PRECISIONSM RESOLUTION TARGETS

PEPPERERS IN ACTION ARE ALLY TESTING SINCE 1974

WIND COUNTY ROAD 5, HAWTHORNE, MN 55037 USA
TEL 612 485 7897 FAX 612 485 7897 TJS BOSTON

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

**Technology Mapping Algorithms for
CMOS Dynamic Logic Circuits**

by

Rolando Ramirez Ortiz, B.Sc.

**A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of**

Master of Engineering

**Department of Electronics
Carleton University
Ottawa, Ontario**

**(c) copyright
1992, Rolando Ramirez Ortiz**

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your title - Votre référence

Our file - Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-79810-6

Canada

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

"Technology Mapping Algorithms for
CMOS Dynamic Logic Circuits"

submitted by Rolando Ramirez Ortiz, B.Sc.
in partial fulfilment of the requirements for
the degree of Master of Engineering

Thesis Supervisor

Chair, Department of Electronics

Carleton University
August 19 1992

ABSTRACT

The last step in logic synthesis is technology mapping, and is defined as the process of transforming an optimized multi-level logic representation into an interconnection of logic elements that correspond to feasible transistor-level circuit designs. In this thesis, a new approach for technology mapping using cell generators and CMOS dynamic logic circuits is given. A new representation model and a dynamic programming approach for technology mapping are introduced for performance optimization. The technology mapping algorithm makes use of a pipelined logic design style for both of combinatorial and sequential logic circuits. The main objective is to optimize circuit throughput. Retiming is used as a compensation technique in order to meet the timing constraints imposed by the selected design style. A prototype is implemented and tested to study the performance of pipelined logic circuit designs. The study shows that pipelined designs yield faster circuits by introducing registers.

ACKNOWLEDGEMENTS

This thesis is the result of a stimulating research environment. The resources provided have been an important factor in the development of this study.

First and foremost I thank my supervisor Dr. Martin Lefebvre for his dedication and support throughout this study. His suggestions, corrections and sound advice are deeply appreciated. I also thank the faculty, staff, and fellow graduate students of the Department of Electronics for enduring my problems. In particular, David Skoll, Richard Griffith and Arthur Castonguay have been particularly helpful.

I thank my family for their support, and to the people in Mexico at the ITESM who made it possible for me to study in Canada. I thank my father Roberto Ramirez G. and Alejandro N. for helping me to leave the wrong place at the right time.

Special thanks are due to my friend Robert L. Stout, with whom I learned what is really needed to learn from a cow, his wife Madeleine Dion Stout and their daughters Tammara and Roberta. Their friendship and continuous support have made an unforgettable experience. I also thank my fast moving friend Jeanne Hackett for her friendship.

Finally, I wish to acknowledge the following sources of financial support: Micronet; the Natural Sciences and Engineering Research Council of Canada (N.S.E.R.C.); the Department of Electronics, Carleton University. The prototype implementation and experimentation were run on a network of SUN workstations provided by the Canadian Microelectronics Corporation (C.M.C.).

Table of contents

	Page
Acceptance Sheet	ii
Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Tables	viii
List of Figures	ix
List of Apendices	xi
CHAPTER 1	1
INTRODUCTION	1
1-1. PERSPECTIVE	1
1-2. OBJECTIVES	3
1-3. THESIS ORGANIZATION	3
CHAPTER 2	5
DYNAMIC LOGIC TECHNIQUES	5
2-1. BASIC CONCEPTS	5
2-1.1. Dynamic Logic Operation	5
2-1.2. Operation Constraints	7
2-2. C ² MOS DYNAMIC REGISTERS	8
2-3. NO-RACE DESIGN TECHNIQUE	9
2-3.1. Description	10
2-4. TRUE-SINGLE-PHASE-CLOCK TECHNIQUE	12
2-4.1. Description	12

CHAPTER 3	15
TECHNOLOGY MAPPING FOR STATIC CELLS	15
3-1. MAPPERS BASED ON CELL LIBRARIES	16
3-1.1. Netlist model and algorithm in Socrates	16
3-1.2. Boolean equation model and algorithm in CERES	17
3-1.3. DAG model and algorithm in MIS	19
3-1.4. Tree model and algorithm in DAGON, MIS and Grenoble ..	20
3-2. MAPPERS BASED ON CELL GENERATORS	21
3-2.1. Eindhoven	21
3-2.2. Rossini	23
CHAPTER 4	25
PROBLEM DEFINITION	25
4-1. TARGET CIRCUITS	25
4-2. TARGET DESIGN TECHNIQUE	26
4-3. TARGET MAPPING	27
4-3. OVERVIEW FOR THE IMPLEMENTED APPROACH	28
CHAPTER 5	35
REPRESENTATION MODEL	35
5-1. GRAPH-LEVEL MODEL	35
5-1.1. Synchronous Network Graph	35
5-1.2. Hierarchical Synchronous Network Graph	37
5-2. GATE-LEVEL MODEL	42
5-3. TIMING-LEVEL MODEL	44
CHAPTER 6	46
TECHNOLOGY MAPPING ALGORITHM FOR NORA	46
6-1. VERTEX DECOMPOSITION	46
6-2. DYNAMIC MATCHING: THE TREE-GRAPH CASE	48
6-2.1. G' Network Matching: P Sub-Network Boundaries Exploration	48
6-2.2. P Network Matching: CG Network Boundaries Exploration .	50
6-2.3. P Network Matching: CG Network Gate-Design Exploration	51

6-3. DYNAMIC COVERING	55
6-3.1. P Network Covering	55
6-3.2. G' Network Covering	56
6-3.3. Limiting Heuristics	58
6-4. RETIMING COMPENSATION	59
6-4.1. Global Retiming	60
6-4.2. Local Retiming	63
CHAPTER 7	65
EXAMPLE	65
7-1. CIRCUIT DECOMPOSITION INTO G' NETWORKS	65
7-2. DYNAMIC MATCHING: P AND CG GRAPH BOUNDARIES ..	67
7-3. LOCAL SELECTION FOR P NETWORK GRAPH	67
7-4. DYNAMIC COVERING: P NETWORK GRAPH COVERING ..	72
7-5. DYNAMIC COVERING: G' NETWORK GRAPH COVERING ..	72
7-6. RETIMING COMPENSATION	77
CHAPTER 8	79
EXPERIMENTAL RESULTS	79
8-1. COMBINATORIAL BENCHMARKS	79
8-2. SEQUENTIAL BENCHMARKS	81
8-3. LIMITING HEURISTICS	82
CHAPTER 9	84
CONCLUSIONS	84
9-1. SUMMARY	84
9-2. CONTRIBUTIONS	85
9-3. RECOMMENDATIONS FOR FUTURE WORK	86
APPENDIX I. PROTOTYPE USER MANUAL	88
APPENDIX II. GLOSSARY	95
REFERENCES	103

LIST OF TABLES

	Page
Table 1	Measured Performance for Combinatorial Benchmarks 73
Table 2	Measured Performance for Sequential Benchmarks 75
Table 3	Measured Performance for Benchmark Circuits COUNT and EX3 76

LIST OF FIGURES

	Page
Figure 2.1 n-type Dynamic NOR Gate	6
Figure 2.2 N-P Structure	7
Figure 2.3 Clocked CMOS register (C ² MOS)	9
Figure 2.4 NO-RACE Design Technique for Pipelined Logic	11
Figure 2.5 TSPC-1 Register and TSPC-2 Register for Pipelined Logic Design	13
Figure 2.6 Doubled n/p-C ² MOS Registers and Split-Output Registers	14
Figure 3.1 Recursive Shannon Decomposition	17
Figure 3.2 NPN-equivalent Definition and Example	18
Figure 3.3 "Compatibility Graph" for 3 Variable Boolean Space	19
Figure 3.4 Directed Acyclic Graph (DAG)	19
Figure 3.5 Single Function Graph Representation	22
Figure 3.6 Example of Constructive Matching	23
Figure 3.7 Example of Covering	24
Figure 4.1 Implemented NO-RACE Design Technique for Pipeline 1 Logic Design	27
Figure 4.2 Grouped Logic expressions and Registers based on the fanout	29
Figure 4.3 Synchronous Network Graph at the Hierarchy Level G	29
Figure 4.4 Synchronous Network Graph at the Hierarchy Level G'	30
Figure 4.5 Synchronous Network Graph at the Hierarchy Level P	32
Figure 4.6 Gate Level Design for a P graph	33
Figure 4.7 Possible gate-level design for the complete circuit	34
Figure 5.1 Synchronous Network Graph Representation Used for Global Retiming	36
Figure 5.2 G' Synchronous Network Graph Representations	38
Figure 5.3 Synchronous Network Sub-Graphs from G'(viol)	39
Figure 5.4 Synchronous Network Graph Representations for Complex Gate Design	41
Figure 5.5 Notation Description	43
Figure 6.1 Binary Partitioning for Vertices	47
Figure 6.2 Give-Output-Vertex Algorithm	48
Figure 6.3 Example of the Give-Output-Vertex Algorithm	49
Figure 6.4 Sample CG Network Boundaries Exploration	51
Figure 6.5 CG Gate-Design for Input Vertices	52
Figure 6.6 CG Gate-Design for Main-Body-Logic Functions where $V_{CG}^0 \neq V_P^0$	53

Figure 6.7 CG Gate-Design for Main-Body-Logic Functions where $V_{CG}^0 = V_P^0$	54
Figure 6.8 Sample Local Selection for P Sub-Network	56
Figure 6.9 Sample Results of Local Selection Process for $G'(viol)$	57
Figure 6.10 Global Retiming Problem Formulated as an Integer Optimization Problem ..	61
Figure 6.11 Sample Global Retiming for Benchmark Circuit DAIO	62
Figure 6.12 Local Retiming Optimization Equations	63
Figure 6.13 Sample Local Retiming for Benchmark Circuit DAIO	64
Figure 7.1 Hierarchical Synchronous Network Graph Representation	66
Figure 7.2 P and CG partitioned networks Generated from $G'(viol)$	68
Figure 7.3 Generated Input Gate-Designs for CG Networks	69
Figure 7.4a Generated Gate-Designs for CG networks	70
Figure 7.4b Generated Gate-Designs for CG networks	71
Figure 7.5 G' Matches Locally Selected	73
Figure 7.6 Throughput and Clock Period for Circuits of (viol)	74
Figure 7.7 Throughput and Clock Period for Circuits of (NOT viol)	75
Figure 7.8 $G'(viol)$ Gate-Design Mappings	76
Figure 7.9 Circuits with Register Weight Requirements	77
Figure 7.10 Retiming Compensation Example	78

LIST OF APPENDICES

	Page
Appendix I. Prototype User Manual	81
Appendix II. Glossary	88

CHAPTER 1

INTRODUCTION

1-1. PERSPECTIVE.

In the past years, the microelectronics industry has seen a rapid growth in the number of applications for integrated circuits. The integrated circuit (IC) market has increased by more than a factor of 15 since the early 1960s. IC sales in the United States were \$11 billion in 1986 and are expected to reach \$90 billion by the year 2000 [Sze]. Important contributions for such a success are the continued technological breakthroughs in integrated circuit technology and the development of Computer-Aided Design (CAD) tools.

Scaling down of transistor device size for CMOS (Complementary Metal Oxide Semiconductor) technology has improved integrated circuit performance and increased the number of components per integrated circuit chip. Integrated circuit complexity has advanced at an exponential rate from the early small-scale integration (SSI) during the 1960s, to medium-scale integration (MSI), to large-scale integration (LSI) and to very-large scale integration (VLSI), which has 10^5 or more components per chip. The advantages in device miniaturization and low power dissipation make CMOS technology the forefront technology for VLSI circuits.

The larger number of devices in VLSI chips and the human limitations are a motivation for the development of Computer-Aided Design tools. Chip designs using CAD tools are virtually free of debugging problems, such as short circuits or logic design errors. Logic circuit design using CAD tools enables faster product development and a better performance optimization.

Logic synthesis is part of the "silicon compilation" techniques developed for CAD. Logic synthesis is the automatic design of digital integrated circuits from a behavioral specification level to the transistor circuit level. The last step in logic synthesis is technology mapping and is defined as the process of transforming an optimized multi-level logic representation into an interconnection of logic elements that correspond to feasible transistor-level circuit designs. Logic elements are designed into CMOS transistor-level circuits using two major logic circuit design techniques named CMOS Static Logic Circuits and CMOS Dynamic Logic Circuits.

The majority of the developed technology mapping algorithms are targeted to cell libraries and CMOS static logic circuits. In VLSI integrated circuits using a limited number of logic elements becomes a design flexibility constraint. Cell generators are an alternative for cell libraries. The technology mapping process with CMOS static logic circuits using automatic circuit generators has been studied and implemented at the Eindhoven University of Technology in The Netherlands [BeJe] and at Carleton University [Liem].

Technology mapping algorithms have been developed in an attempt to extract the advantages of simple design techniques such as the Domino logic technique or the dynamic logic CVS technique. Technology mapping using Domino logic and dynamic logic CVS have been implemented in the Yorktown silicon compiler [BrBr]. Mapping of Domino logic circuits has proven to be an efficient circuit design technique with acceptable performance [Shoj] at the expense of design flexibility and sensitive clocking schemes. Logic circuit design using the dynamic logic CVS technique has proven to be an expensive technique for logic circuit designs implemented with expressions different than a logic EXOR [ChPu]. The circuit design techniques developed for CMOS dynamic logic circuits are the NO-Race (NORA) technique [GoMa], Domino logic technique [KrLe], True-Single-Phase-Clock (TSPC) technique [YuSv1], and dynamic logic Cascode-Voltage-Switch (CVS) technique [HeGr].

In this thesis, an objective analysis is given for different approaches of technology mapping and CMOS dynamic logic circuit design techniques. A new approach for technology mapping based on speed performance optimization is given for the pipelined logic design of combinatorial and sequential logic circuits using CMOS dynamic logic techniques. The transformation of combinatorial and sequential logic circuits into a pipelined logic circuit design is a logic circuit design method used for speed performance optimization of CMOS dynamic logic circuits.

1-2. OBJECTIVES.

The first objective of this thesis is to explore the potential of a technology mapping algorithm for CMOS dynamic logic circuits using pipelined logic. This objective is met by a study of existing technology mapping algorithms and CMOS dynamic logic techniques, in an attempt to develop a technology mapping algorithm.

The second objective is to describe a working prototype for experimentation with different pipelined logic designs. This prototype should be a technology mapper based on cell generators capable to optimize the performance of logic circuits.

1-3. THESIS ORGANIZATION.

The thesis is organized as follows. Chapter 2 explains the circuit level operation of dynamic logic circuits. It describes the most relevant dynamic logic design techniques and explains circuit design constraints. Chapter 3 reviews the representation models and algorithms for technology mappers based on libraries and cell generators. Chapter 4 defines the target circuits and design technique for technology mapping using CMOS dynamic logic circuits. Chapter 5 presents a new approach for the representation model used by the technology mapping algorithm. The representation model has three levels of representation: circuit level, gate level and timing level. Chapter 6 describes the technology mapping algorithm for CMOS dynamic logic circuits using the circuits and technique defined in Chapter 4. Chapter 7 describes an example following every step of the technology mapping algorithm using a

benchmark circuit. Benchmark circuits are extracted from the MCNC benchmark library for logic synthesis of 1991. Chapter 8 presents performance results for sequential and combinatorial logic circuits using benchmark circuit designs. Chapter 9 concludes with a summary, contributions, and recommendations for future work.

CHAPTER 2

DYNAMIC LOGIC TECHNIQUES

This chapter explains the mechanism, constraints and basic operation principles of Dynamic Logic as well as related circuit design techniques. It focuses on an objective presentation of the most relevant design techniques: Domino[KrLe], NO-RACE [GoMa] and True-Single-Phase-Clock [YuSv1]. This presentation of techniques is used in Chapter 4 to determine the objective Dynamic Logic design technique.

2-1. BASIC CONCEPTS.

Dynamic Logic is a CMOS circuit design technique based on the conditional discharge of an unconditionally pre-charged output capacitance. A number of techniques have been developed in an attempt to extract its advantages, which rely on the same operation principle explained in the next subsection.

2-1.1. Dynamic Logic Operation.

Dynamic Logic uses a timed control over the charge and discharge of the output load. Under ideal timing, current control is accomplished using n and p transistors driven by a clock signal. The clock signal is the last arriving input signal to the logic gate. In Figure 2.1, the Dynamic Logic gate is arranged with an unconditional pre-charge state during the V_L state of the clock signal and a conditional discharge during the V_H state. Clocked transistors, while providing control over output transitions, allow the use of circuit designs with a single logic evaluation block.

The gate-level design in Figure 2.1 has an *n*-type evaluation block, and therefore is named an *n*-type gate. A *p*-type gate is defined as a *p*-type evaluation block driven by transistors with a $\overline{\text{clock}}$ signal. A *clock* section is defined, if the gates pre-charge during V_L state of the clock signal and evaluate during the V_H state. A $\overline{\text{clock}}$ section is defined, if the gates are driven by an opposite clock phase having to precharge during the V_H state of the clock signal and evaluate during the V_L state.

Figure 2.1 *n*-type Dynamic NOR gate. A V_L clock signal precharges C_o , and a V_H clock signal discharges C_o subject to the conditional paths in the evaluation block.

The opposite clock signal phase for *n*-type and *p*-type gates controls the data exchange in a compatible gate structure named *n-p structure*[FrLi]. In Figure 2.2, an *n-p* structure using *n*-type and *p*-type gates is connected to static gates. A *clock* section in the *n-p* structures defines a design where the evaluation starts at the rising edge of the clock signal. A $\overline{\text{clock}}$ section in the *n-p* structures defines a design where the evaluation starts at the falling edge of the clock signal.

Figure 2.2 N-P structure. *n-type* gate connected to a *p-type* gate, or vice versa, using opposite clock signals guarantees a safe precharge state and a simultaneous evaluation phase. Static inverters connected in a Domino style[KrLe] allow for a greater flexibility.

Logic circuits designed with *n-p* structures and within the same *clock* section exchange information at the evaluation state of the clock signal (V_H for *clock* section) propagating a logic evaluation from inputs to output. An *n-type* gate for instance, is precharged during the clock V_L state whereas a *p-type* gate in the same section precharges during $\overline{\text{clock}}$ V_H state. Opposite clock signals for *n-type* and *p-type* gates serves this way for a simultaneous precharge and logic evaluation within the same section. Also, alternating *n-type* and *p-type* gates keeps the precharged levels for all gates until they are evaluated.

2-1.2. Operation Constraints.

Dynamic logic gates with a precharged logic level switch to the opposite logic level with a conditional path connecting the output node to the corresponding clock transistor. The discharge path can either make a complete path allowing a logic transition, or an incomplete path remaining in the same logic state. Incomplete paths cannot change the logic level at the output, although voltage levels may drop due to charge redistribution. *Charge redistribution* takes place once an incomplete path connects the load at the output node with parasitic loads in the evaluation structure. Charge redistribution drops the output voltage level, and for ex-

treme cases decreases below the threshold level resulting in a faulty logical level at the output.

An obvious advantage from using Dynamic Logic gates is the ability to implement a logic state with a single evaluation block. *n-type* gates provide a minimum area usage if they can feed *n-type* gates exclusively. Same gate type is an unfeasible design style, because the precharged state of an *n-type* gate feeding another *n-type* gate turns *ON* all paths, in the evaluation block of the output gate, corrupting this way the precharged level of the output. Such a race problem is defined as the *internal delay problem* in [GoMa]. The most common technique to avoid internal delay problems is named the Domino Logic technique. Domino logic uses a static inverter at the outputs, turning *OFF* all transistors in the evaluation block with a V_L precharged logic state. In Figure 2.2, designs using Domino logic are connected to *n-type* or *p-type* dynamic gates. Domino Logic designs can interact with registers in a pipelined logic design style as explained in Section 2–3.

