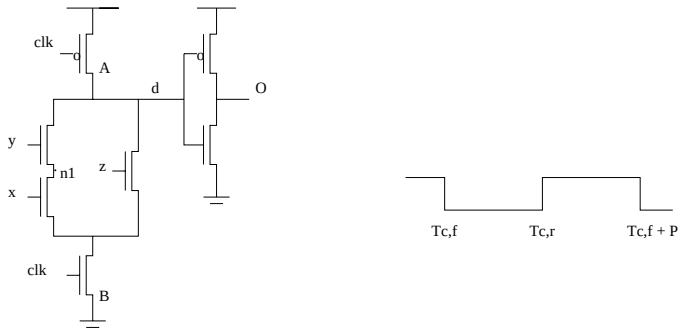

Figure 2.1: A typical domino circuit

Domino circuits offer the advantages of faster speed and glitch-free operation. This is because (1) There is no fighting during transitions since the clock transition is sharp (2) Instead of both NMOS and PMOS transistors at the fanout having to be driven as in static CMOS, only NMOS transistors need to be driven. (3) Large loads are driven through inverters, instead of series logic stacks. (4) Since only the evaluation speed is important to domino gates, the inverting static gate can be easily skewed to favor the critical monotonically rising evaluation edges. In summary, domino logic runs 1.5 to 2 times faster than static CMOS logic [16].

However, domino logic circuits have some drawbacks. They are susceptible to charge-sharing and noise, and all signals must satisfy strict timing constraints to maintain the functional correctness. In addition, they can only implement non-inverting logic. Hence, complements of internal signals must be realized through separate cones of logic using complements of the primary inputs. This results in a significant area overhead when both a signal and its complement must be generated since the fanin cone of the signal must be replicated, with the signal inversion implemented by using DeMorgan's laws to push the inverters to the primary inputs. This is referred to as logic duplication. Therefore, domino logic requires more careful design than its static

counterpart and currently most domino circuits are manually designed.

2.2 Clock Strategies

Domino logic is conventionally divided into multiple phases. Single-phase domino logic is impractical since precharging all domino gates simultaneously would result in wasted time during which no useful computation occurs. Depending on the source from which the precharge control signal is taken and the timing relation between control signal and logic signals, various clock strategies for domino implementations are defined. In the following, we will briefly introduce several strategies used in current domino logic designs.

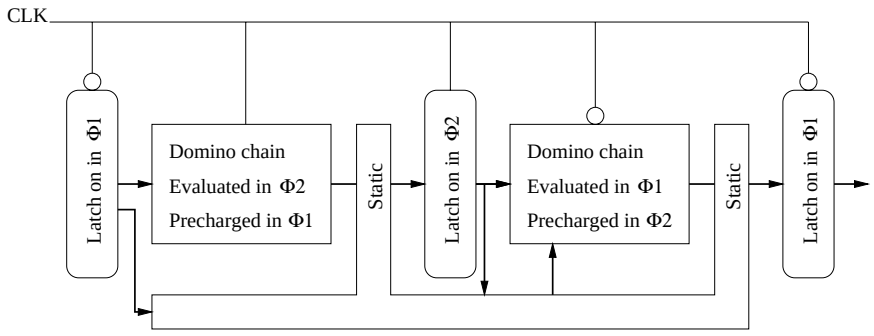

Figure 2.2: A common two-phase non-overlapping clocking scheme [17,18]

A common two-phase non-overlapping clocking scheme Since domino logic operates in alternating precharge and evaluate modes, it is frequently implemented with a two phase clock, as shown in Figure 2.2 [17, 18]. The logic between the latches may be static, domino or a mix of static and domino logic. This is the classical clocking strategy used with domino logic and has been widely employed in various designs due to its robustness. In the remainder of this thesis, when we refer to this as the two-phase clocking scheme, and we will denote the clock phases by the symbols Φ_1 and Φ_2.

Skew tolerant domino clocking scheme In the two-phase non-overlapping clocking strategy, the latch between two phases forms a hard edge that prevents time borrowing between the two phases. In addition, the clock skew and delay of latches may reduce the effective evaluation time. The work in [16] suggested a novel clocking scheme that can overcome the above problems and make the design more skew tolerant. In this clocking scheme, circuits are partitioned into several stages and an overlapping clock is applied to each stage. Figure 2.3 shows an example of a four-phase overlapping domino clock scheme. Each block consists of several levels of domino gates, or clocked buffers if no domino gates are available.

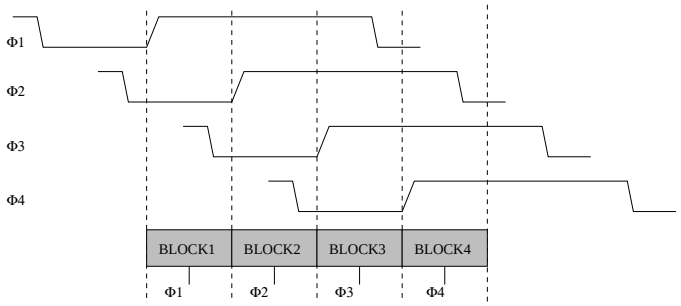

Figure 2.3: Four-phase overlapping clocking scheme [16]

Clock-delayed domino clocking scheme The work in [19] presents a clock-delayed domino clocking scheme. The method overcomes the non-inverting property of domino logic. The simplest clock scheme is shown in Figure 2.4. The delays, ϕ_i, corresponds to the largest delay of gates of stage i, and are synthesized by delay elements (PE) such as transmission gates.

Other clocking schemes include self-timed pipelines [17] and wave-domino pipeline [20]. Self-timed pipelined domino circuits are asynchronous circuits where the precharge signal is generated from the completion signals from neighboring stages. The wave-domino pipeline scheme is used to implement wave-pipelines with domino logic.

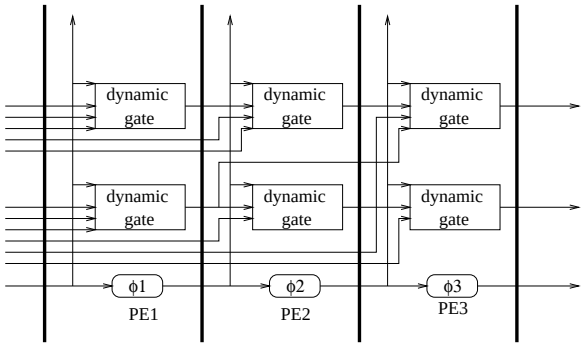

Figure 2.4: Clock-delayed domino clocking scheme [19]

2.3 Research on Domino Logic

In the recent years, with the wide use of domino logic in high performance designs, a significant amount of research has been carried out on domino logic. In this section, we present a brief literature survey of research related to domino logic in recent years, with emphasis on automation tools. A good introduction to domino logic can be found in [17].

Domino circuit design The area of domino circuit design has been widely researched. The research in [2–8], to name just a few, describes high performance circuit designs that predominantly contain domino-like dynamic logic. Enhancements to domino logic using multiple-output domino gates, and their application to adder circuits to improve their area and speed are discussed in [21, 22]. Various novel domino-like dynamic gate configurations are presented, for example in [23–25]. Specific applications include multiplier designs using mixed static/dynamic implementations [26].

Clocking schemes Several clocking strategies have been proposed for domino circuit, as summarized in Section 2.2. These are described in greater detail in [16, 17, 19, 20].

Timing issues Timing and functionality are closely coupled in domino logic circuits. The research in [27–29] addresses the timing constraints between logic signals and clocking signals for domino gates. In [18], a further description of timing constraints at the static-dynamic interface are presented.

Transistor Sizing Gate delays can be reduced significantly by transistor sizing. In [30], a transistor scaling procedure that considers individual domino logic gates is presented. The work in [31] solves the sizing problem of domino circuits without considering timing constraints of domino gate. In [32], the domino timing and sizing problem in PowerPC microprocessor is discussed.

Technology independent optimization and technology mapping In [33], a modified synthesis procedure that unates the circuit first and then performs unate technology independent optimization on the resulting network, is proposed. An approach to technology mapping for domino logic, using on-the-fly cell generation, as opposed to using a fixed cell library is suggested in the same work. While [19] discusses the synthesis of clock-delayed domino, the research in [34] extends this work to address the synthesis of the most general form of a domino gate, which consists of the pairing of a dynamic gate with any inverting static gate. In [35], a new approach to the problem of inverter elimination in domino logic synthesis is proposed.

Output phase assignment The area overhead caused by the non-inverting property of the domino logic can be substantially reduced by selecting an optimal output phase assignment. In [36], the output phase assignment problem is first addressed and the optimal and heuristic algorithms to minimize logic duplication are presented. The research in [37] illustrated the significant impact of output phase assignment on power consumption and an algorithm similar to [36] is applied to the objective of power minimization.

Partitioning Given a clocking frequency, it is a non-trivial to partition the design into static and dynamic implementation to maximize circuit timing performance. Section 3 of [18] provides a heuristic method to solve this problem. The circuits are first mapped to static logic and the outputs on the critical path are detected; next the entire fan-in cones of these outputs are replaced with domino logic.

Power Power is one main concern of contemporary circuit design. Compared with static logic, domino logic provides advantages since it has no spurious transitions, has smaller short-circuit currents and lower parasitic capacitances. However, the corresponding disadvantage is that it may waste more power since the switching activity is typically higher than that of a static implementation. In Section 3.A of [38], the power dissipation model of domino logic and the comparisons between domino and static logic are described. In [37], the problem of reducing the power consumption of domino circuits by output phase assignment is presented and an algorithm for this purpose is provided.

Noise Noise susceptibility is the major drawback of domino logic. Traditionally, weak feedbacks are added to the domino gates to overcome the problem. Choosing the size of feedback transistors involves a tradeoff between the noise margin and switching speed of domino gates. In [39], noise sensitivity and noise generation issues in dynamic circuits are discussed and the theory, simulation and measurements of the noise problem are covered. In [40], the growing problems of noise and the design techniques used to ensure the noise immunity are described.

Testing The techniques for testing multiple output domino logic CMOS circuits are presented in [41] and a new approach for detecting bridging faults in domino logic circuits is described in [42]. In [43], several types of faults that can occur

in domino logic are analyzed.

2.4 Synthesis Flow

Due to issues arising from the precharge and evaluation mechanism of domino logic and its non-inverting property, synthesizing domino logic is more complicated than standard two stage static logic synthesis methods that perform technology-independent optimization, followed by technology mapping.

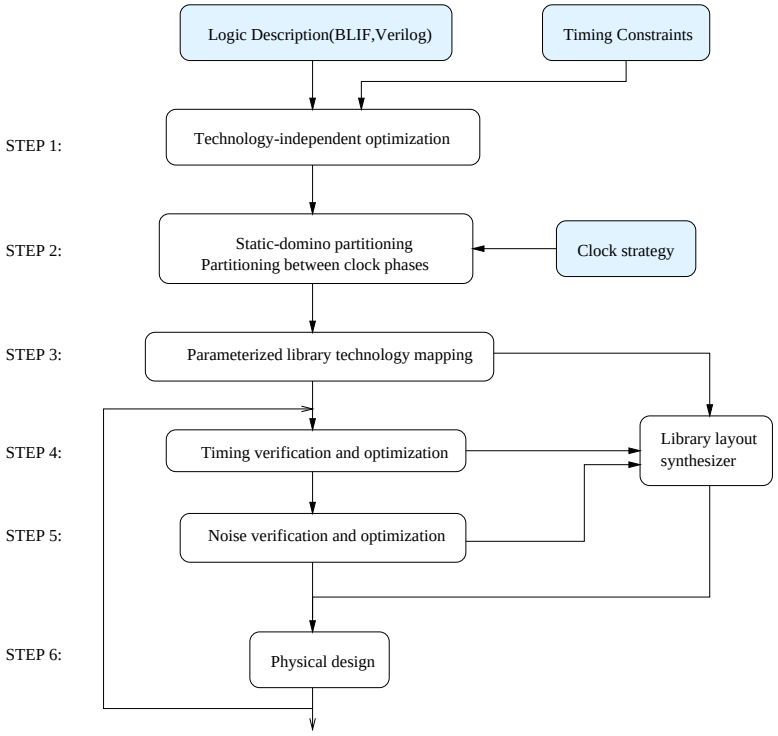

Figure 2.5: Domino logic synthesis flow

To build a high-performance domino circuit taking these factors into account, we suggest the synthesis flow of Figure 2.5 instead of a traditional library based synthesis flow. The first step corresponds to a standard technology-independent optimization. In step two, according to timing specifications, possible noise influences, and the area

or power cost function, the input network is partitioned into the part implemented as static logic and the part implemented using domino gates. If domino gates are used, the choice of a clock scheme, and the partitioning of domino gates between clock phases is essential for correct and efficient performance of domino gates.

In step three, a technology mapping method that considers the characteristics of domino logic is applied. Due to the large number of candidate domino gates, the use of a parameterized library and a library layout synthesizer is considered a good choice for this step [33,34]. In addition, an optimal output phase should be assigned to minimize the cost of technology mapping. This can serve as a pre-processing phase before technology mapping.

While coarse timing optimization measures may be built into the preceding steps, there is a need for a rigorous timing verification step, followed by timing optimization to meet the set of applied constraints. This is performed in step 4, where, for example, optimization methods such as transistor sizing are applied to correct to meet long path constraint violations.

After these procedures, noise verification and optimization, performed in step 5, is necessary. This is particularly important since domino circuits can be susceptible to noise effects such as charge sharing. This circuit is then passed on for physical design in Step 6, with iterations being performed until timing closure.

In this thesis, we develop a static/domino partitioning algorithm for step 2 of the flow, provide technology mapping algorithms of domino logic for step 3, and address issues related to Steps 4 and 5 of the flow by proposing a transistor sizing procedure with a post-processing step that performs noise verification and optimization due to charge sharing. In addition, a pre-process of step 3, output phase assignment problem is addressed.

Chapter 3

Technology Mapping Algorithms for Domino Logic

3.1 Introduction

Technology mapping is an optimization phase at the logic synthesis stage that binds the gates in the circuit network to the cells of the specific technology library. The objective of this chapter is to examine special properties and the flexible configurations of domino style gates and explore efficient technology mapping methods for domino logic.

Technology mapping based on a static standard cell library is a well established problem that is addressed in [44, 45] and further extended in [46–54]. Although technology mapping for domino logic follows the basic static technology mapping procedure of dynamic programming, special features of domino style gates such as large NMOS pull-down network, flexible gate configurations and logic duplication, demand further consideration. Compared to the mappers that simply migrate the algorithms for static logic, a procedure that makes use of these properties of domino

logic will significantly improve the performance in both area and delay.

In [33], a complex CMOS domino logic circuit synthesis flow for technology independent optimization and technology mapping of CMOS domino logic circuits was addressed. The work in [34] introduces the the general form of a domino gate (dynamic-static domino), which consists of the pairing of a dynamic gate with any inverting static gate, and describes a methodology for synthesizing circuits with dynamic-static domino. Other related work includes [18, 19, 34].

The chapter is organized as follows. In section 3.2, we present a parameterized library mapping algorithm and illustrate its application on domino logic. In section 3.3, we describe a greedy method that permits the duplication of input network nodes during technology mapping to obtain improved area and delay performance. Section 3.4 suggests a technology mapping methodology that maps dual-rail logic cones independently for AND/OR logic, and together for XOR/XNOR logic. The method of incorporating the dual-monotonic domino gates into the framework of parameterized library mapping is also described. Section 3.5 explores more flexible domino style gates and their technology mapping methods. Multiple-output domino gates and wide AND/OR domino gate are considered. Section 3.6 extends the parameterized library technology mapping algorithms to static logic and a more general library cost formula is considered. The static mapper applies the DAG mapping approach in Section 3.3 and a flexible phase assignment method. The results for the technology mapping methods described in sections 3.2-3.6 are presented in section 3.7. Our results are compared with previous work, static technology mapping results of SIS system, between each other to show the efficacy of the algorithms. Finally, we summarize and present some concluding remarks on section 3.8.

3.2 Parameterized Library Mapping Algorithm

3.2.1 Motivation and Previous Work

The concept of a parameterized library was originally proposed in [55] for static complex gate mapping. A parameterized library is defined as a collection of gates that satisfy the constraints on the width (maximal number of parallel chains) and height (maximal number of series chains) of the pull-down or pull-up implementations of a gate. Such a library is then used to map the input network nodes. The layout of the parameterized library is produced by an on-the-fly cell generator, instead of a fixed cell library.

The application of a parameterized library and a on-the-fly cell generator on domino logic technology mapping was suggested in [33], which further alluded to [56, 57]. This is particularly useful in the context of domino circuits since the small magnitude of the short circuit current and small driven load motivates the usage of large NMOS pull-down networks. With the increase in the size of NMOS network, the number of cells in a library increases super-exponentially [45]. In such a case, it is difficult to implement the flexible functionality of domino gates with fixed cell libraries since the number of required cells is prohibitively large. Hence, a parameterized library is an appropriate option for domino gate mapping.

In an environment that uses a parameterized library, we modify the definition of the domino technology mapping problem as: Given an optimized boolean network and constraints on the width and height of the domino gates, implement the nodes in the network with domino logic gates, such that the objective cost is minimized.

There are two existing technology mapping methods for parameterized library mapping. In [55], the Boolean net is decomposed into a multiple input AND-OR-NOT directed acyclic graph (DAG) network and a greedy method is used for node

mapping. In other words, at each mapping step, the work in [55] limits its view of a certain expression to one level and maps as many current level nodes as possible into a complex gate. In [33], all possible domino gates are enumerated at each node of the DAG. However, at a particular node, under the width and height constraints, the number of possible mapping schemes can be extremely large. Hence, recursively enumerating all possible solutions to obtain the optimal solution can be computationally expensive.

In this section, we will provide a new parameterized library mapping algorithm that provides an optimal mapping for a tree structure with a reasonable computational complexity.

3.2.2 The Algorithm

Given an arbitrarily optimized network, it is first unated [36], since domino circuits can only implement unate logic functions of the primary inputs. It is then mapped into a two-input AND-OR DAG network, after which the DAG network is decomposed into two-input AND-OR trees. This is the starting point of our algorithm, which is based on the traditional dynamic programming method [58].

3.2.2.1 Node Data Structure

At each tree node, a set of subsolution is stored, each of which is optimal for its subtree under some specified constraints. Specifically, these are the optimal subsolutions for all possible [height,width] combinations from [1,1] to [H,W] for the current gate. Therefore, there are a maximum of H $\times$ W optimal subsolutions that can be possibly stored for every node. Each optimal subsolution can be represented as $\{S, P, C, \{S_l, P_l\}, \{S_r, P_r\}\}$. Here, S $(1 \leq S \leq H)$ is the height constraint of the current node, P $(1 \leq P \leq H)$ is the width constraint of the current node and C is

the area cost. Unless otherwise stated, in this chapter, minimization of the area, measured in terms of the number of transistors, is the objective. Different combinations of the child node constraints can lead to the same parent node constraint, and of all these, $\{S_l, P_l\}$, $\{S_r, P_r\}$ is the combination that provides the minimal cost under the current constraints $\{S, P\}$.

Physically, $\{S, P\}$ represents a segment of a domino pull-down whose maximum height and width are S and P, respectively. The area cost is the accumulated area of its child transition cones, including the domino gate input cost and the number of transistors in the current domino circuit segment.

3.2.2.2 Node Constraint Functions

The computation of the height and width of a subsolution at a parent node, formed by combining subsolutions at child nodes, is illustrated here. Due to the series-parallel structure of the domino pull-down, we only need to consider two types of nodes: AND nodes and OR nodes. Here, we assume the height and the width, respectively, of the subject node to be given by the pair $\{S, P\}$. The constraint on its left child is $\{S_l, P_l\}$, while that on its right child is $\{S_r, P_r\}$. The AND and OR formulas are similar to [55].

1. OR operation: $S = \max(S_l, S_r)$, $P = P_l + P_r$
2. AND operation: $S = S_l + S_r$, $P = \max(P_l, P_r)$
3. gate formation operation: $S = 1$, $P = 1$

A gate formation operation corresponds to a situation where the structure collected so far is converted (during the dynamic programming procedure) to a domino gate with an output at that node.

3.2.2.3 Node Cost Functions

The area cost function for each of the above possibilities is as shown below.

1. literal operation: if **LITERAL** $C = C + 1$; else C is unchanged .
2. OR/AND operation: $C =$ literal operation(C_l) + literal operation(C_r)
3. gate formation operation: $C = C_{min} + 4$

In Case 1, the **LITERAL** corresponds to a primary input or a situation where the pull-down structure accumulated so far was consolidated into a domino gate and therefore a new domino pull-down structure will be started at this point. In this situation, the input transistor of the next level of domino gates adds one more transistor to the cost; otherwise, the node is an inner node the domino gate and no additional cost is required.

In Case 2, C_l is the optimal subsolution for the left child under the constraint $\{S_l, P_l\}$ while C_r is the optimal subsolution for the right child under the constraint $\{S_r, P_r\}$. The number of transistors at an OR/AND operation is found by simply summing up the number in the subtrees, after taking the literal operation into consideration.

Finally, in Case 3, the gate formation operation involves the formation of a gate from the above domino NMOS circuit segment. The minimal cost subsolution, C_{min}, from the set of subsolutions at that node is chosen. Above and beyond this cost, which measures the number of transistors in the domino pull-down network, we add the overhead of two clock control transistors and an inverter, corresponding to four more transistors. Therefore, we have $C = C_{min} + 4$. We emphasize that the gate formation operation corresponds to the effect of creating a gate at the *current* node of the DAG, and the literal operation considers its effect at the *next* node in the DAG.

These functions can be illustrated as shown in Figure 3.1. The AND operator places the transistors in series, and the corresponding $\{S, P\}$ values and cost are computed as shown. If, at this point, we were to decide to consolidate these into a gate, then the corresponding situation would correspond to four extra transistors as shown. The $\{S, P\}$ value of the last transistor would then be set to {1,1}.

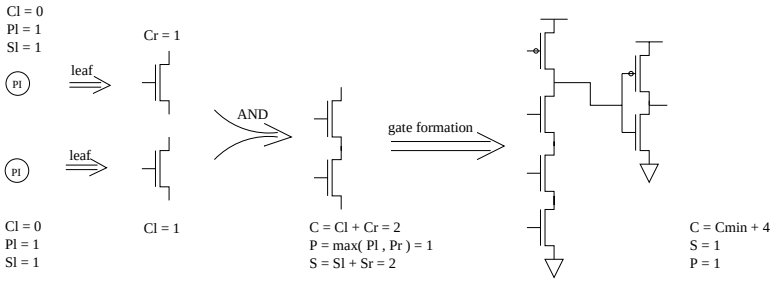

Figure 3.1: An illustration of node cost functions

3.2.2.4 Node Mapping Algorithm

ALGORITHM: **Node Mapping**

```
for each valid [height,width] subsolution of the left child
    for each valid [height,width] subsolution of the right child
    {
      {S, P} = NODE CONSTRAINT FUNCTIONS ({S_l, P_l}, {S_r, P_r}) ;
      if {S, P} is within the constraints (H, W)
      {
        C = NODE COST FUNCTION (C_l, C_r)
        if (C < C[S, P]_min) then C[S, P]_min = C .
        if (C < C_min) then C_min = C .
      }
    }
C[1, 1] = GATE FORMATION (C_min)
```

The $\{1,1\}$ subsolution of at a node is obtained from the subsolution $\{S,P\}$ of the same node whose implementation cost is minimal. A subsolution $\{S,P\}$ $(S > 1 \parallel P > 1)$ at a parent node is obtained by combining optimal subsolutions at children nodes. The solution $C[1,1]$ of the root of the decomposed tree is the optimal subsolution of the subject AND-OR tree.

3.2.2.5 An Example

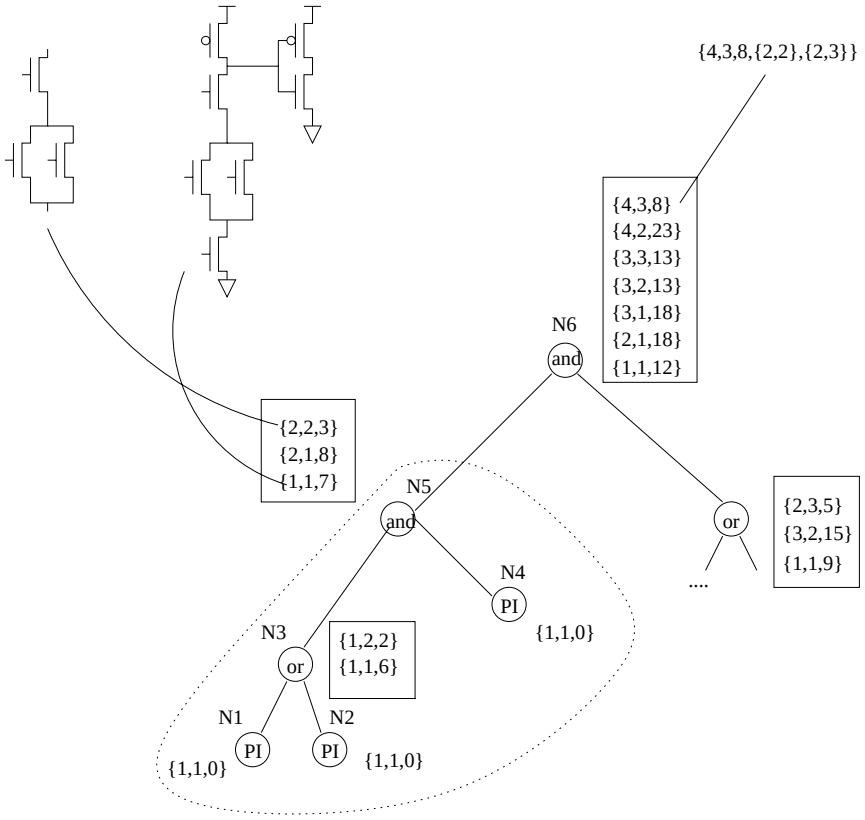

Figure 3.2: An example illustrating the parameterized library mapping procedure

The procedure in Algorithm Node Mapping is illustrated in Figure 3.2. In this example, three-tuples are used to represent combinations of $\{S,P,C\}$. All primary inputs are initialized with the tuple $\{1,1,0\}$. The tree is traversed from the literal node

upwards, and all subsolutions are enumerated using dynamic programming, eliminating any solutions that are suboptimal. From the dynamic programming viewpoint, the optimal solution under each constraint $\{S, P\}$ is a nonsuboptimal substructure. The nonsuboptimal subsolutions at a node are listed and the problem is solved by recursive enumerations of its higher level nodes.

