

DISSERTATION

MIXED PTL/STATIC LOGIC SYNTHESIS USING GENETIC ALGORITHMS:
THEORY AND APPLICATIONS

Submitted by

Geun Rae Cho

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements
for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring, 2003

UMI Number: 3092660

UMI[®]

UMI Microform 3092660

Copyright 2003 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

COLORADO STATE UNIVERSITY

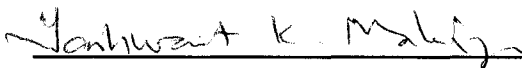
October 29, 2002

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY GEUN RAE CHO ENTITLED MIXED PTL/STATIC LOGIC SYNTHESIS USING GENETIC ALGORITHMS: THEORY AND APPLICATIONS BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

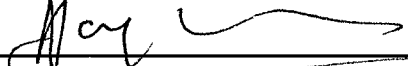
Committee on Graduate Work



Committee Member



Committee Member



Committee Member



Advisor



Department Head



ABSTRACT OF DISSERTATION

MIXED PTL/STATIC LOGIC SYNTHESIS USING GENETIC ALGORITHMS: THEORY AND APPLICATIONS

With the increasing demand of high-performance and high-density VLSI (Very Large Scale Integrated circuits) designs, technology scaling has been meeting the increasing demand and will continue in UDSM (Ultra Deep-SubMicron) regimes. As the complexity of a chip increases almost exponentially as CMOS (Complementary Metal-Oxide Semiconductor) technology scales to under $100nm$, the difficulty of designing VLSI system has increased. Thus, the gap between design productivity and technology capability is increasing. Because of this gap, efficient and practical EDA (Electronic Design Automation) solutions have become more and more relevant in ensuring successful chip designs and time-to-market.

Static CMOS logic style has long been used widely to realize a VLSI system because of ease to use and well-developed synthesis methods. With power becomes an increasingly limiting factor in high density and high-performance VLSI designs, a great deal of effort has been made to explore low-power design options without sacrificing performance. Pass-transistor logic (PTL) is being considered as an alternative logic style of static CMOS because static logic consumes a large amount of power due to large short circuit current. Generally, PTL consumes less power than static logic because of its small capacitive load and smaller area. However, PTL only circuits would increase the circuit delay caused by a long transistor chain in series.

In this dissertation, a new mixed pass-transistor logic (PTL) and static CMOS logic synthesis method is presented. The main advantage of mixed PTL/Static circuits is that the static CMOS acts as a buffer as well as performs a logic function. The proposed synthesis method searches for possible matches between a logic structure and a set of predefined PTL/Static logic gates using Binary Decision Diagrams (BDDs). This proposed mixed PTL/Static synthesis method also optimizes the final mapped circuits on a global level by using genetic algorithms (GAs). Our experimental results demonstrate that circuits synthesized using our proposed mixed PTL/Static synthesis method outperform their static counterparts in delay or power consumption, or both in various CMOS technologies. The impacts of technology scaling on mixed PTL/Static circuits in terms of performance and power consumption are also examined by means of both theoretical projections and experiments. The projection and experimental results give more evidence that the mixed PTL/Static circuit style is a promising alternative to static and domino circuit styles for high-performance and low-power applications in the future.

Geun Rae Cho
Department of Electrical and Computer Engineering
Colorado State University
Fort Collins, Colorado 80523
Spring, 2003

ACKNOWLEDGEMENTS

I would like to express my heartfelt thanks to my advisor, Dr. Thomas Wei Chen for his great enthusiasm, constructive criticism, and encouragement throughout my Ph.D. program. His invaluable assistance on works at Hewlett Packard, writing and presenting research results is also greatly appreciated.

I want to thank Dr. Gary Robinson, Dr. Anura P. Jayasumana, and Dr. Yashwant Malaiya for their invaluable advice, assistance, and encouragement as committee members. Also, I would like to give special thanks to System VLSI Technology Center at Hewlett Packard Co. in Fort Collins for their partial support for this research.

My warmest thanks go to my graduate and undergraduate school advisors in Korea, Dr. Euy-Yeol Park at Pusan National University and Dr. Tag-Gon Kim at Korea Institute of Advanced Science and Technology, for their invaluable suggestions and encouragement during my collage years as well as Ph.D. program.

I would like to thank my colleagues and friends, Alkan Gengiz, Dr. Jinsang Kim, and Dr. Amjad Hajar in the VLSI System Architecture Laboratory for their support and suggestion. I also thank Korean friends in our department, Dr. Younghoon Kim, Yoonggoog Cho, and Sanghoon Lim for their friendship and encouragement. Also, I would like thank Hyuk-Sang Kwon, Kwang-Seok Oh, and my colleagues and friends who were with me in Samsung Electronics Co. from 1986 to 1997, for their encouragement.

I am always thankful for the love and support of my parent-in-law, Soonshin Song and Jinak Kim, my brothers, Seongrae, Samrae, and Hongcheol, and my sisters, Deognam and Deugim, and my brother-in-law, Chang-Gab Kim. Thanks, father and mother. I miss you so much, Mom and Dad. They had sacrificed their whole life for me, and could not see me from a closer distance during my Ph.D program. All I can

remember is their love and patience. Finally, my special thanks go to my beautiful son and daughter, Hynusu and Eunjeong (Helen), and my lovely wife, Mija Song for their support, trust, and love.

DEDICATION

To the memory of my father, Chunje Cho (1932-1982),
and mother, Soonrye Oh (1937-1998),
to my brother Seongrae, my sister Deognam, my parent-in-law,
and most especially, my wife, Mija Song.

CONTENTS

1	Introduction	1
1.1	Background	1
1.1.1	System and software level	1
1.1.2	Architecture or behavioral level	2
1.1.3	Logic level	2
1.1.4	Circuit level	3
1.2	Problem Descriptions	4
1.3	Research Objectives and Organization of the Dissertation	6
1.3.1	Objectives	6
1.3.2	Organization	7
2	On mixed PTL/Static logic	8
2.1	Introduction	8
2.2	Mixed PTL/Static Circuit	9
2.2.1	Properties of Pass-Transistor Chains	10
2.2.2	Conventional PTL and Static Circuits	11
2.2.3	PTL/Static Hybrid Circuit Implementation	12
2.2.4	Simulation Results of mixed PTL/static Logic	15
2.2.5	Summary on mixed PTL/static Logic	17
2.3	PTL Cell Types	18
2.3.1	Bulk CMOS PTL Cells	18
2.3.2	Simulation and Analysis of Single-Rail PTL Cells	21
2.3.3	Dual-Rail PTL Cells	25
2.3.4	PTL Cells for Floating-Body PDSOI	26
2.4	SOI versus DTMOS	27
2.4.1	SOI	27
2.4.2	Dynamic Threshold-Voltage CMOS	29
2.4.3	Pass-transistor Logic in PD SOI and DTMOS	32
2.4.4	Summary on SOI Pass-Transistor Logic	42
2.5	Conclusions	43

3	Boolean Matching	45
3.1	Introduction	45
3.2	Overview of the proposed Logic Synthesis Flow	46
3.3	Structural and Boolean Matching	47
3.4	Logic and Graph Decomposition	48
3.5	Cluster and Its Function	50
3.6	Equivalence Checking	51
3.7	Symmetry Set Finding	52
3.8	Pin Assignment Based On Signal Depth	55
3.8.1	Data structure of a match	55
3.8.2	Pin mapping algorithm	56
3.9	Matching Algorithm	59
3.10	Conclusions	61
4	Covering Using Genetic Algorithms	63
4.1	Introduction	63
4.2	Graph Covering Problem	64
4.3	Previous Works and Limitations	66
4.4	Overview of the proposed covering algorithm	68
4.5	GA Encoding	69
4.6	Initializing sub/global populations	71
4.6.1	Sub-Chromosome and Sub-population	71
4.6.2	Global Chromosome and Global Population	73
4.7	GA Operators	74
4.7.1	Crossover Operation	74
4.7.2	Mutation Operation	77
4.8	Covering Algorithm	79
4.9	Evaluation of GA Parameters	83
4.10	Performance Enhancement	85
4.10.1	Delay Model	85
4.10.2	Redundant Buffer Removal	86
4.10.3	Fanout optimization	88
4.10.4	Experimental Results of Performance Enhancement	90
4.11	Miscellaneous	92
4.12	Conclusions	97
5	Technology Scaling Impact on mixed PTL Circuits	98
5.1	Introduction	98
5.2	Technology Scaling	100
5.2.1	ITRS Road Map	100
5.2.2	Technology Scaling Theory	100
5.3	Delay Models for Different Circuit Styles	103
5.3.1	Static and Dynamic Logic Delay	104