2-2. C²MOS DYNAMIC REGISTERS.

Clocked CMOS registers (C²MOS) are the registers used by a circuit design technique developed by Suzuki/Odagawa [SuOd]. The C²MOS register operation is based on the conditional discharge of a conditionally precharged output capacitance. Current control is accomplished using *n* and *p* transistors driven by complementary clock signals. The clock signals are the last arriving signals to the register.

In Figure 2.3 the C²MOS register precharges the output load C_o according to the logic evaluation blocks, *n* and *p*, during the V_H state of the clock signal. The V_L state of the clock signal leaves the output load C_o at the precharged level. Series connected C²MOS registers have an exchange of logic states if the clock phase is complemented from one register to the next one. In Figure 2.3 the boxed transistors can be replaced by more complicated logic evaluation blocks. C²MOS registers that evaluate complex logic functions are named *C²MOS complex gate registers*.

The output of a C^2 MOS register has a dynamic operation that is affected by the delay between clock and $\overline{\text{clock}}$ signals on the same section. Clock skew between the clock phases is a type of race problem for designs that use C^2 MOS registers or n-p structures and may corrupt evaluation levels. C^2 MOS registers driven by skewed clock signals allow one of the evaluation blocks to gain an earlier access to the output load. This earlier access drops the output voltage generating glitches that can irreversibly corrupt precharged levels in the dynamic logic gates. This race problem is solved with a design technique named NORA and is explained in the next section.

Figure 2.3 Clocked CMOS register (C^2 MOS).

2-3. NORA/DOMINO DESIGN TECHNIQUE.

Asynchronous logic circuit design is often subject to imbalance of signal arrival due to the different evaluation delays through paths. Gate design techniques always need to compensate such differentials with logic transformations (in static logic) or sophisticated clock distribution designs (in Domino logic). Either alternative may become too complex or expensive. NORA (which stands for NO-RACE) is a dynamic logic technique designed to overcome race problems and signal imbalance using pipelined logic with n-p structures and C^2 MOS dynamic registers within the logic. C^2 MOS dynamic registers are compatible with dynamic logic gates and static gates under certain constraints that allow race-free designs as explained in the following subsection.

2-3.1. Description.

Gate designs in a Domino logic style use a static inverting gate at the output to overcome the race problem from precharged states. The use of static inverters add design constraints that limit the number of inverting static gates between Domino gates to an even number in order to preserve the V_L precharged state. Also, static inverters at the outputs of dynamic logic gates result in non-inverting logic circuits. Therefore, Domino logic is a constrained technique in itself. N-P structure designs solve the non-inverting constraint in Domino logic, and add design flexibility by interfacing with Domino logic designs. In Figure 2.2 static inverters make n-type Domino logic gates and p-type Domino logic gates.

NORA is a pipelined logic technique based on n-p structures that uses C²MOS dynamic registers for data storage between pipeline stages. Stages in the pipeline have inverted clock phases as a protection against races from precharged levels. This inverted clock phase in relation to the next *clock* section also serves to stop glitch propagation.

In Figure 2.4, the design constraints for the NORA technique are specified for race-free pipelined logic designs. The *clock* section has an even number of static inversions between nodes N_1 and N_2 making their effect transparent in terms of signal propagation. Static gates have a direct contact with C²MOS registers because in general static gates driven by dynamic gates are prone to glitches[GoMa]. This even number of static gates for the complete section makes nodes N_1 , N_2 , N_3 and N_4 the same operational nodes. A p-type block for instance, connected to node N_3 in precharge state uses a $\overline{\text{clock}}$ signal in V_H state; this same $\overline{\text{clock}}$ signal floats node N_5 for V_L input signals. As a result, pre-charged low levels in nodes N_3 and N_4 may not switch the output of the register. The same happens if an n-type block is connected to node N_3 , since the clock signal used on V_L state to precharge also floats node N_5 for V_H precharged input signals.

Figure 2.4 NO-RACE design technique for pipelined logic using n-p structures and C²MOS dynamic registers. Constraints for a race-free operation are also specified.

Time differentials between clock and $\overline{\text{clock}}$ signals have no effect at nodes N_3 and N_4 , but in extreme cases it may have catastrophic influence over precharged states for gates at output nodes like N_1 and N_5 that have input signals from C²MOS registers. The following clock scheme prevents the corruption of states from one section to the next section; the $\overline{\text{clock}}$ section needs a V_L clock signal to switch node N_1 , whereas the evaluation period of the clock section requires a clock signal in V_H state to propagate the logic evaluation of node N_1 .

The generation of the $\overline{\text{clock}}$ signal from the clock signal add a delay that is reflected as a sequence of logic states where both signals have the same logic level. In Figure 2.4 a negative transition on the clock signal generates for a short period of time a V_L state for both clock and $\overline{\text{clock}}$ signals that in turn load a V_H state in node N_1 . The even number of inversions propagates the V_H level to node N_4 . This V_H level in node N_4 needs a clock signal in V_H to force a corrupted logic level. This way an even number of inversions is a protection against undesirable transitions at node N_5 due to the clock skew from the generated $\overline{\text{clock}}$ signal.

2-4. TRUE-SINGLE-PHASE-CLOCK TECHNIQUE.

True-Single-Phase-Clock technique is a pipelined logic design technique for dynamic logic gates that uses n - p structures and True-Single-Phase-Clock registers (TSPC). TSPC registers are dynamic logic non-inverting registers designed to operate with the same clock phase for both $clock$ and $\overline{clock}$ sections. Pipelined logic using n - p structures in dynamic logic is a simple solution to imbalance of signal arrivals. NORA is a pipelined logic design technique where a number of design constraints are necessary to implement the latching process using C^2MOS registers, and to protect against races between sections. Inside these sections, the n - p structures are an efficient anti-race technique and a flexible design concept that is limited by constraints from the NORA technique. True-Single-Phase-Clock registers, as explained in the following section, make an alternative design technique for dynamic pipelined logic where no even inversion constraints are necessary between sections.

2-4.1. Description

The clock signal transistors in the C^2MOS registers give premature discharge protection from pre-charged levels. However, a C^2MOS register fed by an n -type dynamic gate only needs to protect against the V_H precharged state. The p -type clock transistor has a redundant function and is eliminated; the C^2MOS register becomes an n - C^2MOS register [YuSv2]. A p - C^2MOS register is a C^2MOS register with the n -type clock transistor eliminated.

The TSPC technique uses n - C^2MOS / p - C^2MOS registers, named n/p - C^2MOS ; and uses the n - p structures between registers that are forced to provide the required gate type sequence. The elimination of one clock transistor and one clock signal phase, from one clock section to the next one, gives register designs driven by a single clock phase in the pipelined design. A circuit with single clock phase needs no constraints to protect against races since the component of clock skew due to complementary clock signals is eliminated. Figure 2.5a is the alternative version of the C^2MOS register named True-Single-Phase-Clock type 1 or

TSPC-1 and is obtained with a combination of n/p-C²MOS registers and dynamic inverters in a single clock design.

Figure 2.5b is an improved register design from TSPC-1 named True-Single-Phase-Clock type 2 or TSPC-2. TSPC-1/2 n-type registers are the output of *clock* sections and are fed by p-type dynamic logic gates, whereas the p-type, TSPC-1/2 registers are the output of $\overline{clock}$ sections and are fed by n-type dynamic logic gates. Design flexibility is increased compared to the NORA technique, because TSPC registers allow any number of inversions between registers using n-p structures and static gates. Therefore, design flexibility for TSPC registers is unconstrained for logic design but it is constrained for technology implementation, because the gate type sequence for n-p structures is constrained by the section type.

Figure 2.5 TSPC-1 register and TSPC-2 register for pipelined logic design. The boxed transistors can be replaced by an evaluation block.

TSPC-1 and TSPC-2 registers have the same logic operation and a similar circuit design. Both registers have been compared in terms of step response, glitch generation and load capabilities [YuSv1]. TSPC-1 registers accept a higher load than TSPC-2 since TSPC-1 discharges through two transistors instead of three. TSPC-2 registers accept a smaller load but perform with a faster step response. For example, in Figure 2.5b, if there

is a V_H input signal for the n-type register the internal common node A is precharged. At the evaluation phase, transistor T_1 is ON for a short period of time while the gate node is being discharged. During that same period of time node A provides a higher charge for the output while T_1 is in the process to turn OFF.

Extreme pipeline designs lead to smaller clock periods due to the evaluation of simple logic expressions that reduce the evaluation period between registers. The TSPC technique may lead to faster designs due to the exploration of more design possibilities, since logic blocks can be designed within the first stage of any TSPC register. A single register with complex logic could allow for shorter paths, from inputs to output in a section, and a reduced clock period [YuSv1].

C^2 MOS and TSPC-1/2 registers have glitch protection under certain restrictions in the n-p structures. The alternative register design within a True-Single-Phase-Clock scheme is named *doubled n/p- C^2 MOS* [YuSv1]. In Figure 2.6 the *doubled n/p- C^2 MOS* registers have the series connection of two n/p- C^2 MOS registers that allow for a completely unconstrained n-p structure section design at the expense of extra delay from a two-stage register. An improved version evolved from doubled n/p- C^2 MOS registers in terms of clock load is named Split-Output register and is shown in Figure 2.6b.

Figure 2.6 doubled n/p- C^2 MOS registers and Split-Output registers for pipelined logic design. The logic gates between registers are not constrained.

CHAPTER 3

TECHNOLOGY MAPPING CONCEPTS

This chapter presents an overview of models and algorithms used in different technology mapping algorithms. The first section presents only the most relevant concept contributions from library based mappers. The second section explains two important contributions based on cell generators that are influential algorithms for the technology mapper using dynamic logic cells.

Technology mapping is the final process in the logic synthesis. A logic circuit is implemented to a transistor-level design either from a library of circuit gates or using a cell generator. This abstract transformation considers technology dependent parameters and takes advantage of the particular logic circuit towards an optimum design.

Technology mapping is identified as a process performed in two steps. First, in the matching step all possible circuit gate realizations, or matches, are extracted from the logic circuit. Second, in the covering step a set of matches are selected for a feasible circuit design towards an optimum performance.

The technology mapping process uses a circuit model to represent the original circuit and manipulates the information of the model during match and cover steps to provide a circuit implementation. The model represents a circuit design before technology mapping, during the match and cover process, and the technology dependent circuit gate design. The representation model aids the design algorithms to take advantage from each representation level, such as optimization of combinatorial logic designs and balanced signal arrivals.

3-1. MAPPERS BASED ON CELL LIBRARIES.

3-1.1. Netlist model and algorithm in Socrates.

Netlist model is the circuit representation model used in the Socrates[GrBa] synthesis system. Netlist representations correspond to a direct circuit design of a possible final implementation. Each element of a netlist corresponds to a partial representation of the logic and a complete representation for the circuit gate, or it may just represent a pin for the case of inputs. Netlist representation is modified by the technology mapper using a library also described by netlist representations.

The mapping algorithm for netlist representation transforms an input netlist using a library of gate transformations. The library is maintained with a group of *if-then* rules to transform a target circuit to a more desirable replacement circuit. The decision for each transformation is based on a cost function dependent on the number of applicable rules and the savings from the replacement, on either area or speed.

Rules may give different designs depending on the order of application. A decision structure based on a tree representation is created to perform a *look-ahead* process that keeps all the possible designs with a set of rules. The growth of the tree is limited by statistic knowledge parameters, reducing this way CPU time consumption. A dynamic approach to modify the parameters in the cost function allow for a global optimization by preventing a concentrated local optimum. The continuous circuit improvement from rule application performs a match process within the covering process. That is, the covering is not terminated until no more replacement matches are found.

3-1.2. Boolean equation model and algorithm in CERES.

The Boolean equation model is the circuit representation used by the CERES[FrGi] library based technology mapper. The objective of the model is to exploit Boolean properties of equations such as don't cares, multiple occurrence of variables and phase information. A Boolean equation model represents the initial desired logic, and after technology mapping each equation corresponds to a gate circuit. The library of circuit gates is also expressed by Boolean equations.

The matching process is the tautology checking process using recursive Shannon decomposition [BrHa]. Shannon decomposition is the act of cofactoring an algebraic Boolean expression. Tautology checking is the process of finding equivalent functions or the complementary in a library. Figure 3.1 has the Shannon decomposition for a Boolean expression over the input variables. Thus, for each component in the library with n input variables there are $n!2^n$ possible tautologies, as defined in [FrGi], that may fit to generate a single match [FrGi].

$$\begin{aligned}
 f &= ab + bc + \bar{a}c \\
 &= a(b + bc) + \bar{a}(bc + c) \\
 &= a(b(1+c) + \bar{b}(0)) + \bar{a}(b(c) + \bar{b}(c))) \\
 &= a(b(1) + \bar{b}(0)) + \bar{a}(b(c(1) + \bar{c}(0)) + \bar{b}(c(1) + \bar{c}(0))) \\
 g &= ab + \bar{a}c \\
 &= a(b) + \bar{a}(c) \\
 &= a(b(1) + \bar{b}(0)) + \bar{a}(b(c) + \bar{b}(c)) \\
 &= a(b(1) + \bar{b}(0)) + \bar{a}(b(c(1) + \bar{c}(0)) + \bar{b}(c(1) + \bar{c}(0))) \\
 h &= ab + bc \\
 &= a(b + bc) + \bar{a}(bc) \\
 &= a(b(1+c) + \bar{b}(0)) + \bar{a}(b(c) + \bar{b}(0)) \\
 &= a(b(1) + \bar{b}(0)) + \bar{a}(b(c(1) + \bar{c}(0)) + \bar{b}(c(0) + \bar{c}(0)))
 \end{aligned}$$

Figure 3.1 Recursive Shannon decompositions. [Liem] Equation $f=g$ is a tautology.

Tautology search is an extensive process that is speeded up using symmetries in the expressions and unate variable properties. Symmetries in the expressions allow for an exchange in variable positions and reduce the number of matches. Unate variables are defined to allow the generation of matches for only one phase of a variable, as opposed to binate variables where both phases are present.

Figure 3.2 has the definition for NPN-equivalent logic expressions to determine symmetries of functions. A function f is said to be NPN-equivalent to a function g if f can be derived from g through the negation of variables in g , the permutation of variables in g or the negation of g . A compatibility graph is defined by $G(V,E)$ with vertices V representing NPN-equivalent sets is defined in Figure 3.3 for the logic function $f=(a+b)c$ with a don't care function of $\overline{bc}$. Vertices are joined by edges E if one vertex performs the same logic function by just adding (or extracting) the miniterm of the edge. Don't care functions provide a larger set of matched library tautologies by expanding the function (one vertex) to a set of possible functions (vertices joined by edges representing options from don't cares). Reductions by symmetry are not applied over the *don't care* logic expressions during the match search process of tautologies.

f is NPN-equivalent to g , if f is derived from g using:

Negation of one or more variables of g
 Permutation of variables in g
 Negation of g

i.e.:

function	NPN-equivalents
$f = abc$	$abc \quad \overline{abc} \quad a+b+c \quad \overline{a+b+c}$
$\begin{matrix} a & & \\ & b & \\ \swarrow & & \searrow \\ & c & \end{matrix}$	$\overline{abc} \quad \overline{\overline{abc}} \quad \overline{a+b+c} \quad \overline{\overline{a+b+c}}$
	$a\overline{b}c \quad a\overline{b}\overline{c} \quad a+b+c \quad \overline{a+b+c}$
	$a\overline{b}c \quad a\overline{b}\overline{c} \quad a+b+c \quad \overline{a+b+c}$

Figure 3.2 NPN-equivalent definition and example. [Liem]

Covering is performed with a dynamic approach that builds up an optimum circuit from a set of matches. The network is partitioned into graphs with one output of multiple

fanout. The graphs are decomposed into a network of two-input gates (AND/OR). The graphs are characterized by sub-graphs called clusters using a recursive construction of equations to determine all the possible gate implementations. The gate-implementations are determined by matching clusters to a library that is also represented with networks of AND and OR expressions. The propagation delay is computed by adding the delays from input clusters to the matched one.

Figure 3.3 Compatibility graph for 3 variable Boolean space. Dotted cubes identify function $F=(a+b)c$ plus don't care cubes from $DC=\bar{b}\bar{c}$.

3-1.3. DAG model and algorithm in MIS.

Directed Acyclic Graph (DAG) model is the circuit representation used for the MIS[DeGa] technology mapper. DAGs are composed of one way directed edges and vertices. In Figure 3.4, vertices correspond to a logic operator or input/output pin. Edges represent the data flow from input pins through operators to output pins. Cycles are not allowed and only simple logical operators are represented (i.e. NOT, AND, NAND, OR, etc.).

Figure 3.4 Directed Acyclic Graph (DAG) for a sub-circuit extracted from MCNC's benchmark DAIO.

The mapping algorithm uses a library of cells with each cell transformed to a push down automaton. Thus the push down automaton performs the matching process with a concurrent automata traversing through the DAG arcs following a depth-first order. Network vertices are bound to the cell vertices and a match is identified if the fanin vertices are the same. DAG matching is executed over the entire network providing a global optimization with matched vertices regardless of their fanout.

Covering uses different approaches to solve a matrix of nodes versus matches found. Heuristics such as greedy strategies or branch and bound techniques may lead to an optimum cost within the limitations of the heuristic itself.

3-1.4. Tree model and algorithm in DAGON, MIS and Grenoble.

Tree model circuit representation is used in the DAGON[Keut], MIS[DeGa] and Grenoble[SaSa] technology mappers. Tree model circuit representation is obtained from a partition according to the internal fanout of nodes. Trees are sub-circuits of one output with a multiple fanout and are composed of nodes and edges. Each node corresponds to a logic operator initially of depth one (AND, NAND, OR, NOT, etc.). Nodes are joined by edges to represent the logic evaluation hierarchy of the logic function that flows from leaves to the root.

The MIS technology mapper uses the tree model as a subgraph definition for a DAG model representation. The tree model subgraph is matched and covered using the mapping

algorithm described in Section 3-1.3. Tree model allows for a local covering, where logic minimization and logic duplication lead to smaller trees and fewer sub-circuits. The circuit gate library is maintained as tree shaped gates where all non-isomorphic and irredundant forms of trees are represented. The covering process is also an heuristic approach using a matrix of nodes versus matches.

The DAGON technology mapper uses tree partitioned graphs from a global DAG representation for the complete network, and a constrained set of Boolean operators for nodes in the tree. Logic functions are represented with AND and NOT operators exclusively, where either of the signal phases are provided at the edges. The library is maintained in a compatible logic representation also with just AND and NOT operators. The matching step is based on string matching [AhCo] techniques. Trees are string matched with the library to provide all possible candidate gates for each node in the tree. Nodes in the tree have an associated subtree depending on which match is implemented. Dynamic programming is used for a selection of such subtree gates to obtain a covered design for each tree.

The Grenoble technology mapper uses isomorphisms between tree portions and standard cells. The matching process follows a minimum logic depth selection. That is, the match objective is to find as big as possible gates with a low logic depth using existent gate matches in the library. The logic functions for nodes in the tree are partitioned to keep feasible match functions by comparing the size limits of cells available in the library.

The match process in the Grenoble mapper is a selection of subtrees that are represented in the library. Matches are transformed to gate designs using inverters, and an automatic dual function generator transforming AND-NOT functions to feasible gate circuits. A dual function is defined as an opposite phase output signal fed by opposite phase input signals. The covering process is identified as the gate circuit design process using the matched sub-trees from the library cells and dual functions.