For example, the operation of AND operation on both $(\{2,2,3\},\{2,3,5\})$ and on $(\{2,1,8\},\{2,3,5\})$ produce $\{4,3,C\}$. Only the tuple of this type with minimal cost and its corresponding child tuples are stored at node N6. This corresponds to $\{4,3,8\}$, which is a partial solution that may be used in the future. For the combination of $S = P = 1$, we choose the minimum cost solution at that node and construct a gate corresponding to that subsolution. For each child tuple that has a $\{1,1,C\}$ configuration, the cost is incremented by 1 by the literal operation. For example, $\{1,1,7\}$ is combined with $\{1,1,9\}$ at node N6 to give the tuple $\{2,1,18\}$. Finally, at node N6, the minimal cost obtained from all combinations of its children is 8, and the subsolution $\{1,1,12\}$ is obtained using the formula $C = C_{min} + 4$, corresponding to the gate formation operation.

3.2.2.6 Complexity Analysis

From the above procedure, we can see that the space complexity of the algorithm is $O(WHN)$ and the time complexity is $O(W^2H^2N)$. where N is the number of tree nodes and W, H are the maximum allowable width and height of the domino gate, respectively. At each node, the AND-OR cost function will be executed at most $W^2H^2/2$ times, but generally the number of executions is much lower than this value. From the statistics of 16 ISCAS85 benchmarks, we find that approximately 4 cost function evaluations are performed, on an average, at every node.

3.3 DAG Covering Mapping

3.3.1 Motivations and Previous Work

There are two common approaches to map a DAG. One method is called DAG mapping [45, 47, 48] and it begins at a primary output and continues until a primary input is encountered, or until another internal node that has been already mapped is encountered. However, the mapping schemes at a node may contradict each other at the multiple fanout nodes for parameterized library mapping method. The other method, referred to as tree mapping, is to partition the DAG into disjoint set of trees at multiple fanout nodes [44, 45] and compute the optimal solution at each tree.

The area cost of a domino gate is $K+4$ while that of a static gate is $2K$, where K is the number of literals. The delay cost of a domino gate is the sum of the pull-down network delay and the inverter delay. If the transistor count of a tree is less than 4, then the area cost of the domino mapping will be larger than that of static mapping, even without considering duplication cost for unating [33]. The delay of the inverter will also offset the advantage of fast switching speed provided by domino logic. Since large multi-level circuits often have a large number of multiple fanout nodes, the tree mapping based procedure that breaks the subject DAG at multiple fanout points often generates very small trees, leading to poor solutions. In our mapper, a greedy method is provided to overcome the problem. The method decomposes the DAG into larger trees as in [45] and permits the duplication of DAG nodes.

For static logic, duplication of network nets is mainly used to reduce the delay of implementation. In domino logic, the area cost of a single input is just one NMOS transistor and hence the duplication of DAG nodes will improve both area and delay.

3.3.2 Algorithm Outline

The DAG of the input network is decomposed at the reconvergent nodes, high fanout nodes and the nodes that have already been mapped. A reconvergent node of a node v is defined as a node from which there are two paths to node v. The high fanout nodes are defined as those nodes whose fanout set has a cardinality that exceeds a given threshold value. An example of DAG decomposition is shown in Figure 3.3.

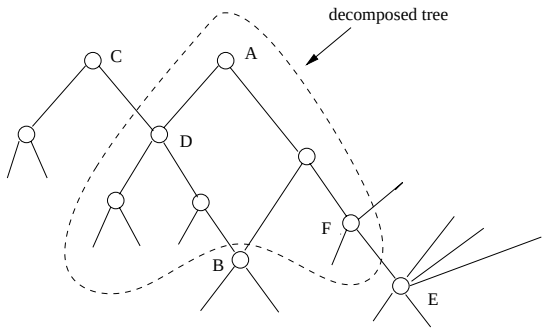

Figure 3.3: DAG decomposition

In Figure 3.3, node A is the root of the current tree and node B is the reconvergent node of node A. In this example, we consider node E to be a high fanout nodes. Therefore, the DAG is decomposed at nodes B and E. Therefore, instead of decomposing the DAG into trees rooted at nodes A, D, B and F, a larger tree rooted at node A is formed and is mapped. A second tree rooted at C goes no further than node D, which is already been mapped by the tree rooted at A.

Once a tree is obtained by decomposition, a postorder node cost calculation procedure and a preorder best solution assignment procedure are performed on the tree. Multiple fanout nodes are greedily assigned the subsolution that favors the first traversal to visit the node. If the multiple fanout node corresponds to an internal node of a gate, then some logic duplication is necessary. This is illustrated in Figure 3.3, where the multiple fanout node D belongs to both the tree rooted at node A and the tree

rooted at node C. Suppose tree A is mapped first and subsolution $\{2, 1\}$ is assigned at node D during the preorder assignment stage. During the mapping of tree C, $\{2, 1\}$ is the only subsolution available at node D. Therefore, the shaded area consisting of nodes D, G, H is duplicated as part of both gate C and gate A. The motivation for

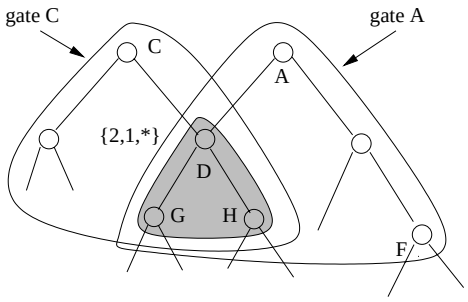

Figure 3.4: Duplication at multiple fanout nodes

decomposition at reconvergent nodes is that it prevents contradictions in mapping schemes at the fanout nodes. The purpose of decomposition at high fanout nodes is to avoid the large area penalty caused by duplication for each fanout.

The algorithm is outlined as follows.

```
ALGORITHM: DAG Covering Mapping(current)
{
  Traverse from node current to input, at each node v
  {
    If v is a reconvergent node or a high fanout node
       DAG Covering Mapping(v)
    else if v is already been mapped
       Stop traversal
  }
  Postorder traverse the tree current, at each node v
    DAG Node Mapping(v)
```

```
  Preorder traverse the tree current, at each node v
    Assign best mapping subsolution for node v
}
```

3.3.3 Duplication Cost Estimation

The DAG node mapping algorithm is similar to the node mapping algorithm described in section 3.2.2.4, except that the duplication cost is also included in the node mapping cost estimation. The node data structure of each optimal subsolution is modified as $\{S, P, C, \{S_l, P_l\}, \{S_r, P_r\}, DC\}$. Here, DC is the duplication cost if the current node is a multiple fanout node and the current subsolution is assigned. Suppose that DC_l is the duplication cost of the left child under the constraint $\{S_l, P_l\}$ while DC_r is the duplication cost of the right child under the constraint $\{S_r, P_r\}$, the duplication cost function is as follows:

1. literal operation: if **LITERAL** $DC = DC + 1$; else DC is unchanged.
2. OR or AND operation: $DC =$ literal operation(DC_l) + literal operation(DC_r)
3. gate formation operation: $DC = 1$

The additional cost caused by duplication will greedily be counted as part of the mapping cost during the postorder node mapping procedure of the first traversal through the node. During the first visit to a node, if the current node is a multiple fanout node, then the cost of the current subsolution provided to its parents is $C + (DC - 1) * (N_{fanout} - 1)$, where N_{fanout} is the number of fanouts of the current node. This is due to the fact that if the duplication subsolution is employed, then a cost of DC transistors is needed for each fanout; otherwise, only one transistor is needed for each fanout.

3.4 Dual-monotonic Gate Mapping

3.4.1 Dual-monotonic Logic

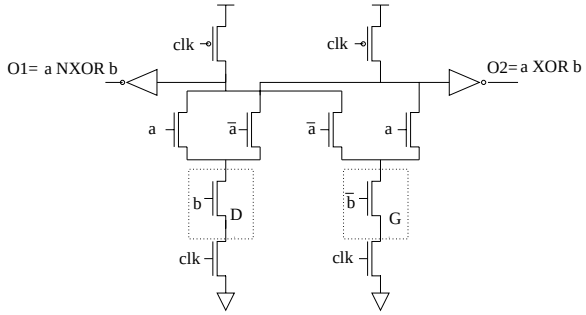

Figure 3.5: An example of a dual-monotonic gate

A dual-monotonic gate is a merged gate that generates both negative and positive polarities of a logic signal [17]. A typical dual-monotonic two-input XOR gate is shown as Figure 3.5. The logic used to implement $a \cdot b$ and $\bar{a} \cdot b$ share the same transistor, D. Due to the complementary relations between a and $\bar{a}$, a sneak path between $O1$ and $O2$ can be prevented. While dual-monotonic implementations of XOR logic use a smaller number of transistors than duplicated gates, this is not true for all logic functions. Indeed, implementations of most common dual-monotonic gates do not share as many transistors as the XOR gate.

3.4.2 Previous Work and Motivation

A domino gate implementation of an input network often requires the synthesis of both positive and negative signals at many nodes due to the unateness requirements; this is known as dual-rail logic. A logic function and its complement can be built as two separate gates or as one merged gate (a dual-monotonic gate).

Recently, several papers on domino logic synthesis have appeared [33,34,9]. All of them follow the synthesis methodology proposed in [33] in which dual-rail logic cones are independently mapped. This approach has the advantage of maintaining the mapping flexibility of each polarity. However, the presence of an XOR function and its recovergence property will decompose the input network into very small mapping trees, which causes a large area and delay cost for tree-by-tree technology mapping. On the other hand, dual-monotonic XOR is a widely used configuration in manually designed domino networks; its application to synthesis of domino logic can effectively solve the above problem. Therefore, we propose a mapping method that can make use of the advantages of both cases by mapping dual-rail AND/OR logic independently with standard domino gates, and mapping XOR/XNOR logic with merged dual-monotonic gate.

3.4.3 Dual-monotonic Mapping Algorithm

The method consists of the following steps:

Step 1 Generate a DAG network of two-input AND/OR gates for the given circuit.

Step 2 Recognize the XOR/XNOR logic inside the DAG by pattern matching. If both positive and negative polarity of XOR/XNOR logic are required, the XOR/XNOR logic networks are collapsed into one XOR/XNOR node to be implemented as a dual monotonic gate.

Step 3 Perform the technology mapping on the AND/OR/XOR/ XNOR subject network, mapping AND/OR nodes to the standard domino gates and XOR/XNOR nodes to various dual-monotonic gates.

The last step of the dual-monotonic mapping algorithm is technology mapping on an AND/OR/XOR/XNOR network. During technology mapping, all possible match-

ings are enumerated at each network node. While the traditional mapping patterns can be applied to AND/OR nodes, the matching patterns available to XOR/XNOR nodes need to be considered. One single XOR/XNOR node can be mapped to a XOR dual-monotonic gate. Moreover, the flexible configurations of dual-monotonic gates enable the exploration of more matching patterns at XOR/XNOR nodes. For example, the matching pattern of Figure 3.6(a) in the network corresponds to a three-input XOR gate as Figure 3.6(b).

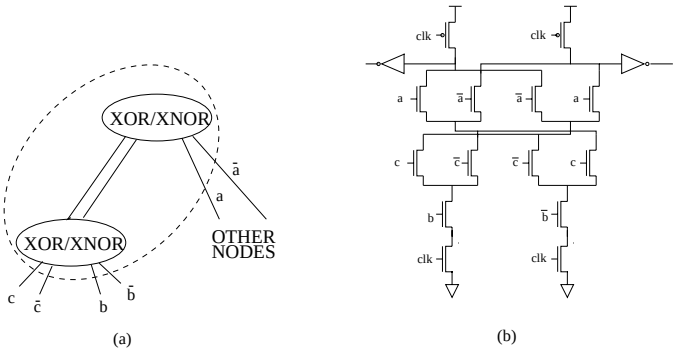

Figure 3.6: Matching pattern 1: three-input XOR gate

The matching pattern of Figure 3.7(a) corresponds to the dual-monotonic gate as Figure 3.7(b). It is obtained by replacing the transistors D and G in Figure 2 by NMOS subnetworks.

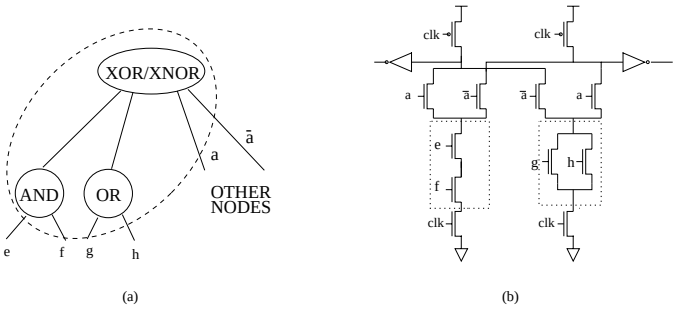

Figure 3.7: Matching pattern 2: arbitrary AND/OR/XOR logic

In this work, we incorporate the above issues in the parameterized mapper in [9], but any other mapper may easily be adapted for this purpose.

3.5 Flexible Domino Configurations

3.5.1 Wide Dynamic AND/OR Gate

The static gate following the precharged stack is not necessarily restricted only to be an inverter. One of the domino gate configurations used by our mapper is the wide dynamic AND gate [17]. As an illustration of the concept, we consider an AND domino gate with a stack of six serial NMOS transistors. To overcome the disadvantages of placing a large number of transistors in series, an equivalent structure that split the long stack and replace inverter with NAND can be used in practical designs. It is illustrated in Figure 3.8, where (a) is a standard domino gate and (b) is its equivalent counterpart. In the same way, NOR function with a high fan-in can be decomposed into two NMOS stacks with a NAND gate in the next stage.

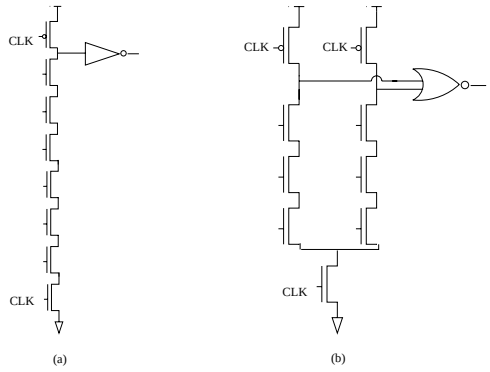

Figure 3.8: An example of a wide domino AND gate

In our mapper, to incorporate the mapping of wide dynamic AND/OR gate into our parameterized library mapping frame, an enlarged subsolution index, illustrated

in Figure 3.9, is used. The subsolutions in region a correspond to standard domino gate mapping, the subsolutions in region b to wide AND domino gate mapping and the subsolutions in region c to wide OR domino gate mapping. The subsolutions in region b are only valid for AND operators; if the subsolution at region b has minimal cost out of all the subsolutions of the current node, a wide domino AND gate is mapped. A similar argument supports the use of OR operations.

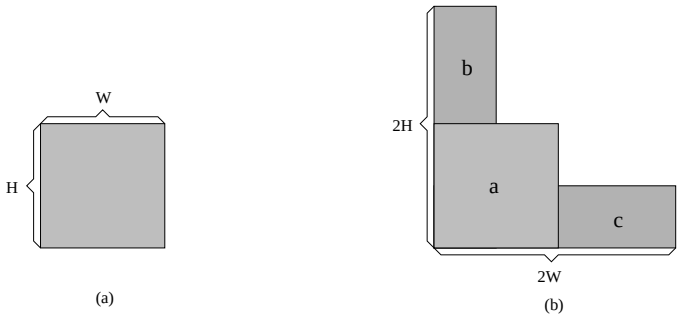

Figure 3.9: Subsolution space to map wide domino AND/OR gates

3.5.2 Multiple-output Gate

Another possible structure, mentioned earlier, is the multiple-output domino gate. Multiple-output gates are most commonly used in high speed adder design [21, 22]. Consider the "regular" domino structure shown in Figure 3.10 (a) with three outputs, O1, O2 and O3. We can see that the O1 logic is evaluated through three levels of domino gates. This may be implemented as a multiple-output domino gate as shown in Figure 3.10 (b) by sharing common substructures used by the three outputs, resulting in a smaller number of transistors. One problem that hinders the application of multiple-output domino logic is the possible existence of a sneak path. In Figure 3.10 (b), although transistor B is off, if the transistors I and A are on, then O2 will be evaluated as '1'. In our mapper, we use separate pull-up transistors for each multiple output branch, as shown in Figure 3.10 (c).

The mapping of multiple-output gates can be incorporated into the DAG covering mapping in section 3.3. During the mapping process, a multiple-output gate is a potential mapping scheme for the multiple fanout node during postorder mapping. Hence, a multiple fanout node could either be duplicated, or mapped as a multiple-output gate, or be mapped as a simple domino gate at the node, depending on the cost tradeoff.

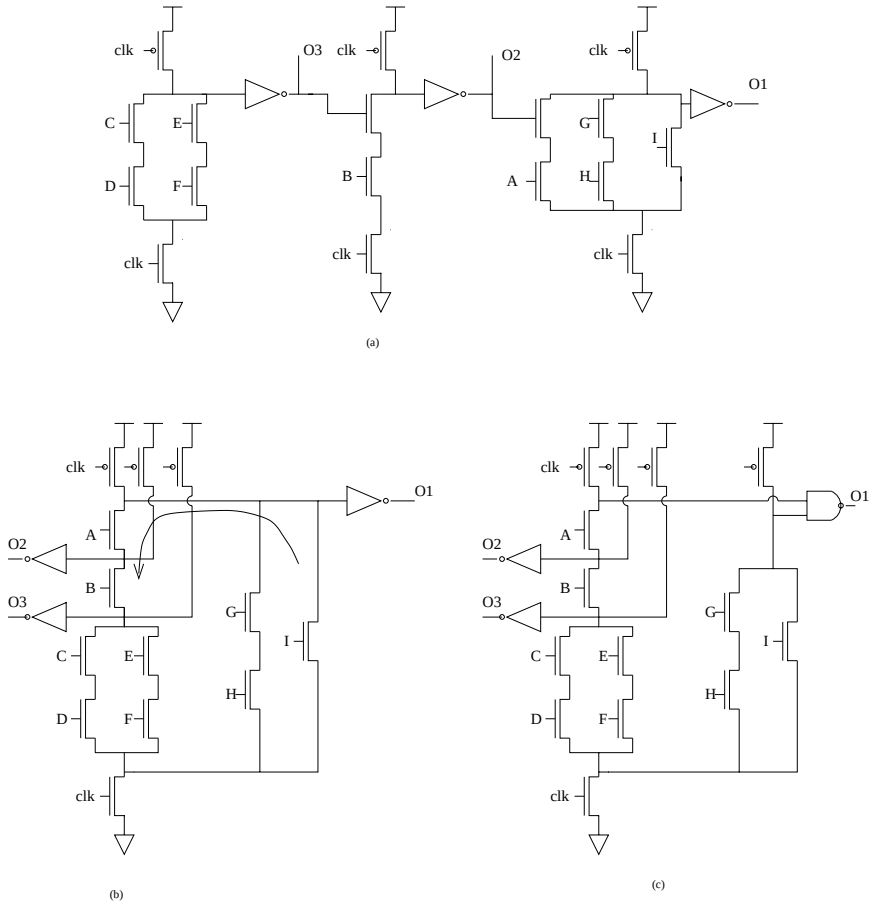

Figure 3.10: Multiple-output domino logic mapping

3.6 Technology Mapping for Static Logic

Technology mapping for static logic has been a cornerstone in the logic synthesis and has been well studied in the past. A good introduction to technology mapping can be found in [59, 60].

Existing technology mapping techniques can be differentiated from each other in the following aspects:

Partitioning Depending on the method by which the input DAG network is partitioned, DAG mapping or tree mapping can be used as described in Section 3.3.

Decomposition The idea of decomposition is to express input network in terms of simple functions, such as two-input NORs or NANDS and a decomposed network is often used as a startup point for technology mapping.

Matching We say a cell matches a subnetwork when they are functionally equivalent. The matching approaches generally can be classified into two categories: the structural approach [44, 45, 54] and the boolean approach [51–53]. The degree of freedom is associating input pins is modeled by the permutation matrix. We refer to this type of matching as Boolean matching because it is based on the property of the function and to distinguish it from another weaker formating of matching. Given a structural representation of two functions by two graphs in a pre-defined format, there is a structural matching if the graphs are isomorphic.

The parameterized library mapping algorithm presented in Section 3.2 is a fast structural matching technique. Its speed advantage can be explained with the help of Figure 3.11. In the input network shown in the figure, cells a and b are matching patterns at node E and F, respectively, and have the same structure at node D. In mapping techniques as far as we know, the cells are matched and their mapping costs

are calculated, which involves visiting nodes that are descendants of node D in the tree. However, in our matching approach, the structure and cost information of one segment of both cell a and b are stored at one subsolution of node D and the nodes below the node D need not be visited repeatedly. This subsolution is shared by all the matching patterns at nodes above node D.

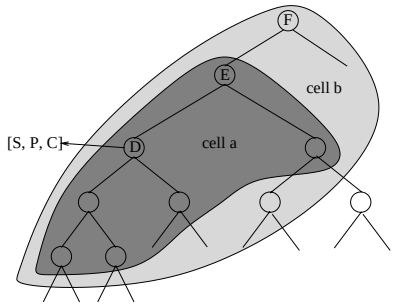

Figure 3.11: Matching technique of parameterized library mapping

In this section, we apply our matching technique described in Section 3.2 and the partitioning technique described in Section 3.3 to technology mapping for static logic, and a flexible on-the-fly phase assignment method is suggested.

3.6.1 Parameterized Library Static Mapping

For deep-submicron technologies, it has been shown in [61] that the ratio of the delay of NAND/NOR gates to the inverter delay is becoming smaller than older technologies, which encourages the use of longer chains of pull-up/pull-down logic in circuits and therefore will lead to an increased usage of complex gate in deep-submicron circuits. Therefore, the parameterized library mapping concept introduced in Sec 3.2 is a promising direction for static logic. Other algorithms for mapping to static CMOS parameterized libraries have been suggested in [62–64].

3.6.1.1 Decomposition and Compressed Network

The input network is decomposed into the compressed network, which is a network consisting of two-input AND, OR nodes and inverting function is represented by polarities of edges(positive and negative).

The input network for static logic technology mapping consists of AND, OR and NOT nodes. To permit the free (virtual) movement of inverters throughout the network in a manner that allows a better exploration of the design space, a compressed network was introduced to represent the original network. Each vertex in the compressed network corresponds to a AND/OR node in the original network, and stores information on the implementation of both the true and complemented forms of the logic function realized at that node; these will be referred to as the positive and negative polarities, respectively. The NOT nodes in the original network are merged into the polarity of edges in the compressed network. In the compressed network, an edge between two vertices can have two polarities: if an inverter exists between these nodes in the original network, then the edge polarity in the compressed network is negative; otherwise, it is positive. The building of the compressed network can be illustrated in Figure 3.12, where inverter node d is collapsed into the edge between a and b by setting its polarity to be negative.

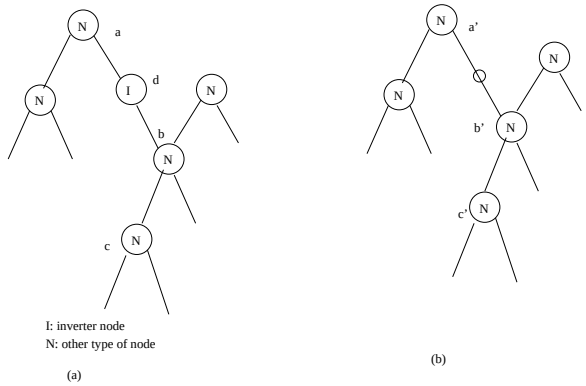

Figure 3.12: Constructing the compressed network

3.6.1.2 Node Data Structure

At each vertex of the compressed network, subsolutions corresponding to both positive and negative logic polarities are maintained. In other words, given the constraints (H, W) on each node corresponding to the height and width constraints on each gate, we generate optimal subsolutions of the form $[B, S, P]$. Here, B represents the polarity which could be either positive $(B = 0)$ or negative $(B = 1)$; S $(1 \leq S \leq W)$ represents the height of pull-down network and width of pull-up network; P $(1 \leq P \leq H)$ represents the width of pull-down network and height of pull-up network.

3.6.1.3 Node Constraint Functions

The operation of node constraints is similar to the node constraints function of domino logic except that the operations must be extended to account for the polarities, B.

1. OR operation: $tmpl = B_l \oplus F_l$, $tmpr = B_r \oplus F_r$
   if $(tmpl == tmpr)$
   $\{\ S = \max(S_l, S_r),\ P = P_l + P_r,\ B = tmpl\ \}$

2. AND operation: $tmpl = B_l \oplus F_l$, $tmpr = B_r \oplus F_r$
   if $(tmpl == tmpr)$
   $\{\ S = S_l + S_r,\ P = \max(P_l, P_r),\ B = tmpl\ \}$

3. gate formation operation: $S = 1$, $P = 1$, $B = \overline{B}_{min}$

Here, P_l, P_r, S_l, S_r, S and P is as defined in section 3.2, B_l and B_r are the polarities of the subsolutions of the left and right children, respectively. The symbols F_l and F_r represent the polarity of the edges to the left and right children, respectively. The polarity of subsolution whose cost is minimal is represented as B_{min}. Note that the inversion of the polarity at gate formation is caused by the inverting nature of static CMOS gates.