5.3.2	Delay Models for Pass-Transistor Logic Gates	107
5.4	Power Consumption Model	109
5.4.1	Leakage Current Model	109
5.4.2	Stacking Effects of Different Logic Styles	112
5.5	Projection Results	115
5.6	Conclusions	119
6	Experimental Results	122
6.1	Experimental Results on Mixed PTL and Static CMOS	123
6.1.1	Experimental Setup	123
6.1.2	Experiment Results in a $0.25\mu m$ Bulk CMOS Process	124
6.1.3	Experiment Results in a $0.1\mu m$ Bulk CMOS Process	127
6.1.4	Experiment Results in a $0.13\mu m$ SOI Process	131
6.2	Experimental Results on Technology Scaling and Leakage	138
6.2.1	Experimental Setup	138
6.2.2	Delay and Dynamic Power	139
6.2.3	Experimental Results on Leakage Power Consumption	143
6.3	Conclusions	147
7	Conclusions and Future Work	148
7.1	Conclusions	148
7.2	Future Work	150
A	Abbreviations and Acronyms	151
B	Notation	152
C	Source Codes	153
	Bibliography	227

LIST OF FIGURES

2.1	Pass-transistor chain delay (HSPICE simulation results in a $0.25\mu\text{m}$ CMOS technology)	11
2.2	Logic implementations of equation (2.2) in conventional PTL and pure static circuits.	13
2.3	PTL cells and mixed PTL implementation of equation (2.2).	13
2.4	Different PTL Cell Types.	18
2.5	Two different full-adder implementations.	20
2.6	Full-adder simulation waveform for three different PTL cell types in $0.25\mu\text{m}$ technology ($V_{dd} = 2.5V$). (a)input waveform, (b)sum output, (c)carry-out waveform, and (d)current waveform.	22
2.7	Simulation results of PTL circuits in a $0.13\mu\text{m}$ CMOS process. The waveforms O_1 and O_2 represent the outputs of Figure 2.4 (a) and (d), respectively. The waveforms X_1 and X_2 represent the node X of Figure 2.4 (a) and (d), respectively. And I_1 and I_2 represent the current of Figure 2.4 (a) and (d), respectively.	24
2.8	Three single-rail PTL+ cell types.	25
2.9	Single-rail and dual-rail PTL+ Type1 cells and their variations.	26
2.10	Sources of body charging mechanism in partially depleted floating body SOI. (a)impact ionization. (b)junction leakage. (c)capacitive coupling.	28
2.11	DTMOS structure[1, 2]. (a)Cross section of an SOI nMOS with body and gate tied together. (b)Gate-to-body connection using aluminum to short the poly gate to the p+ region.	30
2.12	Different types of DTMOS Inverter. (a)Conventional DTMOS. (b) and (c) DTMOS with limiting transistors.	31
2.13	Different body bias control circuits for DTMOS.	33
2.14	The body voltage of main transistors in Figure 2.13	35
2.15	Normalized average delay of pass-transistors with different body control scheme.	36
2.16	DTMOS PTL using symmetrical and ADTPT structure. (a)Symmetrical structure. (b)ADTPT structure.	37
2.17	The body voltages of the pMOS and nMOS main transistors. $V_z(1)$, $V_z(2)$, and $V_z(3)$ represent the output voltage of DTPTL+, Symmetrical, and ADTPT PTL gates, respectively.	38

2.18	(a)DTPTL+ structure. (b)A variation of DTPTL+ when signal B is connected to ground.	38
2.19	Normalized average delay vs. supply voltage.	41
2.20	Normalized average power vs. supply voltage.	42
2.21	Normalized average power-delay-product vs. supply voltage.	43
2.22	Normalized average energy-delay-product vs. supply voltage.	44
3.1	Top level flow of the proposed synthesis method. PTL and DTL represent single-rail and dual-rail mixed PTL circuits, respectively.	46
3.2	The difference between graph matching and Boolean matching. (a) graph representation of f_1 . (b) graph representation of f_2 . (c) BDD representation of both f_1 and f_2	49
3.3	Example of subject graphs generated based on multiple fanout points. S_i represents a subject graph.	51
3.4	Data structure of a match and static and PTL match list.	56
4.1	Logic network and matches. (a)Logic network. (b)Static and PTL matches.	64
4.2	Flow chart of the proposed covering algorithm.	68
4.3	An example of Graph Encoding for GA. (a)Genes. (b)Sub-chromosomes. (c)A global-chromosome.	71
4.4	The hierarchical relationship between sub-chromosome and global chromosome.	72
4.5	Crossover operation.	75
4.6	Mutation Process. (a)before mutation. (b)after mutation.	78
4.7	Example of mapped graph crossover.	81
4.8	Example of a mapped graph mutation.	82
4.9	Number of iterations to converge (C499, $N_{subpop} = 25$, $N_{gpop} = 15$).	83
4.10	Convergence value and number of iterations to converge with respect to various genetic parameters. (a)Normalized convergence value with respect to sub-population size (C499, $\delta = 0.85$, $\mu = 0.35$, $N_{gpop} = 15$). (b)Number of iterations to converge with various mutation rate and global population size (C499, $\delta = 0.85$, $N_{subpop} = 25$).	84
4.11	Convergence result of the proposed method. (a)Convergence result of C6288. (b)Convergence result of C1355.	85
4.12	Delay reducing method. (a)before reduction. (b)after reduction.	88
4.13	Critical path isolation fanout optimization method. (a)before optimization. (b)after optimization.	89
4.14	Synthesis based ASIC design flow.	93
4.15	Possible signal connection patterns around latch. In each case, the signal connected to latch is the candidate of the primary input or output of the new combinational block.	94

4.16	Proposed mixed PTL design flow that is incorporated into a general ASIC design flow.	96
5.1	Pass-transistor logic operational modes. (a) and (c) are voltage and current relationship for logic 1 and logic 0 transfer, respectively. (b) and (d) are the operating regions for logic 1 and logic 0 transfer, respectively.	107
5.2	Static, Dynamic, and PTL logic implementation of a OR gate. (a) , (b), and (c) represent static, dynamic, and PTL, respectively. For PTL, it is assumed that the input b is connected to V_{dd}	112
5.3	Gate capacitance, junction capacitance, and area projection results. . . .	115
5.4	Delay projection results.	116
5.5	Dynamic power projection results.	117
5.6	Sub-threshold leakage and tunneling leakage power projection results. . .	118
5.7	Overall power projection results. (a) Total leakage power. (b) Total power including dynamic, sub-threshold leakage, and tunneling power. . . .	119
5.8	Power-delay-product projection results.	120
5.9	Energy-delay-product projection results.	120
6.1	Normalized area, delay, and power on ISCAS85 benchmark circuits in $0.1\mu m$ CMOS process. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.	129
6.2	Normalized area, delay, and power on the circuits from a microprocessor in $0.1\mu m$ CMOS process. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.	131
6.3	Normalized area, delay, and power of the benchmark circuits from a microprocessor in $0.13\mu m$ bulk CMOS process. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.	135
6.4	Normalized area, delay, and power of the benchmark circuits from a microprocessor in $0.13\mu m$ SOI process. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.	135
6.5	Normalized area, delay, and power of ISCAS85 benchmark circuits in $0.13\mu m$ SOI process. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.	136
6.6	Normalized delay and power in $0.13\mu m$ SOI and bulk CMOS processes. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.	137
6.7	Power-Delay curve of four different implementations of a circuit in $0.18\mu m$.	139
6.8	Group 1 normalized average (a)delay and (b)power.	140
6.9	Group 2 normalized average (a)delay and (b)power.	141
6.10	Group 3 normalized average (a)delay and (b)power.	142
6.11	Overall normalized average (a)delay and (b)power.	143
6.12	Overall normalized average (a)power-delay-product and (b)energy-delay-product in each technology node.	144

6.13	Normalized measured and estimated leakage current of ISCAS85 benchmark circuits.	145
6.14	Normalized measured and estimated leakage current of the benchmark circuits from a microprocessor.	146