3-2. MAPPERS BASED ON CELL GENERATORS.

3-2.1. Eindhoven

The technology mapper developed at the Eindhoven University of Technology in The Netherlands [BeJe] is presented on this section. The Eindhoven technology mapper presents an algorithm based on subgraph generation from Boolean equations and gate circuit generators.

Each Boolean equation is represented by a directed graph $G(V,E)$, where V is the set of vertices and E the edges. The graph is constructed to represent an equivalent of an n -type evaluation block. In Figure 3.5, the vertices are circuit nodes and the edges represent the charge path through transistors.

Longest path and largest cut-set in the graph $G(V,E)$ are properties for each expression constrained by limitations in the cell generator. Longest path corresponds to the series combination of n -type transistors, and also the parallel combination of p -type transistors. Largest cut-set is the parallel combination of n -type transistors and the series combination of p -type transistors. Thus, technology limitations are included by constraining longest path and cut-set parameters.

Figure 3.5 Single function graph representation [BeJe] for the expression:
 $(+ (* a b c) (* d e (+ f g) (+ h i)) (* j k))$

The functions are taken one at a time and have a fanout greater than one. Technology mapping starts with the match of such functions by determining matches with a recursive

match construction. The recursive match construction expands a given logic function by adding logic depth from the feeding operators and are constraint verified using the graph representation in Figure 3.5. Matches are designed as big as possible to minimize the logic depth and the number of gates.

The objective of the covering process is to generate a design with as few gates as possible. Matches are selected to minimize the number of inverters along critical paths.

3-2.2. Rossini.

The Rossini technology mapper developed at Carleton University by Liem [Liem] is presented in this section. The Rossini technology mapper relies on circuit gate generators and is based on the tree partitioned representation model.

Trees are logic functions with one output of multiple fanout that are optimized independently. Trees are composed of nodes and edges. Nodes represent simple Boolean functions (AND, OR, NOT) and edges represent the logic evaluation hierarchy from leaves to root. The matching process is based on a recursive subtree construction process with an auto-dual cell generator. A topological sort is applied at nodes with more than two input edges to balance the logic depth towards a binary tree structure. Trees get transformed into binary tree structures to guarantee a maximum match search and a balanced signal arrival.

Figure 3.6 example of constructive matching.[Liem]

Match generation is subject to the cell generator and to the technology constraint parameters (SPLN). Constraint parameters limit the number of n-type transistors in series (S), the number of p-type transistors in series (P), the logic depth (L) and the number of inputs (N) per circuit gate. Matching is a recursive construction process where for each match there is a set of nodes that represent the logic function implemented in that match. Figure 3.6 explains if a match represented in the sample tree, is fed by n nodes there are $2^n - 1$ possible new sub-tree matches. Subtree matches are generated for all possible tree patterns, together with the dual, and are kept in sets according to the output.

Figure 3.7 Example of covering. [Liem]

Covering is solved with dynamic programming. Subcircuits are designed for a node, having for each phase a circuit design. Covering follows a reversed depth-first order towards the root and it determines for each node the match with the best delay from the leaves of the tree. The logic phase for the subcircuit match is determined from the output logic expression of the node. The dual is used by adding input inverters. In Figure 3.7, inverters are added only at the inputs of each match generating more design options from the inverted phase inputs. The final process is an heuristic for fanin optimization where the number of inverters at the boundaries of trees can be reduced.

CHAPTER 4

PROBLEM DEFINITION

This chapter defines the specific objectives sought for the implementation of a prototype technology mapping system.

4-1. TARGET CIRCUITS

The type of logic circuits to be implemented using technology mapping are combinatorial logic circuits and sequential logic circuits. The objective of the technology mapping algorithm is to realize these logic circuits as pipelined data paths with optimized throughput. Pipelined logic design is a design method that introduces registers within the logic of the circuit with no feedback loops. The introduction of registers transforms asynchronous propagation paths into synchronous propagation paths driven by a common clock signal. The clock period is minimized by optimizing the logic circuit implementations between registers.

Performance in combinatorial circuits is measured with the data throughput. The throughput is related only to the clock period (T) since there are no asynchronous feedback loops. The throughput is estimated as $th=1/T$. The technology mapping of combinatorial circuits is aimed to determine an optimum register placement that minimizes the clock period.

Performance in sequential circuits is also measured with the data throughput. However, because of the possibility of synchronous feedback paths, the throughput is related to the clock period (T) and the number of registers (R) along paths from primary inputs to primary outputs. It is estimated as $th=1/(R*T)$. Technology mapping of sequential circuits is aimed to determine an optimum register placement that maximizes the throughput.

4-2. TARGET DESIGN TECHNIQUE.

The objective CMOS dynamic logic technique includes n-type/p-type dynamic logic gates and C²MOS registers. C²MOS registers are used since they are part of the NO-RACE technique. For prototype implementation, a modified NO-RACE technique is selected because it has a stronger research background than TSPC. The C²MOS registers used for the NO-RACE technique have a more general application than registers used in the True-Single-Phase-Clock technique (TSPC technique). The clock phase driving C²MOS registers does not affect the type of input dynamic logic gates, as opposed to TSPC registers that have a constrained n-type/p-type dynamic logic gate sequence with the clock phase.

In addition to the NO-RACE design technique, C²MOS complex gate registers are also supported. C²MOS complex-gate-registers have increased design flexibility and become the merged-NO-RACE design technique. Figure 4.1 describes the objective CMOS dynamic logic design technique.

A design constraint implemented on the target design technique is on the data exchange between registers and is restricted to registers driven by opposite clock phases. The circuit is divided into "tree like" subcircuits that have an even number of registers for all paths. Registers within subcircuits are assigned opposite clock phases that make a compatible latching scheme from outputs to inputs between subcircuits.

Finally, circuit designs with TSPC registers are used in the last step of the mapping process for compensating all paths in the subcircuits to an even number of registers. TSPC registers are used due to their data storage and non-inverting logic properties.

Figure 4.1 Implemented NO-Race design technique for pipelined logic using n-type/p-type dynamic gate structures and C²MOS dynamic registers. Boxed transistors can be replaced by complex logic. Constraints are also specified.

4-3. TARGET MAPPING.

The technology mapping process explores all the feasible circuit designs using a matching and covering process and selects an optimum design using a covering process. The matching process is restricted to "tree like" graph representations of subcircuits. The reason is that the matching for a general graph representation is a far more complex process that should be the object of future research.

The circuit is partitioned into "tree like" subcircuits based on the fanout and from there the technology mapping finds an optimum circuit design. The circuit partitioning process into "tree like" subcircuits transforms combinatorial and sequential logic circuit designs into combinatorial sub-circuits. The technology mapping algorithm with a matching process for "tree like" circuits is a compatible design concept for both types of logic circuits. Mapped subcircuits make locally optimum circuit designs that are joined for a globally optimum design. The objective for a globally optimum design is the highest possible throughput.

4-4. OVERVIEW FOR THE IMPLEMENTED APPROACH.

A logic circuit is implemented to the gate-level by using a representation model and a technology mapping algorithm. The representation model has the mathematical definition of the circuit interconnection using graph theory. The original circuit structure is partitioned by the technology mapping algorithm into substructures represented by graphs that have assigned hierarchy levels. The hierarchy levels represent the subcircuits that are generated by the matching process and realized into a complete circuit using the covering process.

The representation model uses *synchronous network graphs* that define the interconnection of input signals, output signals, logic expressions and registers. The synchronous network graph is defined with sets of vertices, edges and weights. The elements of the vertex set represent the input signals, output signals and logic expressions. The elements of the edges set represent the circuit interconnection between inputs, outputs and logic expressions. And the elements of the weight set represent the number of registers between logic expressions.

The circuit structure is represented with the synchronous network graphs and is partitioned into different hierarchy levels that correspond to the circuit, or parts of it, used at different stages of the technology mapping. The highest hierarchy level for the graph representation is named G. The G graph separates registers from logic expressions and groups together those logic expressions based on the fanout. Figure 4.2 is an example for the grouping process for logic expressions used for the construction of the hierarchy graph level G. Figure 4.3 is the corresponding synchronous network graph at the hierarchy level G where registers are expressed as a weight along edges. The hierarchy graph level G re-arranges the placement of registers to the input level of those vertices with logic expressions.

Synchronous circuit schematic from MCNC's benchmark DAIO.

Figure 4.2 Grouped logic expressions and registers based on the fanout. The dashed lines determine the combinational logic that is being grouped.

Hierarchical Synchronous Network Graph

Figure 4.3 Synchronous Network Graph at the hierarchy level G. The registers are re-timed to the input level of the vertices with logic expressions.

The logic expressions generated at each vertex in the G graph are structures with one output, many inputs and the corresponding set of logic expressions. The inputs at the boundaries of the structure can be shared by more than one logic expression, this type of structures are named "tree like" structures. The 'tree like" structures of the G graph are represented with the next hierarchy graph level named G'. In Figure 4.4 the logic expressions represented by the vertex "VIOL" from the hierarchy level G is presented as a Synchronous Network Graph at the hierarchy graph level G' and its named the G'(viol) network graph. The represented subcircuit by the G' graph is the input structure for the technology mapping algorithm. The technology mapping algorithm determines for this hierarchy level a gate-level design of the circuit that includes logic gates and registers.

a) logic circuit.

b) Synchronous Network Graph at the hierarchy level G'.

Figure 4.4 Synchronous Network Graph at the hierarchy level G'. The dotted subcircuits determine the boundaries of the subgraphs that define the synchronous network graph at the hierarchy level P.

The exploration of the possible register placements is the first circuit partition for technology mapping and is performed with the next hierarchy level of the synchronous network graph named P. The name of the hierarchy graph level P stands for Register Partition Boundaries. The P graph determines with its boundaries the only places where registers can be placed for the gate-level design of the partitioned subcircuit (P graph). The output of a P graph is always generated by the gate-level design of a register and the inputs can optionally place registers. Figure 4.4a is an example of the grouping process to determine the boundaries of the hierarchy graph level P. The grouping process or boundaries exploration process is also shown in Figure 4.4b over the G' graph. The boundaries exploration process for P graphs is a recursive process that determines from a G' graph all the possible P subgraphs.

The last hierarchy level determines from the hierarchy graph level P a subcircuit partition represented by the hierarchy graph level named CG. The hierarchy graph level CG stands for Complex Gate Boundaries and it determines the logic expression implemented with a single logic complex gate and logic inverters. Figure 4.5a is an example of the grouping process to determine the boundaries of the hierarchy graph level CG. The grouping process or boundaries exploration process for the CG graphs is also determined from the P graph as illustrated in Figure 4.5b. The boundaries exploration process for CG graphs is a recursive process and determines all the possible CG subgraphs from a P graph.

The technology mapping algorithm determines the boundaries for all the hierarchy graphs and optimizes the performance of the gate-level designs for the G' subgraphs. The algorithm is a recursive process that determines the boundaries for the hierarchy graph levels. A subject vertex with more than two feeding edges in the G' graph is modified before the boundaries exploration for the P graph may include the subject vertex. The subject vertex is decomposed into a binary substructure where only two input edges per vertex are allowed.

a) Logic circuit.

b) Synchronous Network Graph at the hierarchy level G' .

Figure 4.5 Synchronous Network Graph at the hierarchy level P . The dashed subcircuits determine the boundaries of the subgraphs that define the synchronous network graph at the hierarchy level CG .

The matching process in the technology mapping algorithms is an exhaustive search for all the possible groupings of logic expressions that define the P and CG hierarchy level graphs. The covering process in the technology mapping algorithms is the ending process at each level of the recursive boundaries exploration. The covering process for P graphs is the gate-level design of a P graph realized by selecting a set of gate-level designs from the CG subgraphs in a design that minimizes the propagation delay at the hierarchy level of the P graph. The CG graphs have assigned property lists that give a description of the gate-level designs at the inputs of a subject structure and a description of the gate-level design of the subject structure. The properties maintained by the property lists are the signal phase, gate type (n or p) and the property in the number of inverting gates from the inputs of the P graph

to the desired subject structure (Odd or Even). In Figure 4.6 a possible gate-level design for the CG graphs described Figure 4.5 are linked with compatible property lists. The gate-level design of Figure 4.6 is the result of two types of gate-level design processes where the input signals to the P graph are given a gate-level design for input-logic expressions and the main-body logic expressions are assigned the logic determined by the CG graphs.

Figure 4.6 Gate-level design for a P graph. The input and output property lists allow for a control over the properties of the designed structures at the CG graph level.

The hierarchy graphs at the G' level represent subcircuits that together realize a circuit that allows for the existence of synchronous feedback loops, therefore there is an uncertainty of which number of registers along paths in the G' graph lead to an optimum throughput. This uncertainty can't be solved with the local solutions generated at the G' graph level having to provide for every G' graph a set of gate-level designs with optimum throughput

for different number of registers along critical paths. That way, the covering process for G' graphs is the gate-level design from the P subgraphs in a design that maximizes the throughput for different register weights along the individual critical register weight paths for each G' graph.

The study of the circuit throughput performance for the complete implementation, using the different gate-level designs for the G' graphs, is measured from a final gate-level design. The final gate-level design is implemented from a set of G' gate-level designs into a circuit that has a compatible timing behavior with the original circuit structure using retiming. Retiming is the reallocation process of registers from the original structure to the desired implementation where the external timing behavior is preserved at a certain extent. Figure 4.7 is a possible final gate-level design for the complete circuit that rearranged the original register placement into a structure with a similar timing behavior from the original.

Figure 4.7 Possible gate-level design for the complete circuit. The gate-level designs for the G' graphs are a selection of all the available optimum designs for different register weights along the critical register weight path.

CHAPTER 5

REPRESENTATION MODEL

This chapter presents a new approach for the representation model in technology mapping. The model has three levels of representation. First, the graph-level model represents the circuit topology. Second, the gate-level model has the notation for logic expressions or gates, and the technology dependent properties of a gate-level circuit. Third, the timing-level model defines the measured performance parameters for gate-level circuits.

The following assumptions are required in the representation model to guarantee an appropriate parameter definition. The graph-level model considers, for technology mapping, circuits with combinatorial logic and registers. The timing-model assumes for every register in technology mapping only one clock signal source, generating clock and $\overline{\text{clock}}$. The latching process is considered as edge triggered since in the gate-level model it is assumed for registers either inverting, non-inverting or *complex gate registers* to be dynamic logic registers. The latching process in dynamic logic for *clock* and $\overline{\text{clock}}$ sections is also considered as edge triggered in relation to the clock signal.

5-1. GRAPH-LEVEL MODEL.

5-1.1. Synchronous Network Graph.

The representation model is based on the synchronous network graph presented by Michelli/Klein [MiKl]. The Synchronous Network Graph $G(V_G, E_G, W_G)$ is a directed weighted multigraph with a vertex set $V_G = V_G^i \cup V_G^g \cup V_G^o$ partitioned into input, gate and output vertices that are a one-to-one correspondence with primary inputs, gates and primary outputs. The edges E_G are unidirectional connections between vertices and have assigned a register weight of $W_G \geq 0$ registers.

Only synchronous cycles are allowed. For each cycle, there are $\sum_x W_{G(x)} > 0$ registers, where x are the edges involved in the cycle. In any graph a weight of zero is omitted for simplicity. Vertices have a single output signal with a *fanout of at least one* or a fanout to a primary output, as shown in Figure 5.1b. Vertices correspond to logic expressions relating inputs with output according to the original circuit. Figure 5.1 is an example of the synchronous network graph grouped for the highest application level in the hierarchical model explained in Section 5.1.2.

Figure 5.1 Synchronous network graph representation used for global retiming. Registers are retimed in (b) to the feeding edges of vertices from the set V_G^s .

5-1.2. Hierarchical Synchronous Network Graph.

The hierarchical synchronous network graph model is a new approach for graph-model representation and is designed for technology mapping applications. The model represents the circuit at every stage of a dynamic programming type of approach for the technology mapping problem using cell generators.

The highest graph-level is a special case of the graph model defined by the synchronous network graph in Section 5.1.1. In the hierarchical synchronous network graph vertices have one output signal with *a fanout of more than one* or a fanout to a primary output. The logic expressions associated for such vertices are "tree like" graphs. Figure 5.1b has a valid graph for Synchronous Network Graphs and for Hierarchical Synchronous Network Graphs.

V_G^s Vertices themselves represent synchronous network graphs using the same graph model and are represented by $G'(V_{G'}, E_{G'}, W_{G'})$ with a vertex set $V_{G'} = V_{G'}^i \cup V_{G'}^s \cup V_{G'}^o$, where the vertex set $V_{G'}^o$ has exactly one vertex. Edges $E_{G'}$ are unidirectional data edges with a register weight $W_{G'}$ initially set to zero. Figure 5.2 is an example of the generated G' networks from the G graph in Figure 5.1b.

Register manipulation along edges in G' is an optimization process towards an optimum throughput. The different register positions involve technology mapping of different logic expressions. Register allocation at the output of a logic expression is characterized by a synchronous network graph P . Network graphs P are partitioned from G' networks and define the logic expressions to implement using complex logic gates. The output is forced to have an output register and the inputs are the only place within P where extra registers can be optionally allocated.

Figure 5.2 G' synchronous network graphs representations for gate vertices V_G^S . Each network is used for local retiming and subcircuit partition in the technology mapping process.

The Network graph P is defined as $P(V_P, E_P, W_P)$ with a vertex set $V_P = V_P^i \cup V_P^S \cup V_P^O$. The vertex set V_P^S is a subset from V_G^S . Also, the Vertex set V_P^O has a single vertex fed by one C^2MOS register. Vertices V_P^i can also feed registers and that is determined after the mapping process described in Section 6-2. Gate vertices V_P^S and edges E_P are recursively determined as explained in Section 6-1. W_P is named *weight requirement* to express the required final weight after the retiming process is used, and are dynamically transformed in the technology mapping process. The following rule determines a valid G' network graph from a set of P networks graphs, see Figure 5.3.

Definition: $\cap_x$ and $\cup_x$ are the indexed functions for intersection and union of the arguments. i.e. Given $G' = P(a) \cup P(b) \cup P(c)$ then G' is defined as $\cup_x P(x)$ where $x = (a, b, c)$.

Rule 1: Given a set of P networks $\{ \dots P(x) \dots \}$ each network is defined as

$\{x \mid P(x), \text{ all } V_{P(x)}^0 \text{ have one element, and all } V_{P(x)}^0 \text{ correspond to a different vertex from } V_{G'}^0 \cup V_{G'}^0\}$.

If $\cap_x V_{P(x)}^0 = \text{NIL}$ and $\cup_x V_{P(x)}^0 = V_{G'}^0$ then $\cup_x P(x)$ is a valid mapping for G' .

Figure 5.3 Synchronous network subgraphs from $G'(\text{viol})$ used to search the optimum dynamic logic design. W_p are determined after the technology mapping.

Register placement is only allowed for edges connected to V_p^i and V_p^0 . Therefore, rule 1 sets a fixed register weight requirement $W_p = 0$ for edges feeding from vertices in V_p^0 set to vertices in the same set. Edges connected to vertices V_p^i or V_p^0 have flexible weight requirements $W_p \geq n$, where $n=1$ for every V_p^0 and $n=0$ or 1 for vertices in V_p^i . Flexible weights can only accept registers that cannot modify the logic, for example adding non-inverting registers or an even number of inverting registers.

A set of valid P networks, within G' network, that meet rule 1 are one possible solution for G' network. Sets of P networks with at least two elements, grouped by rule 1, always lead to junctions of weight requirements. Also, the graph model definition forces all P and G' networks to have a single output. Therefore, if more than one P network make a valid G'

network there are P type networks that feed P type networks from the same set. A flexible weight requirement exists at the common edge between P networks and meets weight requirements from both networks. Rule 2 determines weight requirements for edges involved in junctions.