3.6.1.4 Node Cost Functions

If the area is measured by the number of transistors, the area cost function C can be desribed as follows:

1. literal operation: if **LITERAL** $C = C + 2$; else C is unchanged.
2. OR/AND operation: $C =$ literal operation(C_l) + literal operation(C_r)
3. gate formation operation: $C = C_{min}$

3.6.1.5 Node Polarity Assignment

For polarity assignment of multiple fanout node, a DAG covering mapping algorithm can be applied. The mapping procedure maps the largest possible logic cone in the network even if it contains multiple fanout nodes. Once a logic cone has been mapped, we find its optimal implementation and assign a polarity to each node; this polarity for multiple fanout nodes is then fixed and used to map the remaining logic cones.

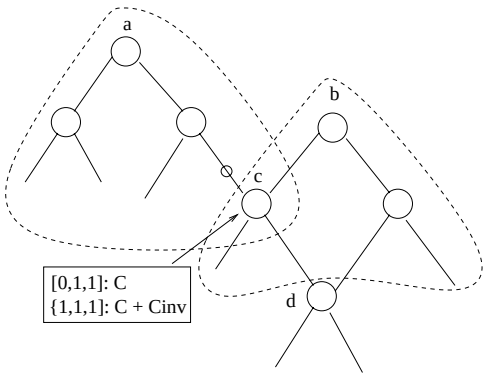

Figure 3.13: Polarity assignment at multiple fanout nodes

The procedure is illustrated in Figure 3.13. The DAG is decomposed into trees at reconvergent nodes such as node d, which contains the multiple fanout node, c, within

its transitive fanin cone. If the tree rooted at b is mapped first, then the resulting polarity at node c is found and then fixed during the remainder of the mapping; for example, during the mapping of the tree rooted at a. If a subsolution other than $[B, 1, 1]$ is chosen at a multiple fanout node, then the duplication of DAG nodes is permitted to reduce the delay with a minor area overhead.

3.6.2 General Cost Functions

So far, the cost functions in this work have used the number of transistors due to its simplicity, and for easy comparison with other work. Other cost models can also be applied to the parameterized library mapping algorithm. By defining different formulas to the AND, OR, gate formation and literal operations, the cost parameters such as delay, power, or area with consideration of routing cost can be used as the minimization objective.

One problem we will address here is the situation where a library is provided with a cost assigned to each library cell. In such a case, we suggest the use of a table lookup method that above matching method can be applied to perform the technology mapping.

The lookup table is in the size of $(H \times W) \times (H \times W)$ for AND(OR) operations and the AND(OR) operation formula in section3.2 is modified as:

$$\text{AND(OR) node:} \qquad C = C_l + C_r + table[AND(OR), [S_l, P_l], [S_r, P_r]] \tag{3.1}$$

By inspecting the library cells and finding the regulation of library cells area variation, lookup tables may be built. Here we illustrate the procedure by a simple example illustrated in Figure 3.14. Suppose that $O1 = !(a + b)$ and $O2 = !(a + b) * (c + d)$ are two cells of a library. The procedure of building up the table is shown in Figure 3.14. $[0, 1]$ is the $[S, P]$ value of the cell $O1$. The item in the table indexed

by $[AND(OR), [S_l, P_l], [S_r, P_r]]$ has the entry $cost(O2) - cost(O1) - cost(O1)$, so that when a tree of the type shown in Figure 3.14 (b) is to be mapped, its cost is correctly computed as $cost(O2)$.

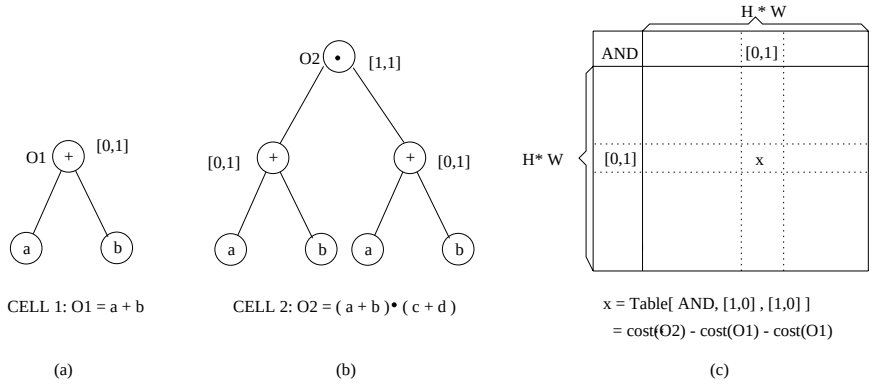

Figure 3.14: An example illustrating the construction of the lookup table

The above example is a simple procedure for building up a lookup table. In more complex situations where significant contradictions exists in table entries between different library cells and there are cells composed of multiple channel-connected components [65] in the library, further efforts need to be made; these are not considered in this work.

3.7 Experimental Results

Our technology mapping package for both domino logic and static logic have been implemented using C++. The experiments were executed on the LGSynth91 multi-level combinational circuit sets.

All of the input circuits are first optimized with *script.rugged* of SIS except for the results in Table 3.1. The input circuits for technology mapping for domino logic are unate equivalents of the benchmark circuits. They are obtained by first being optimized, then pushing the inverters as close to the inputs as possible, and finally

duplicating the fanin cones of the inverters. Except for the results in Table 3.1, all of our results were obtained under the constraints of $W = H = 4$.

In the result tables, the results are presented in terms of both area and delay, where the area is estimated as the transistor count while the delay is estimated by a coarse measure that counts the number of gate levels. In the following discussion, several sets of results show the comparison of our work with previous work, with a static CMOS implementation, as well as comparisons among various methods we discussed in section 2-6.

3.7.1 Comparison with Previous Work

Table 3.1: A comparison of our results with [5]

Circuits	Results #trans/#levels	[5] #trans	Reduction %
c8	289/6	328/7	13.5 %
count	283/6	348/16	23.0 %
i6	890/2	890/3	0 %
C880	1056/9	1499/7	42.0 %
C499	1324/6	1856/12	40.2 %
dalu	2201/10	2142/15	-2.8 %

A comparison of our results with [33] are shown in Table 3.1. The first column shows our approach, while the second column is reproduced from [33]. Both techniques use $W = 6$, $H = 4$ ($W = H = 4$ for dalu) as the constraints of the parameterized library. In the results of [33], the optimized input circuits were obtained by converting a raw CMOS circuit to its unate equivalent and then simplifying it using their unate optimization procedure, named unate optimization. In our procedure, due to the difficulty of implementing unate optimization, our optimized input circuits

were simplified using the same script file as [33] without unate modification, and then converted to its unate equivalent, a procedure was called binate optimization. In [33], it was shown that unate optimization is better than binate optimization in most cases. We expect that we would have further reductions in the transistor count if we were to implement the unate optimization procedure instead of binate optimization.

3.7.2 Comparison with Static Mapping with SIS

Table 3.2 shows a comparison of our results with technology mapping to a static CMOS library using SIS. The cell library $44-3.genlib$, adapted with the cell area taken to be the transistor count, was applied for SIS static mapping, and the objective for both mappers was area minimization.

The second column shows our results in terms of the number of transistors and number of gate levels. The third column contains the results of SIS. The comparative results are listed in column 5 and show that domino logic always has better delay performance than its static counterpart in terms the number of levels. The area cost of the domino circuits is smaller than or close to that of the corresponding static circuit for the small and intermediate sized benchmarks. In large circuits, with the increase of the depth of the circuit, the number of nodes that need to be duplicated becomes larger. Therefore, the area cost of domino logic is larger than that of the corresponding static implementation. In the last column, we report the ratio of nodes that must be duplicated in comparison with the number of nodes in the unated circuits. In the situation where the amount of duplication is larger, the static implementation has the advantage in terms of area cost.

In addition to the transistor count as the area cost of domino and static logic, there are other factors that need to be considered for area cost comparisons. Domino logic uses a very small fraction of PMOS transistors while half of the total number of

Table 3.2: A comparison of our domino mapping algorithm with SIS

Circuits	Domino #tran/#levels	CPU time(s)	44-3.genlib #tran/#levels	CPU time(s)	Reduction %	Dup-ratio %
b9	240/4	0.1	322/8	11.5	25.5%	10%
c8	272/4	0.1	316/8	12.6	13.9%	24%
count	326/6	0.1	354/18	12.5	7.9%	22%
i6	761/3	0.4	1194/5	50.3	36.3%	13%
C880	1019/14	0.3	1048/33	40.9	2.8%	47%
C1355	1360/7	0.4	1378/20	36.8	1.3%	77%
C1908	1456/12	0.4	1344/25	39.9	-8.3%	74%
C2670	1713/13	0.8	1952/17	70.7	12.2%	58%
C3540	4002/20	1.4	3140/34	133.7	-27.5%	92%
C6288	10795/47	4.0	8752/104	228.0	-23.3%	97%
C7552	6653/15	2.8	5834/26	178.6	-13.9%	79%
t481	1587/11	0.5	1838/19	82.9	13.7%	5.6%
rot	1667/9	0.6	1754/20	57.0	5.0%	33%
dalu	2245/11	0.9	2306/23	99.4	2.8%	43%
k2	2492/12	0.9	2928/23	97.0	15.6%	2%
des	9493/10	4.1	8694/19	178.6	-9.3%	49%

transistors of static logic are PMOS transistors, which are typically sized to be twice as large to obtain the same driving ability. Also, it may be noted that the flexible clock scheme of domino logic usually splits the large combinational circuit into smaller circuits associated with different phases, which may reduce the duplication cost. A major disadvantage in terms of area to domino implementation is the overhead for clock routing.

The CPU time of our mapping algorithm and SIS mapper are listed in columns 3 and 5, respectively. We can see that the computation time required by our domino mapper is very small.

3.7.3 Effectiveness of Various Methods

In Table 3.3, we compare the efficacy of various algorithms described in sections 2-6. The parameterized library mapping procedure of Section 3.2 constitutes the basic framework of the mapper. Other methods are incorporated in this framework separately or in various combinations.

Table 3.3: DAG covering mapping and dual-monotonic mapping

Circuits	Basic #tran/#levels	DAG-mapping #tran/#levels	Reduc %	Dual-mono #tran/#levels	Reduc %	#XOR
b9	261/5	249/4	4.6%	261/5	0 %	0
c8	282/6	274/5	2.2%	282/6	0 %	0
count	357/16	332/9	7.0%	357/16	0%	0
i6	763/3	763/3	0%	763/3	0 %	0
C880	1163/20	1131/14	2.7%	1051/20	9.6 %	16
C1355	1824/9	1696/7	7.0%	1360/7	25.4 %	72
C1908	1978/18	1809/14	8.5%	1588/14	19.7 %	50
C2670	1992/12	1940/13	2.6%	1777/12	12.1 %	40
C3540	4527/23	4259/19	5.9%	4241/20	6.3 %	38
C6288	13702/71	12814/45	6.8%	10629/57	28.9 %	419
C7552	7919/18	7493/16	5.4%	6613/16	16.6 %	213
t481	1697/12	1656/12	2.4%	1697/12	0 %	0
rot	1777/10	1695/9	4.6%	1775/10	0.1 %	1
dalu	2360/13	2294/11	2.8%	2349/13	0.5%	3
k2	2884/16	2759/13	4.3%	2884/16	0 %	0
des	9945/10	9736/10	2.1%	9843/10	1.0 %	51

Table 3.3 shows the efficiency of DAG covering mapping and dual-monotonic mapping. Column 2 contains the results obtained from the basic tree-by-tree parameterized library mapping method described in section 3.2. Column 3 shows the results obtained from the DAG covering mapping algorithm of section 3.3 and Column 4

shows the corresponding reduction relative to the basic algorithm. We can see that this method improves the area in some measure and is always effective in reducing the number of levels of the mapped circuit.

Column 5 shows the results obtained from the dual-monotonic mapping algorithm of section 3.4 while Column 6 shows the corresponding area reduction and Column 7 reports the number of XOR/XNOR gates detected in the circuits. We can see that in the circuits with significant XOR substructures, the dual-monotonic gate mapping is quite effective.

Table 3.4: Comparisons of various mapping methods

Circuits	Basic #tran/#levels	Wide-gate #tran/#levels	Multi-output #tran/#levels	Combination #tran/#levels
b9	261/5	261/5	247/4	240/4
c8	282/6	280/6	276/4	272/4
count	357/16	355/16	336/6	326/6
i6	763/3	763/3	763/3	761/3
C880	1163/20	1161/20	1131/14	1019/14
C1355	1824/9	1824/9	1696/7	1360/7
C1908	1978/18	1965/18	1720/13	1456/12
C2670	1992/12	1976/12	1940/13	1713/13
C3540	4527/23	4421/23	4262/19	4002/20
C6288	13702/71	13702/71	12812/45	10795/47
C7552	7919/18	7905/18	7476/16	6653/15
t481	1697/12	1628/11	1653/12	1587/11
rot	1777/10	1767/7	1687/9	1667/9
dalu	2360/13	2339/13	2289/11	2245/11
k2	2884/16	2738/15	2599/12	2492/12
des	9945/10	9819/9	9695/10	9493/10

Table 3.4 shows the efficacy of other methods. Column 2 contains the results

obtained from the basic tree-by-tree parameterized library mapping method same as Column 2 of Table 3.3. Column 3 lists the set of results using the wide AND/OR configuration, and Column 4 shows the results obtained by mapping multiple output configuration with node duplication. Column 5 displays the cumulative results of combining all of the above methods and is identical to Column 2 of Table 3.2.

3.7.4 Parameterized Library Static Mapping vs SIS

A comparison between our static mapper with results of SIS is shown in Table 3.5 using the constraints of $H = W = 4$. The largest example library available, $44-6.genlib$ is used for SIS technology mapping, with the objectives of both competing algorithms being area minimization.

Two sets of comparison between our mapper and SIS are shown. The first set uses the number of transistors as the area cost model, with the library 44-6.genlib being altered so that the cell area equals the transistor count. Columns 2 and 3 show the area cost and CPU time for our parameterized library mapper, and Columns 4 and 5 show the corresponding figures for the SIS technology mapper. The second set uses the cost model of library 44-6.genlib, and the cell area assignment of 44-6.genlib is fitted into the parameterized library cost model using the lookup table technique described in Section 3.6.2. Column 6 shows the results of the parameterized library mapper using this cost model, and column 7 shows the SIS results under the same model. Since the CPU times are similar to the first set of comparisons, they are omitted here.

Table 3.5 shows that the parameterized library mapper can be ten to a hundred times faster than the SIS mapper. The area cost result provides improvements of 0 to 12.3%, except for the circuit "count." From these results, we can see the parameterized library mapping method in Section 3.6 is a good choice when a faster and

Table 3.5: Comparison of the parameterized library static mapping method with SIS

Circuits	Lib-free 4-4	CPU (s)	SIS 44-6	CPU (s)	Lib-free 4-4	SIS 44-6
b9	306/7	0.07	316/8	33	213	227
c8	314/8	0.06	314/8	35	209	208
count	352/31	0.06	324/17	36	240	212
i6	1120/7	0.31	1194/5	136	739	813
C880	1024/19	0.21	1046/33	114	688	711
C1355	1352/17	0.27	1378/20	108	962	978
C1908	1324/30	0.27	1344/33	112.7	941	960
C2670	1770/20	0.56	1936/17	202.1	1188	1356
C3540	2820/34	0.94	3112/35	368	1864	2076
C6288	8388/107	3.24	8728/104	662	6016	6400
C7552	5546/25	2.14	5744/26	537	3768	3942
t481	1798/16	0.39	1832/19	219.6	1198	1235
rot	1668/15	0.45	1714/20	167.2	1162	1214
dalu	2106/22	0.67	2284/24	278	1330	1497
k2	2852/19	0.69	2910/23	254.4	2024	2101
des	8226/18	3.16	8628/19	1030	5337	5728

rougher mapper is required.

3.8 Conclusion

In this chapter, we have mainly explored technology mapping techniques for domino logic. Based on principles of dynamic programming, an efficient parameterized library mapping algorithm is presented. Several other technology mapping methods, considering various features of domino logic, are proposed and incorporated in the mapper. In addition, the parameterized library method is extended to static mapping.

The results of our work have provided a smaller area and much lower CPU time than the previous work for both static and domino mappers. As in static logic technology mapping, the area minimization mapping can be extended to delay minimization mapping, power minimization mapping, as well as area minimization mapping under timing constraints.

Compared to static CMOS mapping, domino logic synthesis results have better delay performance (measured in terms of the number of levels), and better or equal area performance in small and middle sized circuits. With the increase in the number of levels in larger circuits, the logic duplication of domino implementation will increase continuously, which lead to a larger area cost than the corresponding static implementation. Fortunately, domino implementations of combinational circuits usually are divided into multiple phases and the proper partitioning of input combinational network may significantly reduce the logic duplication and therefore reduce the area cost. We will address this problem in Chapter 5.

Chapter 4

0-1 Programming Output Phase Assignment Problem

4.1 Introduction

A major problem in domino logic synthesis is related to its non-inverting property. This feature necessitates logic duplication of some nodes of an input network for which both polarities of a logic signal are required, as illustrated in Figure 4.1. The manner in which the logic duplication problem is handled is the main factor that influences the quality of the domino logic synthesizer. In this chapter, a 0-1 programming formulation was applied to assign the output phase to reduce the duplication cost and minimize the implementation cost.

The output phase assignment problem was first addressed and solved in [36]. The problem was defined as follows: Given a combinational logic network and all primary inputs in the true and complemented form, choose an optimal phase (i.e., polarity) assignment for the primary outputs so as to require minimal logic duplication for obtaining inverter-free logic.

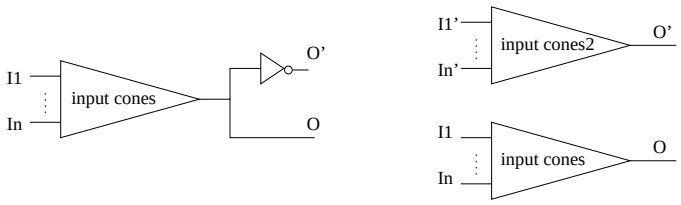

Figure 4.1: Logic duplication in domino logic synthesis

In [36], given an input network, the network is divided into the region that must be duplicated and region whose duplication cost can be minimized by output phase assignment, referred to as the optimizable logic region. The fanout nets in the optimizable logic region are called candidate nets. The problem of finding optimal output phase assignment to minimize the duplication of optimized logic region was formulated as a 2SAT unate covering problem and solved by using binary decision diagrams.

Our solution to the output phase assignment problem improves over the work in [36]. In our mapper, the output assignment problem is formulated into a 0-1 integer programming problem and a standard linear programming package is used to solve the problem.

The objective of our phase assignment problem formulation is to minimize the implementation cost of the mapped network, instead of minimizing logic duplication in the unmapped network as in [36]. From our observations, the output phase assignment accomplishes the objective of reducing implementation cost through several factors. Reducing logic duplication is one such factor, and the cost difference between the implementations of positive and negative polarity logic is another important consideration of output phase assignment. Suppose W and H are the constraints on the width (maximal number of parallel chains) and height (maximal number of series chains) of the NMOS pull-down network of domino gate, respectively. Due to the absence of a complementary PMOS network in domino gate, domino gates usually have

large W with limited H, which causes the cost difference between implementations of positive and negative polarity.

For example, given the constraints that $H = 2$ and $W = 4$, the logic network of Figure 4.2(a) will be mapped to three domino gates as Figure 4.2(a) while its complement logic in Figure 4.3(a) requires only one domino gate implementation as 4.3(b).

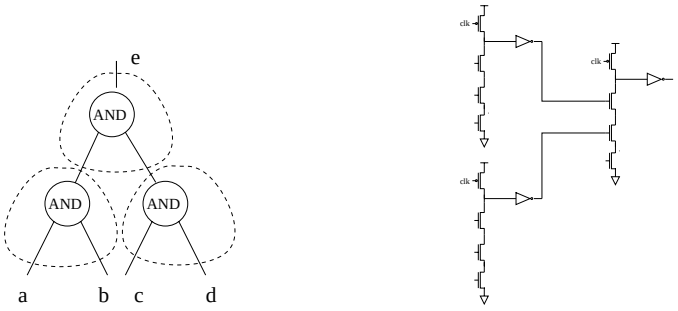

Figure 4.2: The implementation cost of positive polarity

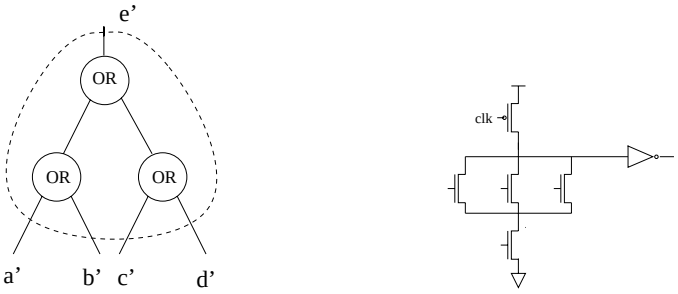

Figure 4.3: The implementation cost of negative polarity

The remainder of the chapter is organized as follows. In Section 4.2, we present an outline of the output phase assignment algorithm. In Section 4.3, we describe the 0-1 integer programming formulation of the output phase assignment problem which is set up with the objective of minimizing the duplication cost. In Section 4.4, the cost

difference between positive and negative polarities is considered and new 0-1 integer constraints are incorporated aimed at total cost minimization. Experimental results are presented in Section 4.5 followed by concluding remarks in Section 4.6.

4.2 Algorithm Outline

The algorithm can be outlined as follows:

1. Decompose the network into a disjoint set of trees and obtain the area cost estimation of each tree by counting number of nodes in the tree or performing tree-by-tree mapping on the network. The area cost of a tree is assigned to the root node of the tree, u, as the weight $C(u)$.

2. Find the optimizable logic region using the method of [36]; build a DAG from the multiple fanout nodes of the optimizable logic region.

3. Write out the 0-1 integer linear programming formulation and solve it.

4.3 0-1 ILP for Minimal Duplication Cost

Given an input network, a DAG $G(V, E)$ can be built as follows. Each vertex $v \in V$ corresponds to a multiple fanout or a primary output node in the optimizable logic region of input network. If vertex u is a literal node of a tree rooted at vertex v in the original input network, there is a corresponding edge $e_{u,v} \in E$ in DAG.

In the DAG, several constants are assigned to an edge or a node from the original input network. These include:

- $O(u, v)$, which represents the inversion polarity between vertex u and vertex v. If there is an even number of inverters from vertex u to vertex v in the input network, $O(u, v)$ is 0; otherwise, $O(u, v)$ is 1.

- $C(u)$, which represents the area cost of the tree whose root is at vertex u in the input network.

- $k(u)$, a constant whose value is twice the number of fanout of vertex u.

The $\{0, 1\}$ integer variables that will be included in the 0-1 programming formulation include:

- $r(u) = 1$ if there is an inverter moving from the inputs of node u towards the outputs of node u; else 0.

- $x(u, v) = 1$ if there is an inverter on the edge $e_{u,v}$ after the output phase assignment; else 0.

- $y(u, v)$ is a dummy variable that transforms a condition statement into a linear constraint.

- $q(u) = 1$ if the fan-in tree of node i needs to be duplicated; else 0.

The weight of an edge $e_{u,v}$ after output phase assignment, denoted by $w(u, v)$ is given by

$$w(u, v) = O(u, v) + r(u) - r(v) \tag{4.1}$$

Since $O(u, v)$, $r(u)$, $r(v) \in \{0, 1\}$, $w(u, v)$ takes a value in the set $\{-1, 0, 1, 2\}$. If $w(u, v) = 0$ or 2, it represents the absence of an inverter between node u and node v. If $w(u, v) = -1$ or 1, then there is one inverter between node u and node v. Therefore, this may be captured by the condition

$$x(u,v) = \begin{cases} 0 & w(u,v) \in \{0,2\} \\ 1 & w(u,v) \in \{-1,1\} \end{cases} \tag{4.2}$$

The above condition may be rewritten as a linear equation.

$$w(u,v) + x(u,v) = 2 \times y(u,v) \tag{4.3}$$

where $y(u,v) \in \{0,1\}$ is a dummy variable introduced to transform a condition statement into a linear constraint.

Combining the equations (4.1) and (4.3), we have

$$2 \times y(u,v) - x(u,v) = O(u,v) + r(u) - r(v) \tag{4.4}$$

If there is an inverter at any position in the fanout cone of a node u, the node will have to be duplicated to ensure the unateness property for domino logic. This condition can be given by the linear formula

$$q(u) \times k(u) - \sum_{i \in dir-succ(u)} (x(u,i) + q(i)) \geq 0 \tag{4.5}$$

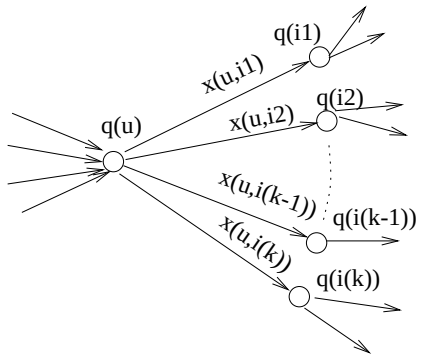

Figure 4.4: An illustration of constraints (4.5) of 0-1 ILP

It can be illustrated as Figure 4.4. Here, $i \in dir - succ(u)$ implies that there is an edge from node u to node i in the DAG $G(V,E)$ defined in Section 4.3. The

constant $k(i)$ was defined earlier and can easily be verified to always be larger than $\sum_{i \in dir-suc(u)(u)}(x(u,i)+q(i))$. This formula implies that if any of its successors needs to be duplicated, or if there is one inverter at the output edges of node u, node u will have to be duplicated as $q(u)$ is forced to be 1; otherwise the objective function will force $q(u)$ to 0.