LIST OF TABLES

2.1	Benchmark Circuits	15
2.2	HSPICE simulation results using $0.25\mu m$, $V_{dd} = 2.5V$	16
2.3	FA Simulation Result of PTL Cells in a $0.25\mu m$ CMOS technology. . . .	21
4.1	Characteristics of benchmark circuits.	91
4.2	Experimental results of applying the redundant buffer removal method with a fanout optimization method.	91
4.3	Experimental results of applying the redundant buffer removal method with a fanout optimization method(ISCAS85 benchmark circuits.) . .	92
4.4	Primary input and output relationship between original and sequential part removed block.	95
5.1	Key Overall Chip parameters for High-performance Logic(data from ITRS 2001 road map).	101
5.2	Scaling relationship for devices.	102
5.3	The leakage current flow of static and dynamic gates in Figures 5.2 (a) and (b), respectively, with different input combinations.	113
5.4	The leakage current flow of a PTL cell in Figure 5.2 (c) with different input and output combinations.	114
6.1	Synthesis results using TSMC $0.25\mu m$ CMOS process, $V_{dd} = 2.5V$	125
6.2	The number of iterations to reach convergence and cost improvement. . .	126
6.3	Experimental results on ISCAS85 benchmark circuits in $0.1\mu m$ CMOS process.	128
6.4	Experimental results on the benchmark circuits from a microprocessor in $0.1\mu m$ CMOS process.	130
6.5	Experimental results on the benchmark circuits from a microprocessor in $0.13\mu m$ bulk CMOS process.	132
6.6	Experimental results on the benchmark circuits from a microprocessor in $0.13\mu m$ partially depleted SOI process.	133
6.7	Experimental results on ISCAS85 benchmark circuits in $0.13\mu m$ partially depleted SOI process.	134
C.1	Summary of source codes for mixed PTL/Static logic synthesis.	153

Chapter 1

Introduction

1.1 Background

Static CMOS logic style has widely been used to realize a VLSI system because of ease to use and well developed synthesis and analysis methods. With power being increasingly a limiting factor in high density and high-performance VLSI designs, a great deal of effort have been made to explore low-power design options without sacrificing performance.

Power reduction techniques can be implemented at various levels in a system. These techniques fall into four general categories: system and software level, architecture or behavioral level, logic level, and circuit level power reduction. This section provides an overview of these four techniques.

1.1.1 System and software level

System and software level power reduction methods relate chiefly to the reordering of the instructions used in a program or the efficient use of registers. Because many mappings from a high-level source code into machine code exist, the optimization method of a compiler or a code-generator can impact power consumption. If a power cost for every instruction is available, the power that a system consumes can

be reduced by reordering the instructions appropriately in the generated code[3]. Because register access is much cheaper than memory access, efficient register allocation can significantly affect power consumption.

1.1.2 Architecture or behavioral level

Architecture or behavioral level low-power technique tries to optimize the overall system power by considering various design alternatives when a system is first being designed. Such analysis can help designers choose the best resource allocation for a given system. Because every architecture resource consumes power, the optimized resource allocation can reduce power consumption.

1.1.3 Logic level

Logic level power reduction methods include several approaches. One is combinational logic, which can be broken down into two phases: technology-independent optimization and technology-dependent optimization.

Technology-independent optimization manipulates logic equations to reduce power dissipation. This optimization method works primarily by means of factorization of logical expression. For example, $ac + ad + bc + bd$ can be factored into $(a + b)(c + d)$, considerably reducing transistor count. Additionally, common subexpressions can be found across multiple functions and reused. In this approach, kernels[4] are usually generated so that the literal counts are reduced. When targeting power consumption, because the cost function is not only the literal count, but also the switching activity, modified kernel extraction methods that target switching activity are proposed in [5]. After a technology-independent optimization of a set of logic equations takes place, the results generally must be mapped in a feasible circuit that optimizes power while it also meets delay constraints.

Don't care optimization is a component of technology-independent optimization. The don't care optimization method exploits controllability and observability don't cares associated with a combinational circuit. Don't cares can be exploited to improve delay[6] or to reduce switching activity and power consumption[7].

Technology-dependent optimization is often called technology mapping. Because technology mapping has the role of finalizing circuit synthesis by performing the gate selection from a given library, technology mapping is a crucial step in logic synthesis.

1.1.4 Circuit level

At the circuit level, static CMOS logic style has been widely used to realize a VLSI system because of its ease of use and its well-developed synthesis and analysis methods. Pass-transistor logic (PTL), first proposed in [8, 9], has attracted much attention from early on. Because the overall capacitance in PTL is less than that of static CMOS logic, PTL generally consumes less power than static CMOS logic. The fact that the transistor's junction capacitance is smaller than gate capacitance is the main reason for the lower overall capacitance of PTL. However, some issues exist about using PTL circuit styles. For instance, a PTL-only circuit would significantly increase the circuit delay caused by long pass-transistor chains. To alleviate this drawback, conventional PTL synthesis methods[10, 11] use dedicated buffers that are inserted every fixed number of pass-transistors in series. However, these buffers can cause a large amount of power consumption and performance degradation.

Mixing PTL with static CMOS has been proposed[12, 13, 14, 15] as an alternative low-power circuit style; this is known as a mixed PTL/Static CMOS circuit. Unlike conventional PTL design, where dedicated buffers are inserted in pass-transistor trees to restore driving strength, the mixed PTL/Static circuit allocates a certain number of static gates at optimal locations within pass-transistor trees to boost driving strength

as well as perform logic functions. Consequently, circuit performance and power consumption of mixed PTL circuits are further improved in comparison to those in conventional PTL circuits. Mixed PTL/Static circuits can take full advantage of both static CMOS and pass-transistor circuit styles and are considered as a promising alternative to static CMOS circuits for use in low-power applications because their use has no performance penalty. Mixed PTL circuit styles are not widely used because the synthesis environments for mixed PTL circuits are not mature in comparison to static synthesis environments. However, mixed PTL synthesis remains very important for achieving low-power and high performance VLSI system.

1.2 Problem Descriptions

Designing mixed PTL circuits for low-power and high-performance highly depends on two tasks: selection of PTL cells and a synthesis technique to produce a mixed PTL structure.

The choice of PTL cells directly impacts the Boolean matching process in the synthesis phase and significantly impacts overall quality of the final synthesized results. In [13, 14, 16], nMOS-only and pMOS-only pass transistor trees are used in PTL cells to reduce cell size. The full rail-to-rail swing of the output signal is restored by the extra level restoring logic at the output of a PTL gate. The existence of level-restoring logic at the output of PTL gates not only slows down the PTL gates due to potential drive-fights, but also increases their power consumption.

During logic synthesis, technology mapping play an important role in reducing power. Because the technology mapping process is to select the best match among the possible different matches at each node of a given decomposed circuit using base functions such as NANDs and INVs, the applied selection method is very important.

The objective function of the mapping process can be area or power under delay constraints, or vice versa.

Dynamic programming[17] has been widely used to solve the technology mapping problem[18, 19, 20, 21]. This method is efficient for an area optimization. The method's main drawback, however, is that it does not guarantee the optimality for timing or power because the load capacitance of a node is not determined yet when a match is considered. Therefore, the delay cannot be calculated at the time of the mapping. In [19], an improved method has been proposed to estimate delay during the mapping process. First, n load bins are created at each node and delay is calculated in the postorder tree traversal. For each bin, the match that gives minimum arrival time at a node output is stored. The final selection is made during preorder traversal by selecting matches depending on the load seen at each node.

In [20, 21], the author proposed an area-delay trade-off curve that utilizes only non-inferior area-delay points, but the load capacitances are still not determined during the postorder traversal. The search for optimal mappings in [19, 20, 21] was applied only within a sub-tree of a given circuit represented by a directed acyclic graph (DAG). Sub-trees are separated by multiple fanout points in the DAG. The cell interactions between sub-trees were not considered, and no attempt was made to optimize the overall circuit at the global level. A greedy search algorithm proposed in [13] to determine the best mappings for mixed PTL/Static circuits also ignores global level optimization.

The limitations of the existing methods can be summarized as follows: (1) The use of PTL cells with nMOS or pMOS only in the signal path significantly impacts power consumption and circuit performance. (2) Technology mapping algorithms using dynamic programming or greedy methods ignore global level optimization. (3) The lack of global view in technology mapping will result in poor quality of the final mapped circuits in terms of power consumption and performance.

1.3 Research Objectives and Organization of the Dissertation

1.3.1 Objectives

The objectives of this research are to explore appropriate PTL cell types for mixed PTL/static logic and to develop a new technique for mixed PTL/Static CMOS circuit synthesis method that can optimize the circuits at global level. Also, the scalability of mixed PTL, including performance, dynamic power consumption, and leakage power consumption, will be examined.

The proposed method is as follows. First, appropriate PTL cell types for mixed PTL/Static circuits will be explored through circuit simulation methods. Second, for the technology mapping method, a circuit will be divided into sub-Boolean functions using multiple fanout points. Each sub-Boolean function will then be mapped to a set of potential static and PTL cells using well-known BDD-based matching techniques[22]. Therefore, each sub-Boolean function may have many legal mappings of mixed PTL and static cells. The global-level search for the best combination of PTL gates and static gates for the given circuit will be achieved by the use of genetic algorithms (GAs).