Rule 2: Given a set $\{P(x) P(y) \dots\}$ that builds a complete G' network as defined by Rule 1 and if the following is true: $V_{P(x)}^o \cap V_{P(y)}^i$ is not NIL; then the edge weight requirement for G' at edge z =(edge from vertex $V_{P(x)}^i \cap V_{P(y)}^i$ to vertex $V_{P(x)}^o$) is $W_{G'}(z) \geq W_{P(y)}(z) + 1$.

P networks are the boundaries for register placement and the logic expression inside each P network is technology mapped towards an optimum delay from inputs to output. Technology mapping of P networks selects a set of complex gate subnetwork graphs CG. Complex gate network graphs CG define the limits of a cell gate and correspond to partitioned subnetwork graphs from P networks. Complex gate network graphs are defined as a synchronous network graph $CG(V_{CG}, E_{CG}, W_{CG})$ whose vertex set $V_{CG} = V_{CG}^i \cup V_{CG}^g \cup V_{CG}^o$, is partitioned with input, gate and output vertices. Edges E_{CG} have W_{CG} register weight requirements, and there is one V_{CG}^o . V_{CG} is a subset of V_P . Figure 5.4 defines some of the possible CG subnetworks generated from a P network with $V_P^i = \{A B C D\}$ from $G'(\text{viol})$ in Figure 5.2.

Rule 1 is applied to a set of CG networks determining a valid P network. CG networks are partitions from P networks. Therefore a set of CG subnetworks with more than one element, joined in a P network, have one CG subnetwork at the output and the other CG subnetworks feed only CG networks from the same set. Edges involved in junctions, for CG networks, always stay in zero since registers are only allowed at the boundaries of P networks.

Figure 5.4 Synchronous network graphs representations for complex gate design. Determined from a P network and used for searching optimum dynamic logic complex gate designs.

A unidirectional circuit is identified when a synchronous network graph is acyclic. In the hierarchical synchronous network graph model cycles are present only in G , for global retiming, having in G' , P and CG unidirectional networks.

5-2. GATE-LEVEL MODEL.

CG networks are the graph model representation for a cell gate. Cell gates are joined to form a valid mapping for a P network that meets design constraints for dynamic logic circuit design. Each CG network can be implemented at the gate-level in different circuit designs that have characteristic output properties and required properties for the feeding CG networks. Property lists are defined for gate-level designs of CG networks. The *input property list* define the required properties of the gate-level designs from the feeding CG networks. The *output property list* define the properties available from a certain CG network fed by CG networks that meet properties in the input property list.

Dynamic logic designs require in the property lists a signal-type property (f or $\bar{f}$), gate-type property (n -type or p -type) and inversions-type property (*Odd* or *Even*). The output property list define the available design at CG , where gate-type is a description of the output gate connected to V_{CG}^o . The input property list define the required output property list from the feeding complex gate networks at vertices in V_{CG}^i . Also, CG network graphs have a *gate-level design* property in addition to the logic expression of the network. The gate-level design property is the complex gate circuit design realization from the logic function of CG . Gate-level designs have input signal supply from gate-level design circuits, of complex gate networks, that meet the input property list. Also the output gate design is described by the output property list.

A type of complex gates named *C²MOS complex gate register* is used for C²MOS registers with n-type/p-type logic evaluation structures merged within the registers. Figure 5.5 has the notation used to express this kind of gates.

Figure 5.5 Notation description.

Technology limitation parameters are introduced to avoid layout congestion or unfeasible gates using the SPLN parameters introduced by Liem [Liem]. The number of input vertices V_{CG}^i is limited by the (N) parameter. The maximum number of n-type transistors connected in series is limited by the (S) parameter and for p-type transistors in series by the

(P) parameter. The (L) parameter is defined as the depth limit or the limit on the number of logic levels for complex gate circuits. Parameter (L) is determined as the maximum number of vertices V_{CG}^8 along paths from V_{CG}^i to V_{CG}^0 .

5-3. TIMING-LEVEL MODEL.

Data ready time t_i is the time between a generated signal and the clock edge for a given gate, or along a path through gates, and is used for the technology mapping process described in Chapter 6. Primary inputs are assumed to have a data ready time of zero, or just synchronized with the clock edge. For V_P^8 vertices, data ready time is the accumulated delay from the critical delay path. The *critical delay* is defined as the worst case for the data ready time measured by accumulating evaluation times of gates along paths from input vertices V_P^i to the target vertex V_P^8 . The data ready time of the critical delay path is compared to the worst precharge time from gates in the P network. The worst case sets up the *minimum clock period* t_{min} for the specific P network. All the P subnetworks are guaranteed to have a C²MOS register as output making safe to assume a data ready time for every V_P^i equal to zero.

Data throughput th_i is measured once the P network is mapped and it is defined as the product of the minimum clock period (t_{min}) and the critical register weight (W_{max}). The minimum clock period is estimated from a network $P(x)$ or from the worst t_{min} of networks P feeding $P(x)$ and so on up to the P networks fed by input vertices in G' . The *critical register weight* W_{max} is the worst register weight along paths from edges feeding input vertices in the set $\cup_y \{V_{P(y)}^i \cap V_{G'}^i\}$ to the edge feeding the output vertex in $V_{P(x)}^0$, where y are all the P networks that make a valid circuit feeding $P(x)$.

Delays are estimated from HSPICE simulations using CMOS4S technology parameters, and an arbitrarily selected device width of 10 microns for n devices and 20 microns for p devices. Each gate circuit is simulated to drive an output charge/discharge through the evaluation block with a fixed input switching signal. It is assumed that for a series connection of dynamic complex gates the signal propagation time can be considered the same as the ac-

cumulated switching time from gates without a big error margin. A library of gate circuits related to their loads within a 2% tolerance is maintained to avoid continuous HSPICE simulations. Although this approach is not the best in terms of CPU time consumption, it provides reliable delay information allowing to focus on the technology mapping problem.

CHAPTER 6

TECHNOLOGY MAPPING ALGORITHM FOR NORA

This chapter presents a dynamic programming approach for technology mapping. The algorithm is organized by succession of tasks and it relies on the model representation presented in Chapter 4. Section 6-1 describes the decomposition process applied to vertices with more than two input edges. Hierarchical graph construction for P and CG networks is part of the matching process for technology mapping and is described in Section 6-2. Section 6-3 describes the rules of a dynamic cover process for technology mapping using the NORA design technique. Section 6-4 describes retiming compensation for mapped designs.

6-1. VERTEX DECOMPOSITION.

Network G' is prepared for technology mapping by pushing inverters, using DeMorgan's theorem[McCl], to vertices in $V_{G'}^0$ set connected to vertices in $V_{G'}^1$ set. Vertices have associated simple logic expressions (AND, OR) that are initially merged for similar operators. G' networks are initially minimum depth networks that allow, for vertices in $V_{G'}^0$ set, more than two feeding edges. A binary decomposition is desired for exploring the maximum number of P network designs, where just two feeding edges per vertex are allowed.

Gate vertices in the G' network are binary decomposed before determining boundaries for each P subnetwork partition. The binary decomposition process uses from P networks the measured throughput to compare performance between feeding P networks. Therefore, binary decomposition is a dynamic programming process in terms of the complete technology mapping process. The binary decomposition process modifies the G' network at the vertex involved to be later considered in the $V_{P'}^0$ set. Figure 6.1 is an example of the structure from a sample G' network modified by the vertex decomposition process.

Binary decomposition is a two step process. First, input edges are grouped in sets of two if there is a differential tolerance of 4% between throughputs from inputs of a given vertex (arbitrary tolerance value). Second, input edge sets are regrouped in a binary network of maximum depth, as shown in Figure 6.1b. Binary decomposition provides from $G'(x_i)$ a new network $G'(x_{i+1})$ with the extra vertices as the difference between x_i and x_{i+1} networks. In Figure 6.1c, the logic functions from $G'(x_i)$ and $G'(x_{i+1})$ networks are the same and allow to continue the boundaries positioning for P from $G'(x_{i+1})$.

Figure 6.1 Binary decomposition for vertices. Throughput from vertex i is expressed as th_i .

6-2. DYNAMIC MATCHING: THE TREE-GRAPH CASE.

6-2.1. G' Network Matching: P Sub-Network Boundaries Exploration.

The boundary exploration process for P subnetworks starts with the *Give-Output-Vertex* algorithm using vertices in V_{G^k} to determine the gate vertex in V_{P^k} directly connected to the output vertex in V_{P^0} . Figure 6.2 gives the algorithm for the Give-Output-Vertex algorithm, following a depth-first-search order, over vertices in V_{G^k} . The deepest vertex level in V_{G^k} has the vertices connected to the input vertices in V_{G^i} .

Algorithm Give-Output-Vertex ($G'(x)$, $V_{G^k}(y)$)

Input: $G'(x) = (V_{G'(x)}, E_{G'(x)}, W_{G'(x)})$ (directed weighted graph
weights are included for completeness)

$V_{G^k}(y)$ (gate vertex y from $G'(x)$).

Output: R (a vertex list with gate vertices from set
 V_{G^k} of the $G'(x)$ network)

begin

$R = (V_{G^k}(y))$

for all edges $V_{G^k}(z)$ feeding $V_{G^k}(y)$ **do**

$R = \text{append} (\text{give-output-vertex} (G'(x), V_{G^k}(z)), R)$

return R

end

Figure 6.2 Give-Output-Vertex algorithm. Adapted from the depth-first search algorithm from Manber [Manb].

The output vertex in $V_{P(x)}^0$ set is one vertex from $V_{G^0} \cup V_{G^k}$ that is fed by the selected gate vertex from the give-output-vertex algorithm. A recursive boundary search process determines as many as possible feasible P networks, with the same output vertex, by adding vertices to the gate vertex set V_{P^k} . Input vertices in $V_{P(x)}^i$ are those vertices from $V_{G^i} \cup V_{G^k}$ that feed vertices in $V_{P(x)}^k$. The recursive boundary search adds vertices from V_{G^k} to the vertex set V_{P^k} and is explained by Rule 3. Figure 6.3b is an example of the P networks generated from the G' network in Figure 6.3a.

Rule 3: Given a circuit $P(x)$ from $G'(y)$ with a vertex set $V_{P(x)} = V_{P(x)}^i \cup V_{P(x)}^s \cup V_{P(x)}^o$. There is a set of networks $expand_all = \{P(x+1) \dots P(x+z)\}$ with the following characteristics:

$$\bigcap_m V_{P(m)}^i = \text{NIL for all } m = x \dots x+z,$$

$z = 2^n - 1$ is the number of extra networks that have the same $V_{P(x)}^o$,

n is the number of vertices from the vertex set V_{expand} .

And the set V_{expand} is defined:

The vertex set V_{expand} is a subset of vertices from the inputs vertex set $V_{P(x)}^i$, that have the longest path.

The longest path is determined from vertices in the set $V_{P(x)}^i \cap V_{G'(y)}^s$ along vertices in the set $V_{P(x)}^s$ to the output vertex in the set $V_{P(x)}^o$.

(Rule 3 is also applied for each of the resulting networks in the $expand_all$ set)

a) Sample G' and the generated sequence list from the Give-Output-Vertex Algorithm.

<u>P network vertices</u> <u>$V_P^i / V_P^s / V_P^o$</u>	<u>vertex in V_P^s</u> <u>directly connected to V_P^o</u>
{x1 in} / {A} / {C}	A
{x1 in} / {B} / {C}	B
{A B} / {C} / {D}	C
{x1 in B} / {A C} / {D}	C
{A x1 in} / {B C} / {D}	C
{x1 in} / {A B C} / {D}	C
{C y0} / {D} / {viol}	D
{A B y0} / {C D} / {viol}	D
{x1 in B y0} / {A C D} / {viol}	D
{A x1 in y0} / {B C D} / {viol}	D
{x1 in y0} / {A B C D} / {viol}	D

b) Generated P networks using rule 3.

Figure 6.3 Example for the *give-output-vertex* algorithm.

6-2.2. P Network Matching: CG Network Boundaries Exploration.

The determined boundaries from a network graph P are supplied to the technology mapping algorithm. Technology mapping of P networks generate more than one mapped solutions as described in Section 6-3. Critical register weight and data throughput are the *design parameters* for P network graphs estimated from the objective P network. Both parameters are measured locally in the objective P network and later accumulated to the design parameters from the feeding P networks if $V_{P(x)}^i \not\subset V_G^i$; otherwise the input design parameters are set to zero. The design parameters are obtained by designing all the feeding P networks to the objective $P(x)$ network graph before $P(x)$ itself.

CG boundaries are determined within P network boundaries and use the give-output-vertex algorithm for vertices in the V_P^s set. The algorithm in Figure 6.2 also determines the vertex in V_{CG}^s set directly connected to the vertex in V_{CG}^o . The output vertex in V_{CG}^o set is one vertex from $V_P^o \cup V_P^s$. Rule 3 is applied: for a given $CG(x)$ from $P(y)$ to determine V_{CG}^i and V_{CG}^s vertex sets. Figure 6.4b is an example of the CG networks generated from the P network in Figure 6.4a. The logic expressions from the gate vertices in V_{CG}^s set are designed with complex gates to create all the possible CG complex gate networks according to the technology mapping step described in Section 6-2.3.

Figure 6.4 Sample CG network boundaries exploration.

6-2.3. P Network Matching: CG Network Gate-Level Design Exploration.

Boundaries exploration for CG network matching provides enough information to generate more than one gate-level design implementation for each CG network. The logic expressions in the vertex set V_{CG}^2 are separated in two groups. First the *input logic expressions*, where the required logic phase for inputs are met with gate-level designs. The logic in such gate-level design is an input phase transformation to meet with gate-level designs all the possible property list combinations. Second the *main body logic expressions*, where the logic in the vertex set V_{CG}^2 is implemented with a single gate-level design and extra output inverters to fulfil all the possible mappings from combinations of the input and output property lists.

CG networks are generated to provide a complete construction set with all the possible combinations of the output property list: $(f, n, even)$ (f, n, odd) $(f, p, even)$ (f, p, odd) $(\bar{f}, n, even)$ $(\bar{f}, n, odd)$ $(\bar{f}, p, even)$ $(\bar{f}, p, odd)$. Figure 6.5 provides the generated gate-level designs with the output property list for input-logic-expressions assigned to vertices in V_P^1 set before designing CG network boundaries. Also, designs without gates are included for completeness. Figure 6.6 provides the gate-level designs of the main-body-logic-expressions

and the dual for a given CG. The dual is explained in Section 3-1.4 and in the Glossary. The output is supplied by dynamic complex gates since vertex $V_{CG}^o \neq V_P^o$.

Input-logic-expressions are implemented using gate-level designs with a single input and a single output. The available input designs for such functions can have either f or $\bar{f}$ type if $V_P^i \subset V_{CG}^i$. In other words, if the input signal of a vertex in V_P^i set comes from a P network there are two design types available f and $\bar{f}$, where only one is used. Complex gate circuits with their property lists provide the required logic for each input vertex V_P^i . (see Figures 6.5 and 5.4). The dual logic expression is used to increase design flexibility of main-body-logic-expressions by expanding the number of gate designs for each CG network. The dual logic expression is obtained by the application of DeMorgan's theorem to find an opposite signal phase in the same boundaries context of CG fed by opposite phase signals. Property lists allow for a correct logic operation of gate-level designs from dual logic expressions.

V_{CG}^i vertex's required Logic expression:	CG gate-level design	($f \mid \bar{f}, n \mid p, \text{Odd} \mid \text{Even}$) Output property List
$f = \text{in}$		$\{ (f, n, e), (f, p, e) \}$ $\{ (f, n, o), (f, p, o) \}$ $\{ (\bar{f}, n, e), (\bar{f}, p, e) \}$ $\{ (\bar{f}, n, o), (\bar{f}, p, o) \}$
$f = (! \text{ in})$		$\{ (f, n, e), (f, p, e) \}$ $\{ (f, n, o), (f, p, o) \}$ $\{ (\bar{f}, n, e), (\bar{f}, p, e) \}$ $\{ (\bar{f}, n, o), (\bar{f}, p, o) \}$

Figure 6.5 CG gate-level design with output property list for input logic functions. The gate-level design is performed for input vertices that meet $V_{CG}^i(x) = V_P^i(x)$

Figure 6.6 CG gate-level design with property lists for main body logic functions. The gate-level design is performed for gate vertices that meet $V_{CG}^o(x) \neq V_P^o(y)$

Property lists indicate the correct input properties supplied to obtain the desired output properties for each possible design. Figure 6.7 provides the matches and property lists when $V_{CG}^o = V_P^o$, where the output is always provided by a C^2 MOS register. Gate type is an irrelevant design information when $V_{CG}^o(x) = V_P^o(x)$ since the output is a C^2 MOS register. Figure 6.7b provides the property list without using complex gate designs inside C^2 MOS registers. Some designs are not included in the complex gate register matches of Figure 6.7a since no special buffering algorithm is used for this implementation, although all designs in Figure 6.7 should be considered.

Figure 6.7 CG gate-level design with property lists for main body logic functions. The gate-level design is performed for gate vertices that meet $V_{CG}^o(x) = V_P^o(x)$.

6-3. DYNAMIC COVERING.

6-3.1. P Network Covering.

The matching process for P networks show that more than one design is available for each of the output property list combinations. P network covering uses the *property lists* for gate-level designs from CG networks in Figures 6.6 and 6.7, and the property lists for input vertices in V_P^i set from Figure 6.5.

A *local selection process for P networks* selects from the matched CG networks one set of gate-level designs. The selected gate-level designs correspond to different combinations of the output property list. The local selection process is the last step of the matching process and is the first step for the covering process for P networks.

The local selection process makes comparisons with the data ready time and selects for each vertex in the set $V_P^i \cup V_P^s$ one gate-level design for each of the output property list combinations. Figure 6.8b is an example from the local selection process performed after the match process for P networks.

The *covering process for P networks* is the gate-level design selection performed by the local selection process when vertices in $V_{CG}^o = V_P^o$. Property types f and $\bar{f}$ are the only types required when vertices $V_{CG}^o = V_P^o$, because that gate-level design is always a register. Gate-level designs are linked with the property lists generating for the covering process a single gate-level design for f and one design for $\bar{f}$.

a) P(C) sub-network graph from G' (viol) network graph

Gate-level designs at $V_{CG}^0 = C$. (*italic-bold* for local selection results)

$A = (! (* x1 \text{ in}))$	$A = (! CG_1); CG_1 = (! (* x1 \text{ in}))$
$CG.o(f,n,e).i(f,p,o) \text{ } t_{min} = 0.7667n$	$CG.o(f,p,e).i(f,p,e) \text{ } t_{min} = 0.840n$
$CG.o(f,n,o).i(f,p,e) \text{ } t_{min} = 0.375n$	$CG.o(f,p,o).i(f,p,o) \text{ } t_{min} = 1.23n$
$CG.o(f,p,e).i(f,n,o) \text{ } t_{min} = 0.822n$	$CG.o(f,n,e).i(f,n,e) \text{ } t_{min} = 0.76n$
$CG.o(f,p,o).i(f,n,e) \text{ } t_{min} = 0.25n$	$CG.o(f,n,o).i(f,n,o) \text{ } t_{min} = 1.058n$
$A = (! (+ x1 \text{ in}))$	$A = (! CG_2); CG_2 = (! (+ x1 \text{ in}))$
$CG.o(f,n,e).i(f,p,o) \text{ } t_{min} = 0.689n$	$CG.o(f,p,e).i(f,p,e) \text{ } t_{min} = 0.763n$
$CG.o(f,n,o).i(f,p,e) \text{ } t_{min} = 0.297n$	$CG.o(f,p,o).i(f,p,o) \text{ } t_{min} = 1.154n$
$CG.o(f,p,e).i(f,n,o) \text{ } t_{min} = 1.009n$	$CG.o(f,n,e).i(f,n,e) \text{ } t_{min} = 0.896n$
$CG.o(f,p,o).i(f,n,e) \text{ } t_{min} = 0.712n$	$CG.o(f,n,o).i(f,n,o) \text{ } t_{min} = 1.193n$

b) Local selection results when $V_{CG}^0 = \{C\}$ Figure 6.8 Sample local selection for P network graphs. Data ready time is given as t_i

6-3.2. G' Network Covering.