Therefore, the output phase assignment problem can be formulated as the 0-1 integer linear programming problem:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{u \in V} C(u) \times q(u) \\
\text{subject to} \quad & 2 \times y(u,v) - x(u,v) = O(u,v) + r(u) - r(v) \quad \forall e_{u,v} \in E \\
& q(u) \times k(u) - \sum_{i \in dir-succ(u)} (x(u,i) + q(i)) \geq 0 \quad \forall u \in V \\
& q(u), r(u) \in \{0,1\} \quad \forall u \in V \\
& y(u,v), x(u,v) \in \{0,1\} \quad \forall e_{u,v} \in E
\end{aligned}
$$

4.4 0-1 ILP for Minimal Implementation Cost

This section modifies the 0-1 formulation of Section 4.3. to incorporate the cost difference between positive and negative polarity implementations to obtain the minimal implementation cost.

In addition to the notation defined in 4.3, the symbols that will be used include:

$Z(u)$ is a constant that represents the cost difference between optimal implementation of each of the two polarities. If the implementation cost of positive[negative] polarity of the node u is $p(u)[n(u)]$, then the value of $Z(u) = p(u) - n(u)$.

We introduce $t(u)$ as a variable that help to maximize $r(u)$. Its value can be

expressed as follows.

$$t(u) = \begin{cases} 0 & q(u) = 1 \\ r(u) & q(u) = 0 \end{cases} \tag{4.6}$$

Similarly, we also define a variable $s(u)$to minimize $r(u)$ as

$$s(u) = \begin{cases} 0 & q(u) = 1 \\ 1 - r(u) & q(u) = 0 \end{cases} \tag{4.7}$$

There are three possibilities for each node after output phase assignment. A node will either be duplicated, implemented in positive polarity, or in negative polarity. In the case of no duplication at node u, if $Z(u) > 0$, it costs less to implement the node with negative polarity; if $Z(u) < 0$, it is better to implement the node u with positive polarity. Hence, using the previous definitions of $q(u)$ and $r(u)$, the problem can be expressed as

$$\text{if } q(u) = 0 \textbf{ and } Z(u) > 0, \text{maximize } r(u);$$
$$\text{if } q(u) = 0 \textbf{ and } Z(u) < 0, \text{minimize } r(u).$$

The first statement can be expressed by linear equations

$$\begin{aligned} \text{minimize} \qquad & -Z(u) \times t(u) \\ \text{subject to} \qquad & t(u) \leq 1 - q(u) \\ & t(u) \leq r(u) \\ & q(u), r(u), t(u) \in \{0, 1\}, \qquad \text{if } Z(u) > 0, \end{aligned}$$

When $q(u) = 1$, $t(u)$ is forced to 0. Hence, there is no limit on $r(u)$ and the duplication cost is given by $C(u) \times q(u)$. When $q(u) = 0$, $r(u)$ is forced to be as large as possible since the negative value of $-Z(u)$. When polarity preferences of two

nodes conflict with each other, $Z(i)$ is the weight that decides which node should win. Similarly, the second statement can be captured by

$$\begin{aligned} \text{minimize} \quad & Z(u) \times s(u) \\ \text{subject to} \quad & s(u) \leq 1 - q(u) \\ & s(u) \leq 1 - r(u) \\ & q(u), r(u), s(u) \in \{0, 1\}, \qquad \text{if } Z(u) < 0 \end{aligned}$$

Combining with the linear equations of section 4.3, the 0-1 ILP formulation of the output phase assignment problem for cost minimization can be rewritten as

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{u \in V} C(u) \times q(u) - \sum_{Z(u)>0} Z(u) \times t(u) + \sum_{Z(u)<0} Z(u) \times s(u) \\ \text{subject to} \quad & 2 \times y(u,v) - x(u,v) = O(u,v) + r(u) - r(v) \;\; \forall e_{u,v} \in E \\ & q(u) \times k(u) - \textstyle\sum_{i \in dir-succ(u)} (x(u,i) + q(i)) \geq 0 \;\; \forall u \in V \\ & t(u) \leq 1 - q(u), \;\; t(u) \leq r(u) \quad \forall Z(u) > 0 \\ & s(u) \leq 1 - q(u), \;\; s(u) \leq 1 - r(u) \quad \forall Z(u) < 0 \\ & q(u), r(u), t(u), s(u) \in \{0, 1\} \quad \forall u \in V \\ & y(u,v), x(u,v) \in \{0, 1\} \quad \forall e_{u,v} \in E \end{aligned}$$

4.5 Experimental Results

The above algorithms are implemented using C++ and incorporated as the preprocess of parameterized library domino mapper addressed in Section 3.2. All of the input circuits are optimized with *script.rugged* of SIS and targeted to the area minimization. In the result tables, the results are presented in terms of area where the area is estimated as the transistor count. To demonstrate the influence of cost difference

between two polarities to total implementation cost, the domino gates were restricted to $W = 8$ and $H = 2$. Table 4.1 shows the efficacy of our 0-1 programming output

Table 4.1: Output phase assignment using a 0-1 ILP

Circuits	#po	no-ass area	opt-ass1 area	opt-ass2 area	reduction %
b9	21	391	389	311	20.5%
c8	18	407	364	330	18.9%
i6	67	1298	1281	766	41.0%
C1355	32	2064	2064	2064	0.0%
C2670	140	2647	2624	2414	8.8%
C6288	32	14257	14257	14252	0.0%
C7552	108	9069	9069	8904	1.8%
rot	107	2332	2296	2196	5.8%
dalu	16	3020	2752	2745	9.1%
k2	45	3974	-	-	-
des	245	13130	13130	11710	10.8%
apex7	37	833	791	736	11.6%
frg1	3	362	362	362	0%
x1	35	861	927	850	1.3%
x3	99	2381	2360	2100	11.8%

phase assignment algorithms. Column 2 contains the number of primary output in the circuits. Column 3 lists the results when the output phase assignment are not used. Column 4 shows the results when 0-1 formulation of 4.3 is applied. Column 5 shows the results obtained by incremental 0-1 programming formulation in 4.4 while Column 6 shows the corresponding area reduction. We use the linear program solver *lp_solve*_2.3 [66] to solve the 0-1 integer linear programming formulas. All of the benchmarks can solved in under one minute with no or minor simplification of 0-1 ILP except for the 0-1 programming problem for Circuit k2, which is too large to

yield the result in reasonable time.

It was observed that in some circuits such as dalu, the cost reduction by output phase assignment mainly comes from the duplication cost reduction; in some circuits such as i6 and b9, the implementation cost difference between two polarities becomes the most significant consideration of output phase assignment. In some case, the two objectives of output phase assignment can be contradictory to each other. In Circuit x1, the optimization for duplication cost minimization causes a cost increase due to the increased cost of implementing the reversed polarity and increases the total implementation cost.

4.6 Conclusion

In this chapter, a 0-1 integer programming formulation is provided for the output phase assignment problem for domino logic. It considers the cost difference between two polarities and enables a standard linear programming package to be used to solve the problem. The results show up to 41.0% improvement in area. By replacing the area cost of logic trees with power cost, we expect our 0-1 ILP of output phase assignment can serve the power minimization objective proposed in [37] similarly.

Chapter 5

Timing-driven Partitioning of Domino and Mixed Static-Domino Circuits

5.1 Introduction

After the technology-independent logic design step, a specified technology must be chosen to implement the input logic network. Both static and domino implementations offer various advantages. While static logic is commonly used for its robustness, its speed may be inadequate to satisfy the clocking constraints. Although various optimization tools like sizing and buffer insertion can be used to increase the speed of the circuit, the area cost may increase exponentially beyond a certain point, and it is worthwhile to alter the logic style to meet such stringent constraints. Domino circuits typically provide higher speeds than static logic while having a higher clock routing overhead and are less noise-tolerant. Depending on the amount of logic duplication required to make the network unate, domino circuits may use a larger or smaller

number of transistors than static circuits. Therefore, for optimality, a determination must be made as to which parts of the circuit should be implemented in static logic, and which parts using domino logic. This must be carried out while keeping the requirements of the specified clock scheme in mind.

In this chapter, we describe an automated design strategy to solve the following problems with the goal of minimizing an objective function such as area or power, under a set of timing constraints:

- We partition a circuit into static and domino regions, under the same clock phase. Note that since one of the partitions may be empty, the two extremes of constructing the circuit entirely using domino logic, or entirely using static logic, are also feasible solutions to the problem.
- Under a two-phase clocking scheme, we address the problem of partitioning the domino region to determine which gates are to be clocked by each clock phase. Since domino implementations always use multiple phases, and most commonly, two phase clocks, this is an essential problem in domino circuit design.
- For a two-phase clocking discipline, we utilize the solutions for the above two problems to arrive at a flow that partitions a circuit into a common two-phase clocking strategy as Figure 2.2.

As will be described through an example in Section 5.2, for a partitioned circuit, the logic duplication penalty for the domino segment depends on the location of the cut between the partitions. Our method finds a partitioning solution that takes this into account while minimizing the area objective.

To the best of our knowledge, the only published work addressing a related problem is [18], which outlines a static-domino partitioning procedure. Their work first implements the input logic using static gates, and then finds a critical path and its

fanin transition region. A greedy approach is taken and the logic in this region is then remapped to domino gates to maximize the use of domino logic gates. No experimental results were presented in this work. In our paper, the problem is approached systematically, and a network flow based algorithm that provides an optimal solution, within the accuracy limitations of the cost estimation, is proposed. Our procedure is based on the following observations that differentiate it from [18]:

- Logic duplication can cause a large area penalty for large combinational circuits [33, 34]. A proper choice of the partitioning cut can reduce the duplication cost. Our partitioning procedure automatically creates the largest possible unate region within the domino partition.

- The critical path and its fanin transition cone may form a large part of the input, or possibly even the entire network (for a circuit with one primary output). Implementing this in domino logic as in [18] may be costly and unnecessary, and as long as the timing constraints are satisfied, it may not be essential to maximize the use of domino gates in the fanin cone.

- The work in [18] attempts to greedily minimize the number of domino transistors by utilizing them only in the critical region. We use an estimator that directly incorporates the area cost (including the duplication penalty) for static gates and domino gates to find optimal combinations between domino and static gates that minimize the overall cost function, while meeting the performance targets.

The chapter is organized as follows. In Section 5.2, we provide the problem definition and motivations. Next, Section 5.3 describes the complete algorithm for timing-driven static-domino partition. This is followed in Section 5.4 by a description the modifications to the algorithm of Section 5.3 for timing-driven two-way domino partitioning. Section 5.5 presents the flow of partitioning for the general two-phase clocking scheme in Figure 2.2, and a description of the cost model used for partitioning

is provided in Section 5.6. Finally, experimental results are presented in Section 5.7 followed by concluding remarks in Section 5.8.

5.2 Problem Definition and Motivations

Synchronous domino logic is conventionally divided into multiple phases. From various clocking schemes for domino circuits described in Section 2.2, we can see that it is important to decide which part of a given segment of combinational logic is to be implemented with static, and which part to be implemented with domino logic. This determination depends on the clocking frequency objective of a design. Moreover, even for a segment of a combinational network to be implemented purely as domino logic, it is necessary to partition the input network into several stages, corresponding to each clock phase.

The clocking strategy of Figure 2.2 is the most commonly used clocking scheme for domino circuits. In the remainder of this chapter, we will address the general domino partitioning algorithm with special emphasis on this common two-phase nonoverlapping clocking strategy.

The problems of static-domino partitioning and domino partitioning between the two phases are critical problems to be solved in determining the optimal circuit implementation. We define these two problems concisely as follows:

Static-domino partitioning Given an optimized combinational circuit and delay specifications on the outputs of the network, implement the nodes in the network using either domino logic gates or static gates such that the cost is minimized, assuming them all to be within the same clock phase.

Two-way domino partitioning Given an optimized combinational circuit to be implemented entirely in domino logic and a two-phase nonoverlapping clock

scheme which permits inverters to be placed at the interface between the two phases, partition the boolean network into two clock phases such that the cost of domino implementation is minimized.

We will use the solutions to the above problems later in this paper to solve a more general statement of the partitioning problem in Section 5.5.

For the problem of static-domino partitioning, we assume that we are provided with a combinational logic network description as an input, and that it can be partitioned into a static and a domino region within the same clock phase. Under this model, an additional precedence constraint that must be satisfied that states that no static logic gate is permitted to fan out to a domino gate in the same clock phase. A violation of this requirement would lead to functional errors in the domino logic [18]. The timing constraint for this situation is that the partitioned circuit must have its outputs ready at the end of the clock phase.

For the problem of two-way domino partitioning, we assume that the input is a combinational logic network that is to be partitioned so that its primary outputs are available in one clock cycle. This imposes a similar precedence constraint, where a domino gate in Φ_2 is not permitted to fan out to a domino gate in Φ_1. Due to the nature of domino logic circuits under two-phase nonoverlapping clocks, the latch between the two phases forms a hard edge that prevents time borrowing between the phases. This imposes a timing constraint that states that the delay within each phase must be restricted to a half-cycle.

The timing driven partitioning method must consider two factors: the timing constraints, described above, and the cost of the implementation, measured in terms of the area or the power. We will first describe the algorithm using the area as a cost measure, and then show how a cost function modeling the power can be incorporated within the same framework. In our experimental results, the area is modeled as the

total number of transistors, but it is easy to incorporate a factor that models the clock routing penalty by adding a factor that is proportional to the number of domino gates.

An important consideration in the area cost relates to the requirement that only unate functions can be implemented in domino logic, as a consequence of the non-inverting nature of the domino logic family. Therefore, intervening inverters are handled by implementing a logic function and its negation separately, which duplicates the cost of logic implementation for that logic cone [36]. However, we observe that the duplication cost can be reduced by partitioning, and our formulation directly incorporates the cost of duplicating non-unate logic within a domino region in the cost function. To see this, consider the example shown in Figure 5.1.

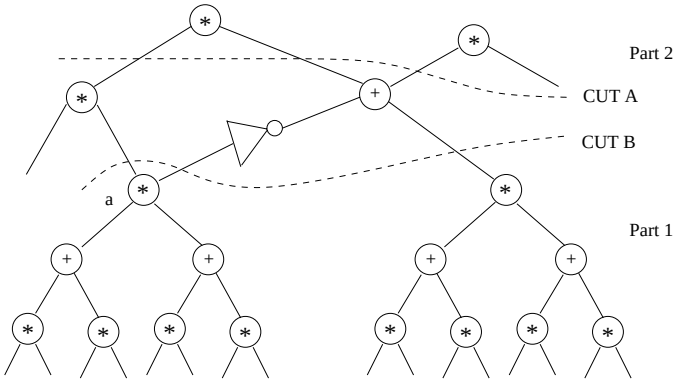

Figure 5.1: An example for static-domino partitioning

In Figure 5.1, assume that the segment labeled "Part 1" corresponds to a domino implementation, while "Part 2" represents a static CMOS implementation. Two possible cut lines are shown in the figure. If cut A is used, then both a and $\overline{a}$ must be implemented as unate functions, resulting in a duplication of the fanin cone of a, doubling the cost of implementing this cone. However, if cut B is used instead, then there is no need to duplicate the fanin cone of a as the inverter may be implemented in static logic. Similarly, under the scenario where Part 1 corresponds to domino logic driven by phase Φ_1 of a two-phase clock, and Part 2 corresponds to phase Φ_2,

with latches being placed at the cut, an identical argument holds, with the difference that the inverter for cut B would now be implemented using the Qbar output of the inserted latch.

5.3 Timing Driven Static-Domino Partitioning

5.3.1 Algorithm Outline

The two subproblems listed in Section 5.2 can be solved using similar algorithms, using different cost formulations. In this section, we will illustrate the algorithm for static-domino partitioning. The algorithm applies a DAG technology mapper based on dynamic programming, a timing analysis method based on PERT, and a maximum flow algorithm to realize a timing-driven two-way partitioning.

The input to the algorithm is an arbitrarily optimized two-input AND-OR DAG network. In outline, the algorithm consists of the following steps that are described in detail in the remainder of this subsection:

1. Perform static technology mapping and domino technology mapping separately on the entire logic network to determine a cost estimate for every vertex.

2. Find the candidate cut nodes in the network. A candidate cut node is defined as a node that satisfies the criterion that a cut passing through it will not violate the timing specification.

3. Build the flow network from the candidate cut nodes. The capacity of the flow network is determined from the cost difference between static and domino implementations in Step 1. A maximum flow algorithm is applied to the network to obtain a minimal cost cut [58].

5.3.2 Cost Estimation

The first step of partitioning is to perform static and domino mapping on the logic network to obtain the cost estimation. The partitioning between domino and static requires a cost comparison of each vertex of the network implemented using a domino or a static implementation. While we would like to perform this determination on the same network for purposes of storage efficiency, this task is complicated by the fact that technology mapping of domino logic requires the input network to be unated; after unating, the topology of the new network will be different from the original network on which static mapping is performed. Moreover, inverters may be pushed through the network using De Morgan's laws, and therefore their location is not fixed. Depending on where the inverters are placed, the cost of logic duplication and the positions at which logic duplication occurs could vary, and a good partitioning algorithm should consider all such possibilities in arriving at a partition.

To address these issues, we use the compressed network introduced in Section 3.6.1.1, denoted as N_{twin}, to represent the original network. Each vertex in compressed network corresponds to a node in the original network, and stores information on the implementation of both the true and complemented forms of the logic function realized at that node; these will be referred to as the positive and negative polarities, respectively. An edge between two nodes can have two polarities: if an inverter exists between these nodes in the original network, then the edge polarity in the is negative; otherwise, it is positive. It is not necessary to store inverters in the original network as nodes in the twin network; rather, these may be merged into the polarity of an edge as shown in Figure 3.12. The lists of positive and negative polarity fanouts of each vertex in N_{twin} are maintained, and in case of domino mapping, the duplication of any fanout is flagged.

The actual procedure used for domino and static mapping is based on the work

in [9], and is not described here. Once the technology mapping results on N_{twin} are available for both for static and domino implementations, cost estimations are available for both the static and domino implementation at each vertex.

Definition 1: We define the following quantities at each node N.

$p\_delay_d(N)[n\_delay_d(N)]$: the positive [negative] polarity delay from the inputs to node N using a domino implementation.

$p\_delay_s(N)[n\_delay_s(N)]$: the positive [negative] polarity delay from the inputs to node N using a static implementation.

$p\_area_d(N)[n\_area_d(N)]$: the area cost of implementing the positive [negative] polarity logic of node N using domino gates.

$p\_area_s(N)[n\_area_s(N)]$: the area cost of implementing the positive [negative] polarity logic of node N using static gates.

The cost estimation at each vertex includes both delay and area information, and the following quantities are calculated at each node. Note that $p\_area_s(N)$ and $n\_area_s(N)$ can differ by at most two since one can be obtained by placing an inverter at the output of an implementation of the other. Therefore, storing only one of the two values will lead to a negligible loss in accuracy. However, for domino circuits, due to the requirement of the unateness property, the positive and negative polarities of a node are implemented separately, which may lead to a significant difference in their values.

5.3.3 Determining the Candidate Cut Nodes

Once a cost estimation at each vertex has been obtained, the next task is to determine the region that can be partitioned without violating the timing constraints. We introduce the idea of a *candidate cut node*, illustrated in Figure 5.2.

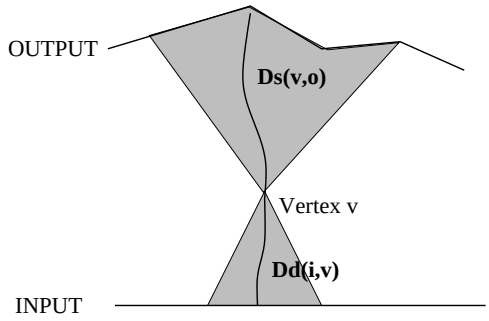

Figure 5.2: Determination of candidate cut nodes

Assume that v is some vertex in N_{twin}. If the cutset passes through the node, then the input transition cone of v will be implemented by domino gates and the output transition cone of v will be implemented by static gates. Let $D_d(i, v)$ be the largest delay from the inputs to node v using a domino implementation, and $D_s(v, o)$ be the largest delay from the node v to outputs through paths using static logic. Then the maximal delay from the input to output that passes through the cut at node v will be $D_d(i, v) + D_s(v, o)$. If this value is smaller than the specified delay, T_{spec}, then the cut through node v is eligible; if not, it is certain to violate the timing constraint. The procedure of finding candidate cut nodes makes this determination at each vertex.

The value of $D_d(i, v)$ is directly obtained from $delay_d(v)$ from the technology mapping phase. The value of $D_s(v, o)$ can be obtained using a PERT-like procedure [65] to traverse from the outputs of the network towards its inputs. The algorithm for finding candidate cut nodes is as follows:

ALGORITHM: **Find_candidate_cut_nodes**

For all nodes i, $D_s(i, o) = 0$;

Perform a PERT traversal from outputs to inputs.

At each node v

{

If $D_s(v, o) + delay_d(v) \leq T_{spec}$

v is a *candidate cut node*;

Slack = $\max[delay_s(v) - delay_s(i)]$, $\forall i \in inputs(v)$;

For each input i of v

$D_s(i, o) = \max[D_s(i, o), \text{Slack} + D_s(v, o)]$;

}

5.3.4 Finding the Minimum Cut

After all of the candidate cut nodes have been found, a maximum flow network is built using these nodes using an identical DAG structure as in the original circuit, with capacities assigned to the edges. The max-flow min-cut algorithm is then applied to this network to find the minimum cut.

5.3.4.1 Maximal Flow Capacity Assignment

The DAG technology mapping procedure performs the mapping of a Boolean network to the gates in a parameterized library. For a multifanout node in the network, the area contribution of the node is divided by the fanout count of the node, as in [47], so that the area value of each node, $area(N)$, is the approximate area contribution of its fanin transition cone. Therefore, for any cut on the graph, the area cost in the region from the primary inputs to each cut set node can be approximated as $\sum_{i \in CutSet} area(N_i)$, where the value of $area_d(N_i)$ is set to $\min[p\_area_d(N), n\_area_d(N)]$. In the example of Figure 5.3, if nodes a, b, c and d form the cut vertices, then the area of region Y is calculated as $area_d(a) + area_d(b) + area_d(c) + area_d(d)$.

For a given cut that divides the network into two parts, referred to as Region X (closer to the inputs) and Region Y (closer to the outputs), the cost can be calculated as follows. Assume the area cost of the entire network implemented in static logic is $A_s(sum)$, and the area costs for Region X to be implemented in static and domino

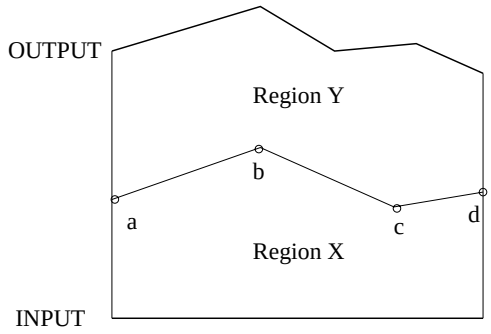

Figure 5.3: Evaluating the cost of a cut

logic are, respectively, $A_s(X)$ and $A_d(X)$. Then the total cost after partitioning, with Region X implemented in domino logic and Region Y implemented with static gates is $A_s(sum) - A_s(X) + A_d(X)$. Since $A_s(sum)$ is constant over all partitions, the objective of minimizing cost is equivalent to minimizing $-A_s(X) + A_d(X) = \sum_{i \in cutset}(area_d(i) - area_s(i))$.

Based on this, we reduce the problem to one of finding a minimum cost vertex cut on the network. The capacity associated with each node i is set to $area_d(i) - area_s(i)$. The vertex cut must maintain the predecessor constraints that dictate that no static node can feed a domino node.

5.3.4.2 Building the Maximum Flow Graph

The maximum flow algorithm is a well-established approach for finding minimum cost cuts. The procedure of building the maximum flow network can be illustrated on the example circuit of Figure 5.1.

After performing static and domino mapping on N_{twin} and determining the candidate cut nodes, we obtain an example circuit shown in Figure 5.4 (a). The shaded part of the network shows the region containing the candidate cut nodes. There are two weights on every node, which represent the result of domino mapping, $area_d(i)$

and the result of static mapping, $area_s(i)$, respectively.

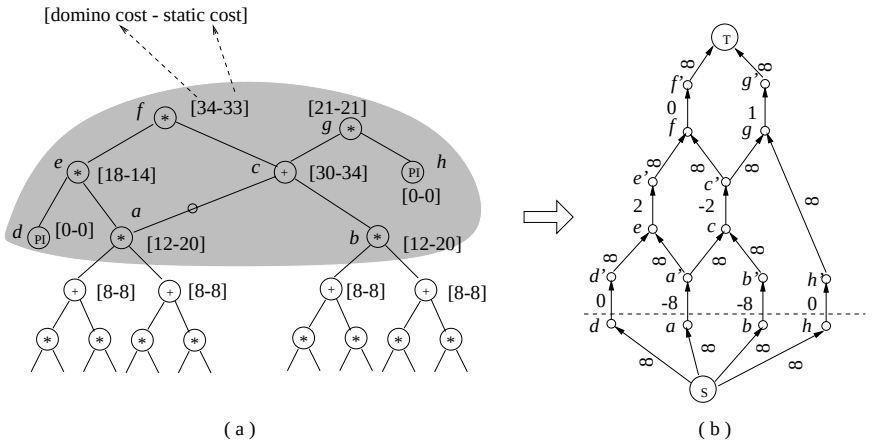

Figure 5.4: Boolean network after mapping and candidate cut node decision

Finding the minimum cost cut of the above network is equivalent to finding the minimum cost cut on the maximum flow graph shown in Figure 5.4 (b). In the list of candidate cutsets, the nodes in the shaded region that are closest to the primary inputs are connected to the source node, and the nodes of the region that are closest to the primary outputs are connected to the sink node. Each vertex in Figure 5.4 (a) is split into two vertices connected by an edge of capacity $area_d(i) - area_s(i)$.

However, before the standard max-flow min-cut algorithm can be applied on the network, two conditions in the network must be considered:

1. The vertex cut must maintain the predecessor constraints that dictate that no static node can feed a domino node. The solution to this problem is provided in the work in [67, 68].

2. Standard maximum flow algorithms cannot handle edges with negative capacities, and the network must be modified suitably.