There are concerns about the ability of PTL to scale well with technology scaling, particularly with V_{dd} scaling. The scalability of mixed PTL, including performance, dynamic power consumption, and leakage power consumption, is not well understood. The existing published results did not address these concerns, because almost all of the previous work on PTL was performed on the $0.25\mu m$ technology generation or older. With current process generation moving beyond the $0.25\mu m$ generation and into the $0.18\mu m$, $0.13\mu m$, and $0.1\mu m$ generations, this paper attempts to directly address the issues of PTL scalability by projecting the trends of performance, dynamic

power consumption, and leakage power consumption on future CMOS technologies beyond $0.1\mu m$, and presenting new comparative results using more up-to-date $0.18\mu m$, $0.13\mu m$, and $0.1\mu m$ CMOS technologies.

1.3.2 Organization

The organization of the dissertation is as follows. In Chapter 2, first, three different logic implementations including mixed PTL/Static logic implementations are examined and preliminary experimental results are presented. Then, PTL cell types that are appropriate for mixed PTL circuits will be explored in bulk CMOS and SOI devices. Boolean matching using BDDs will be discussed and implemented in the Chapter 3. In Chapter 3, the proposed logic mixed PTL/Static logic synthesis is also presented. In Chapter 4, the proposed mixed PTL/static logic covering method based on genetic algorithms and the performance enhancement methods are presented. In Chapter 5, the scalability of mixed PTL circuit styles will be presented by projecting the performance and power consumption of the mixed PTL circuits for the future technology generations. In Chapter 6, first, the experimental results on the mixed PTL/Static and static CMOS circuits in a $0.25\mu m$, $0.1\mu m$ CMOS, and a $0.13\mu m$ SOI technologies will be presented. Second, the impact of technology scaling on mixed PTL circuits will be presented by comparing with the results of static and dynamic CMOS circuit styles using three different modern CMOS technologies. Finally, the experimental results on leakage power consumption will be presented. Finally, concluding remarks and future work will be given in Chapter 7.

Chapter 2

On mixed PTL/Static logic

2.1 Introduction

This Chapter has two purposes. The first is to present more evidence in a deep-submicron (DSM) CMOS processor to validate the advantage of mixed PTL circuits. The second is to explore appropriate PTL cell structure for mixed PTL circuits in a deep-submicron CMOS process and a deep-submicron silicon-on-insulator (SOI) processor.

Conventional static CMOS logic gates have been widely used in ASIC (application-specific integrated circuit) design today mainly due to the advantages of easy synthesis and a well-developed synthesis and analysis environment. Pass-transistor logic (PTL), which was first proposed in [8, 9], attracted a lot of attention early on. In general, PTL has less overall capacitance than static CMOS because of small area, and consumes less power than static CMOS. The advantage of PTL as an alternative low power circuit design approach in $0.5\mu\text{m}$ and $0.3\mu\text{m}$ CMOS technology has been well documented in [23, 24]. However, the advantage of PTL in deep-submicron CMOS technology is not well understood yet. There are speculations [23] that with aggressive scaling in transistor critical dimension and supply voltage, the advantages of PTL over conventional static CMOS gates will gradually diminish.

The adoption of PTL is further exacerbated by the lack of synthesis environment to support the use of PTL gates. The well-known method of synthesizing the PTL circuit from a set of logic equations was proposed in [10]. The use of binary decision diagrams (BDDs)[25, 26, 27] made the synthesis process easy and straightforward. To minimize delay, a buffer is inserted every fixed number of transistors in series in the PTL chain. The process of buffer insertion can also be viewed as the process of BDD decomposition as suggested in [11], where buffers are inserted at the root of each decomposed BDD.

For complex logic functions, the number of buffers to be inserted in a PTL circuit can be significant. One way to alleviate this problem is to use static logic gates to both act as buffer and perform logic functions. Mixed PTL with static logic gates has been proposed in [12]. This chapter presents more evidence in a deep-submicron CMOS process to validate the benefit of mixed PTL circuits. By eliminating explicit buffering between PTL stages, mixed PTL circuits can further improve the low power nature of PTL circuits without sacrificing performance.

The remainder of this chapter consists of four sections. In section 2.2, pure static and conventional PTL will be reviewed, then a mixed PTL/Static logic structure will be presented to show more evidence of mixed PTL/Static's potential advantages. Because PTL cell type is very important for mixed PTL, an appropriate PTL cell type for the mixed PTL circuits will be presented in section 2.3. Finally, the behavior of PTL cells in silicon-on-insulator (SOI) CMOS process will be reviewed and a new SOI DTMOS PTL structure will be presented in section 2.4.

2.2 Mixed PTL/Static Circuit

In this section, we compare mixed PTL circuits with conventional PTL and pure static circuits in terms of power, delay, and power-delay-product (PDP). The mapping method for each circuit style will be described briefly.

2.2.1 Properties of Pass-Transistor Chains

The average dynamic power, P_{av} , consumed by a CMOS gate is given by

$$P_{av} = V_{dd}^2 \cdot f \cdot C_L \cdot A_i \quad (2.1)$$

where V_{dd} is the supply voltage, f is the operating frequency, C_L is the total load capacitance driven by the gate, and A_i is the switching activity of the gate¹. V_{dd} , f , and A_i are fixed. Decreasing C_L will have a positive impact on power consumption. Since the input capacitance of a PTL gate is smaller than the input capacitance of a static CMOS gate, replacing CMOS gates with PTL gates should result in a reduction in the power consumed.

We cannot arbitrarily connect PTL gates together, however, because the delay through pass-transistors is a quadratic function of the number of transistors in series. The delay in PTL can be estimated as $t_d = \tau_n \cdot N^2$ where τ_n and N are the time constant and the number of transistors in series, respectively[28]. This quadratic dependence can be seen in Figure 2.1 where the results from an HSPICE[29] simulation of a $0.25\mu\text{m}$ process are shown. The delay is plotted versus the number of nMOS transistors in series in Figure 2.1. Figure 2.1 plots the percentage of increase in delay over one pass-transistor against the number of transistors in series. These graphs show that the delay is unacceptably high when the number of pass-transistors is three or more. Therefore, we limited ourselves to a maximum of three pass-transistors in series without an intervening static gate for this process.

Another issue with pass-transistors is that they do not have full output voltage swing. The high output voltage is limited to $V_{dd} - V_{tn}$ for an nMOS transistor chain

¹In this case, we ignored the power consumed by short circuit.

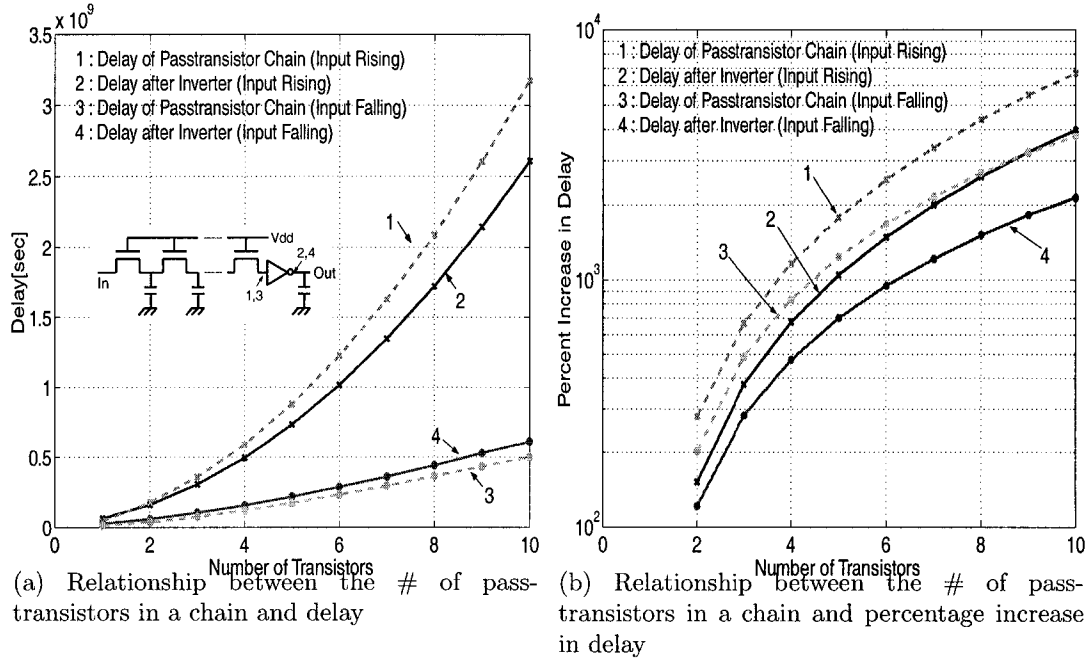


Figure 2.1: Pass-transistor chain delay (HSPICE simulation results in a $0.25\mu\text{m}$ CMOS technology)

where V_{tn} is the nMOS threshold voltage. We had hoped to use this limited voltage swing to reduce the dynamic power consumption. However, this output voltage was not high enough to completely turn off the pMOS transistors in the following static gates, and, therefore, the static current became unacceptably high. To fix this problem, an inverter and a pull-up pMOS transistor had to be added at the output of each PTL cell to restore the swing all the way to V_{dd} . The size of inverter and pull-up pMOS was chosen in such a way that the loading effect at the cell output is minimized. This arrangement is shown in the inset of Figure 2.2(a).