Gate-level designs from the dynamic matching process are designs without restrictions to the increased throughput, created by placing input registers. G' network covering is a gate-level design selection from P networks based on throughput. The objective for the G' network covering is to select the gate-level designs of P networks that realize a design with an optimum throughput. The G' network covering determines many optimum designs for different critical register weights.

The mapping process described in Section 6-3.1 has a *mixed register allocation* type of mapping, since some inputs may have registers and others may not. *Forced register allocation* type of mappings are two extra sets of P networks matched and covered that use if feasible P network matches from Figure 6.5 without input registers, and matches with only input registers. Mappings of the mixed-register-allocation type are compared with mappings of the forced type. The comparison selects those gate-level designs with a minimum data-ready-time along the critical register weight path.

The covering process in Section 6-3.1 is considered the matching step for G' , where the covering process for G' is a selection of gate-level designs for P networks. A *local selection process for G'* chooses one gate-level design of the P network for different critical register weights.

The *covering process for G' network* is the gate-level design selection performed by the local selection process when vertices $V_P^0 = V_G^0$. The covering process for G' network selects a set of gate-level designs with f -type output, where each element in the set of gate-level designs has an optimum throughput for different critical register weights. In Figure 6.9 gate-level designs are linked with the critical register weight (W_{max}), along the critical register weight path, and the output phase. Also in Figure 6.9, the covering process of the G' (viol) network is the result from the local selection generating a gate-level design set for f -type output and a set for $\bar{f}$ -type output when vertices $V_P^0 = V_G^0 = \{D\}$.

Figure 6.9 Sample results of local selection process for G' (viol).

6-3.3. Limiting Heuristics.

Limiting heuristics are included to save CPU time consumption once technology limitations are known. The number of transistor level gates and size of the critical register weight in CG and P networks are limited by *network limitation parameters*. The number of transistor-level gates, excluding complex gate registers, in either CG or P networks are (c) for lower limit and (C) for upper limit. And the parameters for the number of registers along paths in G' networks are (g) for lower limit and (G) for upper limit.

Parameter (c) takes values from zero to (C) if complex gate registers are allowed; otherwise parameter (c) takes values from one to (C). Lower values for (c) may not provide enough information for the G' decomposition process explained in Section 5-1 and also can give NIL design sets for P, such network boundaries are invalid. In parameter (C) a value of zero forces the complete logic expression inside C²MOS registers and a higher limit allows for more flexibility with *n*-type and *p*-type dynamic gates. The boundaries design for CG verifies for (c) and (C) parameter limits eliminating designs out from range [c C]. Parameters (c C) force the local covering for CG with fewer circuits to compare providing a suboptimal design for a complete search, but an optimal design for range [c C].

Network G' should have at least one set of P networks that make a valid G' circuit design. Also, the G' decomposition process needs at least one P network for each gate vertex in $V_{G'}$; therefore parameter (g) gets a value of at least two, since P networks may need one or two registers along paths. CPU savings come from parameter (G) at the local selection and matching of G', where as CPU savings from parameter (g) are effective just for the covering process of G' networks.

6-4. RETIMING COMPENSATION.

The obtained gate-level designs from the technology mapping algorithm are structures that have modified the original timing behavior. The available G' gate-level design sets provide a number of options for different critical register weights. Therefore, timing is different between gate-level designs. A greedy selection of a single gate-level design for each G' network selects the circuit with the best throughput performance. A final circuit design for G requires a compatible timing behavior between G' circuits, and that is met using *retiming*.

Retiming is a register manipulation technique towards an optimum throughput that was developed by Lieserson [LiRo]. The retiming process assumes a fixed circuit design for all register allocations and optimizes the clock period while keeping the external timing behavior. Retiming also introduces the slow-down/speed-up concept. In the slow-down/speed-up process the total number of registers for the circuit is increased/decreased by a positive integer product/division factor that results in integer number of registers between gates. The slow-down/speed-up process modifies the data throughput but not the external timing behavior.

In this thesis, retiming is only used as a register manipulation process since the clock period is optimized for different register allocations. The register manipulation process is defined as: if r registers are added to the inputs r registers are taken from the outputs always keeping the weight greater or equal to zero for any edge.

Retiming is carried out in two phases. The *global retiming* modifies the original register allocation in the G network graph to supply at the inputs of gates vertices, or G' networks, the required critical register weight in a compatible phase scheme. The *local retiming* distributes the register weights at the inputs, supplied by the global retiming, along edges to meet all the weight requirements of the edges in a distribution that minimizes the total count of registers.

6-4.1. Global retiming.

Mapped circuits of G' networks have weight requirements for the proposed mapping. G' is assigned a *flattened network* design by including all CG networks inside P , and including all P networks inside G' . Junctions between edges of CG networks have no effect since all joined edges have a register weight (W_{CG}) set to zero, and junctions for P networks are handled with Rule 2. Weight requirements from paths determine the critical register weight W_{max} which is increased to the next upper even number for phase compensation during global retiming. The NORA technique imposes a clocking scheme for phase compatibility between clock sections by alternating the clock phase assignment. There is no special heuristic for phase compatibility between gate-level designs in G' networks other than making for G' networks all inputs part of a $\overline{clock}$ section, and outputs part of a *clock* section. This clocking scheme is met with the forced W_{max} weights to the next even number.

The global retiming is solved using integer optimization algorithms. Figure 6.10 gives the global retiming problem formulated as an integer programming optimization problem. The objective is to minimize the total number of registers in the circuit since that also minimizes the clock signal load. Critical register weight is increased to the closest even number, and is assigned to constants CW . The total number of registers for an edge is estimated from the initial number of registers increased by the slow-down factor plus those registers re-arranged using register manipulations.

Variables:

T_j integer total number of registers at edge j
 r_j integer number of register transfer at vertex j
 SD integer slow-down factor

Constants:

CW_j positive even integer, critical register weight for network G_j'
 increased from W_{max} to the closest even number.
 W_k positive integer, minimum register weight from
 edge k to G' output

(a) Definitions.**Objective:**

$$\text{MIN } z = \sum T_j \quad (\text{for all edges } j)$$

Equations:

$$T_k = r_{vin} - r_{vout} + W_k * SD \quad (\text{for all edges } k \text{ with data flow from vertex } vout \text{ to vertex } vin)$$

$$T_k \geq CW_j \quad (\text{required register weight for all feeding edges } k \text{ to network } G_j')$$

$$r_j \geq 0 \quad (\text{for all } j)$$

Freedom constraints:**(implemented constraints)**

$$r_{vi} = r_{vj} \quad (\text{for all } i \text{ and } j \in V^i, \text{ all primary inputs have equal source/sink capacity})$$

$$r_{vi} = r_{vj} \quad (\text{for all } i \text{ and } j \in V^o, \text{ all primary outputs have equal source/sink capacity})$$

$$SD \geq 1 \quad (\text{if slow-down is not allowed the } SD=1)$$

(b) Global retiming formulated as an integer optimization problem.

Figure 6.10 Global retiming problem formulated as an integer programming optimization.

Global retiming is given *freedom constraints* that relies on the application itself. The addition of more freedom constraints to the global retiming optimization would not allow for slow-down or the assumption of primary signals as infinite register sources or sinks. Less freedom constraints allow for slow-down and the use of primary signals as infinite sinks and sources, up to the degree where the primary signals do not have to source or sink the same amount of registers. Designs with many freedom constraints are more likely to result with unfeasible solutions. Figure 6.11a is the G network graph before retiming with CW constants

or critical register weights. Figure 6.11b is the G network graph modified after the global retiming process.

$$\begin{array}{lll} CW_{x1}=0 & CW_{y0}=2 & CW_{y4}=2 \\ CW_{x2}=0 & CW_{viol}=2 & CW_{sample}=2 \end{array}$$

a) G circuit before global retiming

b) G circuit after global retiming

Figure 6.11 Sample global retiming for benchmark circuit DA10.

6-4.2. Local retiming.

The local retiming process for the G network is a register manipulation of G' networks using the supplied registers at the inputs of vertices in the G network. The retiming is solved using an integer optimization. Figure 6.12 gives the local retiming problem, for each G' network, formulated as an integer programming optimization problem. The objective is to meet all the gate-level design weight requirements and have a total number of registers minimized. Local retiming uses edge totals T_j from the global retiming as the register supply from vertices in V_{G^i} .

In Figure 6.13b the local retiming process uses dynamic non-inverting registers for the purpose of phase compensation. Figures 2.4 and 2.5 provide a set of non-inverting True-Single-Phase-Clock registers. There is no heuristic for a selection of non-inverting registers.

Variables:

r_j integer number of register transfer at vertex j

LT_j integer, local total number of registers at edge j

Constants:

T_j results from global retiming, only for edges in contact with vertices in $V_{G^i} \cup V_{G^o}$.

a) Definitions.

Objective:

$$\text{MIN } z = \sum LT_j \quad (\text{for all edges } j)$$

Equations:

$$LT_j = r_{vin} - r_{vout} \quad (\text{for all vertices } j \text{ with data flow from vertex } vout \text{ to vertex } vin)$$

$$r_{vj} = -T_j \quad (\text{for } T_j \text{ results from global retiming, vertex } j \text{ from set } V_{G^i})$$

$$r_{vj} = 0 \quad (\text{vertex } j \text{ from set } V_{G^o})$$

(register re-arrange is forced to flow only from inputs to output)

$$LT_j = 0 \text{ or } LT_j \geq n \quad (\text{according to mapping requirements})$$

b) G' Local retiming solved as integer optimization.

Figure 6.12 Local retiming optimization equations.

b) G circuit after local retiming.
Figure 6.13 Sample local retiming for benchmark circuit DA10.

CHAPTER 7

EXAMPLE

This chapter explains the technology mapping process for the MCNC benchmark circuit DAIO. The mapping process is presented as a concept having many circuit gates generated more than once. Also, mixed register allocation and forced mappings triplicate the mapping efforts and are only included in the results for performance comparisons.

7-1. CIRCUIT DECOMPOSITION INTO G' NETWORKS.

Figure 7.1a shows the schematic for the benchmark circuit used in this chapter. The circuit is represented with a weighted hierarchical synchronous network graph. Figure 7.1b is the G network graph representation for the benchmark circuit with all registers retimed to the inputs of vertices. The fanout limits the logic expressions inside the vertices to "tree like" structures.

Figure 7.1c has the synchronous network graphs for each vertex in the G graph. Vertices in the G' graphs correspond to simple logic expressions. Inverting expressions are moved to the input level by application of DeMorgan's theorem [McCl]. This way gate vertices have non-inverting logic expressions (AND, OR) except for those connected to input vertices that allow for an inverted input phase. The example has to be clear and well related to the mapping concept; for this reason only G'(viol) is fully extended to all the P and CG networks in Section 7-2.

a) Synchronous circuit schematic from MCNC benchmark DAIO.

$$V_G = \{ in \}; V_G^* = \{ x1 \ x2 \ y0 \ viol \ y4 \ sample \}; V_G^* = \{ sample_o \ viol_o \}$$

b) Synchronous network graph G from schematic. With enough detail for global retiming.

c) G' Synchronous Network Graphs.

Figure 7.1 Hierarchical Synchronous network graph representation.

7-2. DYNAMIC MATCHING: P AND CG GRAPH BOUNDARIES.

V_P^B gate vertex sets are the result from the application of the give-output-vertex algorithm and a recursive application of rule 3. The give-output-vertex algorithm is applied to $G'(viol)$ with a gate vertex set $V_G^B = \{A B C D\}$, and it returns the vertex list $(A B C D)$. The vertex list $(A B C D)$ corresponds to the gate vertices in V_P^B connected to the output vertex in V_P^O . The gate vertex list is expanded to all the possible partitions according to the recursive construction from Rule 3 determining new sets of input vertices.

Each network P is explored for all the possible CG complex gate network boundaries using the give-output-vertex algorithm over V_P^B and expanded with the recursive construction from Rule 3. Figure 7.2 shows all the generated P and CG networks for the network graph $G'(viol)$.

7-3. LOCAL SELECTION FOR P NETWORK GRAPH.

A few CG complex gate networks have input vertices that also correspond to the input vertices in the P network graph. This input vertices in V_{CG}^i generate the appropriate set of gate-level designs for all the possible combinations in the output property list. Figure 7.3a shows an arbitrarily selected P network graph with all the generated CG networks. The required gate-level designs for the input logic expressions are shown in Figure 7.3b. Gate-level designs for the input logic expressions are generated using the rules from Figures 6.4 and 6.5. Gate-level designs from vertex $V_{CG}^i(B)$ also indicate an input property list, because the local selection process selects for that signal one of the two available designs of opposite phase.

The match generation process is shown in Figures 7.4a and 7.4b for the P network where $V_P^B = \{A C D\}$ and follows a depth-first-search order $(A C D)$. In Figures 7.4a and 7.4b are shown, in italic-bold case, the results from the local selection process based on the data ready time (t_{min}) that are used to generate a complete set for the output property list.

<u>V_P^s vertex connected to V_P^o from</u>	<u>P network vertices $V_P^1 / V_P^s / V_P^o$</u>	<u>CG network vertices $V_{CG}^1 / V_{CG}^s / V_{CG}^o$</u>	<u>Logic function</u>
A	{x1 in} / {A} / {C}	{x1 in} / {A} / {C}	A=(* (! x1) (! in))
B	{x1 in} / {B} / {C}	{x1 in} / {B} / {C}	B=(* x1 in)
C	{A B} / {C} / {D}	{A B} / {C} / {D}	C=(+ A B)
	{x1 in B} / {A C} / {D}	{x1 in} / {A} / {C}	A=(* (! x1) (! in))
		{A B} / {C} / {D}	C=(+ A B)
	{A x1 in} / {B C} / {D}	{x1 in B} / {A C} / {D}	C=(+ (* (! x1) (! in)) B)
		{x1 in} / {B} / {C}	B=(* x1 in)
		{A B} / {C} / {D}	C=(+ A B)
		{A x1 in} / {B C} / {D}	C=(+ A (* x1 in))
	{x1 in} / {A B C} / {D}	{x1 in} / {A} / {C}	A=(* (! x1) (! in))
		{x1 in} / {B} / {C}	B=(* x1 in)
		{A B} / {C} / {D}	C=(+ A B)
		{x1 in B} / {A C} / {D}	C=(+ (* (! x1) (! in)) B)
		{A x1 in} / {B C} / {D}	C=(+ A (* x1 in))
D		{x1 in} / {A B C} / {D}	C=(+ (* (! x1) (! in)) (* x1 in))
	{C y0} / {D} / {viol}	{C y0} / {D} / {viol}	D=(* C y0)
	{A B y0} / {C D} / {viol}	{A B} / {C} / {D}	C=(+ A B)
		{C y0} / {D} / {viol}	D=(* C y0)
	<i>{x1 in B y0} / {A C D} / {viol}</i>	<i>{A B y0} / {C D} / {viol}</i>	<i>D=(* (+ A B) y0)</i>
		<i>{x1 in} / {A} / {C}</i>	<i>A=(* (! x1) (! in))</i>
		<i>{A B} / {C} / {D}</i>	<i>C=(+ A B)</i>
		<i>{x1 in B} / {A C} / {D}</i>	<i>C=(+ (* (! x1) (! in)) B)</i>
		<i>{C y0} / {D} / {viol}</i>	<i>D=(* C y0)</i>
		<i>{A B y0} / {C D} / {viol}</i>	<i>D=(* (+ A B) y0)</i>
		<i>{x1 in B y0} / {A C D} / {viol}</i>	<i>D=(* (+ (* (! x1) (! in)) B) y0)</i>
	{A x1 in y0} / {B C D} / {viol}	{x1 in} / {B} / {C}	B=(* x1 in)
		{A B} / {C} / {D}	C=(+ A B)
		{A x1 in} / {B C} / {D}	C=(+ A (* x1 in))
		{C y0} / {D} / {viol}	D=(* C y0)
		{A B y0} / {C D} / {viol}	D=(* (+ A B) y0)
		{A x1 in y0} / {B C D} / {viol}	D=(* (+ A (* x1 in)) y0)
	{x1 in y0} / {A B C D} / {viol}	{x1 in} / {A} / {C}	A=(* (! x1) (! in))
		{x1 in} / {B} / {C}	B=(* x1 in)
		{A B} / {C} / {D}	C=(+ A B)
		{x1 in B} / {A C} / {D}	C=(+ (* (! x1) (! in)) B)
		{A x1 in} / {B C} / {D}	C=(+ A (* x1 in))
		{x1 in} / {A B C} / {D}	C=(+ (* (! x1) (! in)) (* x1 in))
		{C y0} / {D} / {viol}	D=(* C y0)
		{A B y0} / {C D} / {viol}	D=(* (+ A B) y0)
		<i>{x1 in B y0} / {A C D} / {viol}</i>	<i>D=(* (+ (* (! x1) (! in)) B) y0)</i>
		<i>{A x1 in y0} / {B C D} / {viol}</i>	<i>D=(* (+ A (* x1 in)) y0)</i>
		<i>{x1 in y0} / {A B C D} / {viol}</i>	<i>D=(* (+ (* (! x1) (! in)) (* x1 in)) y0)</i>

Figure 7.2 P and CG partitioned networks generated from $G'(viol)$. *Italic-bold* case is used for the P and CG networks mapped in Figures 7.3 and 7.4.

a) P and CG networks extracted from Fig.6.1.

at $V_{CG}^i(x1)$:		at $V_{CG}^i(B)$:	
$CG.o(f,n,o)$:	(! x1)	$CG.o(f,n,o).i(\tilde{f})$:	(! B)
$CG.o(f,n,e)$:	(! (@ x1 clock))	$CG.o(f,n,e).i(\tilde{f})$:	B
$CG.o(\tilde{f},n,o)$:	(! I_1); I_1=(! (@ x1 clock))	$CG.o(\tilde{f},n,o).i(\tilde{f})$:	(! I_5); I_5=(! (@ B clock))
$CG.o(\tilde{f},n,e)$:	x1	$CG.o(\tilde{f},n,e).i(\tilde{f})$:	B
$CG.o(f,p,o)$:	(! x1)	$CG.o(f,p,o).i(\tilde{f})$:	(! B)
$CG.o(f,p,e)$:	(! (@ x1 clock))	$CG.o(f,p,e).i(\tilde{f})$:	B
$CG.o(\tilde{f},p,o)$:	(! I_2); I_2=(! (@ x1 clock))	$CG.o(\tilde{f},p,o).i(\tilde{f})$:	(! I_6); I_6=(! (@ B clock))
$CG.o(\tilde{f},p,e)$:	x1	$CG.o(\tilde{f},p,e).i(\tilde{f})$:	B
at $V_{CG}^i(in)$:		at $V_{CG}^i(y0)$:	
$CG.o(f,n,o)$:	(! in)	$CG.o(f,n,o)$:	(! I_7); I_7=(! (@ y0 clock))
$CG.o(f,n,e)$:	(! (@ in clock))	$CG.o(f,n,e)$:	y0
$CG.o(\tilde{f},n,o)$:	(! I_3); I_3=(! (@ in clock))	$CG.o(\tilde{f},n,o)$:	(! y0)
$CG.o(\tilde{f},n,e)$:	in	$CG.o(\tilde{f},n,e)$:	(! (@ y0 clock))
$CG.o(f,p,o)$:	(! in)	$CG.o(f,p,o)$:	(! I_8); I_8=(! (@ y0 clock))
$CG.o(f,p,e)$:	(! (@ in clock))	$CG.o(f,p,e)$:	y0
$CG.o(\tilde{f},p,o)$:	(! I_4); I_4=(! (@ in clock))	$CG.o(\tilde{f},p,o)$:	(! y0)
$CG.o(\tilde{f},p,e)$:	in	$CG.o(\tilde{f},p,e)$:	(! (@ y0 clock))

b) Gate-level designs at $V_{CG}^i = V_P^j$ for each possible property list. Performance selection is made when more than one gate-level design is generated. This is $V_{CG}^i = V_P^j \in V_G^*$, i.e.: $V_{CG}^i(B)$. Minimum clock period t_{min} is specified.