To solve the problem, we heuristically transform the vertex-cut maximum flow

network into an edge-cut maximum flow network and and translate it into a standard maximum flow network with nonnegative edge capacities.

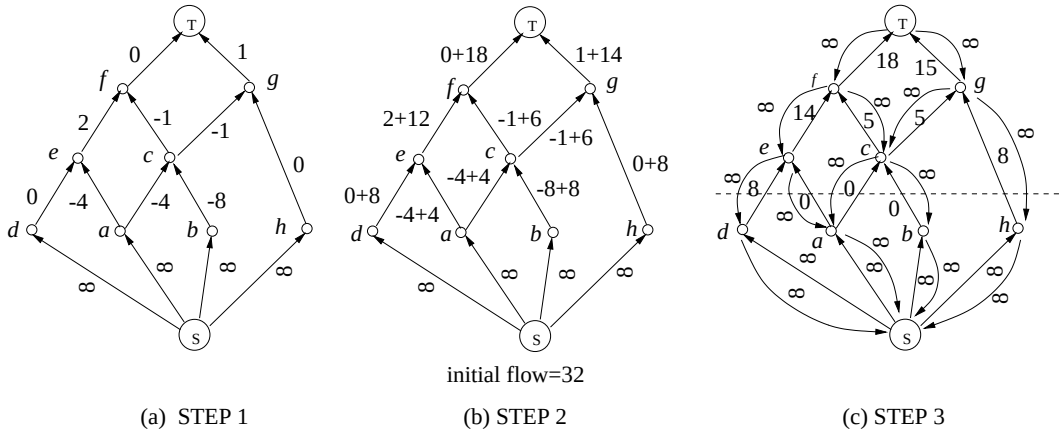

Figure 5.5: Constructing the edge-cut maximum flow network

The procedure consists of the following steps:

1. Building the edge cut maximum flow network.
   If (u, v) is an edge originating at the candidate cut node u in N_{twin}, the capacity of the edge is heuristically assigned to $C_{init} = (area_d(u) - area_s(u))/fanout(u)$, where $fanout(u)$ is fanout number of node u.

2. A positive initial flow is injected into the source node, and the initial flow is distributed into the whole network. The flow at each node is calculated by a PERT-like traversal on the DAG, with the flow from node u to a fanout node v being calculated as $Flow(u, v) = \sum_{i \in input(u)}(Flow(i))/fanout(u)$. Since this is a feasible flow, updating the capacity C of each edge as $C_{modified}(uv) = C_{init}(uv) - Flow(uv)$ leaves the identity of the minimum cost cut unchanged. This procedure is repeated until the value of $C_{modified}$ for each edge is nonnegative; it is easily verified that this method must converge.

3. For each edge (u, v) in the graph, a new edge (v, u) with a capacity of ∞ is

introduced into the graph to force the predecessor constraint.

The max-flow min-cut algorithm is then applied to the network to obtain the minimum cost cut.

5.4 Timing Driven Two-way Domino Partitioning

The problem of two-way domino partitioning of a circuit under a two-phase clock is to determine which gates in the circuit should be clocked by the first clock phase, and which by the second clock phase. Like the static-domino partitioning scenario described earlier, there is an inherent precedence constraint that states that no gate clocked by Φ_2 may fan into a gate clocked by Φ_1. The common features of the two problems is that the partitioning itself influences the cost of implementation area. The differences between the two lie in the manner in which the candidate cut node is chosen, and how the capacities in the max-flow network are assigned. These differences are summarized as follows:

Determining the candidate cut nodes For any vertex N of the input network, let $D_d(i, N)$ be the largest delay from the inputs to node N and let $D_d(N, o)$ be the largest delay from node N to the outputs, calculated using a reverse PERT traversal as before. The physical meaning of the cut in this situation is that if the cutset passes through some node N, then all gates in the fanin transition cone of N will be clocked by clock phase Φ_1, and all gates in the fanout transition cone of N will be clocked by Φ_2. Therefore, since candidate cut nodes are those that satisfy the timing constraints in each phase and no time borrowing between two phases, a vertex is a candidate cut node if both $D_d(i, N)$ and $D_d(N, o)$ are smaller than the clock pulse width.

Maximal flow capacity The two clock phase regions must be separated by a latch, which provides the ability to invert a signal. The cost function here is the cost of latches and the cost of logic duplication.

In the input network, some of the logic nodes must be duplicated since the logic function is required in both true and complemented forms, while some of the other logic outputs are required only in one polarity and need no duplication. There are two possible cases:

- Assume that N is a node at which both the true and complemented form of the logic function are required. If this node does not belong to the cutset, then the logic must be duplicated, and hence the area cost at this node is $p\_area_d(N) + n\_area_d(N)$. If the node lies on the cut, then only one polarity of the logic needs to be generated and the other is generated by inverter or within the latch placed at the cut. Therefore, the area contribution of the node can be modeled as $\min[p\_area_d(N), n\_area_d(N)]$. Therefore, the area cost difference between placing node N on the cutset or not is the $p\_area_d(N)$ $(n\_area_d(N)])$, plus the latch area.
- On the other hand, for a node N at whose output only one polarity is required, the area cost difference between passing a cut through N or not is merely the area of the latch.

In either case, the capacity corresponding to a cut at this node is represented by the area cost difference values defined above.

5.5 A Partitioning Flow for a General Two-phase Clocking Strategy

Using the solutions developed in previous sections, we solve the problem under the general two-phase clocking scheme in Figure 2.2. We present two design flows, the first of which is **Flow 1** below:

1. We first perform static-domino partitioning on the entire network to divide the circuit into domino and static regions, labeled, respectively, as Region 1 and Region 2. The timing constraint specified here is a full clock cycle.

2. We now specify the required time at the output of Region 1. Next, we perform two-way domino partitioning on this region to obtain the domino partition to be (tentatively) clocked by phase Φ_1 of the clock (Region 3) and the region to be clocked by phase Φ_2 of the clock (Region 4).

3. Region 3 is now assigned to Phase 1; however, we now investigate the possibility of implementing a part of it in static logic. Therefore, we perform static-domino partitioning on this region to divide it into a phase Φ_1 domino region (Region 5) and a phase Φ_1 static region (Region 6).

The end result of this procedure is that Region 2 and Region 6 are implemented in static CMOS, Region 5 is implemented as phase Φ_1 domino, and Region 4 as phase Φ_2 domino.

Another possible partitioning design flow (**Flow 2**) first performs two-way domino partitioning on the input circuit, followed by a static-domino partitioning step on each of the two resulting partitions.

5.6 Cost Modeling

In the discussion so far, we have used the area as the cost function for partitioning. However, the algorithm may be adapted to other cost objectives, and in this section we will describe a power model incorporated into this algorithm.

5.6.1 Power Model

The dynamic power dissipation is given by

$$Power = P_t \cdot C_L \cdot V_{dd}^2 \cdot f_{clk} \tag{5.1}$$

where P_t is the transition probability, C_L is the loading capacitance, V_{dd} is supply voltage, and f_{clk} is the clock frequency. Since we assume that V_{dd} and f_{clk} are constant over the optimization, it is enough to consider $P_t \cdot C_L$ as the objective. In the succeeding description, we denote the probability that a node is at logic 0 or 1 as $P(0)$ and $P(1)$, respectively.

Power Model for a Domino Gate

A domino gate must be precharged during every clock cycle. Therefore, some nodes may be immediately discharged during evaluation after being precharged, possibly leading to a higher activity factor [38]. For domino logic, the probability of a 0-to-1 transition at the output of the domino gate is the probability that domino gate evaluates to 1. This is identical to the probability of a 1-to-0 transition at the dynamic node d in Figure 2.1. Since each 1-to-0 transition must correspond to a 0-to-1 transition, the transition probability at these nodes is given by $P_t = P(1)$.

Under the assumption all transistors of the same type have the same size, and that pmos transistors are sized to be larger than nmos transistor by a factor *cpnratio*, the

pmos to nmos gate capacitance ratio is *cpnratio*. Therefore, the power dissipated by the gate is $P(1) \times ((1 + cpnratio) + N_{fanout}) \times C_{gate}$, where N_{fanout} is the number of fanouts driven by the gate and C_{gate} is the gate capacitance of an nmos transistor. In the formula, the quantity "$1 + cpnratio$" represents the nmos and pmos transistors of the inverter, corresponding to the load driven by the dynamic node, and N_{fanout} represents the load driven by the domino gate.

We also model the power dissipation of the clock tree required to drive the domino gates as part of power consumption. The two clock controlled transistors make a transition each time the clock signal changes, and have a transition probability of 1. Therefore we model the power consumption for the domino due to the clock network by $(1 + cpnratio) \times C_{gate}$. The power due to additional routing may also be modeled by an additive factor here, but we have not considered it in our experimental results.

Therefore, the total power dissipation of the gate is

$$P_{domino} = C_{gate} \cdot [(P(1) \cdot (1 + cpnratio + N_{fanout}) + (1 + cpnratio)] \tag{5.2}$$

Power Model for a Static Gate

For static gates, the transition probability P_t is well known to be $P(1) \times (1 - P(1))$, and the equation for the power dissipation is

$$P_{static} = P(1) \cdot (1 - P(1)) \cdot C_{gate} \cdot ((1 + cpnratio) \cdot N_{fanout}) \tag{5.3}$$

Other Considerations

Since domino logic has the advantage of having no glitching transitions and significantly smaller short-circuit dissipation, we alter the power equations above to reflect this fact. We make a simple modification to the static power dissipation equation

given by (5.3) as

$$P_{static} = P(1) \cdot (1 - P(1)) \cdot C_{gate} \cdot ((1 + cpnratio) \cdot N_{fanout}) \cdot c_{glitch}, \quad (5.4)$$

where $c_{glitch} > 1$ is a constant to compensate for the additional power dissipation error caused by glitches. Similarly, the domino power dissipation equation of (5.2) is modified to

$$P_{domino} = P(1) \cdot C_{gate} \cdot ((1 + cpnratio + N_{fanout}) \cdot c_{short} + (1 + cpnratio)), \quad (5.5)$$

where $c_{short} < 1$ is a constant to make up the power estimation error caused by short circuit consumption.

5.6.2 Delay Model

A transistor level Elmore delay model that is similar to that in [69] was used here. While the Elmore delay is known to be limited in its accuracy, it is reasonable to use it here for a fast and approximate delay estimate early in the design flow shown in Figure 2.5.

Delay Model for a Domino Gate

The delay of a domino gate is given by

$$D_{domino} = R_n \cdot (1+H) \cdot C_{gate} \cdot (1+cpnratio) \cdot c_{domino} + R_p \cdot C_{gate} \cdot N_{fanout} + Delay_{para}, \quad (5.6)$$

where R_n, R_p are the driving resistances of the NMOS transistor and the PMOS transistor, respectively, H is the maximum height of the NMOS pull-down network, c_{domino} is a delay reduction factor attributed to the absence of short circuit current during transition of domino gate. The first term in Equation (5.6) represents the delay of NMOS pull-down network that is driving an inverter. The existing clock controlled

transistor at the bottom of the NMOS pull-down network contributes the $(1 + H)$ factor in Equation (5.6). The second term in Equation (5.6) represents the delay of inverter that is driving the fanout gate capacitance. The third term represents the delay caused by parasitic capacitance due to source/drain capacitance within the gate. The $Delay_{para}$ is calculated by modeling a gate as a distributed RC network and finding the corresponding Elmore delay expression.

Delay Model for a Static Gate

Separate rising and falling delays were used for static gate. They are given by

$$D_{r,static} = R_n \cdot H \cdot C_{gate} \cdot ((1 + cpnratio) \cdot N_{fanout}) + Delay_{para}$$
$$D_{f,static} = R_p \cdot W \cdot C_{gate} \cdot ((1 + cpnratio) \cdot N_{fanout}) + Delay_{para},$$

where H, W is the maximum height of the NMOS segment and the PMOS segment of the static gate, respectively.

5.7 Experimental Results

The entire partitioning mapping flow has been implemented using C++. The parameterized library domino technology mapping algorithm of [9] is used for the domino technology mapping. The parameterized library contains all possible cells with up to 4 transistors in series and 4 transistors in parallel. The static technology mapping method modified from [9] is used for the static technology mapping to a parameterized library that contains all possible cells with up to 3 transistors in series and 3 transistors in parallel. The mapping objective is area minimization. An initial technology-independent logic minimization for all circuits was performed using SIS with *script.rugged* script file. The number of transistors is used to represent the area.

The delay estimation is obtained from the delay model described in Section 5.6.2. The primary inputs are assumed to be available in true and complemented form, and it is assumed that each of these is driven by a flip-flop that provides that inverted signal at no cost. The partitioning algorithm is applied on the LGSynth91 multi-level circuits.

Table 5.1: Results of the static-domino partitioning algorithm

Circ-	Domi	Static	No spec		Spec=(×1.25)		Spec=(×1.05)		CPU
uits	N_{tran}	N_{tran}/Del	N_{tran}	G_d/G_s	N_{tran}	G_d/G_s	N_{tran}	G_d/G_s	(s)
c1355	1824	1302/2.25	1302	0/260	1800	170/104	1800	170/104	1.4
dalu	2360	2192/2.16	2098	97/198	2096	147/132	2096	189/75	7.9
c880	1163	982/2.08	958	21/124	1015	56/88	1027	62/85	1.4
count	357	336/2.77	344	5/54	350	23/30	353	32/18	0.3
c1908	1978	1308/1.78	1306	5/263	1723	174/86	1928	238/34	1.4
c2670	1992	1754/1.75	1775	79/173	1775	79/173	1774	81/170	3.5
c3540	4527	2850/1.43	2748	88/349	3312	260/218	3987	461/26	10.9
c6288	13702	8350/1.78	8340	16/1771	12079	1301/493	13456	1733/73	33.5
k2	2884	2896/1.54	2884	368/68	2884	368/68	2884	368/68	8.6
des	9945	8134/4.25	7527	160/915	7536	165/911	7536	165/911	60.2
c7552	7919	5464/2.35	5370	78/296	5987	375/578	6198	456/504	30.9
t481	1697	1832/1.35	1695	203/19	1695	203/19	1693	206/15	3.7
rot	1777	1536/1.99	1462	55/171	1514	87/137	1611	126/103	3.0

The results of static-domino partitioning are shown in Table 5.1. The second and third columns show the results of pure one-phase domino mapping and pure static mapping, respectively. The delay of the static implementation is also shown. All delays in this table are normalized so that "×1.0" corresponds to the delay of a purely domino implementation. Next, for various delay specifications, we list the number of transistors, N_{tran}, and the number of domino/static gates (G_d/G_s) using our method.

From the table, we can see that when the timing constraints are larger, the cost is smaller than the minimum of columns 2 and 3, as expected. We observe that domino implementations of the benchmarks usually have a speed advantage over static circuits, but tend to have larger areas. Therefore, the tighter the timing constraints, the more domino gates are required, and the larger the area cost. The average CPU execution time of static-domino partitioning including technology mapping is shown in column 10.

Table 5.2: Results of the two-way domino partitioning algorithm

Circuits	Domino N_{tran}	No spec N_{tran}	S=(×1.05) N_{tran}	CPU (s)
c1355	1824	1656	1600	1.8
dalu	2360	1971	2080	10.0
c880	1163	933	1037	4.6
count	357	267	347	0.4
c1908	1978	1867	1838	2.7
c2670	1992	1703	1705	6.0
c3540	4527	3499	3499	11.7
c6288	13702	13173	13170	96.4
k2	2884	2856	2920	9.6
des	9945	8265	10835	111.5
c7552	7919	6434	6607	44.6
t481	1697	1638	1812	5.43
rot	1777	1422	1638	4.9

The results of two-way domino partitioning are shown in Table 5.2. Column 2 shows that the results of one phase domino implementation are identical to Column 2 of Table 5.1. Column 3 shows the two way partitioning results without any timing specification and Column 4 shows the results under timing specification of ×1.05. If

the maximum delay for the domino mapping is x, then $\times 1.05$ specification represents the fact that a delay constraint of $x \times 1.05 \times 0.5$ is assigned to each phase of domino partition, corresponding to the fact that the total delay must be evenly distributed over the two partitions for a symmetric clock scheme. The area cost for the latches and inverters between two phases is not counted. In most cases, the resulting area is about the same or larger for the tighter specification, as expected. In a few cases, a slightly smaller area is obtained (but within a reasonable margin of error); this is due to approximations in the cost estimation for the maxflow problem. The differential between N_{tran}(no spec) here and column 2 shows the reduction in logic duplication obtained by exploiting the inverters at the partition boundary. The CPU execution time for the two-way domino partitioning procedure is shown in the last column.

Table 5.3: Results of applying the partitioning flows for the two-phase clocking scheme

Circuits	N_{tran}/N_{latch}			
	Flow 1 $\times 1.25$	Flow 1 $\times 1.05$	Flow 2 $\times 1.25$	Flow 2 $\times 1.05$
c1355	1408/8	1456/8	1452/48	1486/48
dalu	1998/56	2050/78	1923/63	2049/117
c880	944/13	953/14	926/43	943/43
count	346/9	345/14	337/23	338/23
c1908	1449/46	1560/46	1519/40	1590/60
c2670	1538/52	1538/52	1548/95	1548/95
c3540	3063/60	3235/68	2943/53	3277/53
c6288	11604/104	12511/115	12105/110	12410/111
k2	2691/157	2795/152	2862/147	2889/156
des	7510/118	7513/119	8452/437	8766/437
c7552	5754/164	5772/164	5701/192	5892/194
t481	1701/84	1752/90	1831/80	1833/85
rot	1463/36	1515/51	1485/118	1538/119

The results of **Flow 1** and **Flow 2** from Section 5.5 for the general two-phase clocking scheme are shown in Table 5.3. We lists the number of transistors, N_{tran}, and the number of latches, N_{latch}. The experiments were executed under two sets of delay specification of $\times 1.05$ and $\times 1.25$. From the results, we can see that that while we finish the task of partitioning the combinational circuits to fit into the two-phase clocking strategy, we use much less area than domino logic and obtain the almost the same speed as the domino implementation. If the timing constraints are more relaxed, the resulting area will be even smaller. Therefore, our method can be used to find an area-delay tradeoff curve for static-domino partitioning. It is observed that **Flow 2** introduces more latches than **Flow 1**. This can be explained by the fact that it begins with an initial two-way domino partition and hence must insert latches at the boundaries even if they are not essential to the final partition. The variation in the number of transistors with the delay specification shows the expected trends.

The partitioning algorithm was also be applied to the power model provided in Section 5.6.1 and the results for static-domino partitioning are shown in Table 5.4. Values of power consumption are in normalized units. The second and the third columns show the power dissipation as a result of pure domino mapping and pure static mapping, and the last three columns show the power corresponding to different delay specifications.

5.8 Conclusion

In this chapter, we have explored algorithms for timing-driven static-domino partitioning and for timing-driven two-way domino partitioning. The experimental results are executed with the aim of minimizing both the area and the power dissipation. Our results show that partitioning can effectively reduce the logic duplication penalty. The two algorithms are then applied to partition an input network so that it con-

Table 5.4: Results of static-domino partitioning for power minimization

Circuits	domino	static	static-domino partitioning		
			×1.5	×1.2	×1.05
c1355	875.1	242.9	709.6	712.2	868.7
dalu	1232.1	303.4	368.6	365.3	654.4
c880	506.1	151.6	178.1	187.2	311.5
count	165.5	26.1	87.7	111.8	119.6
c1908	1016.5	230.5	458.0	913.8	968.6
c2670	842.9	340.2	296.3	319.5	493.6
c3540	2211.1	539.4	698.1	1276.3	1697.1
c6288	7112.9	2144.0	3867.6	6047.4	6960.6
k2	2205.7	185.4	367.7	520.8	640.7
des	5347.4	1533.1	1274.5	1321.9	1321.9
c7552	3963.3	1481.4	1619.1	1891.3	1994.5
t481	1300.6	161.8	199.0	525.7	551.6
rot	773.4	248.8	346.5	459.0	465.1

forms to a common two-phase nonoverlapping clocking scheme. Our results show that the area of the original one-phase domino network implementation is reduced significantly, while maintaining the circuit speed of a pure domino implementation.

Chapter 6

Timing Verification and Sizing Optimization of Mixed Static-Domino Circuits

6.1 Introduction

The correct functionality of a circuit containing domino logic is contingent on correct timing relations between the clock and logic signals of domino gates. For a given circuit topology consisting of static and domino gates, timing verification and optimization is an essential step of the design process. In different designs, domino circuits may utilize various clocking schemes; some typical clocking strategies are shown in Figures 2.2, 2.3 and 2.4 of Section 2.2. In addition, various self-timed domino and wave-domino circuits are also used. The goal of the section is to provide a general timing verification and sizing tool for mixed static-domino circuits that presents the problem in a similar framework as the corresponding solutions for static circuits.

The basis for this work lies in the timing analysis technique used in [28, 29] for

timing analysis at the gate level. Although several sizing algorithms have been published in the past (a survey is provided in [65]), most of them have not considered domino logic. Although the research of [30, 31] performs sizing for domino circuits, both techniques perform local optimizations, optimizing only one domino block at a time. In [32], a sizing tool for domino style circuits called Focus was developed.

The rest of this chapter is organized as follows: In Section 6.2 we present the long path and short path time constraints domino logic gate, followed by the timing verification and sizing algorithms described in Section 6.3. Section 6.4 discusses the charge sharing measurement and correction problem after timing verification and sizing. Finally, we present the experimental results in Section 6.5 and conclude the chapter in Section 6.6.

6.2 Domino Logic Timing Constraints

Let us first consider the timing constraints of an individual domino gate used in our timing analysis and sizing optimization tool. The domino logic timing constraints listed below are based on [28, 29], but we express them at the transistor level. In addition, we differ from that work in offering further insights into domino timing constraints by using more accurate short path timing constraints.

The clock input to the domino gate is shown in Figure 2.1. The precharge phase begins at $T_{clk,f}$ and continues until $T_{clk,r}$, and the evaluate phase begins at that time and ends at time $T_{clk,f} + P$ where P is the period of the clock signal feeding the domino gate. The reference time $t = 0$ is set with respect to the clock signal at the primary input of the circuit block. If more than one clock signal is used, any one of them may be used as a reference, and the transition times of the other clocks may be expressed according to the reference.

6.2.0.1 Long Path Timing Constraints

We list the node timing constraints for domino logic as follows, in terms of the signal arrival times and the clock arrival time. In case of multiple clocks, the clock signal *clk* should be set to be the clock signal that feeds the gate that is currently under consideration.

(i) Any falling event at a data input should meet the setup-time requirement to the rising edge of the evaluate clock. If $T_f(in)$ refers to the falling event time of the input node, then we require that

$$T_f(in) \leq T_{clk,r} - T_{setup} \tag{6.1}$$

where the setup time T_{setup} is a constant that acts as a safety margin.

(ii) The rising event of the output node of the domino gate must be completed before the falling edge of evaluate clock. If $T_r(out)$ refers to the rising event time at the output node, then the circuit operates correctly only if

$$T_r(out) \leq T_{clk,f} + P \tag{6.2}$$

In other words, before the beginning of the precharge for next cycle, the correct evaluation result must have traveled to the output node. For example, in Figure 2.1, the rising event at the output node o of the domino gate must satisfy (6.2). Since we can write

$$\begin{aligned} T_r(out) = \quad & \max((T_r(x) + D_f(x,d), T_r(y) + D_f(y,d), T_r(z) + D_f(z,d), T_{clk,r} + D_f(clk,d))) \\ & + D_r(d, out)), \end{aligned}$$

where $T_r(x)$, $T_r(y)$, $T_r(z)$ are the rising event times at inputs x, y and z, respectively, $D_f(i,d)$ represents the delay of a falling transition at the dynamic node d due to a

rising transition at input $in \in \{x, y, z\}$, and $D_r(d, out)$ represents the rise delay of the inverter feeding the gate output node out. Therefore for $in \in \{x, y, z\}$, we get

$$T_r(in) + D_f(in, d) + D_r(d, o) \leq T_{clk,f} + P \tag{6.3}$$

$$T_{clk,r} + D_f(clk, d) + D_r(d, o) \leq T_{clk,f} + P \tag{6.4}$$

The relation (6.3) corresponds to the requirement that the rising edge of each input should appear in time for the falling edge of the evaluate clock so as to allow sufficient time for the output to be discharged [29]. To be more conservative, we also add the delay through the inverter and ensure that it is allowed sufficient time to discharge. The relation (6.4) ensures that the pulse width of the evaluate clock is sufficient for pulling down the output node [29].

(iii) The rising event d of the domino gate must be completed before the rising edge of the evaluation clock, i.e.,

$$T_r(d) \leq T_{clk,r} \tag{6.5}$$

If we denote the rise time of the dynamic node through the p-transistor fed by the clock as $D_r(c, d)$, then the rising event time can be expressed as:

$$T_r(d) = T_{clk,f} + D_r(clk, d) \tag{6.6}$$

This leads us to the constraint given by

$$D_r(clk, d) \leq T_{clk,r} - T_{clk,f} \tag{6.7}$$

This constraint implies that the pulse width of precharge clock must be capable of pulling up the output node.

6.2.0.2 Short Path Timing Constraints

Instead of using the conservative constraints of [29] or more aggressive constraints of [28], we use different constraints for falling and rising edges, which is more accurate timing request for the correct function of domino logic.

Suppose $t_f(in)$ and $t_r(in)$ are, respectively, the earliest falling and rising event time of input signal. A falling input data line may not go down to logic 0 but must be held at the logic 1 level until the output transition in that cycle has been completed. It is given by

$$t_f(in) + P \geq T_r(out) \tag{6.8}$$

For any rising input data line, the hold constraint states that a transition should occur only after $T_{clk,f}$, i.e.,

$$t_r(in) \geq T_{clk,f} \tag{6.9}$$

Note that if the falling signal arrives before the evaluation phase, it will not influence the output as long as the dynamic node has already been discharged. If, however, the rising signal arrives earlier than the end of the evaluation phase, it may cause a glitch at the output. The presence of $T_r(out)$ in Equation (6.8) causes the short path timing analysis to be related to the long path timing analysis.