2.2.2 Conventional PTL and Static Circuits

One of the purposes of this chapter is to compare a type of mixed PTL structure with the conventional PTL structure. To implement a given set of Boolean functions in a conventional PTL circuit, we followed a method similar to the one described in [10]. In this method, the logic equations are first converted to a reduced and ordered

BDD (ROBDD)[26, 27]. This BDD is then converted to a PTL circuit simply by replacing each node with two nMOS pass transistors in a Y-shaped pattern, as shown in the inset in Figure 2.2(a). This is a straightforward method, and it assures that only one path is on at a time. The final step is to add buffers after every two pass-transistors in series for the reasons outlined in Section 2.2.1. The buffers use pull-up transistors to restore full swing at the output of the pass-transistors.

As an example of a conventional PTL circuit, the Boolean equation

$$f = y'_0 s'_2 s'_1 x'_2 + s_1 s_0 x_2 x_1 + s'_2 x_2 x_1 x_0 + y_0 s'_2 s_1 s_0 x_2 \quad (2.2)$$

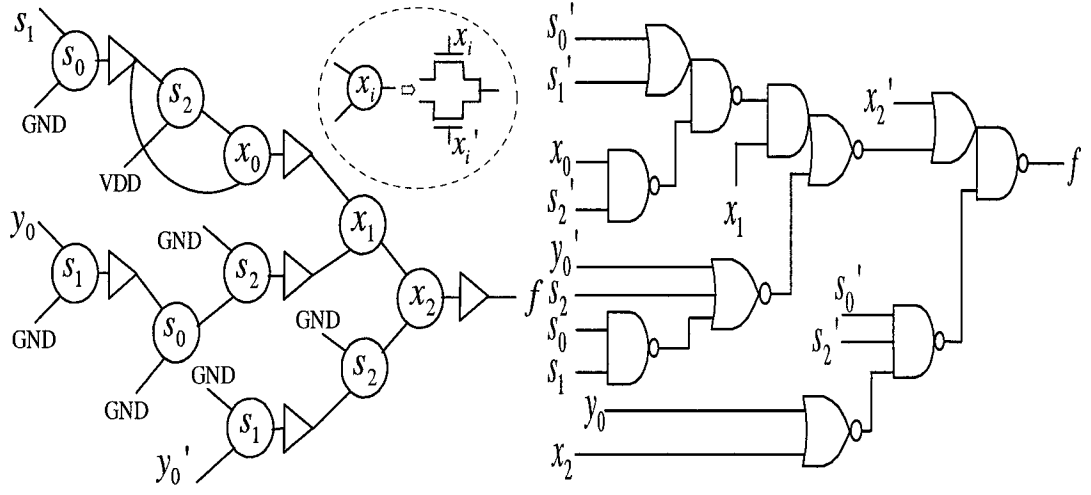
was mapped using the above process. The results are shown in Figure 2.2(a).

For the pure static implementation, we used the SIS program, as described in [30, 31]. Since technology mapping is an NP-hard problem[32, 33], SIS uses a set of heuristic algorithms. The first algorithm decomposes the given Boolean function into a graph containing only simple gates, such as 2-input NAND's and inverters. In the second step, this large graph is then partitioned into smaller pieces called subject graphs[19, 34] to make the third step tractable. The third step is called covering[19, 34, 22] and consists of trying to match portions of the subject graph to graphs created from the cell library. The cell library used by SIS contains only static CMOS gates. In the final step, the algorithm chooses which library cells to use, usually based upon some sort of timing constraint.

SIS was used to implement the function f given in (2.2). The result is shown in Figure 2.2(b).

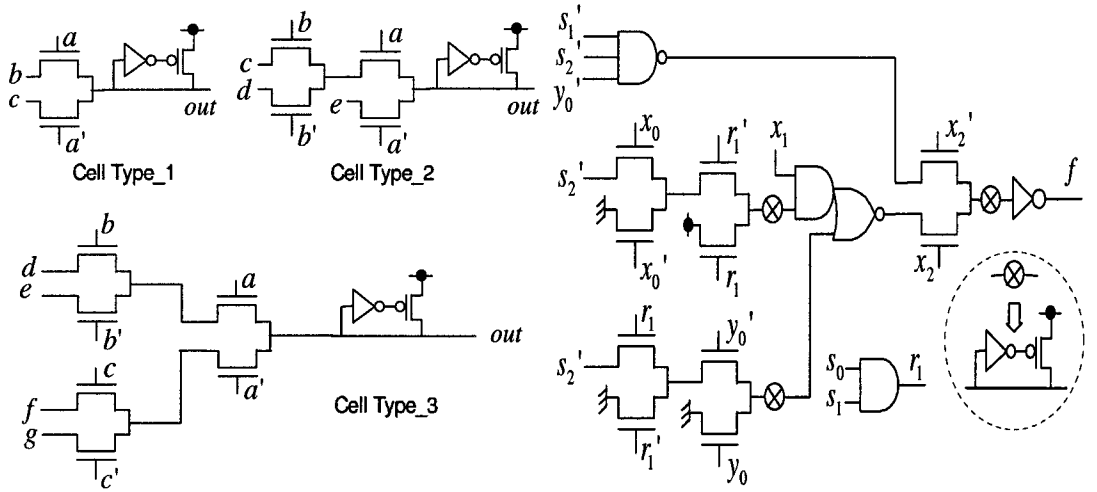
2.2.3 PTL/Static Hybrid Circuit Implementation

The approach used in this chapter to the mixed PTL/Static circuit implementation is similar to the method as described in SIS[30, 31]. The mapping process begins



(a) Conventional PTL logic implementation (b) Pure Static logic implementation

Figure 2.2: Logic implementations of equation (2.2) in conventional PTL and pure static circuits.



(a) Pass-transistor logic cells (b) Mixed PTL/Static logic implementation

Figure 2.3: PTL cells and mixed PTL implementation of equation (2.2).

by decomposing the Boolean functions into a graph containing only simple gates called base functions: 2-input AND's, 2-input OR's, and inverters in our case. The primary purpose of this step is to express all local functions as simple functions and is required to ensure that every node is covered by at least one library cell. In order to ensure the existence of a solution, the library has to include the base functions.

During partitioning, the large graph is broken into subject graphs. Since the technology mapping problem is NP-complete in nature, we can reduce the problem size, and the problem becomes tractable by breaking the graph into cones. We have chosen multiple fanout points as the boundary of the subject graphs.

We used a similar covering method as that used in SIS. Our method differs from the SIS method in two ways. First, three PTL gates are added to the library of static gates. Second, the static gate is used both to perform logic functions and to act as buffers between PTL gates. These PTL gates have a maximum of two pass-transistors in series. Figure 2.3(a) shows the schematic of these three cells. With the addition of the PTL gates, some new rules must be added to the conventional covering scheme to generate a mapped circuit after finding possible matches at every node in the subject graph. In this mixed PTL implementation, we used the following rules. First, all inputs to a PTL cell must be from static cells. Second, all outputs from a PTL cell must be to static cells. These rules assure that two PTL cells are never connected in series, and thus no more than two pass-transistors are in series at any point in the circuit. These rules need to be followed to keep the delay at acceptable levels as discussed in Section 2.2.1.

Because the main purpose of discussing this mixed PTL approach is to examine the ability of the mixed PTL circuit in power and performance, the details of the mapping process are not presented in this chapter. Instead, the details of the mixed PTL mapping process and algorithm will be presented in Chapters 3 and 4. A sample

of mixed PTL implementation is shown in Figure 2.3(b), where the function f given in equation (2.2) is implemented.