Figure 7.3 Generated input gate-level designs for CG networks of P where $V_P^j = \{A\ C\ D\}$. Input property type f and $\tilde{f}$ is indicated for vertex $V_{CG}^i(B)$ since there are two possible designs from the feeding P(B) network

Gate-level designs at $V_{CC}=C$. (*italic-bold* for local selection process for P sub-network results)

<p>A=(! (* x1 in))</p> <p><i>CG.o(f,n,e).i(f,p,o) $t_{min}=0.7667n$</i></p> <p><i>CG.o(f,n,o).i(f,p,e) $t_{min}=0.375n$</i></p> <p><i>CG.o(f,p,e).i(f,n,o) $t_{min}=0.822n$</i></p> <p><i>CG.o(f,p,o).i(f,n,e) $t_{min}=0.525n$</i></p> <p>A=(! (+ x1 in))</p> <p><i>CG.o(f,n,e).i(f,p,o) $t_{min}=0.689n$</i></p> <p><i>CG.o(f,n,o).i(f,p,e) $t_{min}=0.297n$</i></p> <p><i>CG.o(f,p,e).i(f,n,o) $t_{min}=1.009n$</i></p> <p><i>CG.o(f,p,o).i(f,n,e) $t_{min}=0.712n$</i></p>	<p>A=(! CG_1); CG_1=(! (* x1 in))</p> <p><i>CG.o(f,p,e).i(f,p,e) $t_{min}=0.840n$</i></p> <p><i>CG.o(f,p,o).i(f,p,o) $t_{min}=1.23n$</i></p> <p><i>CG.o(f,n,e).i(f,n,e) $t_{min}=0.76n$</i></p> <p><i>CG.o(f,n,o).i(f,n,o) $t_{min}=1.058n$</i></p> <p>A=(! CG_2); CG_2=(! (+ x1 in))</p> <p><i>CG.o(f,p,e).i(f,p,e) $t_{min}=0.763n$</i></p> <p><i>CG.o(f,p,o).i(f,p,o) $t_{min}=1.154n$</i></p> <p><i>CG.o(f,n,e).i(f,n,e) $t_{min}=0.896n$</i></p> <p><i>CG.o(f,n,o).i(f,n,o) $t_{min}=1.193n$</i></p>
--	---

Gate-level designs at $V_{CC}=D$. (*italic-bold* for local selection process for P sub-network results)

<p>C=(! (+ A B))</p> <p><i>CG.o(f,n,e).i(f,p,o) $t_{min}=0.935n$</i></p> <p><i>CG.o(f,n,o).i(f,p,e) $t_{min}=1.103n$</i></p> <p><i>CG.o(f,p,e).i(f,n,o) $t_{min}=1.126n$</i></p> <p><i>CG.o(f,p,o).i(f,n,e) $t_{min}=1.517n$</i></p> <p>C=(! (* A B))</p> <p><i>CG.o(f,n,e).i(f,p,o) $t_{min}=0.917n$</i></p> <p><i>CG.o(f,n,o).i(f,p,e) $t_{min}=1.140n$</i></p> <p><i>CG.o(f,p,e).i(f,n,o) $t_{min}=0.948n$</i></p> <p><i>CG.o(f,p,o).i(f,n,e) $t_{min}=1.340n$</i></p> <p>C=(! CG_3); CG_3=(! (+ A B))</p> <p><i>CG.o(f,p,e).i(f,p,e) $t_{min}=1.610n$</i></p> <p><i>CG.o(f,p,o).i(f,p,o) $t_{min}=1.440n$</i></p> <p><i>CG.o(f,n,e).i(f,n,e) $t_{min}=1.632n$</i></p> <p><i>CG.o(f,n,o).i(f,n,o) $t_{min}=1.240n$</i></p> <p>C=(! CG_4); CG_4=(! (* A B))</p> <p><i>CG.o(f,p,e).i(f,p,e) $t_{min}=1.610n$</i></p> <p><i>CG.o(f,p,o).i(f,p,o) $t_{min}=1.385n$</i></p> <p><i>CG.o(f,n,e).i(f,n,e) $t_{min}=1.570n$</i></p> <p><i>CG.o(f,n,o).i(f,n,o) $t_{min}=1.180n$</i></p>	<p>C=(! (+ (* x1 in) B))</p> <p><i>CG.o(f,n,e).i(f,p,o) $t_{min}=0.763n$</i></p> <p><i>CG.o(f,n,o).i(f,p,e) $t_{min}=0.372n$</i></p> <p><i>CG.o(f,p,e).i(f,n,o) $t_{min}=1.103n$</i></p> <p><i>CG.o(f,p,o).i(f,n,e) $t_{min}=0.806n$</i></p> <p>C=(! (* (+ x1 in) B))</p> <p><i>CG.o(f,n,e).i(f,p,o) $t_{min}=0.828n$</i></p> <p><i>CG.o(f,n,o).i(f,p,e) $t_{min}=0.437n$</i></p> <p><i>CG.o(f,p,e).i(f,n,o) $t_{min}=1.002n$</i></p> <p><i>CG.o(f,p,o).i(f,n,e) $t_{min}=0.705n$</i></p> <p>C=(! CG_5); CG_5=(! (+ (* x1 in) B))</p> <p><i>CG.o(f,p,e).i(f,p,e) $t_{min}=0.882n$</i></p> <p><i>CG.o(f,p,o).i(f,p,o) $t_{min}=1.273n$</i></p> <p><i>CG.o(f,n,e).i(f,n,e) $t_{min}=0.966n$</i></p> <p><i>CG.o(f,n,o).i(f,n,o) $t_{min}=1.263n$</i></p> <p>C=(! CG_6); CG_6=(! (* (+ x1 in) B))</p> <p><i>CG.o(f,p,e).i(f,p,e) $t_{min}=0.904n$</i></p> <p><i>CG.o(f,p,o).i(f,p,o) $t_{min}=1.295n$</i></p> <p><i>CG.o(f,n,e).i(f,n,e) $t_{min}=0.526n$</i></p> <p><i>CG.o(f,n,o).i(f,n,o) $t_{min}=1.223n$</i></p>
---	---

Figure 7.4a Generated gate-level designs for CG networks of P where $V_P=\{A\ C\ D\}$. x1 and in are references to the nodes. Property lists are references to the signals. Minimum clock period is specified

Gate-level designs at $V_{CC}^0 = \text{viol.}$ (*italic-bold* for local selection process for P sub-network results)

```

D=(! (@ (* C y0)))
  CG.o(f,n,e).i(f,p,e)  $t_{min}=1.519n$ 
  CG.o(f,p,e).i(f,n,e)  $t_{min}=1.441n$ 
D=(! (@ (+ C y0)))
  CG.o(f,n,e).i(f,p,e)  $t_{min}=1.320n$ 
  CG.o(f,p,e).i(f,n,e)  $t_{min}=2.089n$ 
D=(! (@ (* (+ A B) y0)))
  CG.o(f,n,e).i(f,p,e)  $t_{min}=1.501n$ 
  CG.o(f,p,e).i(f,n,e)  $t_{min}=1.459n$ 
D=(! (@ (+ (* A B) y0)))
  CG.o(f,n,e).i(f,p,e)  $t_{min}=1.258n$ 
  CG.o(f,p,e).i(f,n,e)  $t_{min}=2.051n$ 
D=(! (@ (* (+ (* x1 in) B) y0)))
  CG.o(f,n,e).i(f,p,e)  $t_{min}=0.699n$ 
  CG.o(f,p,e).i(f,n,e)  $t_{min}=0.699n$ 
D=(! (@ (+ (* (+ x1 in) B) y0)))
  CG.o(f,n,e).i(f,p,e)  $t_{min}=1.366n$ 
  CG.o(f,p,e).i(f,n,e)  $t_{min}=1.366n$ 
D=(! (@ CG_7 clock)); CG_7=(! (* C y0))
  CG.o(f,p,e).i(f,p,o)  $t_{min}=1.826n$ 
  CG.o(f,n,e).i(f,n,o)  $t_{min}=1.373n$ 
D=(! (@ CG_8 clock)); CG_8=(! (+ C y0))
  CG.o(f,p,e).i(f,p,o)  $t_{min}=1.749n$ 
  CG.o(f,n,e).i(f,n,o)  $t_{min}=1.606n$ 
D=(! (@ CG_9 clock)); CG_9=(! (* (+ A B) y0))
  CG.o(f,p,e).i(f,p,o)  $t_{min}=1.860n$ 
  CG.o(f,n,e).i(f,n,o)  $t_{min}=1.444n$ 
D=(! (@ CG_10 clock)); CG_10=(! (+ (* A B) y0))
  CG.o(f,p,e).i(f,p,o)  $t_{min}=1.626n$ 
  CG.o(f,n,e).i(f,n,o)  $t_{min}=1.748n$ 
D=(! (@ CG_11 clock)); CG_11=(! (* (+ (* x1 in) B) y0))
  CG.o(f,p,e).i(f,p,o)  $t_{min}=1.716n$ 
  CG.o(f,n,e).i(f,n,o)  $t_{min}=1.510n$ 
D=(! (@ CG_12 clock)); CG_12=(! (+ (* (+ x1 in) B) y0))
  CG.o(f,p,e).i(f,p,o)  $t_{min}=1.569n$ 
  CG.o(f,n,e).i(f,n,o)  $t_{min}=1.878n$ 

```

Figure 7.4b Generated gate-level designs for CG networks of P where $V_P^8 = \{A C D\}$. x_1 and in are references to the nodes. Property lists are references to the signal. Minimum clock period is specified.

7-4. DYNAMIC COVERING: P NETWORK GRAPH COVERING.

The local selection process for P networks is a greedy selection of the best gate-level design using the determined minimum clock period. In Figure 7.4b, the covering process for P sub-networks is a selection between CG networks with gate-level designs that have CG.o(f) output properties for f -type output signal D , and CG.o($\bar{f}$) for $\bar{f}$ -type output signal D . The CG networks selected have compatible input and output property lists that meet conditions set by Rule 1. Figure 7.4b shows the local selection results used for the covering process, being the best designs:

D.f=(! (@ (+ (* A B.f) I_1)))	D. $\bar{f}$ =(! (@ (* (+ (* x1 in) B.f) y0)))
A=(! CG_2) p-type	
CG_2=(! (* x1 in)) n-type	
I_1=(! (@ y0 clock)) $t_{min}=1.258n$	$t_{min}=0.699n$

7-5. DYNAMIC COVERING: G' NETWORK GRAPH COVERING.

The P network covering process is also the matching process for G' networks. And the G' network covering is a selection of P networks by comparing the throughput. A local selection process for gate-level design matches used for G'(viol) follows the depth-first-search order (A B C D) obtained from the algorithm give-output-vertex. Designs from the local selection process are gate-level designs with optimum throughput for all the possible number of critical register weights. G' local covering result sets are shown in Figure 7.5.

The covering process for G' is a selection of P network gate-level designs with different critical register weight. The covering process for G' networks selects gate-level designs of P networks with compatible input/output property lists that meet conditions set by Rule 1. Figures 7.6 and 7.7 show the measured clock period and throughput for all the possible gate-level designs resulted from the covering for G'(viol) network.

Figure 7.5 G' matches, Gate-level design sets locally selected for $V_P^s(A)$, $V_P^s(B)$ and $V_P^s(C)$ feeding V_P^o . Elements are optimum designs with different weight in the critical register weight path. Minimum clock period (t_{min}) and critical register weight (W_{max}) are specified.

C²MOS complex gate registers and *forced mappings* are independent options that allow for a bigger search space, such options are explored for $G'(viol)$ in Figures 7.6 and 7.7. Also, Figure 7.8 shows the complete $G'(viol)$ gate-level designs for the measured circuits in Figure 7.6 with forced mappings and with/without using complex gate registers.

Figure 7.6 Throughput and clock period for circuits of VIOL in benchmark DAIO.

(...) for mappings with logic not merged in registers and forced input registers

(___) for mappings with logic merged in registers and forced input registers

(ooo) for mappings with logic not merged in registers and without forced input registers

(xxx) for mappings with logic merged in registers and without forced input registers.

Figure 7.7 Throughput and clock period for circuits of (NOT VIOL) or (! VIOL) in benchmark DAIO.

(...) for mappings with logic not merged in registers and forced input registers
 (___) for mappings with logic merged in registers and forced input registers
 (ooo) for mappings with logic not merged in registers and without forced input registers
 (xxx) for mappings with logic merged in registers and without forced input registers.

Figure 7.8 $G'(\text{viol})$ gate-level design mappings. minimum clock period ($t_{\min}$), external clock period (T), and critical register weight ($W_{\max}$) are given. Performance is measured on Fig.6.5.

7.6. RETIMING COMPENSATION.

Retiming compensation is the final step and performs a feasible circuit integration process. Selected G' gate-level designs are connected with a compatible clocking scheme between clock sections. G' network outputs are part of a *clock* section and G' network inputs are part of a *clock* section, where $W_{\max}$ is always met in even numbers. Retiming compensation is illustrated with a greedy selection of G' with the best throughput. Figure 7.9 shows the selected G' network graphs with their gate-level design structures. Global retiming results towards a circuit integration are shown in Figure 7.10a. The freedom constraints are application dependent optimization parameters that provide in this example an uniform register manipulation among inputs, and an uniform register manipulation among outputs.

Figure 7.9 G' circuits with register weight requirements.

a) G circuit after global retiming

b) G circuit after local retiming.

Figure 7.10 Retiming compensation example. Results for G and G' in the final circuit schematic for the benchmark circuit DAIO.

CHAPTER 8

EXPERIMENTAL RESULTS

This chapter presents performance results measured for sequential and combinatorial benchmarks. The benchmark circuits are extracted from the 1991 MCNC benchmark library for logic synthesis.

8-1. COMBINATORIAL BENCHMARKS

Combinatorial circuits have structures with data evaluation flowing from primary inputs to primary outputs without asynchronous feedback loops. The structure transformation of a combinational circuit to a pipelined logic design introduces alterations on the external timing behavior. In combinational circuits the modified timing behavior does not affect the performance of the circuit since the throughput is measured as an independent factor from the number of registers along paths.

Table 1 shows the measured clock period and throughput against the number of *complete clock sections* along paths (N_{clock}) in tree like sub-circuits. A complete clock section is comprised of a *clock* and a $\overline{\text{clock}}$ section together. The reason to consider complete sections comes from the data exchange behavior between sections. A compatible data latching process with external circuits is driven by complete clock cycles to perform the latch and evaluation process. Because dynamic logic registers evaluate data and load a register during half a clock cycle it is necessary to evaluate the circuit performance in terms of complete clock cycles or *complete clock sections*.

In Table 1, the clock period (T) is measured from the minimum clock period ($t_{\min}$). The minimum clock period ($t_{\min}$) corresponds to the minimum time for a section to propagate and evaluate through paths. The clock period (T) is measured as $T = (2 * t_{\min})$. The throughput (th) is measured as the data output rate and it is an independent factor from the number of registers along paths. The throughput (th) is measured from $th = 1/T$. The circuit latency response (L) is the time required to propagate a signal from a primary input to a primary output. The circuit latency response is the product between the number of registers along the worst case of the shortest paths from primary inputs to primary outputs, and the minimum clock period ($t_{\min}$).

Benchmarks and number of equations		$N_{\text{clock}}=1$ T, th, L	$N_{\text{clock}}=2$ T, th, L	$N_{\text{clock}}=3$ T, th, L
cm138	9	4 ns, 250 MHz, 8 ns	3.09ns, 323 MHz, 9.27 ns	3.09 ns, 323 MHz, 15.45 ns
cm150	17	2.42 ns, 412 MHz, 9.69 ns	1.35 ns, 740 MHz, 9.46 ns	1.35 ns, 740 MHz, 15 ns
pcl	16	5.97 ns, 168 MHz, 41 ns	5.97 ns, 168 MHz, 84 ns	5.97 ns, 168 MHz, 125 ns
pcler8	24	5.97 ns, 168 MHz, 48 ns	5.97 ns, 168 MHz, 96 ns	5.97 ns, 168 MHz, 143 ns
cm162	19	3.92 ns, 255 MHz, 16 ns	3.92 ns, 255 MHz, 27 ns	3.92 ns, 255 MHz, 43 ns
cm163	16	3.92 ns, 255 MHz, 16 ns	3.92 ns, 255 MHz, 31 ns	3.92 ns, 255 MHz, 47 ns
cm42	13	3.25 ns, 307 MHz, 9.8 ns	3.25 ns, 307 MHz, 20 ns	3.25 ns, 307 MHz, 29 MHz
cm82	6	2.09 ns, 479 MHz, 4.1 ns	1.88 ns, 532 MHz, 7.5 ns	1.88 ns, 532 MHz, 11.3 ns
cm85	24	2.84 ns, 352 MHz, 11.4 ns	2.79 ns, 358 MHz, 17 ns	2.79 ns, 358 MHz, 22 ns
count	47	2.835 ns, 353 MHz, 45 ns	2.835 ns, 353 MHz, 90 ns	2.835 ns, 353 MHz, 0.14 us
decod	18	3.92 ns, 255 MHz, 7.8 ns	3.92 ns, 255 MHz, 16 ns	3.92 ns, 255 MHz, 24 ns
parity	15	1.82 ns, 549 MHz, 7.3 ns	1.27 ns, 786 MHz, 10 ns	1.27 ns, 786 MHz, 15 ns
unreg	32	2.502 ns, 400 MHz, 2.5 ns	1.409 ns, 710 MHz, 2.8 ns	1.409 ns, 710 MHz, 4.2 ns

Table 1 Measured performance for combinatorial benchmarks. N_{clock} is the number of complete clock cycles to evaluate through tree-like sub-circuits, T is the minimum clock period for the complete circuit, th is the circuit throughput, and L is the circuit latency response.

Table 1 demonstrates for combinational circuits that the throughput increases as the number of registers is increased. The throughput has a limit and for small scale circuits the limit is less than three complete *clock* sections. The latency is not a circuit speed parameter but it reflects the performance alterations to the external circuit behavior. Bigger benchmarks may show a similar performance since the throughput is better related to the size of tree like subcircuits than to the size of the complete circuit. The benchmarks tested were all limited to have subcircuits with up to ten inputs. Larger subcircuits are expected to reach the throughput limit at low values in the number of complete clock sections.

8-2. SEQUENTIAL BENCHMARKS.

Sequential circuits are structures with data evaluation flowing from primary inputs, through gates and registers to primary outputs. The application of registers allows for synchronous feedback. The structure transformation of a sequential circuit using pipelined logic introduces timing alterations that modify the circuit performance.