6.3 Timing Verification and Sizing

6.3.1 Timing Verification

The timing analysis procedure described here is based on the PERT procedure and uses a table-lookup delay model for delay calculation. The long path timing analysis consists of two steps.

First, the circuit is forward-traversed, beginning with the primary input nodes and the clock node. The rising and falling event arrival times for each node v are calculated as follows:

$$T_r(v) = \max(T_f(u) + D_r(u, v)) \tag{6.10}$$

$$T_f(v) = \max(T_r(u) + D_f(u, v) \tag{6.11}$$

where $T_r(u)$ and $T_f(u)$ are, respectively, the rising and falling event times for nodes u and v, and $D_f(u,v)$ and $D_r(u,v)$ are, respectively, the worst fall delay and rise delay from input u to output v. The domino clock input node is treated in the same way as any primary input node, and the rising or the falling edge of the clock provide the corresponding event times for the clock node. The rising and falling event arrival times at the output node of a domino gate can be obtained similarly to the static gate arrival time computations, using equations (6.10) and (6.11). The only difference is that the rising event at the dynamic node is related only to the falling edge of the domino clock and is independent of the other input nodes. This fact is captured by setting the value of D_r from each input node of the domino gate to the output node as $-\infty$. At the end of this traversal, the arrival time at each node has been computed.

Next, a backward traversal is carried out to calculate the required time at each nodes. Before traversal, the required time of domino gates from constraints (6.1), (6.2), (6.5) and the required time at each primary output from synchronizer have been assigned the corresponding nodes. Beginning with the primary output nodes, we make a reverse PERT pass back through constraint graph to compute the required time at each node and the slack associated with every edge. At the same time, the critical path, defined as the path with maximum negative slack, is found. During this second traversal, the algorithm keeps a record of the minimum negative slack. If an edge with same slack is encountered, then it is incorporated the into critical path. If an edge with smaller (more negative) slack is traversed, then the previous critical path is discarded and the critical path is updated to begin at that edge. Note that due to the presence of constraints at each domino gate, the critical path does not necessarily terminate to a primary output and could terminate at a domino gate instead. Additionally, it is possible for the origin of the critical path to be either at a PI or at the clock node.

Constraints (6.8), (6.9) provide the short path constraints. After long path timing

analysis is performed, $T_r(out)$ in constraint (6.8) has been computed. A similar procedure as long path timing verification can be applied to support short path timing verification.

6.3.2 Sizing Algorithm

If long path timing constraints are found to be violated after applying the above timing analysis procedure, a sizing algorithm is applied. For short path violations, delay insertion techniques [70] are more appropriate, and are not addressed in this work. However, constraint (6.8) is incorporated into the sizing procedure. The sizing problem is formally stated as follows:

$$
\begin{aligned}
&\text{minimize} && Area && (6.12)\\
&\text{subject to} &&&&\\
&\max(T_r(o), T_f(o)) \leq T_{spec} && \forall o \in PO &&\\
&T_f(in) \leq T_{clk,r} - T_{setup} && \forall in \in I_{domino} &&\\
&T_r(out) \leq T_{clk,f} + P && \forall out \in O_{domino} &&\\
&T_r(d) \leq T_{clk,r} && \forall d \in D_{domino} &&\\
&T_r(out) \leq t_f(in) + P && \forall out \in O_{domino} &&\\
&K_1 \leq \frac{W_p}{W_n} \leq K_2 && \forall \text{ gates in the circuit.} &&
\end{aligned}
$$

where *Area* is the area of the circuit and, as in other work on transistor sizing [69], is approximated as a sum of transistor sizes, PO is the set of primary outputs, I_{domino}, O_{domino} and D_{domino} are, respectively, the set of inputs, outputs dynamic nodes of all of the domino gates in the circuit. The last constraints come from the noise margin that we will address in Section 6.3.3.

The sizing algorithm used here is an adaptation of the TILOS algorithm [69]. Beginning with a circuit where all transistors are minimum-sized, each iteration selects

one transistor and increases its size by a constant factor. In each iteration, a timing analysis is performed to identify the constraint $g(\mathbf{w}) \leq 0$ with the largest violation, where $g(\mathbf{w})$ denotes the fact that the constraint g is a function of the vector $\mathbf{w}$ of transistor widths. The traceback procedure described above is used to determine the critical path of the circuit, which corresponds to that constraint. The sensitivity of the constraint function g to each transistor width is computed, and the width of the transistor with the most negative sensitivity is increased. The iterations continue until the timing specifications are all met, or until no further improvement is possible.

6.3.3 Noise Margins

Noise margin constraints are applicable to both static and dynamic gates. In [31], Chen and Kang describe a technique for deriving bounds K_1 and K_2 on W_p/W_n that will ensure that noise margin constraints are satisfied:

$$K_1 \leq Ratio = W_p/W_n \leq K_2 \tag{6.13}$$

For an inverter, it is a simple matter to verify whether *Ratio* satisfies the specified bounds or not. For complex gates, each domino gate is reduced to an equivalent inverter corresponding to the largest and smallest value of *Ratio*. During the sizing process, these are compared with K_1 and K_2, respectively, to ensure that during the sizing process, these bounds are not violated.

In other words, the constraint above corresponds to the following two constraints that are always maintained during sizing:

$$Ratio_{min} = W_{p(min)}/W_{n(max)} \geq K_1$$
$$Ratio_{max} = W_{p(max)}/W_{n(min)} \leq K_2$$

The value $W_{p(max)}(W_{n(max)})$ corresponds to the equivalent inverter width when all pmos (nmos) transistors in the complex gate are on, and $W_{p(min)}$ $(W_{n(min)})$ is the

equivalent inverter width when only the largest resistive path [65] of the complex gate is on.

6.4 Charge Sharing Measurement and Correction

After the timing verification and sizing algorithm is executed, charge sharing is measured from the timing information obtained earlier.

6.4.1 Estimation of Worst-case Charge Sharing

Charge-sharing noise is produced by charge redistribution between a dynamic evaluation node and internal nodes within the gate. The usual way [31, 71, 72] of estimating worst case charge sharing is as follows. During the precharge stage, the uppermost device of every n-stack is assumed to be off, so that only the capacitance at the dynamic node, C_d is precharged. In the evaluate stage, the bottommost devices in the n-stack are configured to be off, and all devices above these in the n-stack are assumed to be on, and the total capacitance that now shares charge with the dynamic output node is $C_d + C$, where C is the sum of all internal node capacitors. Therefore, charge sharing can cause the dynamic node can have a worst-case voltage given by the following expression instead of being at V_{dd}:

$$V_{worst} = V_{dd} \cdot \frac{C_d}{C_d + C} \tag{6.14}$$

For correct operation of the circuit, it is required that V_{worst} must be no smaller than a specified voltage, V_{spec}.

However, the above expression may be too pessimistic. If the worst-case arrival time for each input is known, and if we can identify a node n such that there is a path from the dynamic node d to n on which the rise transition on all transistors is

guaranteed to arrive sufficiently before time $T_{clk,r}$, then node n will be precharged and will not trigger charge-sharing. If C_{pre} is the total capacitance of all such nodes n, then we can arrive at a less conservative estimate of charge sharing that states that

$$V_{worst} = V_{dd} \cdot \frac{C_d + C_{pre}}{C_d + C} \tag{6.15}$$

where C_d and C are as defined in Equation (6.14).

The calculation of C_{pre} is illustrated by the example of Figure 6.1, which is taken from a fast adder [22]. The value of C in Equations (6.14) and (6.15) is $C_1 + C_2 + C_3 + C_4 + C_5 + C_6 + C_7 + C_8$. If we know that signal a4 arrives before $T_{clk,r}$ and that the arrival of a3, b3 is later than $T_{c,r}$, then we know C_1, C_3 should be precharged and C_2 may not be precharged. Therefore, the value of C_{pre} in (6.15) is $C_1 + C_3$. If instead, b3 were to arrive before $T_{c,r}$, then C_2 can also be added to C_{pre} and this would correspond to a smaller value of V_{worst}.

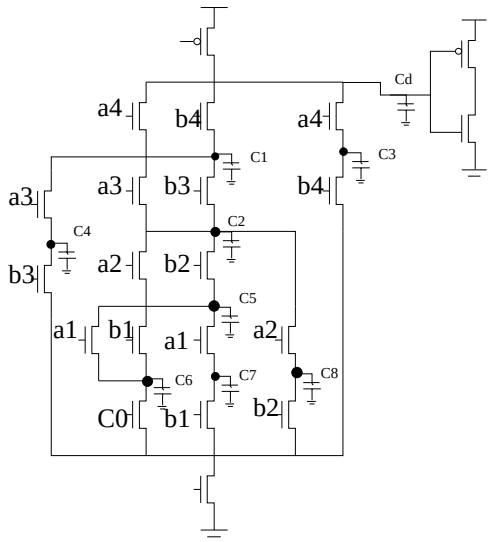

Figure 6.1: An example to illustrate charge sharing

6.4.2 An Algorithm for Reducing Charge Sharing

Charge sharing can be reduced by various methods, e.g., adding a keeper, changing the size of the inverter, using separate parallel branches by replacing the inverter with a NAND gate, and adding more pmos transistors [17]. In the CAD tool presented here, we consider two methods that are performed as a post-processing step after sizing has been completed.

One common method used by designers is to add one or more pmos transistors from the V_{dd} node to nodes the cannot be charged during precharge. Our algorithm suggests methods for choosing the best node to which a pmos transistor should be connected, and adds it automatically when charge sharing drops the output voltage below a specified bound, V_{spec}. As mentioned above, the worst case signal arrival time information is used. The procedure is interactive, and at each step, the designer can specify one of two possible ways to reduce charge sharing. This is continued until charge sharing constraints are met.

Method 1

The basic idea is that this added pmos should precharge the largest internal node capacitance values. For simple domino gates, a pmos transistor can be added intuitively. However, when the domino gate becomes more complex, the algorithm suggested here is helpful.

We note that if two nodes are connected by a transistor whose input arrives before evaluation, then precharging one node will also precharge the other node; if this is not true, then precharging one node would not precharge the other. We refer to any set of such nodes as a *channel-connected precharge set*. Our algorithm finds the channel connected precharge set with the largest total capacitance and connects a pmos to a node in that set. This procedure requires one traversal of the graph representing the channel-connected component [65]. This total capacitance is then added to C_{pre} in

Equation (6.14) and the worst case voltage due to charge sharing is calculated.

Method 2

A second method is to alter the node timing constraints for some input node. We note that at any input transistor, if the input rising event time lies between the end of precharge and end of evaluation begins at time $T_{c,r}$, then our worst-case charge sharing estimation assumes that even if the drain node of the transistor is precharged, the source node cannot be precharged. To relieve this problem, we may increase the value of C_{pre} by ensuring that the signal at the transistor gate node arrives before evaluation by sizing transistors in the circuit and set the required arrival time for the rise transition to be to be the rising edge of domino clock instead of the falling edge of domino clock.

As in the first method, we are faced with the problem of determining which nodes should have a modified timing specification. Here again, we search for the section with the largest sum of node capacitances, and we restrict the candidate sections to those whose neighbors can be precharged.

However, this method will not always work. When the input under consideration is the output of another domino gate with same domino clock, it is impossible for its rising event time to be smaller than precharge time. Moreover, if the arrival time has to be changed by a large amount, this will require a large amount of sizing elsewhere in the circuit, which may introduce charge-sharing problems at other points in the circuit. Therefore, we only permit this sizing operation if the input arrival time is greater than $T_{clk,r}$ by T_{thre}, where T_{thre} is the threshold value defined by the user.

6.5 Experimental Results

6.5.1 Timing Verification and Sizing

Table 6.1: Transistor sizing on circuits for the two-phase clocking scheme

Circuit	Unsized Area	Clk Spec Period(ns)	Optimized Area	CPU (s)
C1355	1582	1.8	1589	1.2
		1.6	1605	1.5
		1.4	-	2.3
dalu	2357	2.3	2453	7.2
		2.08	2524	10.4
		1.86	2711	15.7
		1.64	-	17.0
C1908	1764	2.6	1780	2.1
		2.2	1871	6.3
		1.8	2107	16.5
		1.4	-	31.8
des	9881	4	9910	10.1
		3.6	9977	13.7
		3.2	10156	26.4
		2.8	10506	57.1
C7552	6948	3.5	6954	5.9
		3.0	7015	14.8
		2.6	7210	38.7
		2.1	-	73.5

The timing verification and sizing tool has been implemented in C++, and takes an input in the format of a SPICE transistor netlist. The specifications applied on the circuits include the clocking scheme, output timing constraints, noise margins, and technology parameters. The characterization was performed in a 0.5 μm technology.

The factor, Bumpsize, for the TILOS-like algorithm is set to $\times 1.5$ of original size and the noise margin constraints require that Ratio $\in [1.0, 4.0]$.

The procedure has been performed on two sets of circuits of different clock strategies. A summary of the results are listed in Table 6.1 and Table 6.2, respectively. For each circuit, the original unsized area is listed. For various domino clock specifications listed in the "Clk spec" column, the results of sizing are listed. The output timing specification is always set to the corresponding clock period. In our sizing experiment, the duty cycle of the clock is fixed as the clock period specification is changed. The area is reported as "-" if the specifications are too tight to be satisfied.

Table 6.2: Transistor sizing on circuits for a four-phase overlapping clocking scheme

Circuit	Unsized Area	Clk Spec Period(ns)	Optimized Area	CPU (s)
C1355	2736	1.36	2744	1.4
		1.22	2774	2.2
		1.08	-	4.9
dalu	3414	2.1	3417	1.7
		1.88	3423	1.9
		1.22	3552	6.9
		1	-	13.3
C1908	2922	2.2	2929	1.6
		1.8	3091	8.6
		1.4	3554	23.3
		1	-	24.7
des	17368	2.8	17369	10.7
		2.4	17370	10.8
		2	-	17.3
C7552	10264	2.12	10321	11.8
		1.66	10614	44.3
		1.2	-	72.1

The first set of results, reported in Table 6.1 uses the two-phase clocking scheme of Figure 2.2, whose duty cycle is set of 1/2. To test the more complex timing relation of clock and domino gate, we apply our procedure to four-phase overlapping domino clocks of Figure 2.3, whose duty cycle is set to 1/3. The results are listed in Table 6.2. Both the long path and short path timing verification are performed for the input circuit.

6.5.2 Charge Sharing

The procedure of charge sharing is a post-processing phase after timing verification and sizing. The results of the application of the charge sharing algorithm on the example in Figure 6.1 is illustrated in Table 6.3. The set of signals that arrive before time $T_{clk,r}$ are listed as the early signals in the first column. The second column shows the parameter $\frac{V_{worst}}{V_{dd}}$, which is defined in Equation (6.15). The node at which a PMOS transistor is added is defined in column Pnode, and the updated value of $\frac{V_{worst}}{V_{dd}}$ is listed after the addition of the first pmos and the second pmos, respectively.

Table 6.3: Application of the charge sharing algorithm on the circuit of Figure 6.1

# N($< T_{cr}$)	Ratio	1st Iteration		2nd Iteration	
		Pnode	Ratio	Pnode	Ratio
NULL	0.210	c5	0.361	c2	0.514
a1,b1,a3,b3,a4,b4	0.635	c5	0.939	c8	1
a1,b1,a2,b2,a3,b3	0.210	c1	0.939	c3	1
a1,b2,a3,a4,b4	0.939	c8	1	-	-
a1,a2,a3,a4	1	-	-	-	-

Table 6.4 shows the results when the above charge sharing algorithm are performed on benchmark circuits. The circuits are the same as those in 6.2. The threshold

value, V_{spec}, is set to 1. The sizing procedure is performed before the charge sharing procedure, with the clock specifications of all circuits set to around 80% of the delay of the minimum sized circuit under a four-phase overlapping clocking scheme. In Table 6.4, Column 2 shows the number of pull-up transistors that will be added without considering timing information and Column 3 shows the number needed with our method, and this shows that our method provides significant improvements. The CPU times on the circuits in Table 6.4 are under several seconds.

Table 6.4: Results of the charge sharing algorithm on large circuits

Circuits	Traditional Method (#added transistors)	Our method (#added transistors)
C1355	358	104
dalu	485	135
C880	225	46
count	64	0
C1908	342	132
C2670	492	76
C3540	915	365
C6288	2345	1574
k2	438	239
des	2209	242
C7552	1612	473
t481	373	36
rot	306	162

6.6 Conclusion

In this chapter, a timing verification and sizing optimization tool dealing with circuits constraining mixed domino and static logic is presented. Timing constraints

of domino logic gate in transistor level are addressed and more accurate short path timing constraints are provided. A timing analysis methodology permitting the application of PERT, allowing mixed static-domino circuits to be handled in a manner similar to static combinational circuits, is developed. Finally, a more accurate charge sharing estimation and correction procedure is proposed as a post-processing phase after sizing.

Chapter 7

Hierarchical Analysis of Power Distribution Networks

7.1 Introduction

With the increase in the complexity of VLSI chips, designing and analyzing a power distribution network has become a challenging task. A robust power network design is essential to ensure that the circuits on a chip operate reliably at the guaranteed level of performance. A poorly designed power network can become the cause for a variety of problems such as loss of circuit performance, noise generation, and electro-migration failures. With the increased power level and device densities of microprocessors in sub-micron technologies, these problems are more likely unless serious attention is given to power network design. Critical to obtaining a robust design is the ability to analyze the network efficiently several times in the design cycle. Several research works [73–76] published earlier discuss methodologies and techniques to accomplish this task efficiently.

The difficulty in power network analysis stems mainly from three sources: (i) the

network is very large, typically 1 million to 100 million nodes, (ii) the network is nonlinear as it contains switching devices, and (iii) the voltage and current distribution in the network is dependent on the instruction executed on the processor. Our work, presented in this chapter, addresses the first problem. The second problem is circumvented traditionally [73] by performing nonlinear simulation of individual circuit blocks without including the parasitics in the power interconnects, and then simulating the power interconnect as a whole using the time-variant current profiles, obtained in the nonlinear simulation as the excitation sources. The power grid needs to be simulated as a whole since it is tightly coupled, and as a result, currents drawn in one part of the chip affect the voltage distribution throughout the chip. However, the complexity of analysis is reduced substantially by not having to perform a nonlinear simulation of the entire interconnect and circuit devices. The third problem is one of obtaining a good coverage of all possible worst case power demand situations. Manually generated "hot loops," an extensive set of input vectors, and statically generated worst case current profiles [77–79] are some of the alternatives that address the worst case coverage issue.

The size based complexity of the problem has been partly addressed earlier [73,74] by using very efficient sparse linear system solution techniques. Cholesky factorization (direct method) [80] and Conjugate gradient techniques with pre-conditioners (iterative method) [80] have been used to solve the linear system associated with the power grid. These specialized techniques operate very efficiently by exploiting the special structure and properties of the underlying linear system. However, the earlier proposed solutions have applied these techniques to a flat (nonhierarchical) model of the power network. As a result, there is a serious limitation on the size of the problem they can solve, the limitation being imposed by the amount of memory available for computation. At the current technological level, it is seen that the available computing resources are insufficient to simulate very large power grids of

today's microprocessors using a flat model. The size of the power grid of a typical high performance microprocessor in 0.18 micron design, and using 6 layers of metal, is in the range of 10 million to 100 million nodes. Thus the power grid simulation would require solving a linear system of similar size at multiple time points. Clearly, the speed and memory capacity of a typical computing environment is insufficient to solve such a large system even with the most efficient linear system solution techniques.

In this work, we propose a hierarchical analysis technique to overcome the limitations of the traditional approach based on flat power grid model. Our approach comprises of the following steps: (1) Partitioning of the power grid into local and global grids, using the hierarchical structure in the design, (2) Generating macromodels for the local grids using efficient numerical methods, (3) Sparsifying the port admittance matrices of the macromodels, while maintaining the accuracy of the solution, (4) Simulating the global grid after augmenting it with the macromodels of the local grids, and finally, (5) Simulating the local grids where desired.

The basic strength of the proposed approach is derived from the well-known strategy of "divide and conquer," which is realized through partitioning. However, the efficiency and usefulness of the hierarchical approach is sensitive to several factors, such as the partitioning technique, the memory and runtime costs involved in generating the macromodels, etc. Our work in this research addresses these problems in order to realize a practical and efficient implementation of the hierarchical analysis strategy. We propose a partitioning strategy that reduces the memory required for storage in our hierarchical simulation approach. Moreover, a novel matrix sparsification technique based on 0-1 integer linear programming is proposed to further reduce the memory requirements. Also, an efficient numerical procedure for calculating the macromodels is given. The computation takes advantage of the fact that the underlying linear system is symmetric and positive definite. The proposed approach has been applied for analyzing the power grid of a number of high performance micropro-

cessors and DSP chips, obtaining significant memory and runtime advantages over the flat model analysis approach. To our knowledge, no work has been reported so far to address this critical issue of limitation on the size of power grids analyzable using current approaches.

The remainder of the chapter is organized as follows. In section 2, we present the concept of macromodeling and the partitioning strategy. Also presented in that section are the computational techniques for generating the macromodels. In section 3, the matrix sparsification technique is explained. Section 4 reports the performance results of the proposed approach for a set of benchmark designs, followed by conclusions in section 5.

7.2 Macromodeling Approach

7.2.1 Overview of Power Grid Simulation

Before presenting the macromodeling approach, let us present an overview of power grid simulation in general. A chip's power distribution system is modeled as a linear RLC network with independent time-varying current sources modeling the switching currents of the transistors. Simulating the network requires solving the following system of differential equations, which are formed in a typical approach such as the Modified Nodal Analysis (MNA) [81] approach:

$$\mathbf{G} \cdot \mathbf{x}(t) + \mathbf{C} \cdot \mathbf{x}'(t) = \mathbf{b}(t), \tag{7.1}$$

where $\mathbf{G}$ is a conductance matrix, $\mathbf{C}$ is the admittance matrix resulting from capacitive and inductive elements, $\mathbf{x}(\mathrm{t})$ is the time-varying vector of voltages at the nodes, and currents through inductors and voltage sources, and $\mathbf{b}(\mathrm{t})$ is the vector of independent time-varying current sources. This differential system is very efficiently solved by reducing it to a linear algebraic system

$$(\mathbf{G} + \mathbf{C}/h) \cdot \mathbf{x}(t) = \mathbf{b}(t) + \mathbf{C}/h \cdot \mathbf{x}(t-h), \tag{7.2}$$

using Backward Euler (BE) technique with a fixed time step, h. The BE reduction with fixed time stepping is advantageous for transient simulation since the left hand side (LHS) matrix $(\mathbf{G} + \mathbf{C}/h)$, referred to as the coefficient matrix, is rendered stationary, allowing either pre-processing or factoring of the matrix for a one-time cost and reusing it efficiently to solve the system at successive time points.

When $\mathbf{x}$ consists only of node voltages, as in the case of a modified nodal formulation of a network with R's, C's, and current sources only, the coefficient matrix can be shown to be symmetric and positive definite. A symmetric positive definite formulation is feasible even when inductive elements are included in the analysis, although this would involve an additional reduction step from the modified nodal formulation. The symmetric positive definiteness of the coefficient matrix, which is also very sparse, is especially attractive as the system can now be solved very efficiently using specialized linear system solution techniques, such as Cholesky factorization (direct method) and Conjugate Gradient (iterative method) techniques. The direct method through Cholesky factors is very cost-effective for simulations at multiple time points, as the expensive step of factoring is performed only once initially and its cost is amortized over multiple time point solutions. Successive solutions would involve only inexpensive forward and backward substitution procedures. Although the macromodeling techniques presented in this chapter are suitable for use with either type of solution approach, direct or indirect, we will assume, for simplicity of presentation, that the underlying linear solver is direct.

7.2.2 Basic Idea

The run time and memory requirement for solving a linear system is determined primarily by the size, sparsity, and structure of the coefficient matrix. If the network is very large (10^7 - 10^8 nodes), the available physical and virtual memory of the system is insufficient even for loading in the data associated with the network. Even when the base memory requirement is met, memory demand quickly grows during the matrix factorization process, due to new fills being created. Given a reordering scheme, the number of fills created is determined by the initial sparsity and structure of the matrix. The sparsity is given by the ratio of the number of elements in the network to the number of nodes. While tree-like network structures have low fills, mesh structures generally tend to have large fills during factorization. The amount of matrix computation being very sensitive to the sparsity and fill pattern, it is very desirable to have the initial matrix as sparse as possible. The objective of the proposed approach is, hence, twofold - (i) to reduce the size of the problem, and (ii) to maintain a high degree of sparsity in the reduced problem.

The first objective is met by partitioning the given network into subnetworks of manageable size, and solving the network by solving the sub-pieces individually. Since the entire network is tightly connected, we cannot ignore the interaction between the various partitions without incurring significant error. So, in order to account for the interactions between partitions, while at the same time not enlarging the size of the problem at hand, we use models for the partitions that capture their behavior as observed at their interface nodes (also called ports). We refer to these models as macromodels. A macromodel is a multi-port linear circuit element that has the same linear relation between the voltages at and currents through its ports as the partition itself. With macromodels for partitions available, the original (unpartitioned) network is efficiently solved after replacing the partitions by the respective macromodels, as the macromodels are much smaller in size than the partitions themselves.

The gains made through partitioning can be quickly lost if the partitions generate very dense macromodels, and thus increase the effective size of the problem. Our approach addresses this issue in two ways. First, the partitioning is performed strategically as explained in section 7.2.4. Then, an optional step of sparsification can be applied to the generated models. The key issue in sparsification is not to compromise accuracy of the final solution. The sparsification technique is covered in section 7.3.

Besides the memory advantage, the macromodeling approach has significant speed up as the creation of macromodels for the partitions can be performed in parallel.