2.2.4 Simulation Results of mixed PTL/static Logic

In order to make a comparison, three different benchmark circuits taken from a microprocessor design are implemented in each of the three logic styles: conventional static, conventional PTL, and mixed PTL/static. The characteristics of three circuits are summarized in Table 2.1. These three circuits are implemented using a commercial $0.25\mu\text{m}$ CMOS process with $V_{dd} = 2.5V$. For each of the three circuits, we randomly chose five paths to compare delay, power consumption (under different input setup to activate the given path), and power-delay product. The transistor sizes in all the circuits were determined such that rise and fall time of all the signal nodes in the circuits are in the range between 300ps and 400ps.

Table 2.1: Benchmark Circuits

	No. of PIs	No. of POs	No. of Gates	Circuit Function
Circuit 1	7	13	52	Logic
Circuit 2	32	16	107	Logic
Circuit 3	93	53	172	Logic

The HSPICE simulation results are shown in Table 2.2, where the paths numbered 1 through 5 are from Circuit 1, 6 through 10 are from Circuit 2, and 11 through 15 are from Circuit 3. This was the original process we designed our cells for, and the results are encouraging. In 9 out of 15 paths (path # 1, 6, 7, 8, 11, 12, 13, 14, 15), the mixed PTL scheme outperformed the conventional static logic and the conventional PTL in either delay or power consumption. Out of these 9 paths, 6 paths

Table 2.2: HSPICE simulation results using $0.25\mu m$, $Vdd = 2.5V$

Path	Delay(sec)			Power(W)			PDP		
	Static	Conv. PTL	Mixed PTL	Static	Conv. PTL	Mixed PTL	Static	Conv. PTL	Mixed PTL
1	4.83E-10	4.22E-10	4.86E-10	5.08E-05	8.72E-05	4.71E-05	2.45E-14	3.67E-14	2.29E-14
2	5.82E-10	5.07E-10	6.64E-10	1.04E-04	6.81E-05	8.52E-05	6.02E-14	3.45E-14	5.66E-14
3	4.95E-10	5.67E-10	6.20E-10	8.29E-05	6.07E-05	6.74E-05	4.11E-14	3.44E-14	4.18E-14
4	3.64E-10	4.27E-10	4.09E-10	3.93E-05	3.98E-05	3.95E-05	1.43E-14	1.70E-14	1.62E-14
5	2.82E-10	2.55E-10	3.70E-10	1.11E-04	7.67E-05	9.48E-05	3.14E-14	1.96E-14	3.51E-14
6	4.92E-10	6.84E-10	4.27E-10	5.48E-05	8.21E-05	5.27E-05	2.70E-14	5.62E-14	2.25E-14
7	5.56E-10	5.45E-10	3.90E-10	6.07E-05	6.77E-05	4.70E-05	3.38E-14	3.69E-14	1.83E-14
8	2.65E-10	2.50E-10	1.87E-10	4.77E-05	9.89E-05	3.88E-05	1.26E-14	2.47E-14	7.25E-15
9	3.90E-10	2.13E-10	2.16E-10	8.78E-05	1.73E-04	1.06E-04	3.42E-14	3.68E-14	2.29E-14
10	3.20E-10	6.33E-10	3.21E-10	3.97E-05	1.61E-04	4.18E-05	1.27E-14	1.02E-13	1.34E-14
11	6.85E-10	1.19E-09	5.73E-10	8.99E-05	3.01E-04	7.27E-05	6.15E-14	3.57E-13	4.16E-14
12	6.16E-10	1.15E-09	6.25E-10	8.95E-05	2.92E-04	7.92E-05	5.51E-14	3.34E-13	4.95E-14
13	5.47E-10	8.94E-10	4.63E-10	4.80E-05	1.36E-04	5.33E-05	2.62E-14	1.21E-13	2.47E-14
14	6.14E-10	5.62E-10	4.28E-10	6.17E-05	1.29E-04	5.28E-05	3.78E-14	7.26E-14	2.26E-14
15	4.30E-10	9.82E-10	3.55E-10	3.92E-05	8.12E-05	3.24E-05	1.69E-14	7.97E-14	1.15E-14
Avg.	4.75E-10	6.18E-10	4.36E-10	6.71E-05	1.24E-04	6.07E-05	3.18E-14	7.66E-14	2.65E-14

outperformed in both delay and power consumption. In 10 out of 15 paths (path # 1, 6, 7, 8, 9, 11, 12, 13, 14, 15), the mixed PTL scheme outperformed the conventional static logic and the conventional PTL in power-delay product.

On average, the mixed PTL has a 30 percent improvement in delay while consuming approximately half of the power when compared to the conventional PTL. More modest, but still substantial, gains are seen when comparing our approach to the static CMOS implementation, where there is approximately a ten percent savings in both delay and power. For power-delay product (PDP), the mixed PTL yields a 17 percent improvement over the static CMOS method.

In addition, the experimental results show that, at $0.25\mu m$, the advantages of conventional PTL over conventional static are disappearing. In fact, conventional static logic significantly outperforms the conventional PTL scheme in all three categories of delay, power consumption, and power-delay product. It is our belief that explicit use of buffering between PTL stages in the conventional PTL added significant delay and additional power consumption to the conventional PTL circuits.

2.2.5 Summary on mixed PTL/static Logic

In this section, a new pass-transistor scheme in which conventional PTL stages are mixed with static logic gates is presented. The static logic gates not only act as buffers between PTL stages, but also perform part of the logic functions of the given block. In comparison to the conventional PTL, the elimination of explicit buffering between PTL stages significantly improved both circuit performance and power consumption. The experimental results confirm that the proposed scheme can yield over 10% improvement in delay and power consumption over the conventional static logic, and up to 50% improvement in power consumption and up to 30% in delay over the conventional PTL.

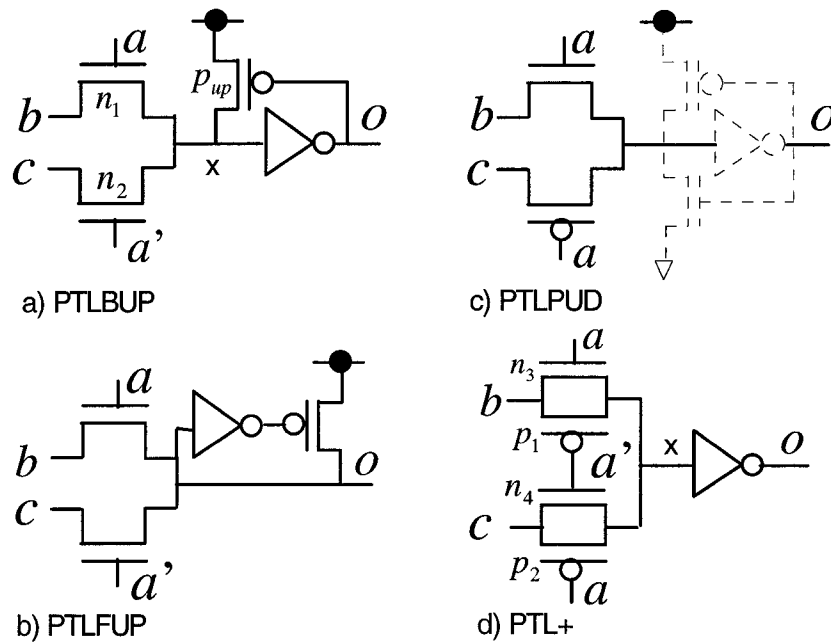


Figure 2.4: Different PTL Cell Types.

The proposed scheme requires a new method of technology mapping. In Chapters 3 and 4, the process of mapping a given logic network using the mixed PTL and static logic gates will be described.

2.3 PTL Cell Types

Power consumption and circuit performance of a PTL-based circuit depend on the PTL cell types used in the circuits. But not enough work has been done in comparing the effects of PTL cell types for mixed PTL/Static circuits. In this section we examine several widely known and well used PTL structures to examine their performance and power consumption in the context of mixed PTL/CMOS circuits. The outcome of this analysis also serves as justifications for our choice of the PTL cell structure used by our synthesis algorithm.