In sequential circuits, performance is measured also with the throughput. However, the throughput is measured from the product between the clock period and the number of complete clock sections or registers along the path with the smallest number of complete clock sections. The latency is the time period determined by the throughput. The clock period is determined from the minimum clock period parameter (t_{min}) as $T=(2 * t_{min})$. Table 2 has the measured clock period and throughput against the number of complete clock sections along paths in tree like subcircuits. Table 2 shows that for medium sized circuits the differential between clock periods is either too small or null. Therefore, the advantage of subsequent introduction of registers towards designs with smaller clock period is outweighed by the decreased throughput.

Benchmarks and number of equations		$N_{\text{clock}}=1$ T, th, L	$N_{\text{clock}}=2$ T, th, L	$N_{\text{clock}}=3$ T, th, L
daio	11	2.516 ns, 132 MHz, 7.6 ns	2.412 ns, 69 MHz, 14.5 ns	2.412 ns, 46 MHz, 21.7 ns
s27	13	2.624 ns, 95 MHz, 10.5 ns	2.624 ns, 48 MHz, 21 ns	2.624 ns, 32 MHz, 32 ns
ex3	72	4.96 ns, 40 MHz, 24.8 ns	3.92 ns, 43 MHz, 23.5 ns	3.92 ns, 32 MHz, 31.4 ns
s344	175	5.97 ns, 3.7 MHz, 0.27 us	5.97 ns, 1.9 MHz, 0.53 us	5.97 ns, 1.3 MHz, 0.79 us
s349	176	5.97 ns, 3.7 MHz, 0.27 us	5.97 ns, 1.9 MHz, 0.53 us	5.97 ns, 1.3 MHz, 0.79 us
s382	179	5.97 ns, 18.6 MHz, 53.7 ns	5.97 ns, 16.8 MHz, 59.7 ns	5.97 ns, 12 MHz, 84 ns
s400	185	5.97 ns, 18.6 MHz, 53.7 ns	5.97 ns, 15 MHz, 65.7 ns	5.97 ns, 12 MHz, 84 ns
s444	202	5.97 ns, 16.8 MHz, 59.7 ns	5.97 ns, 11.2 MHz, 90 ns	5.97 ns, 7.6 MHz, 131 ns

Table 2 Measured performance for sequential benchmarks. N_{clock} is the number of complete clock cycles to evaluate through tree-like sub-circuits, T is the minimum clock period for the complete circuit, th is the worst output signal throughput, and L is the circuit latency response.

8-3. LIMITING HEURISTICS.

Limiting heuristics are designed to save CPU time consumption at the subcircuit level. The limiting heuristics use the parameters (cCgG) for upper and lower limits in the number of registers along the critical register weight path and in the number of complex gates. Circuit performance and CPU consumption are evaluated for parameter (C) since limiting the number of registers along paths has a more direct relationship with the performance parameters in table 1 and table 2. Table 3 shows the measured throughput, clock period and CPU time consumption used by the technology mapper for different (C) parameter values.

Table 3 demonstrates that circuits with small subcircuits show CPU savings only at low values in the upper limit parameter (C). It is expected more significant CPU savings for circuits with a larger number of tree like subcircuits that have a larger number of inputs.

Benchmark circuit: COUNT

Limitation parameter: CPU:	C=10 2569 sec	C=9 2543 sec	C=8 2546 sec
T, th, L ($N_{\text{clock}}=1$):	2.835 ns, 353 MHz, 45 ns	2.835 ns, 353 MHz, 90 ns	2.835 ns, 353 MHz, 0.14 μ s
T, th, L ($N_{\text{clock}}=2$):	2.835 ns, 353 MHz, 45 ns	2.835 ns, 353 MHz, 90 ns	2.835 ns, 353 MHz, 0.14 μ s
T, th, L ($N_{\text{clock}}=3$):	2.835 ns, 353 MHz, 45 ns	2.835 ns, 353 MHz, 90 ns	2.835 ns, 353 MHz, 0.14 μ s
Limitation parameter: CPU:	C=7 2543 sec	C=6 2545 sec	C=5 2542 sec
T, th, L ($N_{\text{clock}}=1$):	2.835 ns, 353 MHz, 45 ns	2.835 ns, 353 MHz, 90 ns	2.835 ns, 353 MHz, 0.14 μ s
T, th, L ($N_{\text{clock}}=2$):	2.835 ns, 353 MHz, 45 ns	2.835 ns, 353 MHz, 90 ns	2.835 ns, 353 MHz, 0.14 μ s
T, th, L ($N_{\text{clock}}=3$):	2.835 ns, 353 MHz, 45 ns	2.835 ns, 353 MHz, 90 ns	2.835 ns, 353 MHz, 0.14 μ s
Limitation parameter: CPU:	C=4 2540 sec	C=3 2538 sec	C=2 1154 sec
T, th, L ($N_{\text{clock}}=1$):	2.835 ns, 353 MHz, 45 ns	2.835 ns, 353 MHz, 90 ns	2.835 ns, 353 MHz, 0.14 μ s
T, th, L ($N_{\text{clock}}=2$):	2.835 ns, 353 MHz, 45 ns	2.835 ns, 353 MHz, 90 ns	2.835 ns, 353 MHz, 0.14 μ s
T, th, L ($N_{\text{clock}}=3$):	2.835 ns, 353 MHz, 45 ns	2.835 ns, 353 MHz, 90 ns	2.835 ns, 353 MHz, 0.14 μ s

Benchmark circuit: EX3

Limitation parameter: CPU:	C=10 2819 sec	C=9 2816 sec	C=8 2814 sec
T, th, L ($N_{\text{clock}}=1$):	4.96 ns, 40 MHz, 25 ns	4.96 ns, 40 MHz, 25 ns	4.96 ns, 40 MHz, 25 ns
T, th, L ($N_{\text{clock}}=2$):	3.92 ns, 43 MHz, 24 ns	3.92 ns, 43 MHz, 24 ns	3.92 ns, 43 MHz, 24 ns
T, th, L ($N_{\text{clock}}=3$):	3.92 ns, 32 MHz, 31 ns	3.92 ns, 32 MHz, 31 ns	3.92 ns, 32 MHz, 31 ns
Limitation parameter: CPU:	C=7 2819 sec	C=6 2809 sec	C=5 2807 sec
T, th, L ($N_{\text{clock}}=1$):	4.96 ns, 40 MHz, 25 ns	4.96 ns, 40 MHz, 25 ns	4.96 ns, 40 MHz, 25 ns
T, th, L ($N_{\text{clock}}=2$):	3.92 ns, 43 MHz, 24 ns	3.92 ns, 43 MHz, 24 ns	3.92 ns, 43 MHz, 24 ns
T, th, L ($N_{\text{clock}}=3$):	3.92 ns, 32 MHz, 31 ns	3.92 ns, 32 MHz, 31 ns	3.92 ns, 32 MHz, 31 ns
Limitation parameter: CPU:	C=4 2802 sec	C=3 2801 sec	C=2 1052 sec
T, th, L ($N_{\text{clock}}=1$):	4.96 ns, 40 MHz, 25 ns	4.96 ns, 40 MHz, 25 ns	4.96 ns, 40 MHz, 25 ns
T, th, L ($N_{\text{clock}}=2$):	3.92 ns, 43 MHz, 24 ns	3.92 ns, 32 MHz, 31 ns	4.96 ns, 20 MHz, 50 ns
T, th, L ($N_{\text{clock}}=3$):	3.92 ns, 32 MHz, 31 ns	3.92 ns, 21 MHz, 47 ns	4.96 ns, 13 MHz, 7.4 ns

Table 3 Measured performance for benchmark circuits COUNT and EX3 subject to limits in the number of registers along paths in tree-like sub-circuits. N_{clock} is the number of complete clock cycles to evaluate through tree-like sub-circuits, T is the minimum clock period for the complete circuit, th is the worst output signal throughput, and L the circuit latency response.

CHAPTER 9

CONCLUSIONS

9-1. SUMMARY.

The development of this study has been successful in its objectives. The analysis of CMOS dynamic logic design techniques and of technology mapping algorithms are the base for a new approach of the representation model and technology mapping algorithm. Following a review of existing CMOS dynamic logic design techniques, a modified NO-Race design technique is selected for the experimentation. This technique uses n-type/p-type dynamic logic gates and C²MOS registers following the logic, topology and electrical constraints imposed by the NO-Race technique.

The technology mapping is targeted to combinatorial and sequential logic circuits, and optimize circuit designs for speed using pipelined logic. The new representation model emerged from the need to efficiently represent in a single structure the logic, topological and electrical properties of a circuit. The model inherited some key parameters from previous models to efficiently adapt the advantages from using cell generators. The levels of representation for the model are used to manipulate the circuit topology during the technology mapping. The manipulated structures are expressed with the graph and gate-level models and are based on technology dependent parameters defined using the timing-level model.

The technology mapping algorithm is based on tree match and cover methods used for cell generators. The balanced tree decomposition process is the first step for technology mapping and is followed by a recursive matching process that determines all the feasible logic circuit gate elements and pipelined sections for subcircuits. A recursive selection process integrated with a dynamic programming algorithm approach finds the optimum designs that are selected by the covering process. Sequential and combinatorial circuits are mapped by the same algorithm at the subcircuit level, and a final subcircuit integration process finds the optimum circuit.

21 of/de 2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

The prototype is named the "Denali" technology mapper, and is implemented using the COMMON LISP programming language using the framework provided by the technology mapping system for static logic circuits [Liem]. The prototype generates circuit designs at different levels of abstraction: from the transistor netlist to logic circuit expressions.

Performance parameters of mapped circuits are studied and have demonstrated that a pipelined logic design technique yields to faster circuits by introducing registers. Results demonstrate that a small number of registers are required to find the optimum speed solution. This results are very encouraging since these demonstrate that pipelined logic is a useful technique and there is the possibility to develop more efficient algorithms oriented for low levels of pipelined logic.

9-2. CONTRIBUTIONS.

A new representation model for technology mapping has been developed. The presented model is a general representation model of logic circuits using CMOS technology. The model has a clear definition for each level of representation that can be applied to other technologies.

A new algorithm for technology mapping using CMOS dynamic logic techniques has been developed. The algorithm is based on concepts developed for technology mapping using cell generators. The introduction of a new approach for the match and cover process using dynamic programming have, for the first time, introduced the ability to implement constraints of the NORA technique in a logic circuit design.

A prototype framework has been implemented and has allowed the investigation of different levels of pipelining for combinatorial and sequential logic circuits. The prototype has also introduced new directions of research.

9-3. RECOMMENDATIONS FOR FUTURE WORK.

The presented algorithms and prototype are the basis for future research. In the short term, the exploration of n-p structures implemented in Domino logic, dynamic logic and static logic gates are expected to improve circuit performance. The property lists can be expanded to manipulate the construction of feasible n-p structures. The experimental results have demonstrated that a few registers are required to find the optimum speed justifying the addition of more register design options such as TSPC registers. Also, experiments dealing with buffering can be included in the matching process and are expected to improve the circuit performance.

The improvement on transistor sizing algorithms is definitively a significant concept for circuit performance improvement. An optimum transistor sizing technique not only improves the circuit performance but it may also improve the circuit design selections made by the technology mapping algorithm.

The clock signal generation is an important aspect for dynamic logic circuits. The large number of clocked transistor devices generate large loads that need specific clock signal generators. Research on clock signal generation oriented to large signal loads and clock skew has been done [Fish]. And in CMOS dynamic logic circuits, the performance is improved if the clock signal generator reduces clock skews and signal overloading.

Analysis of dynamic CVS circuits [HeGr] has been studied for gate level implementations [ChPu] and it is expected to become a significant improvement for circuit performance with the development of better circuit design techniques to exchange data between single-ended and complementary logic circuits. The matching process implemented is for tree-like sub-graphs and is a special case of graph matching. A matching process for general graph structures is expected to generate the better circuit designs. Graph matching is a well researched area from the graph-theory community [Lora] [Manb] and the only possibility to generate optimum designs is with a better understanding of multiple-output circuits

[HwFi], complementary logic and the dynamic programming approach for integration of combinational logic minimization techniques [MaSe] within the technology mapping at nodes with a fanout to more than one gate.

Finally, it is desirable to compare against other technology mappers such as the technology mapper in the Yorktown Silicon Compiler [BrBr], oriented for Complementary Voltage Switch Logic circuits; and compare against technology mappers for Static Logic circuits such as Rossini [Liem] and MIS [DeGa].

APPENDIX 1. PROTOTYPE USER MANUAL

START UP:

The Denali technology mapping prototype is started from the Common LISP interactive interface. An interactive (or batch) session should be started in the same directory where the target logic circuit file is stored.

To start an interactive session from UNIX type:

```
> cl
```

```
user:: (load "~/lisp/code/denali.fasl")
```

A batch session uses a batch file and is executed from UNIX. The first line in the batch file is:

```
(load "~/lisp/code/denali.fasl")
```

followed by the commands used in a normal interactive session. A batch session is started with the UNIX command:

```
> nice +20 cl -batch < batch-file >& log-file &
```

this command executes all the command in the batch-file. The output is stored in the log-file file. The sessions runs in the background without disturbing other users and it will exit LISP automatically if an error happens or if the commands in the batch file are terminated.

IMPORTANT:

+ The host CPU should be able to run the following commands:

```
cl  
hspice  
gams  
nawk
```

LOADING LOGIC CIRCUITS

Two standard file formats for logic circuit definition are supported: SLIF and EQN. The SLIF format supports both combinatorial and sequential logic circuits. The EQN format supports combinatorial logic circuits. SLIF circuit files are loaded with the command:

(read_slif 'fn)

fn is the file name without the ".slif". i.e. file daio.slif is loaded typing:

(read_slif 'daio)

EQN circuit files are loaded with the command:

(read_latched 'fn)

fn is the file name without the ".eqn". i.e. file cm180.eqn is loaded typing:

(read_latched 'cm180)

IMPORTANT:

- + The variable names *i* and *r* are reserved for internal use in Denali. Any primary signal or equation named *i* or *r* must be modified.
- + The command *read_slif* may not accept lines bigger than 200 characters.

TECHNOLOGY MAPPING.

There is one command to start the technology mapping:

(table C G 'fn c g)

parameters *C*, *G*, *c*, and *g* are the performance parameters defined in the timing level model, the recommended values are *C*=4, *G*=1000, *c*=1, and *g*=0. *fn* is the output file name for a file command in MATLAB stored as *fn.m*. This file shows the performance graphs for all tree-like sub-circuits. The performance data in the graphs are $t_{\min}$ and th_i .

RETIMING.

Retiming is executed once the sub-circuits have been determined. To set up the correct set of sub-circuits for retiming the user must know the desired limit for the critical register weight $W_{\max}$. The following command finds the best sub-circuit with a critical register weight up to the limit or less, and adds non-inverting registers to meet the limit:

(set_to_limit limit)

limit is an even and positive integer number greater than zero (limit=2, 4, ...).

The retiming compensation from the original register setup to the desired one is solved with the following command:

(retime)

RESULTS INSPECTION.

The mapped circuit is available in two formats. A transistor netlist file or logic expressions. To generate a netlist in hspice format:

(write_net 'fn)

fn is the output file named *fn.net*.

To generate the logic expressions:

(write_expr 'fn)

fn is a file with no specific format that shows logic gates, registers and complex registers. The output file name is *fn.dyn*.

Performance parameters from the circuit are obtained with:

(determine_t)

determines the minimum clock period of the retimed design.

(old_path)

determines the register weight used for throughput estimation in the original circuit.

(new_path)

determines the register weight used for throughput estimation in the retimed circuit.

MISCELLANEOUS.

HSPICE simulations are used for the delay model of gates. The results are stored in a database to avoid a massive use of HSPICE. Denali automatically maintains the database by making the required simulations every time a new gate is required. The data stored in the database may grow every time a new circuit is mapped. The new database information is stored with the following command:

(store_tech 'fn)

fn is the output file that keeps the data for all gates. The output file is *fn.tech*. The database is automatically loaded every time Denali is started.

The Rossini technology mapper for static logic gates is integrated inside Denali. The following commands switch between both systems:

(rossini) to change from Denali to Rossini

(denali) to change from Rossini to Denali.

The following LISP variables control the type of mapping:

SPLN The SPLN parameters. Default: '(10 10 10 6)

tech_map_mode A value of '*mix_all*' mappings to try both types of forced mappings. A value of '*mix*' doesn't include the forced mappings. Default: '*mix_all*'.

merge_latch Controls the option of using complex registers. Default: 't

Detailed data from the mapped circuit is available with the command:

(dump_part_term)

Complete HSPICE files with simulation parameters generate data files for inspection with the Graphics-Interface-Software from HSPICE using the following command:

(test 'fn 'input-signals)

fn is the output file for HSPICE using CMOS4S parameters. The output file name is *fn.spi*.

input-signals is a list of waveform definitions for each primary input:

'((input1 waveform) (input2 waveform) . . .)

input1, input2, ... are the primary inputs to the circuit. Waveform is any of the following:

1	generates a constant +5v signal
0	generates a constant +0v signal
01	generates a repetitive 01 waveform
10	generates a repetitive 10 waveform
0011	generates a repetitive 0011 waveform
1100	generates a repetitive 1100 waveform
000111	generates a repetitive 000111 waveform
111000	generates a repetitive 111000 waveform
00011101	generates a repetitive 00011101 waveform
11100010	generates a repetitive 11100010 waveform
11100000	generates a repetitive 11100000 waveform

the clock signal uses the same period of time for a logical one and for a logical zero.

Analysis parameters in HSPICE and waveforms use the following variables:

t	clock period. Default: 4e-9
tr_data	transition time for data signals. Default: 0.1e-9
tr_ck	transition time for clock signal. Default: 0.1e-9
tran	transient parameters definition. Default: "0.2n 80n"
fixed_w	fixed n-type transistor width parameter. Default: 10e-6
wp_factor	scale factor between n-type and p-type devices. Default: 2

DEBUGGING.

A debugger is used to study all the steps in the technology mapping. Results are stored in the file *cl.debug*. The following command is executed before technology mapping and after technology mapping:

(turn_debug)

this command gives basic data from the match and cover processes. A better detail of the process is controlled with the following variables assigned to a value of 't' before technology mapping and after the first *turn_debug* command:

debug2	gives detailed delay data from match and cover.
debug3	gives data from local selection and covering of G' circuits.
debug4	minimum data generation for performance comparisons used

in the thesis.

debug5	delay debugging for data generation in the thesis.
-----------------	--

the variable **debug** can be set to 'nil' if **debug3** or **debug4** are 't'.

WARNING:

+ The debugging tool can generate data files bigger than 30 Mega-bytes.

SESSIONS OR BATCH FILES.

1) This batch reads the circuit , executes the technology mapping and retimes the circuit for all the possible sub-circuit limits in W_{max} :

```
(load "~/lisp/code/denali.fasl")
(batch_slif 'fn mode C G c g)
(store_tech "~/lisp/code/cmos4s")
```

fn is the input circuit file in SLIF format. C, G, c, and g are the limitation parameters. (suggested to have values of 4, 1000, 1 and 0 respectively)

mode can have any of the following values:

"m"	sets *tech_map_mode* to 'mix_all and *merge_latch* to 't
"nm"	sets *tech_map_mode* to 'mix_all and *merge_latch* to 'nil
"mi"	sets *tech_map_mode* to 'mix and *merge_latch* to 't
"nmi"	sets *tech_map_mode* to 'mix and *merge_latch* to 'nil

The following batch processes EQN files:

```
(load "~/lisp/code/denali.fasl")
(batch_eqn 'fn mode C G c g)
(store_tech "~/lisp/code/cmos4s")
```

2) This batch reads the circuit DAIO, executes the technology mapping, retimes the circuit and generates the executable HSPICE file.

```
(load "~/lisp/code/denali.fasl")
(read_slif 'daio)
(table 4 1000 'daio 1 0)
(set_to_limit 4)
(ptime)
(setq *t* 6e-9)
(test 'daio '((in *00011101*))
(store_tech "~/lisp/code/cmos4s")
```

APPENDIX II. GLOSSARY

B

BINARY DECOMPOSITION: Manipulation of a graph such that each vertex has two input edges.