7.2.3 Hierarchical Modeling

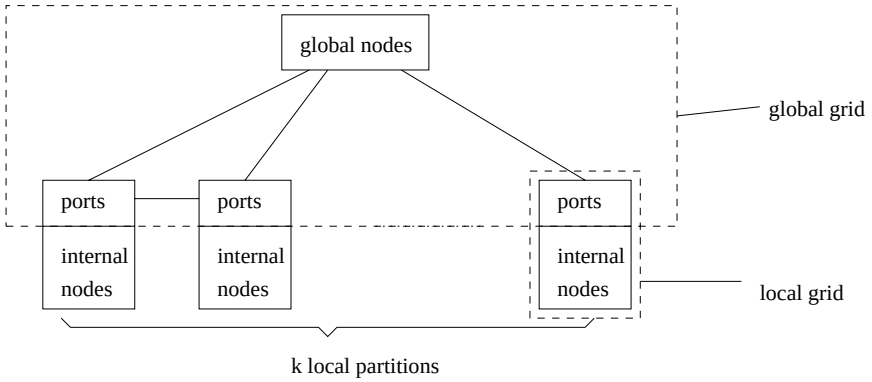

Figure 7.1: Hierarchical power network analysis

The macromodel approach to power grid analysis is illustrated in Figure 7.1. Let us consider a division of the entire power network into one global partition and k local partitions. A node in a local partition having links only to other nodes in the same partition is called an *internal node*, a node in the global partition is called a *global node*, and a node in a local partition that is connected to some node outside the local partition (i.e., in the global partition or in another local partition) is called a *port*. The *global grid* is then defined to include the set of nodes that lie in the global

partition and the port nodes, while the grid in a local partition constitutes a *local grid*.

Now each of the k local grids can be modeled as a multi-port linear element with transfer characteristics given by

$$\mathbf{I} = A \cdot \mathbf{V} + \mathbf{S}, \quad \mathbf{I} \in \mathbf{R}^m, A \in \mathbf{R}^{m \times m}, \mathbf{V} \in \mathbf{R}^m, \mathbf{S} \in \mathbf{R}^m \tag{7.3}$$

where m is number of ports in the local grid, A is the port admittance matrix, $\mathbf{V}$ is the vector of voltages at the ports, $\mathbf{I}$ is the current through the interface between the local and global grids, and $\mathbf{S}$ is a vector of current sources connected between each port and the reference node. $\mathbf{S}$ essentially has the effect of moving all the current sources internal to a local grid to the ports of the multi-port model. We refer to the set (A,$\mathbf{S}$) as the macromodel of the respective local grid.

The macromodel (A,$\mathbf{S}$) in equation (7.3) is obtained through a reduction procedure starting from the modified nodal equations of the local grid expressed in the form:

$$G \cdot \mathbf{U} = \mathbf{J}, \quad G \in \mathbf{R}^{n \times n}, \mathbf{U} \in \mathbf{R}^n, \mathbf{J} \in \mathbf{R}^n \tag{7.4}$$

where n is number of nodes in the local grid, G is the coefficient matrix, $\mathbf{U}$ is the voltage vector of the nodes of the local grid, and $\mathbf{J}$ is vector of currents that flow out of each node in the local grid. For the port nodes, $\mathbf{J}$ would also include the currents through the interface between the local and global grids. The procedure of deriving the transfer characteristic in Equation (7.3) from the modified nodal equation of (7.4) is referred to as macromodeling, and will be addressed in detail in section 7.2.5.

Once the macromodels for all the local grids are generated, the entire network can be abstracted simply as the global grid, with the macromodel elements connected to it at the port nodes. This is achieved by combining the coefficient matrix and the

RHS current vector of the global grid with the macromodels, (A,$\mathbf{S}$); Equations (7.3) of each local grid may be stamped into the modified nodal equations of the global grid as follows.

$$\begin{bmatrix} G_{00} & G_{01} & G_{02} & \dots & G_{0k} \\ G_{01}^T & A_1 & G_{12} & \dots & G_{1k} \\ G_{02}^T & G_{12}^T & A_2 & \dots & G_{2k} \\ \vdots & & & & \vdots \\ G_{0k}^T & G_{1k}^T & G_{2k}^T & \dots & A_k \end{bmatrix} \begin{bmatrix} \mathbf{V_0} \\ \mathbf{V_1} \\ \mathbf{V_2} \\ \vdots \\ \mathbf{V_k} \end{bmatrix} = \begin{bmatrix} \mathbf{I_0} \\ \mathbf{-S_1} \\ \mathbf{-S_2} \\ \vdots \\ \mathbf{-S_k} \end{bmatrix} \tag{7.5}$$

In the above equation,

- global nodes are labeled as partition 0
- A_i is the port admittance matrix of partition i, where $i \in [1, k]$.
- G_{ij} represents the conductance links between partition i and partition j.
- $\mathbf{I_0}$ is the vector of currents that flow out of the global nodes.
- $\mathbf{S_i}$ is the constant vector of partition i.
- $\mathbf{V_i}$ is voltage vector of partition i.

This is a system of $(n_0 + m_1 + m_2 + \ldots + m_k)$ linear equations, where n_0 is the number of global nodes and m_i is the number of ports in each partition.

From the above reduction scheme, the voltages and currents in the entire power grid can be solved in the following steps:

- Obtain global grid voltages by solving equation (7.5).
- For each partition, obtain $\mathbf{I}$ from equation (7.3) using the port voltages

- Solve equation (7.4) for each partition using $\mathbf{I}$ on the right hand side, to obtain voltages at the internal nodes of partitions.

The flow of the macromodel approach is illustrated in Figure 7.2.

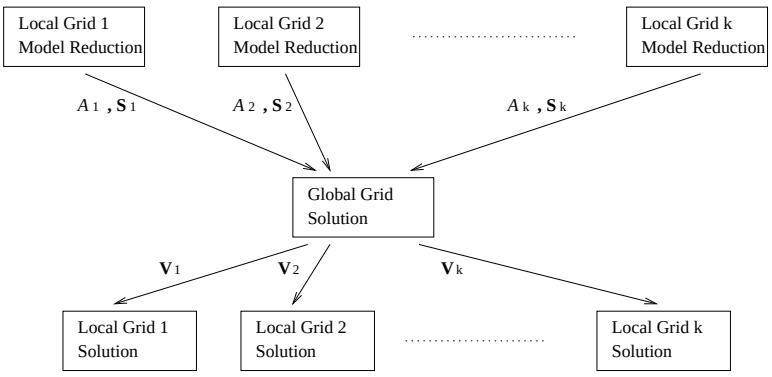

Figure 7.2: Flow of the hierarchical analysis

7.2.4 Partitioning Strategy

The main difficulty in macromodeling is that the model is often fully dense even though the partition from which it is created itself may be very sparse. Note that the entries of matrix A in equation (7.8) are admittance of paths between pairs of ports. Thus, a nonzero entry at position (i, j) results if there is a conducting path in the partition between these ports, even though there may not be a direct link between these ports. As a result, the number of nonzero entries in A is $\mathcal{O}(m^2)$, where m is the number of ports, unless the grid inside the partition is heavily fragmented. Nevertheless, there is a substantial win if m^2 is much smaller than the number of nodes in the partition that are abstracted away by this model. Thus, the key idea in the partitioning strategy is to identify a subnetwork and a interface boundary such that the number of internal nodes is much larger than the square of the number of nodes at the interface.

Fortunately, the natural hierarchical boundaries of circuit blocks often meet the above criteria. For instance, a large memory array with 3 local metal layers may have several millions of internal nodes, but it may have very few (hundreds of) nodes interfacing with the upper layer of the global grid, and almost none with other circuit blocks. Although one can have a sophisticated partitioning strategy, we have found in practice that a simple inspection procedure of checking every circuit block or a group of adjacently placed blocks for the above criteria works very well.

7.2.5 Macromodeling

Macromodeling is the procedure of deriving Equation (7.3) from the modified nodal equations of the partition. The modified nodal equations for a partition can be written as

$$\begin{bmatrix} G_{11} & G_{12} \\ G_{12}^T & G_{22} \end{bmatrix} \begin{bmatrix} \mathbf{U_1} \\ \mathbf{V} \end{bmatrix} = \begin{bmatrix} \mathbf{J_1} \\ \mathbf{J_2} + \mathbf{I} \end{bmatrix} \tag{7.6}$$

where

- $\mathbf{U_1}$ and $\mathbf{V}$ are vectors of voltages at the internal nodes and ports respectively
- $\mathbf{J_1}$ and $\mathbf{J_2}$ are vectors of current sources connected at the internal nodes and ports respectively
- $\mathbf{I}$ is the vector of currents through the interface
- G_{12} is the admittance of links between the internal nodes and the ports
- G_{11} is the admittance matrix of internal nodes
- G_{22} is the admittance matrix of ports.

From (7.6), we may rewrite the first set of equations as

$$\mathbf{U_1} = G_{11}^{-1}(\mathbf{J_1} - G_{12}\mathbf{V}) \tag{7.7}$$

Substituting this value of $\mathbf{U_1}$ into the second equation of (7.6), we get

$$\mathbf{I} = (G_{22} - G_{12}^T G_{11}^{-1} G_{12})\mathbf{V} + (G_{12}^T G_{11}^{-1}\mathbf{J_1} - \mathbf{J_2}) \tag{7.8}$$

Here, $G_{12}^T G_{11}^{-1}\mathbf{J_1} - \mathbf{J_2}$ is the constant vector $\mathbf{S}$ in Equation (7.3) and $G_{22} - G_{12}^T G_{11}^{-1} G_{12}$ is the port admittance matrix A in Equation (7.3).

It may be noted that the pre-multiplication and post-multiplication operations with G_{11}^{-1} can be carried out without explicitly inverting G_{11}, but through multiple invocation of the direct or iterative solver.

The above calculation can be made very efficient using the fact that the coefficient matrix, G is symmetric and positive definite. We show below how A and $\mathbf{S}$ can be computed efficiently from the submatrices of the Cholesky factors, rather than the Cholesky factors themselves.

Relating G_{11}, G_{12}, and G_{22} to the submatrices of the Cholesky factors of G, we have

$$\begin{aligned}
\begin{bmatrix} G_{11} & G_{12} \\ G_{12}^T & G_{22} \end{bmatrix} &= \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix} \\
&= \begin{bmatrix} L_{11}L_{11}^T & L_{11}L_{21}^T \\ L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T \end{bmatrix}
\end{aligned}$$

Now computing A in terms of submatrices of factors, we get

$$\begin{aligned}
A &= G_{22} - G_{12}^T G_{11}^{-1} G_{12} \\
&= L_{21}L_{21}^T + L_{22}L_{22}^T - L_{21}L_{11}^T (L_{11}L_{11}^T)^{-1} L_{11}L_{21}^T \\
&= L_{22}L_{22}^T
\end{aligned}$$

Similarly, vector $\mathbf{S}$ is given by

$$\begin{aligned} \mathbf{S} &= G_{12}^T G_{11}^{-1} \mathbf{J_1} - \mathbf{J_2} \\ &= L_{21} L_{11}^T (L_{11}^T)^{-1} L_{11}^{-1} \mathbf{J_1} - \mathbf{J_2} \\ &= L_{21} L_{11}^{-1} \mathbf{J_1} - \mathbf{J_2} \end{aligned} \tag{7.9}$$

The above simplified technique reduces computation dramatically over the direct computation based on equation (7.6), as L_{22} and L_{11} used in equation (7.9) are already triangular.

7.2.6 Analysis of the Computation Cost

In this section we present the computational advantage of macromodeling over the flat model analysis approach.

Suppose the cost of factorizing a matrix is $C_1(l)$, and the cost of one forward and one backward substitution is $C_2(l)$, where l is the size of the matrix, and $C_2(l) << C_1(l)$. Let N be the number of nodes in the entire power network.

If no macromodels are used for the power analysis, the computation cost of the first run is $C_1(N)$ and the computation cost of a subsequent run is $C_2(N)$. In the macromodeling approach, the computation cost of the first run can be expressed as

$$C_1(n_1)+C_1(n_2)+\ldots+C_1(n_k)+C_1(n_0+m_0+m_1+\ldots+m_k)+C_2(n_1)+C_2(n_2)+\ldots+C_2(n_k) \tag{7.10}$$

Here, $n_i, i \in [1,k]$ is number of nodes in each partition and $n_0+n_1+n_2+\ldots+n_k = N$. The computation cost from macromodeling is given by $C_1(n_1)+C_1(n_2)+\ldots+C_1(n_k)$ by using the simplified macromodeling method described in section 7.2.5. The cost of finding the solution to the global network is $C_1(n_0+m_0+m_1+\ldots+m_k)$ and the cost

of solving the local grids is $C_2(n_1) + C_2(n_2) + \ldots + C_2(n_k)$, since the factors obtained from macromodeling can be used for solving the local grid.

The computation cost of the subsequent run can therefore be approximated as

$$C_2(n_0 + m_0 + m_1 + \ldots + m_k) + 2C_2(n_1) + 2C_2(n_2) + \ldots + 2C_2(n_k) \qquad (7.11)$$

where the computation cost of macromodeling is $C_2(n_1) + C_2(n_2) + \ldots + C_2(n_k)$ since the A_i's are unchanged and only the $\mathbf{S_i}$'s must be recalculated during the subsequent run in macromodeling.

Expressions (7.10) and (7.11) provide a rough estimate of computation costs based on the size of the network and its partitions. In reality, the density of a matrix is an important factor that influences the solution speed. Generally, the conductance matrices for partitions are denser than the conductance matrix of the entire network, and thus the conductance matrix in equation (7.8) used for the global solution is a dense matrix.

Typically, Cholesky factorization requires $n^3/6$ multiplications and substitution requires $n^2/2$ multiplications. However the sparsity of the conductance matrix, combined with efficient reordering, enables the observed computation cost to be near-linear with the dimension of the matrix. However, with an increase in the size of the conductance matrix, the computation cost will approach the $n^3/6$ or $n^2/2$ curve gradually. In such cases, the computation cost for the macromodel approach will be lower than that for the flat analysis even with the overheads associated with partitioning.

Most important of all, the *divide and conquer* procedure applied to the power network makes parallel execution of power network simulation possible. During parallel execution, the execution time of the first run is given by

$$\max(C_1(n_1), C_1(n_2), \ldots, C_1(n_k)) + C_1(n_0 + m_0 + m_1 + \ldots + m_k) + \max(C_2(n_1), C_2(n_2), \ldots, C_2(n_k))$$

where $\max(C_1(n_1), C_1(n_2), \ldots, C_1(n_k))$ represents the maximum execution time among macromodeling of partitions, $C_1(n_0 + m_0 + m_1 + \ldots + m_k)$ is the global solution time and $\max(C_2(n_1), C_2(n_2), \ldots, C_2(n_k))$ represents the maximal execution time out of partition solutions. Similarly, the execution time of the subsequent run is given by

$$C_2(n_0 + m_0 + m_1 + \ldots + m_k) + 2 \times \max(C_2(n_1), C_2(n_2), \ldots, C_2(n_k))$$

Moreover, the memory requirement with macromodels is the maximum memory required for solving any partition, rather than the sum of memory requirement of each partition.

Besides run time and memory advantage, macromodeling provides a certain flexibility to a design/analysis situation so that significant analysis effort can be saved. Given below are few examples of design/analysis situations when such flexibility is useful.

Example-1: When a designer is interested in the detailed analysis only of a specific circuit block, then significant design time is saved by not simulating the other partitions, but while accounting accurately the effect of switching of these other blocks on the block s/he is interested in.

Example-2: A designer knows *a priori* in which circuit block or blocks the worst drop is to be expected, and the objective of the analysis is to only to find the worst IR drop estimate for the design. Then, it will be necessary to simulate only few blocks (partitions) in the last step of the macromodel approach.

Example-3: The process of fixing problems in a power grid is usually an iterative one. The process consists of detecting an error, making local changes to the grid to correct the problem, and re-running the analysis. In this case, only the macromodeling of the partition whose grid was changed needs to be recalculated. The speed-up in analysis due to this makes it possible for the designer to fix the problems interactively with the analysis tool.

7.3 Sparsification of Macromodels

In section 7.2.4, we pointed out that the number of entries in the macromodel has $\mathcal{O}(m^2)$ complexity for model size m. Although the macromodels reduce the size of the system to the smaller system described in equation (7.5), the sparsity of the coefficient matrix of equation (7.5) decreases considerably due to the density of the A_i submatrices. For an iterative solver, this is undesirable as the number of floating point operations (FLOPs) to solve the system increases. For a direct solver, this affects both the FLOPs, as well as the memory required to factorize. The additional memory demand is caused by excessive fills created by the dense parts during factoring. So, to derive the most benefit out of the macromodeling approach, it is important that the coefficient matrix in equation (7.5) is kept sparse. While the partitioning strategy explained in section 7.2.4 is a natural way of achieving this, other sparsification techniques in conjunction with good partitioning schemes are very useful for making the macromodeling approach practical. In this section, we present a novel technique to sparsify the port admittance matrices of the macromodels.

Our sparsification method is motivated by the observation that although the matrix A is dense, it consists of a large number of values that are numerically small and will have little influence on the results if approximated to zero. We provide an algorithm to sparsify the coefficient matrix A by dropping some of its entries, while keeping the error introduced by this process below a specified value. The proposed sparsification technique also preserves the symmetry and the positive definite property of the matrix. Note that the sparsification procedure needs to be performed only once (during the first run).

7.3.1 Problem Definition

The problem is stated as follows: Given the transfer characteristic equation of each partition

$$\begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ \vdots \\ i_m \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,m} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,m} \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots & a_{3,m} \\ \vdots & & & & \\ a_{m,1} & a_{m,2} & a_{m,3} & \dots & a_{m,m} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_m \end{bmatrix} + \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_m \end{bmatrix}, \tag{7.12}$$

the nominal value of v_j, $j \in [1, m]$: B, $B > 0$, and

the error in i_j, $j \in [1, m]$: e_j

transform equation (7.12) into

$$\begin{bmatrix} i'_1 \\ i'_2 \\ i'_3 \\ \vdots \\ i'_m \end{bmatrix} = \begin{bmatrix} a'_{1,1} & a'_{1,2} & a'_{1,3} & \dots & a'_{1,m} \\ a'_{2,1} & a'_{2,2} & a'_{2,3} & \dots & a'_{2,m} \\ a'_{3,1} & a'_{3,2} & a'_{3,3} & \dots & a'_{3,m} \\ \vdots & & & & \\ a'_{m,1} & a'_{m,2} & a'_{m,3} & \dots & a'_{m,m} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_m \end{bmatrix} + \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_m \end{bmatrix} \tag{7.13}$$

to maximize

the number of $a'_{j,k}, j \neq k,$ such that $a'_{j,k} = 0,$

subject to

$$| i_j\prime - i_j | \leq e_j, \ j \in [1, m]$$

$a'_{j,k} = a'_{k,j}$ (maintaining matrix symmetry).

7.3.2 Problem Formulation

This problem can be formulated into a 0-1 multidimensional knapsack problem [58, 82]. In this section, we describe the transformation from the above problem to the knapsack problem.

The task here involves zeroing out off-diagonal elements of the matrix A. It is easy to show that these sparsification operations maintain the positive definite property of the matrix. To see this, we note first that the partition can be thought of as being purely resistive (for example, any capacitors are linearized). Given this "resistive" network, one may build an equivalent network of a set of equivalent resistances R_{jk} between each pair of ports j and k. The matrix A is then simply the conductance matrix for this network of R_{jk}'s, and is therefore diagonally dominant. This leads to two conclusions: (1) all off-diagonal elements must be non-positive, and (2) zeroing out off-diagonal elements of A maintains the diagonal dominance of the matrix, and therefore its positive definite property.

The problem formulation is described as follows. First consider the maximum error that an element of matrix $a_{j,k}$ can cause if it is rounded off to 0. Since B is positive and $a_{j,k} \leq 0, j \neq k$, the maximum negative error caused by rounding off $a_{j,k}$, $en_{j,k}$, is given by

$$en_{j,k} = a_{j,k} * B, j \neq k$$

Let $X_{j,k}$ represent a Boolean value, 1 when element $a_{j,k}$ is rounded to zero, and 0 otherwise. The matrix sparsification problem can be formulated as 0-1 knapsack problem as follows.

$$\begin{aligned} \text{Maximize} \quad & z(x) = \textstyle\sum_{k=1}^{m} \sum_{j=1}^{k} X_{j,k} \\ \text{subject to} \quad & -\textstyle\sum_{k=1}^{j-1} en_{j,k} * X_{j,i} - \sum_{k=j+1}^{m} en_{j,k} * X_{j,k} \leq e_j, j \in [1,m] \\ & X_{j,k} \in \{0,1\} \text{ for all } X_{j,k},\ j < k \end{aligned} \tag{7.14}$$

In (7.14), the indices of the variables $X_{j,k}$ are required to satisfy the relation, $j < k$, so that $X_{j,k} = 1$ indicates the rounding-off of both $a_{j,k}$ and $a_{k,j}$ to maintain the symmetricity of A. Therefore, the resulting sparsified matrix is symmetric and positive definite.

The 0-1 knapsack problem can be solved optimally either by dynamic programming or using an ILP solver. In our implementation, we use the latter, but with some modifications for speed considerations. First, we relax the integer requirement and solve the fractional knapsack problem using a linear programming solver [66]. Next, the fractional x_{jk}'s are sorted, and greedily applied successively until the maximum error in $|i_j|$ reaches the specified limit, e_j.

7.4 Experimental Results

The hierarchical analysis method using macromodels was implemented using C and embedded in an existing in-house power analysis tool [73]. An efficient direct linear solver based on Cholesky factors was used in all the experiments. The extracted power grids of four high performance general purpose/DSP microprocessor chips were used to benchmark the performance of macromodeling (Tables 7.1 and 7.2) and sparsification (Table 7.3) techniques. Chips 1 and 2 are DSP and communication chips whose power grids are implemented in 3 layers of metal. Chips 3 and 4 are high performance microprocessor chips using 5 and 6 metal layers respectively. The analyses were run on dedicated Sun workstations with 2 GBytes of physical memory and 2 GBytes of swap space. The run time measures used for comparison are based on the actual time required to complete the task.

7.4.1 Performance of Macromodeling Technique

Table 7.1: Run-time and memory comparison for the first simulation

Chip	#nodes (millions)	Without macromodel		With macromodel				
		Run-time (minutes)	Peak Memory (GB)	# part	#nodes (max) (millions)	Total Run-time Serial (min)	Total Run-time Parallel (min)	Peak Memory (GB)
Chip-1	3.9	93	1.5	12	0.40	43	7	0.2
Chip-2	2.7	57	1.2	9	0.58	25	6	0.3
Chip-3	7.5	629	2.6	11	0.79	136	26	0.4

Table 7.1 compares the performance of the proposed hierarchical approach using macromodels with that of the nonhierarchical approach. Two metrics are compared - the peak memory demand and the total run-time. The number of nodes, in millions, for the entire power network is given in column 2, and the number of nodes in the largest partition (or the global grid if it is larger than other partitions) is given in column 5. Column 3 shows the total time, in minutes, taken for completing the analysis on the flat model, while columns 7 and 8 show the total time required by the hierarchical approach. The run-time in column 7 corresponds to the cases when the macromodels for the various partitions were generated serially on a single computer, whereas column 8 is for the cases when these computations are performed in parallel. The run-time reported in this table is the time taken for analyzing the power network at the first time point in a sequence of simulations. Columns 4 and 9 show the peak memory demand, in Gigabytes, during the analysis without macromodels and with macromodels, respectively.

It is evident from the above table that the problem size tackled with the proposed approach is substantially reduced from the original problem. This is the primary goal of the proposed approach so that a chip-level analysis of very large designs is made possible. Based on the benchmarks, it can be seen that the size of the linear system that needs to solved with the new approach is about 10X smaller than the traditional

approach.

The effect of problem size reduction is clearly reflected in the peak memory requirements of the different approaches shown in the table. Again, a 10X to 20X reduction in memory requirement is seen possible with the hierarchical approach. This implies that with the available computing resources (memory and speed), the new method enables analyzing much larger designs that will become common in the near future.

From the results, we can see that without macromodels the run time can be several hours (e.g. 10.5 hours for Chip 3) for a supply network with millions of nodes. As a result of reducing the size complexity, the run-time is reduced by a factor of 2X to 5X even when the macromodels are computed one after another on a single computer. The run-time is dramatically reduced by 10X to 23X, if the parallelism created by the macromodel is utilized. Observing that the designs for the various circuit blocks become available at different time points in the design cycle, all the macromodels need not be created all at once. That is, the effort of creating macromodels, which is a significant part responsible for the total run-time, can be distributed through out the design cycle, saving time during the final analysis stage.

It should be noted that the performance of hierarchical approach reported in Table 7.1 does not consider the additional performance gain resulting from the proposed sparsification technique.

Table 7.2 compares the performance of the two approaches based on the time required to perform simulations at 1000 successive time points, after the first one. Thus, the run-times here are independent of the time taken to generate the macromodels. Column 2 shows the run time without macromodels. For the hierarchical approach, run-times for both serial (column 3) and parallel (column 4) execution are shown. Since the memory requirement of these runs is less than that of the first run, these figures are omitted in Table 7.2.

Table 7.2: Comparison of run time for 1000 subsequent simulations

Chip	Without macromodel	With macromodel	
	Run-time(hours)	Total run-time(hours)	Parallel run-time(hours)
Chip-1	8.4	28.0	4.0
Chip-2	8.0	22.5	4.4
Chip-3	33.7	43.5	6.6

The hierarchical approach executed in serial mode recorded unfavorable run-times for the benchmarks. However, the disparity in run-times between the nonhierarchical and hierarchical approach (in serial mode) diminishes as the size of the original network becomes larger, as evidenced from the results for Chip-3, which has 7.5 million nodes. This behavior is not unexpected, and can be explained by the fact that the overhead associated with computing the $\mathbf{S}$ vector for each partition at every time step, and back-solving each partition again in the final step of the solution, is a dominant factor. This behavior is exhibited for networks up-to a certain size, where the original matrix and the reduced matrix do not differ greatly in terms of the time required for a back-solve. However, as the network becomes larger, the difference in problem sizes with and without macromodels are significantly different, and the overhead cost of handling the partitions becomes negligible in the overall cost. As a result, the hierarchical approach becomes favorable for very large networks even in the serial execution mode.