2.3.1 Bulk CMOS PTL Cells

2.3.1.1 Previous methods

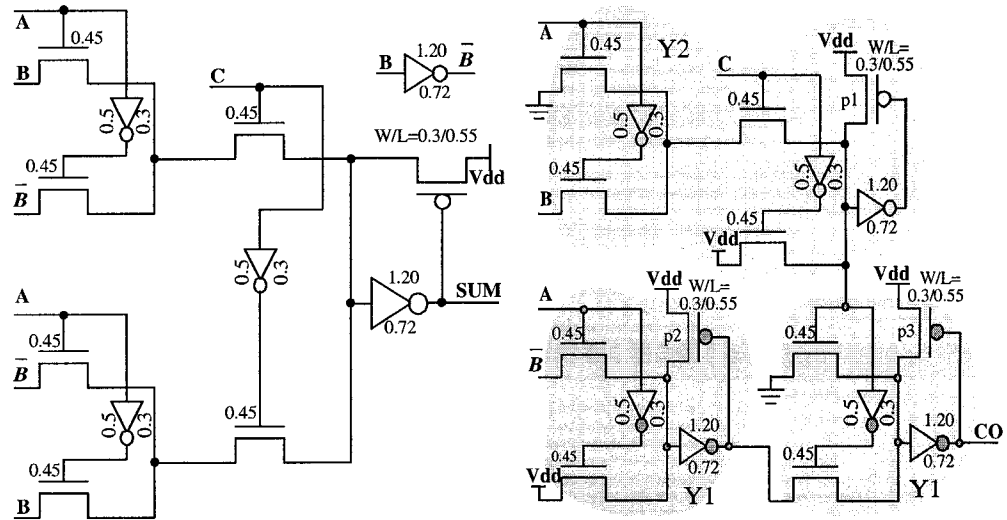
Figure 2.4 shows several single rail PTL cells². The cell type in Figure 2.4(a) is used in [10]. There are two other cell types proposed in [10] that are not shown here. Figure 2.4 (b) is used in [23], (c) is used in [13] and [14, 16]. For the circuits in Figure 2.4 (a) and (b), the output of the cells swings from $0V$ to $V_{dd} - V_{tn}$, where V_{tn} is the threshold voltage of the nMOS transistor. Therefore, the cell outputs need to have swing restoring circuit in order to reduce leakage current of the static gates at the next stage. Having swing restoring circuit would increase power consumption and delay of the circuit because of potential drive fight at the PTL output.

The PTL cell in Figure 2.4(c) uses nMOS and pMOS to form paths for non-control input signals. The worst case output voltage of the pass-transistor section is from V_{tp} to $V_{dd} - V_{tn}$, because nMOS passes zero well, while pMOS passes high well. In order to recover the voltage drops due to the use of nMOS and pMOS in the cell, two swing restoring transistors need to be used. One is for pull-up and the other one is for pull-down. This introduces more power consumption and delay than the cell types PTLBUP and PTLFUP, which are shown in Figure 2.4 (a) and (b), respectively.

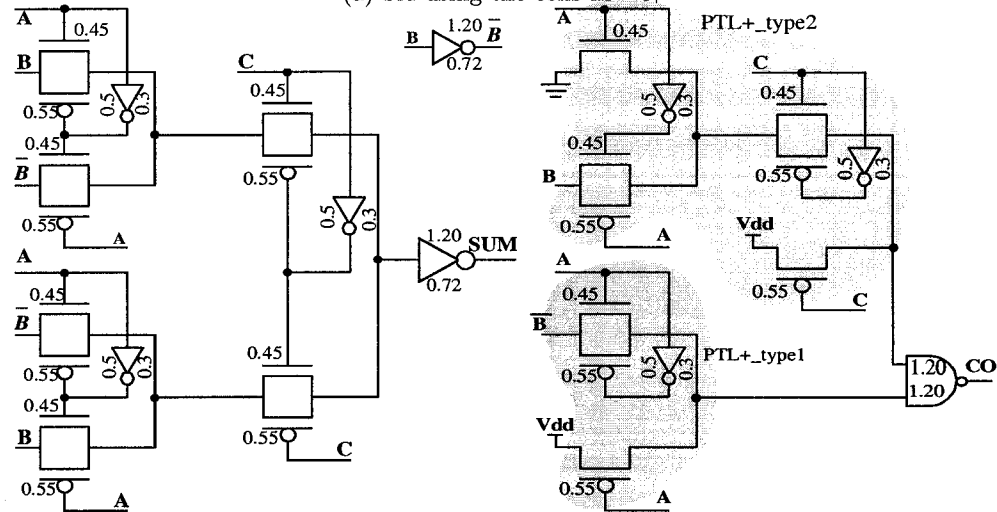
2.3.1.2 PTL+ cell types

The PTL cell type in Figure 2.4 (d) is a MUX-like structure that uses transmission gates. This cell type does not need any swing restoring circuitry because the output swing range of this cell is rail-to-rail ($0V$ to V_{dd}). The transistors pMOS $p1$ or $p2$ (nMOS $n1$ or $n2$) can be removed, as shown in Figure 2.9 (c), if terminal b or c (*non-control inputs*) is connected to Gnd (V_{dd}). This removal will positively impact

²The inverters to generate signal a' are not drawn. The inverters are needed because these cells generate single-rail output voltage.



(a) FA using the cells in [10]



(b) FA using PTL+ cells

Figure 2.5: Two different full-adder implementations.

Table 2.3: FA Simulation Result of PTL Cells in a 0.25 μm CMOS technology.

Cell Type	Power(W)	Delay(ns)	PDP(J)	Area(μm^2)
PTLBUP	1.914E-06	0.462	8.842E-07	7.16
PTLFUP	1.940E-06	0.459	8.902E-07	8.65
PTLPUD	5.743E-06	1.921	1.103E-05	7.50
PTL+	1.591E-06	0.347	5.520E-07	6.86

power consumption and performance of the circuits where this type of cell is used. This type of cell is referred to as the PTL+ cell.

2.3.2 Simulation and Analysis of Single-Rail PTL Cells

To illustrate that the PTL+ type cells are faster and consume less power than others, each PTL cell type from Figure 2.4 is used to implement a 1-bit mixed PTL/CMOS full adder (FA) circuit in a commercial 0.25 μm CMOS process. The number of pass-transistors in series is limited to two.

The FA circuits using the cells in [10] and PTL+ cells are shown in Figure 2.5. The simulation results for all cell types are shown in Table 2.3, and Figure 2.6 shows the full adder simulation waveform for the PTLBUP, PTLFUP, and PTL+ cell types.

The PTLPUD cell used in [13, 14] showed the highest power and worst delay. Because this cell's output swing is V_{tp} to $V_{dd} - V_{tn}$ in the worst case, extra level restoring logic (dashed symbols) is added at the output of the cell, as shown in Figure 2.4 (c). This extra level restoring logic introduces a significant amount of delay and a large amount of power consumption due to short circuit current.

The PTLBUP and PTLFUP cells show very similar results in both power consumption and delay. This is due to the fact that both cell types use one pull-up pMOS and one inverter for level restoring. The area of PTLFUP cell, measured using

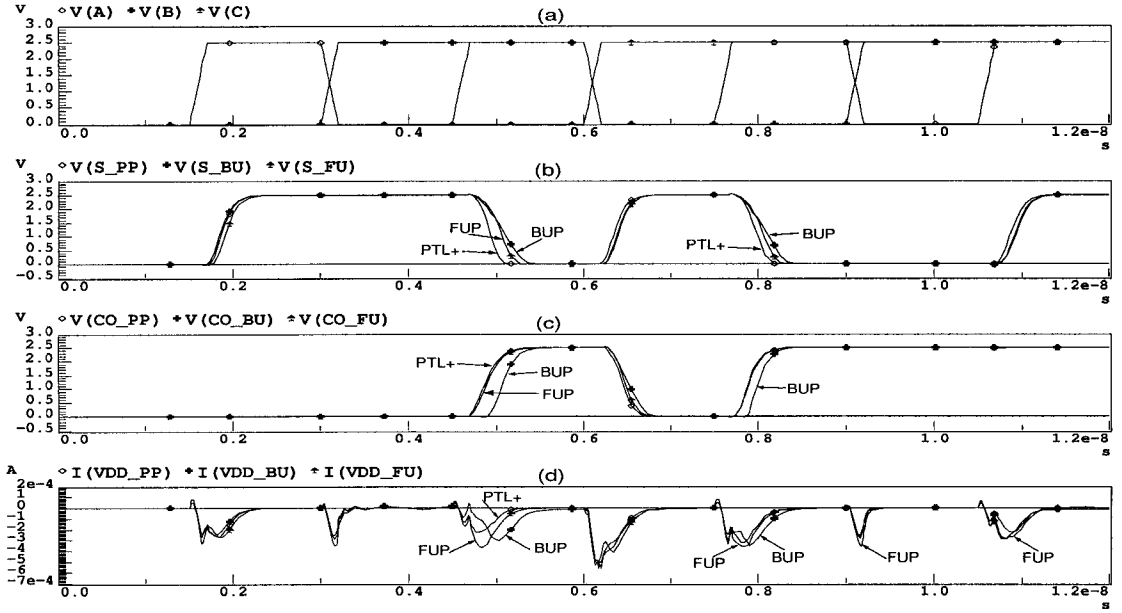


Figure 2.6: Full-adder simulation waveform for three different PTL cell types in $0.25\mu\text{m}$ technology ($V_{dd} = 2.5\text{V}$). (a)input waveform, (b)sum output, (c)carry-out waveform, and (d)current waveform.

the total transistor gate area, is larger than PTLBUP cell due to added inverter in PTLBUP cell.