BINATE VARIABLE: A logic variable where both phases of the signal are present.

C

C²MOS COMPLEX GATE REGISTER: C²MOS register that evaluates inverting complex logic functions.

reference Section 2-2, Figure 2.3

C²MOS REGISTER: Clocked CMOS registers are a circuit design technique developed by Suzuki/Odagawa [SuOd]. C²MOS registers are based on the conditional discharge of a conditionally pre-charged output capacitance.

reference Section 2-2, Figure 2.3

CELL GENERATOR: In this theses, is a custom gate designer that makes a transistor-level gate designs from a logic expression.

CELL LIBRARIES: A set of transistor-level gate designs that correspond to logic expressions.

CG: Graph hierarchy level for Complex Gates. See CG network graph.

CG NETWORK GRAPH: Synchronous sub-network graph extracted from the synchronous network graph P. The logic expression from all vertices are implemented in a single transistor-level gate design or complex-gate register. The gate-level design of a CG network can use extra dynamic logic inverters at the output.

CHARGE REDISTRIBUTION: Voltage level drop at the output of a gate due to connected parasitic loads in the evaluation block from an incomplete evaluation path.
reference Section 2-1.2

CLOCKED CMOS REGISTER: See C²MOS register.

CLOCK SECTION: Sub-circuit that propagates a logic evaluation during the V_H state of the clock signal. The inputs to a section are either static or dynamic logic gates, the output is always a register. A clock section propagates a logic evaluation during the V_L state of the clock signal.

reference Sub-section 2-1.1

CLOCK SIGNAL: Input signal that globally controls the operation states of dynamic logic. This clock signal generates the $\overline{\text{clock}}$ signal.

CLOCK SKEW: Clock skew is a delay between clock and $\overline{\text{clock}}$ signals. C²MOS registers driven by clock signals with a clock skew allow to one of the evaluation blocks for an earlier access to the output load causing races.

CMOS: Complementary Metal Oxide Semiconductor.

COMPLEMENTARY LOGIC CIRCUITS: Logic circuit design method where all gates generate both phases for a given output expression.

COMPLEX GATE REGISTER: See C²MOS Complex–Gate Register.

COVER: Performance oriented selection of matches for a feasible circuit design.

P network covering reference Section 6–3.1

G' network covering reference Section 6–3.2

CRITICAL REGISTER WEIGHT: Critical register weight ($W_{\max}$) is the worst case for the number of registers along a path from an edge feeding a vertex in V_p^i to the edge feeding the output vertex in V_p^o .

CRITICAL REGISTER WEIGHT PATH: The path where the critical register weight is estimated.

CRITICAL DELAY: Worst case from the accumulated data ready times along paths from inputs to output of a network graph.

CVS: Complementary Voltage Switch. See Dynamic CVS Circuits.

D

DATA READY TIME: Data ready time (t_i) is the time between the clock edge and an evaluated signal for a give gate–level design. Data ready time is measured either for a gate or a binary path through gates.

DATA THROUGHPUT: Data throughput (th_i) in the timing–level model is the product of $t_{\min}$ and $W_{\max}$ for either G' or P network graphs. For retimed synchronous circuits it is $1/(2*t_{\min}*W_{\max})$ at the G network graph level. For retimed asynchronous circuits it is $1/(2*t_{\min})$ at the G network graph level.

DESIGN PARAMETERS: Variables described in the timing–level model, in Section 5–3, to measure the performance of gate–level circuit designs. The design parameters are data ready time, critical delay, minimum clock period, data throughput and critical register weight.

reference Section 5–3

DOMINO LOGIC: Domino logic gates are dynamic logic gates, n or p, with a static inverter at the output.

reference Sub–section 2–1.2

design constraints reference Section 2–2.1

DOUBLED N–C²MOS REGISTER: Series connection of two n–C²MOS registers driven by the same clock signal phase.

reference Figure 2.6a

DOUBLED P–C²MOS REGISTER: Series connection of two p–C²MOS registers driven by the same clock signal phase.

reference Figure 2.6b

DUAL: See Dual Logic Expression.

DUAL FUNCTION: See Dual Logic Expression.

DUAL LOGIC EXPRESSION: The dual logic expression is obtained by the application of DeMorgan's theorem to find an output signal with inverted phase, and is fed by signals with inverted phase.

DYNAMIC CVS CIRCUITS: Dynamic logic circuit design technique based on the input and output of complementary signal phases.
reference [HeGr]

DYNAMIC LOGIC: CMOS circuit design technique based on the conditional discharge of an unconditionally pre-charged output capacitance.
reference Section 2-1

E

EDGE: See GRAPH

EVALUATION BLOCK: Transistor-level design that corresponds to a logic expression. The circuit uses a single type of transistors, and the current flows in one direction.
reference Figure 2.1

F

FANOUT: Fanout is the number of logic expressions, or logic circuit gates, fed by the output expression or logic circuit gate.

FLATTENED NETWORK: Gate level design of the G network graph obtained by including all CG networks inside P networks, and by including all P networks inside a final G' network.

FLEXIBLE WEIGHT: Minimum register requirements used at the edges connected to input or output vertices of a P sub-circuit.
reference Section 5-1.2

FORCED REGISTER ALLOCATION: Technology mapping type of process where the placement of registers at the inputs of P networks is constrained.
reference Section 6-3.2

FREEDOM CONSTRAINTS: Application dependent timing level constraints used during the global retiming process.
reference Section 6-4.1

G

G: Highest graph hierarchy level for the complete circuit. See G network graph.

G NETWORK GRAPH: Highest hierarchy level in the hierarchical Synchronous Network Graph.

G': Graph hierarchy level for subcircuits of one output with a fanout of more than one. See G' network graph.

G' NETWORK GRAPH: Synchronous network graph with one output and no loops.

GATE LEVEL DESIGN PROPERTY: Network graph property that corresponds to the transistor level implementation of the logic expression for all vertices in the network graph. The output properties of the gate-level design are expressed by the output property list, and all the inputs meet the requirements in the input property list.

GRAPH: "A graph $G=(V,E)$ consists of a set V of vertices, and a set E of edges. Each edge corresponds to a pair of vertices. The edges represent relationships among the vertices". "The edges in a directed graph are ordered pairs -The order between the two vertices the edge connects is important. In this case we specify an edge as an arrow pointing from one vertex (the tail) to another (the head). Trees are examples of graphs"

reference [Manb]

GATE-LEVEL DESIGN: Logic circuit expressions or transistor-level designs that correspond to a feasible circuit gate using CMOS complex gates and registers.

GLOBAL RETIMING: Integer programming optimization process that manipulates the original register allocation to implement the desired circuit subject to the freedom constraints.

reference Section 6-4.1

H

HSPICE: Industrial grade circuit analysis system for the simulation of electrical circuits in steady-state, transient and frequency domain.

I

INPUT LOGIC EXPRESSIONS: Simple logic expressions that represent the required input logic phase of a signal.

reference Section 6-2.3

INPUT PROPERTY LIST: Define the required properties of the gate-level designs of a network graph at the inputs. Input properties for dynamic logic are signal type property (*f* or *f'*), gate type property (*n-type* or *p-type*) and inversions type property (*Odd* or *Even*).

INTERNAL DELAY PROBLEM: A dynamic logic gate with a pre-charged input logic level that irreversibly corrupts the output evaluation of the fanout gate, is a circuit with a race problem named internal-delay-problem.

reference Section 2-1.2

J

JUNCTIONS: Repeated edges while merging at least two sub-networks of the same type (P or CG) as to generate a feasible network of the next higher hierarchy (G' or P respectively).

reference Section 5–1.2, Rule 2.

L

L PARAMETER: Limits the logic depth. See SPLN technology limitation parameters.

LOCAL RETIMING: Integer programming optimization process that re–arranges the supplied input registers by the global retiming into the desired set of sub–circuits.
reference Section 6–4.2

LOCAL SELECTION: Selection process of generated matches for a given vertex.
local selection for P networks reference Section 6–3.1
local selection for G' networks reference Section 6–3.2

M

MAIN BODY LOGIC EXPRESSION: Complex logic expressions without inverted input signals used during the gate design exploration process.
reference Section 6–2.3

MATCH: Exploration of all the possible circuit gate realizations, or matches, extracted from a logic evaluation.

MCNC: Microelectronics Center of North Carolina.

MINIMUM CLOCK PERIOD: Minimum clock period ($t_{\min}$) for a P network graph determined by either the critical delay path or by the pre–charge time of gates in P networks, whichever is worst.

MIXED REGISTER ALLOCATION: Technology mapping type of process where the placement of registers at the inputs of P networks is not constrained.
reference Section 6–3.2

N

N PARAMETER: Limits the number of inputs for a complex gate. See SPLN technology limitation parameters.

N–C²MOS REGISTER: A C²MOS register with the p–type clock transistor eliminated.
reference Section 2–4.1

NETWORK GRAPH: See GRAPH

NETWORK LIMITATION PARAMETERS (cCgG): Upper and lower limiting parameters on the number of gates (excluding registers) and the number of registers along the critical register weight paths. The parameters for the number of gates are g for lower limit and G for upper limit. The parameters for the number of registers along the critical register weight path are g for lower limit and G for upper limit.
reference Section 6–3.3

NORA TECHNIQUE: The NO–RACE technique developed by Gonclaves/DeMan [GoMa] is a pipelined logic design technique for dynamic logic that uses a constrained design of n–p structures and C²MOS registers.

reference Section 2-3, Figure 2.4

N-P STRUCTURES: Gate structure composed of n-type dynamic logic gates, p-type dynamic logic gates and Domino logic gates. n-type and p-type dynamic logic gates are alternated and use opposite clock phase to provide a simultaneous evaluation and protection from internal-delay-problems.

reference Figure 2.2

in NORA reference Section 2-3

NPN-EQUIVALENT: Two logic functions which are equivalent through negation of its variables, permutation of its variables and/or negation of the function itself.

O

OUTPUT PROPERTY LIST: Define the properties available of the gate-level design of a network graph provided that input gate-level designs meet the input property list. Output properties for dynamic logic are signal type property (f or $\bar{f}$), gate type property (n -type or p -type) and inversions type property (*Odd* or *Even*).

P

P: Graph hierarchy level for partitions or register placements. See P network graph.

P NETWORK GRAPH: Synchronous sub-network graph extracted from the synchronous network graph G' . The output has a register and the inputs can optionally have registers.

P PARAMETER: Limits the number of p-type transistors in series. See SPLN technology limitation parameters.

P-C²MOS REGISTER: A C²MOS register with the n-type clock transistor eliminated. reference Section 2-4.1

PIPELINED LOGIC: Logic circuit design style where logic expressions are partitioned in latched sections to compensate imbalance from asynchronous signal arrivals, towards a clock driven logic evaluation.

PRIMARY INPUTS: External input signals to the circuit. Primary inputs can be generated by a primary output only if there are no asynchronous loops.

PRIMARY OUTPUT: Output signal fed to the exterior of the circuit. Primary outputs can feed primary inputs only if there are no asynchronous loops.

PROPERTY LISTS: See Input Property List. See Output Property List.

R

RACE PROBLEMS: see internal-delay-problem, and clock-skew.

S

S PARAMETER: Limits the number of n-type transistors in series. See SPLN technology limitation parameters.

SHANNON DECOMPOSITION: The Shannon decomposition of a logic expression refers to writing the logic expression in a maximal form of sum of products of etc., or product of sums of etc...
reference Section 3–1.2

SPLIT–OUTPUT N/P–TYPE REGISTER: Improved versions of the doubled n/p–C²MOS registers.
reference Figure 2.6b

SPLN TECHNOLOGY LIMITATION PARAMETERS: Constraints for limiting the size of gates introduced by Liem [Liem]. The parameters are based on the knowledge of the available capacity in the cell generator and technology.
reference Section 3–2.2

T

TAUTOLOGY: Equivalent logic expression or the complementary.
reference Section 3–1.2

TECHNOLOGY LIMITATION PARAMETERS: See SPLN technology limitation parameters.

TECHNOLOGY MAPPING: Final process in a logic synthesis system. A logic circuit, synchronous or asynchronous, is implemented to a transistor level design.

TH_i: See Data Throughput.

THROUGHPUT: See Data Throughput.

T_i: See Data Ready Time

T_{min}: See Minimum Clock Period

TREE LIKE: Hierarchical structure with one output vertex and many input vertices and no feedback loops, where it is permitted to have duplicated input vertices.

TRUE–SINGLE–PHASE–CLOCK TECHNIQUE: TSPC technique developed by Yuan/Karlsson/Svensson [YuSv2] is a pipelined logic design technique for dynamic logic gates that uses n–p structures and dynamic non–inverting True–Single–Phase–Clock registers. See TSPC.
reference Section 2–4

TSPC: True–Single–Phase–Clock. See True–Single–Phase–Clock technique, TSPC–1/2 register, doubled n/p–C²MOS register and split–output n/p–type register.

TSPC–1 REGISTER: An n/p–C²MOS register fed by an n/p–type dynamic logic gate. Both n–type and p–type registers are driven by the same clock signal phase.
reference Section 2–4.1, Figure 2.5a

TSPC–2 REGISTER: A modified version of the TSPC–1 register.
reference Section 2–4.1, Figure 2.5b

U

UNATE VARIABLE: A logic variable, where only one phase of the signal is present.

V

VERTEX: see GRAPH

VERTEX DECOMPOSITION: Manipulation of the input edges of a vertex such that its decomposed in a set of vertices restricted to have a fanin of two.

W

WEIGHT: Number of registers on a certain edge of a directed weighted graph. Total number of registers from edges along a path in a directed weighted graph.

WEIGHT REQUIREMENT: The constant weight requirement (W_p) sets the minimum register weight for an edge or along a path.

W_{max} : See Critical Register Weight

W_p : See Weight Requirement

REFERENCES.

1. [AhCo]: A. Aho, M. Corasick, "Efficient String Matching: An Aid to Bibliographic Search", *Communications of the ACM*, 18, 6, June 1975, pp. 333–340.
2. [BeJe]: M.R.C.M. Berkelaar, J.A.G. Jess, "Technology Mapping for Standard–Cell Generators", *Proceedings of the International Conference on Computer–Aided Design*, 1988, pp. 470–473.
3. [BrBr]: R.K. Brayton, N.L. Brenner, C.L. Chen, G. DeMichelli, C.T. McMullen, and R.H.J.M. Otten, "The YORKTOWN Silicon Compiler", *Proceedings of ISCAS'85*, pp. 391–394.
4. [BrHa]: R.k. Brayton, G.D. Hachtel, C.T. McMullen, A.L. Sangiovanni–Vicentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
5. [ChPu]: Kan M. Chu, David L. Pulfrey, "Design Procedures for Differential Cascode Voltage Switch Circuits", *IEEE Jour. Solid–State Circuits*, Dec. 1986, vol–21, pp 1082–1087.
6. [DeGa]: E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni–Vicentelli, A. Wang, "Technology Mapping in MIS", *Proceedings of the International Conference on Computer–Aided Design*, 1987, pp 116–119.
7. [Fish]: John P. Fishburn, "Clock Skew Optimization", *IEEE Trans. on Computers*, July 1990, vol–39, pp 945–951.
8. [FrGi]: F. Mailhot and G. DeMichelli, "Technology Mapping Using Boolean Matching and Don't Care Sets", *Proceedings of the European Design Automation Conference*, 1990, pp. 212–216.
9. [FrLi]: V. Friedman, S. Liu, "Dynamic Logic CMOS Circuits", *IEEE Jour. Solid–State Circuits*, April 1984, vol sc–19, pp 263–266.
10. [GoMa]: Nelson F. Gonclaves, Hugo J. de Man, "NORA: A Racefree Dynamic CMOS technique for Pipelined Logic Structures", *IEEE Jour. Solid–State Circuits*, June 1983, vol sc–18, pp 261–267.
11. [GrBa]: D. Gregory, K. Bartlett, A. deGeuss, G. Hatchel, "Socrates: A System for Automatically Synthesizing and Optimizing Combinational Logic", *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, June 1986, pp. 79–85.

12. [HeGr]: Lawrence G. Heller, William R. Griffin, "Cascode Voltage Switch Logic: A Differential CMOS Logic Family", 1984 IEEE International Solid-State Circuits Conference. pp 16–17.
13. [HwFi]: Inseok S. Hwang, Aaron L. Fisher, "A 3.1ns 32b Adder in Multiple Output Domino Logic", IEEE International Solid-State Circuits Conference, 1988, pp 140–141.
14. [Keut]: K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching", Proceedings of the 24th ACM/IEEE Design Automation Conference, 1986, pp. 79–85.
15. [KrLe]: R.H. Krambek, Charles M. Lee, Hung-Fai Stephen Law, "High Speed Compact Circuits with CMOS", IEEE Jour. Solid-State Circuits, June 1982, vol sc-17, pp 614–619.
16. [Liem]: Clifford B. Liem, "Performance Driven Technology Mapping for Cell Generators", August 1991, Master's theses at Carleton University, Ottawa, Canada.
17. [LiRo]: C.E. Lieserson, F.M. Rose, J.B. Saxe, "Optimizing Synchronous Circuitry by Retiming", Third Caltech Conference on VLSI (1983).
18. [Lor]: L. Lovasz, M.D. Plummer, *Matching Theory*, Annals of discrete mathematics, vol 29, North Holland, 1986.
19. [Manb]: Udi Manber, *Introduction to Algorithms*, Addison Wesley, 1989.
20. [MaSe]: Sharad Malik, Ellen M. Sentovich, Robert K. Brayton, and Alberto Sangiovanni-Vicentelli, "Retiming and Resynthesis: Optimizing Sequential Networks with Combinational Techniques", IEEE Transactions on Computer-Aided Design, January 1991, vol. 10, pp. 74–84.
21. [McCl]: McCluskey, Edward J., *Introduction to the Theory of Switching Circuits*, McGraw-Hill, 1965.
22. [MiKl]: Giovanni De Michelli, Thierry Klein, "Algorithms for Synchronous Logic Synthesis", ISCAS'89.
23. [SaSa]: K. Sakouti, G. Saucier, "A fast and effective technology mapper on an autodual library of standard cells", Proceedings on the IFIP Working Conference on Logic and Architecture Synthesis, 1990, pp. 137–150.

24. [Shoj]: Masakazu Shoji, *CMOS Digital Circuit Technology*, Prentice Hall, 1988, pp.366–369.
25. [SuOd]: Yasoji Suzuki, Kaichiro Odagawa, Toshio Abe, "Clocked CMOS Calculator Circuitry", *IEEE Jour. Solid-State Circuits*, Dec. 1973, pp 462–469.
26. [Sze]: S.M. Sze, *VLSI Technology*, MacGraw–Hill, 1988.
27. [YuSv1]: Jiren Yuan, Christer Svensson, "High-Speed CMOS Circuit Technique", *IEEE Jour. Solid-State Circuits*, Feb. 1989, vol-24, pp 62–70.
28. [YuSv2]: Yuan Ji–Ren, Ingemar Karlsson, Christer Svensson, "A True Single-Phase-Clock Dynamic CMOS Circuit Technique", *IEEE Jour. Solid-State Circuits*, Oct. 1987, vol sc-22, pp 899–901.

END

24| 05| 93|

FIN