The run-time advantage of parallel execution mode is very clear from Table 7.2. Results show that the parallel execution utilizing hierarchy is $1.8 - 5.1$ times faster than the nonhierarchical approach. As designers would like to simulate the power grid with long traces of current signatures in order to obtain good coverage of the IR-drop situations, efficiency of simulation in this phase is crucial. The parallel execution mode, as well as the flexibility in the hierarchical analysis discussed in section 7.2.6,

make the hierarchical analysis approach extremely attractive.

7.4.2 Performance of the Sparsification Technique

The sparsification procedure described in section 7.3 reduces the number of nonzero elements while maintaining an acceptable level of accuracy. In our implementation, the specified error e_j is defined as $e_j = \max(const, \mid s_j \times x\% \mid)$, where $const$ is a small positive constant, $s_j, 1 \leq j \leq m$, is as defined in equation (7.12), and $x\%$ is the user-defined error limit, which is typically $0\% - 10\%$. The sparsification technique was implemented using a linear programming solver *lp_solve_2.3* [66].

Table 7.3 reports the sparsity and run-time improvements achieved for two benchmark examples, analyzed at different levels of accuracy. The second column in the table shows the voltage value of the clean power supply and the value of the maximum voltage drop observed in the circuit. Columns 3 and 4 report the number of nodes and the total number of ports respectively in the global grid. The number of nonzero elements in the coefficient matrix of equation (7.5) are shown in Column 5. Column 6 shows the maximum voltage error caused by the sparsification procedure. The ratio of maximum observed error in voltage to the maximum voltage drop is shown in column 7. Finally, column 8 reports the time required to solving equation (7.5).

For each benchmark, the proposed sparsification technique was tested at four levels of accuracy. The benchmark Chip-4 is a 6-layer, mesh type, power grid. Its power grid is much denser than the other examples, and this example also has some partitions with large number of ports. As a result, the coefficient matrix obtained for this example could not be solved with the available computing resources without sparsification.

The results clearly show that the sparsity of the coefficient matrix is improved by as much as 11X, incurring only 2.6% error in the final results. The improved sparsity

improved the run-time for the dense example, Chip-4 significantly, besides greatly reducing the memory requirement.

Table 7.3: The effect of sparsification

Chip	Clean/ Max drop(v)	#nodes	#ports	#nonzeroes	Max-err (v)	Error %	Run-time (secs)
Chip-3	2.0/0.12	23261	379	106909	0.0	0.0%	38.3
				98505	0.000043	0.04%	36.0
				98253	0.00058	0.5%	32.4
				98005	0.0013	1%	30.6
Chip-4	1.8/0.02	2932	2849	2067572	0.0	0.0%	-
				366612	0.000045	0.2%	36.5
				249588	0.00021	1.0%	24.4
				197868	0.00051	2.6%	20.1

7.5 Conclusion

In this chapter, we have presented a hierarchical power network analysis method using novel macromodeling and matrix sparsification techniques. The proposed techniques were shown to gain significant memory and run-time advantages over the traditional approach of analyzing the power network without using the hierarchy. The experimental results based on analyzing the entire power network of four high performance microprocessor designs confirmed these claims. Our work addressed several issues that arise in designing a practical and efficient hierarchical analysis methodology. These issues related to partitioning of the network and handling the density growth in the matrices of the underlying linear system. The hierarchical analysis approach shows excellent promise as a viable alternative to the traditional nonhierarchical analysis

method, capable of handling the increasing size of power grids in modern microprocessors.

It was shown that the method of partitioning has a significant influence on the performance of this approach. One of our future directions is to explore optimal partitioning techniques that can be applied with minimal user-intervention. It may be still important to maintain the natural hierarchical boundaries alongside the automatically-created boundaries in order to preserve the flexibilities of hierarchical analysis discussed earlier. Another research direction is to develop more efficient sparsification techniques.

Chapter 8

Conclusion

8.1 Summary

In this thesis, we have addressed several analysis and optimization problems in high speed circuits, related to domino logic and supply network.

In Chapter 2, we provided a brief introduction to domino logic, and suggested a synthesis and optimization flow for domino logic is suggested. Several parts of this flow were described in subsequent chapters.

In Chapter 3, we addressed the problem of technology mapping for domino logic, and presented an efficient algorithm for parameterized library mapping using a DAG covering mapping method and a dual-monotonic gate mapping method. In our mapper, specific consideration has been given to practical domino circuit design techniques, such as wide dynamic AND/OR gates and multiple-output gates. The results show up to 42.0% improvement in area over existing approaches. A comparison with a static implementation shows that the domino logic synthesized by this method shows better performance in terms of delay. Moreover, the area cost of our domino implementation is better than or close to the cost of a static implementation, even though

the non-inverting property of domino logic may require the duplication of numerous nodes of in the input network. In addition, the parameterized library mapping algorithm was extended to static technology mapping. The application of our algorithm provides an average of 3.8% improvement in the area and an average speedup of 418x in the CPU time over the static solution provided by SIS using its largest built-in library, $44-6.genlib$.

The output phase assignment has a significant impact on both the logic duplication cost and the implementation cost of a circuit. In Chapter 4, a 0-1 integer programming formulation was provided for the output phase assignment problem for domino logic. It considers the cost difference between two polarities and enables a standard linear programming package to be used to solve the problem. The results show up to 41.0% improvement in area.

An important problem for domino logic synthesis is to partition a circuit to determine which parts of the circuit should be implemented as static logic, and which parts as domino logic, in conformance with the specified clock scheme. In Chapter 5, we presented an efficient timing driven static-domino partitioning algorithm. Our algorithm finds a partition of a logic network between static and domino implementations that minimizes the cost, subject to timing constraints specified by the clocking scheme. This algorithm was extended to develop a method for partitioning domino logic into two phases, with inverters permitted between the two phases. The results show that the effect of partitioning is to reduce the logic unating penalty. Our partitioning algorithms were then applied to the most common two-phase clocking strategy, and we showed that the area of original domino network is reduced significantly, while maintaining the circuit speed of a pure domino implementation.

In Chapter 6, we addressed a timing verification and sizing optimization tool for circuits containing mixed domino and static logic. The timing constraints from both synchronizer and domino gates were considered during the timing analysis and opti-

mization procedure. After sizing, the circuit was evaluated for charge sharing using the a procedure that is more accurate and less pessimistic than previous proposed methods, and the circuits was corrected to eliminate any charge-sharing problems. These techniques were shown to work well on the standard benchmark circuits.

The last part of the thesis addressed the problem of analyzing large supply networks hierarchically since many of today's designs have power networks are too large to be analyzed in the traditional way as flat networks, both in terms of being compute-intensive and memory-intensive. In Chapter 7, we proposed an analysis technique to overcome these capacity limitations. We presented a new technique for analyzing a power grid using macromodels that were created for a set of grid partitions. Efficient numerical techniques for the computation and sparsification of the port admittance matrices of the macromodels were presented. A novel sparsification technique using a 0-1 integer linear programming formulation was proposed to achieve superior sparsification for a specified error. The run-time and memory efficiency of the proposed method was illustrated through analysis case studies of several multi-million node power grids, extracted from real microprocessor and DSP designs.

8.2 Directions for Further Research

8.2.1 Future Work for Domino Logic

Although a large amount of research has been performed on domino logic, many issues are still open problems that remain to be solved. We now present some of these issues, and our thoughts on them.

Effective Noise Estimation and Correction Noise susceptibility is the major drawback of domino logic, as well as the most difficult to be overcome. The two major

sources of noise, i.e., crosstalk and power/ground bounce [1], have great impact on the correct operation of domino logic. In both cases, accurate noise values are extremely difficult to predict and calculate in an early phase of the design process as they depend on the detailed layout and timing information. With the rapid increase of coupling capacitance with the technology scaling, the capacitive coupling crosstalk will become more and more significant. Therefore, the handling of noise problem is crucial for domino logic to maintain the predominant role in the next several generations of high performance designs. Although in [83], there are some discussions on noise in dynamic circuit and in [40], design methodologies for noise in digital circuits are described, further efforts must be made for noise estimation and correction problem of domino logic, such as exploring more robust dynamic circuits, more accurate noise estimation, as well as domino logic design methodology to reduce the impact of noise.

Low Power Domino Logic Design Power consumption has become one of the main concern of contemporary VLSI circuit designers. If the output signal of a domino gate remains a "1", the dynamic node and its driven inverter will continuously be precharged and evaluated even though there is no actual activity on the output signal, which causes a significant amount of power wastage. Thus, the power problem is another important factor that must be considered during domino logic design. The work in [37] proposes a method of reducing the power consumption of domino logic by optimal output phase assignment. The 0-1 ILP formulation proposed in Section 4 should be capable of being adapted for a power minimization objective. The partitioning problem addressed in Section 5 could also be used to reduce power. Moreover, other techniques such as gated clocks and asynchronous domino logic style circuits should be considered to provide an effective solution to this problem.

Clock Network Generation All the synthesis tools for domino circuits so far have concentrated on the logic part of the domino logic design. An indispensable part of domino logic is related to the clocking strategies and their implementations. As we can see from configurations of domino logic in Figures 2.1, 3.5, 3.8, 3.10, each domino gate requires the clock input signal of various skews. Typically, these clock signals are generated locally rather than from the global clocking signals [16, 17, 19], in order to provide more flexible clock signals and to avoid the difficulties associated with the magnitude of the physical clock distribution problem. These clocks are usually generated manually by the designers and it is difficult to match them with the delay of domino gates. Therefore, it is an important task to explore the methodology and develop automated synthesis tools for the clock network design and routing for domino logic. In the context of static CMOS circuits, the clock generation and routing problem has been widely addressed [84–88]. Both [16] and [19] mention some configurations for clock generation circuits without an extensive exploration of the problem. A systematic method aimed at the special properties associated with clocking for domino logic must be established in the future.

Domino Logic Cell Synthesis Domino logic consists mainly of NMOS transistors, as against static CMOS logic, which uses a balanced number of PMOS and NMOS transistors. Therefore, the layout of a domino cell may be greatly different from that of the static library cell. Making effective use of the features of domino logic to produce compact layouts for domino cells is non-trivial problem that has not been addressed in current research.

Delay Balanced Partitioning Partitioning of an input combinational network to fit into the clocking scheme is an essential problem of domino logic synthesis. Besides the objective we addressed in Section 5, delay balancing is an important objective for some clocking schemes. For example, in clocking scheme of Figure

2.4, the delay of each stage is the largest delay of all the gates of the same stage; in clocking scheme of Figure 2.3, if the entire circuits is partitioned into one stage and the other is empty, clocked buffers have to be used to replace the logic gates in the latter. Therefore, delay balanced partitioning is important for domino logic design with the specified clocking scheme.

Asynchronous Domino Techniques Self-timed pipelined domino circuits are a promising class of asynchronous domino design techniques addressed in [17]. In these circuits, the precharged signal comes from the completion information from neighboring stages, offering the potential advantages of reduced power consumption and high speed. However, the design of asynchronous domino circuits is quite difficult and may result in a large area overhead. Exploring the asynchronous domino logic synthesis technique is a key issue before the asynchronous domino techniques can be widely used.

8.2.2 Future Work for Supply Network Analysis

With the increase in the complexity of VLSI chips, designing and analyzing a power distribution network has become a challenging task. The hierarchical analysis approach show excellent promise as a viable alternative to the traditional flat analysis method, capable of handling the increasing size of power grids in model microprocessors. The method requires further refinement in the following aspects.

Automatic Partitioning Techniques The method of partitioning has a significant influence on the improvements provided by a hierarchical analysis of the power network. Section 7.2.4 addresses the partitioning strategy for our hierarchical simulation reduction. In the future work, an automatic partitioning tool that incorporates the connectivity information of the power network and satisfies the objective of our partitioning criterion must be developed.

Port Collapsing A well-designed power supply network should be a strongly connected network. This makes the task of partitioning network difficult since it is difficult to find clusters that interact through a small cut, and increases the computation cost of macromodel, as shown by the cost formula of (7.10). While sparsifying of the port admittance matrix is one way to prevent this, other methods will be explored to overcome the problems caused by the presence of a large number of ports. One such a method would judiciously collapses ports that have similar voltage levels into a single port to reduce the size of the cut between clusters, and hence the computational overhead with minimal sacrifices in the accuracy.

Hierarchical analysis for RLC networks At present, our hierarchical power network analysis method is used on a RC model of power network. We expect this *divide and conquer* algorithm to be applied to more complex RCL analysis of power networks in the future.

Bibliography

[1] Semiconductor Industry Association, "National technology roadmap for semiconductors," 1997.

[2] P. E. Gronowski, W. J. Bowhill, R. P. Preston, M. K. Gowan, and R. L. Allmon, "High-performance microprocessor design," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 676–686, May 1998.

[3] S. Naffziger, "A sub-nanosecond 0.5um 64b adder design," in *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 362–363, 1996.

[4] C. Heikes and G. Colon-Bonet, "A dual floating point coprocessor with an FMAC architecture," in *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 354–355, 1996.

[5] P. E. Gronowski, P. J. Bannon, M. S. Bertone, and *et al.*, "A 433MHz 64b quad-issue RISC microprocessor," in *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 1687–1696, 1996.

[6] M. Benes, A. Wolfe, and S. M. Nowick, "A high-speed asynchronous decompression circuit for embedded processors," in *Proceedings of the Conference on Advanced Research in VLSI*, pp. 219–236, 1997.

[7] F. Lu and H. Samueli, "A 200-mhz CMOS pipelined mutiplier-accumulator using a quasi-domino dynamic full-adder cell design," *IEEE Journal of Solid-State Circuits*, vol. 28, pp. 123–132, Feb. 1993.

[8] R. P. Colwell and R. L. Steck, "0.6um BiCMOS proessor with dynamic execution," in *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 176–177, 1995.

[9] M. Zhao and S. S. Sapatnekar, "Technology mapping for domino logic," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 248–251, 1998.

[10] M. Zhao and S. S. Sapatnekar, "Timing optimization of mixed domino and static logic," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1998.

[11] M. Zhao and S. S. Sapatnekar, "Timing-driven partitioning for two-phase domino and mixed static/domino implementations," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 107–110, 1999.

[12] M. Zhao and S. S. Sapatnekar, "Dual-monotonic domino gate mapping and optimal output phase assignment of domino logic," *submitted for publication*, 1999.

[13] M. Zhao, R. Panda, S. S. Sapatnekar, T. Edwards, R. Chaudhry, and D. Blaauw, "Hierarchical analysis of power distribution networks," *submitted for publication*, 1999.

[14] M. Zhao and S. S. Sapatnekar, "Technology mapping algorithms of domino logic," *submitted for publication*, 1999.

[15] M. Zhao and S. S. Sapatnekar, "Timing-driven partitioning and timing optimization of mixed static-domino implementations," *submitted for publication*, 1999.

[16] D. Harris and M. A. Horowitz, "Skew-tolerant domino circuits," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 1702–1711, Nov. 1997.

[17] T. Williams, "Dynamic logic: Clocked and asynchronous." Tutorial notes at the International Solid State Circuits Conference, 1996.

[18] R. Puri, "Design issues in mixed static-domino circuit implementations," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 270–275, 1998.

[19] G. Yee and C. Sechen, "Dynamic logic synthesis," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 345–348, 1997.

[20] W. H. Lien and W. P. Burleson, "Wave-domino logic: Theory and applications," *IEEE Transactions on Circuits and Systems*, vol. 42, pp. 78–90, Feb. 1995.

[21] J. Wang, Z. D. Wang, G. A. Jullien, and W. C. Miller, "Area-time analysis of carry lookahead adders using enhanced multiple output domino logic," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 59–62, 1994.

[22] Z. Wang, G. A. Jullien, W. C. Miller, J. Wang, and S. S. Bizzan, "Fast adders using enhanced multiple-output domino logic," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 206–213, Feb. 1997.

[23] D. H. K. Hoe and C. A. T. Salama, "Dynamic GaAs capacitively coupled domino logic (CCDL)," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 844–849, June 1991.

[24] S. M. Menon, A. P. Jayasumana, and Y. K. Malaiya, "A novel high-speed BiCMOS domino logic family," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 21–24, 1995.

[25] J. R. Yuan, C. Svensson, and P. Larsson, "New domino logic precharged by clock and data," *Electronics Letters*, vol. 29, pp. 2188–2189, Dec. 1993.

[26] J. Kernhof, M. Selzer, M. A. Beunder, B. Hoefflinger, B. Laquai, and I. Schindler, "Mixed static and domino logic on the CMOS gate forest," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 396–402, Apr. 1990.

[27] D. V. Campenhout, T. Mudge, and K.Sakallah, "Modeling domino logic for static timing analysis," Tech. Rep. CSE-TR-295-96, The University of Michigan, 1996.

[28] D. V. Campenhout, T. Mudge, and K. A. Sakallah, "Timing verification of sequential domino circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 127–132, 1996.

[29] K. Venkat, L. Chen, I. Lin, P. Mistry, and P. Madhani, "Timing verification of dynamic circuits," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 452–455, Mar. 1996.

[30] L. T. Wurtz, "An efficient scaling procedure for domino CMOS logic," *IEEE Journal of Solid-State Circuits*, vol. 28, pp. 979–982, Sept. 1993.

[31] H. Y. Chen and S. M. Kang, "A new circuit optimization technique for high performance CMOS circuits," *IEEE Transactions on Computer-Aided Design*, vol. 10, pp. 670–676, May 1991.

[32] A. Dharchoudhury, D. Blaauw, J. Norton, S. Pullela, and J. Dunning, "Transistor-level sizing and timing verification of domino circuits in the PowerPC microprocessor," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 143–148, 1997.

[33] M. R. Prasad, D. Kirkpatrick, and R. K. Brayton, "Domino logic synthesis and technology mapping," in *Workshop Notes, International Workshop on Logic Synthesis*, 1997.

[34] T. Thorp, G. Yee, and C. Sechen, "Domino logic synthesis using complex static gates," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 242–247, 1998.

[35] K. W. Kim, C. L. Liu, and S. M. Kang, "Implication graph based domino logic synthesis," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 111–114, 1999.

[36] R. Puri, A. Bjorksten, and T. E. Rosser, "Logic optimization by output phase assignment in dynamic logic synthesis," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 2–8, 1996.

[37] P. Patra and U. Narayanan, "Automated phase assignment for the synthesis of low power domino circuits," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 379–384, 1999.

[38] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 473–484, Apr. 1992.

[39] P. Larsson and C. Svensson, "Noise in digital dynamic CMOS circuits," *IEEE Journal of Solid-State Circuits*, vol. 29, pp. 655–662, June 1994.

[40] K. Shepard, "Design methodologies for noise in digital integrated circuits," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 94–99, 1998.

[41] N. K. Jha and Q. Tong, "Testing of multiple-output domino logic (MODL) CMOS circuits," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 800–805, June 1990.

[42] A. Walker, A. P. Henry, and P. K. Lala, "An approach for detecting bridging faults in CMOS domino logic circuits using dynamic power supply current monitoring," in *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 272–280, 1997.

[43] R. Morien and R. Anthony, "Testing of domino and latched domino circuits using current sensors," in *Proceedings of Midwest Symposium on Circuits and Systems*, pp. 572–575, 1991.

[44] K. Keutzer, "DAGON: technology mapping and local optimization," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 341–347, 1987.

[45] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology mapping in MIS," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 116–119, 1987.

[46] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang, "Performance-oriented technology mapping," in *MIT Conference on Advanced Research in VLSI*, pp. 79–97, 1990.

[47] K. Chaudhary and M. Pedram, "Computing the area versus delay trade-off curves in technology mapping," *IEEE Transactions on Computer-Aided Design*, vol. 14, pp. 1480–1489, Dec. 1995.

[48] Y. Kukimoto, R. K. Brayton, and P. Sawkar, "Delay-optimal technology mapping by DAG covering," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 348–351, 1998.

[49] D. Gragory, K. Bartlett, A. de Geus, and G. Hachtel, "SOCRATES: a system for automatically synthesizing and optimizing combinational logic," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 79–85, 1986.

[50] M. Lega, "Mapping properties of multi-level logic synthesis operations," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 257–260, 1988.

[51] F. Mailhot and G. DeMicheli, "Technology mapping using boolean matching and don't care sets," in *Proceedings of the European Design Automation Conference*, pp. 212–216, 1990.

[52] F. Mailhot and G. DeMichel, "Technology mapping with boolean matching," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 599–620, May 1993.

[53] H. Sato, N. Takahashi, Y. Matsunaga, and M. Fujita, "Boolean technology mapping for both ECL and CMOS circuits based on permissible functions and binary decision diagrams," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 286–289, 1990.

[54] C. R. Morrison, R. M. Jacoby, and G. D. Hachtel, "Techmap: Technology mapping with delay and area optimization," in *Logic and Architecture synthesis for Silicon Compilers, North-Holland, Amsterdam, The Netherlands*, pp. 53–64, 1989.

[55] M. R. C. M. Berkelaar and J. A. G. Jess, "Technology mapping for standard-cell generators," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 470–473, 1988.

[56] R. K. Brayton, C. L. Chen, C. T. McMullen, R. H. J. M. Otten, and Y. J. Yamour, "Automated implementation of switching functions as dynamic CMOS circuits," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 346–350, 1984.

[57] R. R. Ortiz and M. C. Lefebvre, "Technology mapping for NORA dynamic logic circuits," in *Proceedings of the European Design Automation Conference*, pp. 310–314, 1993.

[58] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. New York, NY: McGraw-Hill, 1990.

[59] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. New York, NY: McGraw-Hill, 1994.

[60] S. Devadas, A. Ghosh, and K. Keutzer, *Logic Synthesis*. New York, NY: McGraw-Hill, 1994.

[61] T. Sakura and A. R. Newton, "Delay analysis of series-connected MOSFET circuits," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 122–131, Feb. 1991.

[62] A. Reis, R. Reis, D. Auvergne, and M. Robert, "The library free technology mapping problem," in *Workshop Notes, International Workshop on Logic Synthesis*, 1997.

[63] Y. Jiang and S. Sapatnekar, "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*, pp. 276–281, 1998.

[64] A. Reis and R. Reis, "Covering strategies for library free technology mapping," in *Workshop Notes, International Workshop on Logic Synthesis*, 1999.

[65] S. S. Sapatnekar and S. M. Kang, *Design automation for timing-driven layout synthesis*. Boston, MA: Kluwer Academic Publishers, 1993.

[66] M. R. C. M. Berkelaar, "LP SOLVE 2.3 Users' Manual," 1998.

[67] H. Liu and D. F. Wong, "Network flow based circuit partitioning for time-multiplexed FPGA's," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 497–504, 1998.

[68] S. Iman, M. Pedram, C. Fabian, and J. Cong, "Finding uni-directional cuts based on physical partitioning and logic restructing," in *4th International Workshop on Physical Design*, 1993.

[69] J. P. Fishburn and A. E. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 326–328, 1985.

[70] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Minimum padding to satisfy short path constraints," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 156–161, 1993.

[71] K. Venkat, L. Chen, I. Lin, P. Mistry, P. Madhani, and K. Sato, "Timing verification of dynamic circuits," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 271–274, 1995.

[72] K. L. Shepard and V. Narayanan, "Noise in deep submicron digital design," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 524–531, 1996.

[73] A. Dharchoudhury, R. Panda, D. Blaauw, R. Vaidyanathan, B. Tutuianu, and D. Bearden, "Design and analysis of power distribution networks in PowerPC microprocessors," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 738–742, 1998.

[74] G. Steele, D. Overhauser, S. Rochel, and Z. Hussain, "Full-chip verification methods for DSM power distribution systems," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 744–749, 1998.

[75] H. Chen and D. Ling, "Power supply noise analysis methodology for deep-submicron VLSI chip design," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 638–643, 1997.

[76] S. Taylor, "The challenge of designing global signals in UDSM CMOS," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 429–435, 1999.

[77] H. Kriplani, F. Najm, and I. Hajj, "Pattern independent minimum current estimation in power and ground buses of CMOS VLSI circuits," *IEEE Transactions on Computer-Aided Design*, vol. 14, no. 8, pp. 998–1012, 1995.

[78] A. Krstic and K. Cheng, "Vector generation for maximum instantaneous current through supply lines for CMOS circuits," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 383–388, 1997.

[79] Y.-M. Jiang, T. Young, and K. Cheng, "VIP – an input pattern generator for identifying critical voltage drop for deep sub-micron designs," in *Proceedings of the International Symposium of Low Power Electronic Devices*, pp. 156–161, 1999.

[80] G. H. Golub and C. F. V. Loan, *Matrix Computations*. Baltimore, MD: The Johns Hopkins University Press, 1984.

[81] C. Ho, A. Ruehli, and P. Brennan, "The modified nodal approach to network analysis," *IEEE Trans. Circuits and Systems*, vol. CAS-22, no. 6, pp. 504–509, 1975.

[82] K. G. Murty, *Operations Research Deterministic Optimization Models*. Englewood Cliffs, NJ: Prentice Hall, 1995.

[83] P. Larsson and C. Svensson, "Noise in digital dynamic CMOS circuits," *IEEE Journal of Solid-State Circuits*, vol. 29, pp. 655–662, June 1994.

[84] S. Ganguly, D. Lehther, and S. Pullela, "Clock distribution methdology for PowerPC microprocessors," *Journal of VLSI Signal Processing*, vol. 16, pp. 181–189, Jun–July 1997.

[85] M. P. Desai, R. Cvijetic, and J. Jensen, "Sizing of clock distribution networks for high performance CPU chips," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 389–394, 1996.

[86] A. B. Kahng and G. Robins, *On Optimal Interconnections for VLSI*. Boston, MA: Kluwer Academic Publishers, 1995.

[87] R. S. Tsay, "An exact zero-skew clcok routing algorithm," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 242–249, Feb. 1993.

[88] D. Lether and S. S. Sapatnekar, "Clock tree synthesis for multi-chip modules," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 53–56, 1996.