The PTL+ cell shows the best results compared to the other three types of PTL cells. This is due to the range of output voltage that exhibits rail-to-rail swing and reduces the amount of short circuit current at the output stage. This configuration of PTL gates has not been used often in most studies on PTL because of fears that the configuration is too big. Our results show that PTL+ cells are slightly bigger than PTLBUP, but smaller compared to others. The benefits of getting a rail-to-rail full swing at the PTL gate output without significant drive fights significantly outweighs the slightly bigger area in terms of performance and power consumption.

Although it is clear that the delay of the PTL+ cell shown in Figure 2.4 (d) is better than that in Figure 2.4 (a), it might be helpful to illustrate that analytically. In deep-submicron technologies, drain current of a transistor can be represented by

the alpha-power law[35]. Assuming that logic 1 is transferred from input c to output o . The final voltage of node X of Figure 2.4 (a) is V_{DD} due to level restoring logic, the nMOS transistor n_2 is in saturation mode during the transfer and the delay of Figure 2.4 (a), t_d^a , is given by

$$t_d^a = \frac{C_X V_{DD}}{2K_s(V_{GSn_2} - V_{tn})^\alpha} + t_d^{INV} \quad (2.3)$$

where K_s is a technology dependent parameter, V_{GSn_2} is $V_{DD} - V_X$, V_{tn} is the threshold voltage of the nMOS transistor, t_d^{INV} is the delay from node x to node o , and α is the velocity saturation index which varies from 1 to 2.

For Figure 2.4 (d), the nMOS transistor, n_4 , is in saturation mode when $V_X \leq V_{DD} - V_{tn}$ and n_4 is in cutoff region when $V_X > V_{DD} - V_{tn}$. The pMOS transistor, p_2 , is in saturation mode when $V_X \leq |V_{tp}|$ and it is in non-saturation mode when $V_X > |V_{tp}|$.

Since V_{DD} is several times higher than the device threshold voltage for sub-180nm technology node[36], for most of the switching time, nMOS and pMOS transistors are in saturation and non-saturation mode, respectively. Therefore, the delay of Figure 2.4 (d) can be approximated as

$$t_d^d = \frac{C_X V_{DD}}{2(I_{Dn_4}^{sat} + I_{Dp_2}^{lin})} + t_d^{INV} \quad (2.4)$$

where $I_{Dn_4}^{sat}$ is the saturation current of nMOS n_4 and $I_{Dp_2}^{lin}$ is the non-saturation current of pMOS p_2 . $I_{Dn_4}^{sat}$ and $I_{Dp_2}^{lin}$ are represented by

$$I_{Dn_4}^{sat} = K_s(V_{GSn_4} - V_{tn})^\alpha \quad (2.5)$$

$$I_{Dp_2}^{lin} = K_l(V_{SGp_2} - V_{tp})^{\alpha/2} V_{SDp_2} \quad (2.6)$$

where V_{GSn_4} is $V_{DD} - V_X$, V_{SGp_2} is V_{DD} , V_{SDp_2} is $V_{DD} - V_X$, and K_l is the velocity saturation index in non-saturation mode. Comparing Equations (2.3) and (2.4), it is

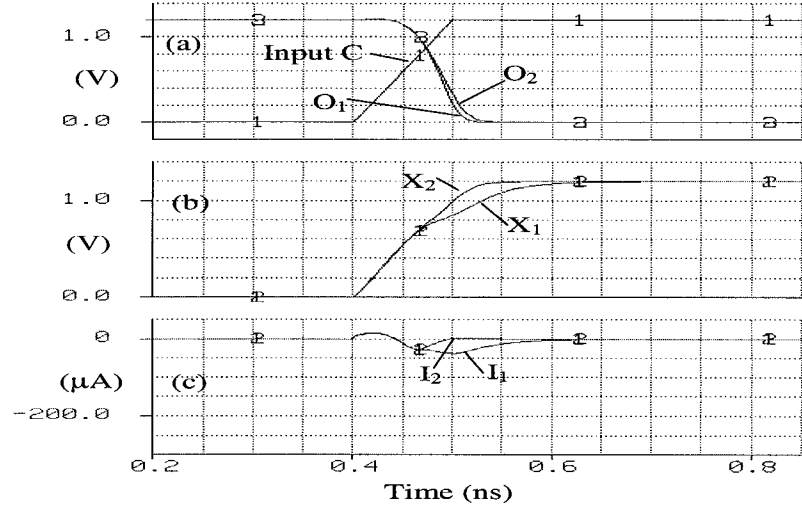


Figure 2.7: Simulation results of PTL circuits in a $0.13\mu m$ CMOS process. The waveforms O_1 and O_2 represent the outputs of Figure 2.4 (a) and (d), respectively. The waveforms X_1 and X_2 represent the node X of Figure 2.4 (a) and (d), respectively. And I_1 and I_2 represent the current of Figure 2.4 (a) and (d), respectively.

clear that the delay of the circuit in Figure 2.4 (d) is lower than that in Figure 2.4 (a) given the delays of the inverters in Figures 2.4 (a) and (d) for the same load capacitance are the same.

Dynamic power dissipation of a CMOS inverter is given by

$$P_{dy} = P_{load} + P_{sc} \quad (2.7)$$

where P_{load} and P_{sc} are the power consumed by load capacitance and short circuit power, respectively. Since the power consumed by the load capacitance is the same for the same load capacitance, the power efficiency of Figure 2.4 (a) mainly depends on P_{sc} of the inverter and the power dissipation caused by the current flow through pull-up transistor p_{up} . However, the power efficiency of Figure 2.4 (d) mainly depend on P_{sc} of the inverter. Assuming that $\beta_n (V_{tn})$ and $\beta_p (V_{tp})$ are the same, P_{sc} of an inverter is given by[37]

$$P_{sc} = \frac{\beta}{12} (V_{DD} - 2V_t)^3 \frac{t_{rf}}{T} \quad (2.8)$$

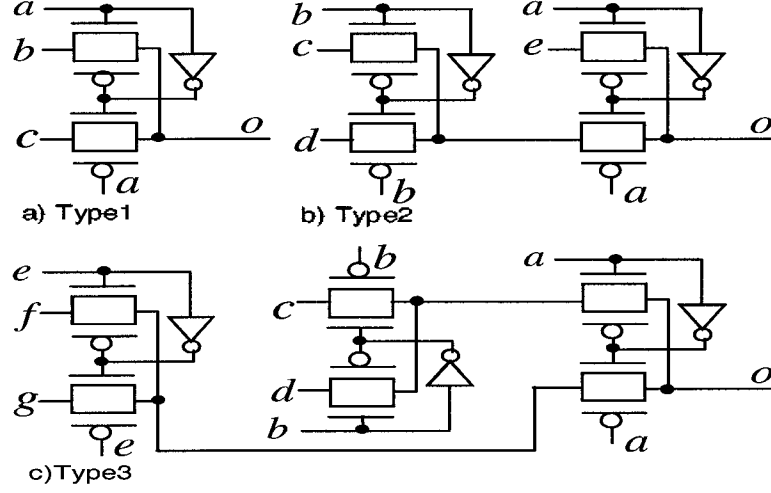


Figure 2.8: Three single-rail PTL+ cell types.

where t_{rf} is the input transition time, T is the period of the input waveform. One of the reasons the power consumption of PTL cell in Figure 2.4 (a) is more than that in Figure 2.4 (d) is the slower slope of node X in Figure 2.4 (a) due to drive fight at the level restoring circuit. Figure 2.7 shows the simulation results where the transition time difference of node X is clearly illustrated. Therefore, the power consumed by Figure 2.4 (a) is great than that of Figure 2.4 (d).

Based on the experimental results and analysis above, we have chosen PTL+ as the basic PTL cell type for our mixed PTL circuit implementations. Figure 2.8 shows three basic single-rail PTL+ cell types used in our approach. The area overhead of PTL+ cells can be reduced by removing nMOS or pMOS transistor in the cells. For example, pMOS p_1 or p_2 (nMOS n_3 or n_4) of 2.4 (d) can be removed when terminal b or c (*non-control input*) is connected to Gnd (Vdd), as shown in Figure 2.9 (c).

2.3.3 Dual-Rail PTL Cells

Dual-rail PTL+ cell structure used in this study was chosen to be very similar to single-rail PTL+ cells based on the analysis above. The difference is that, since

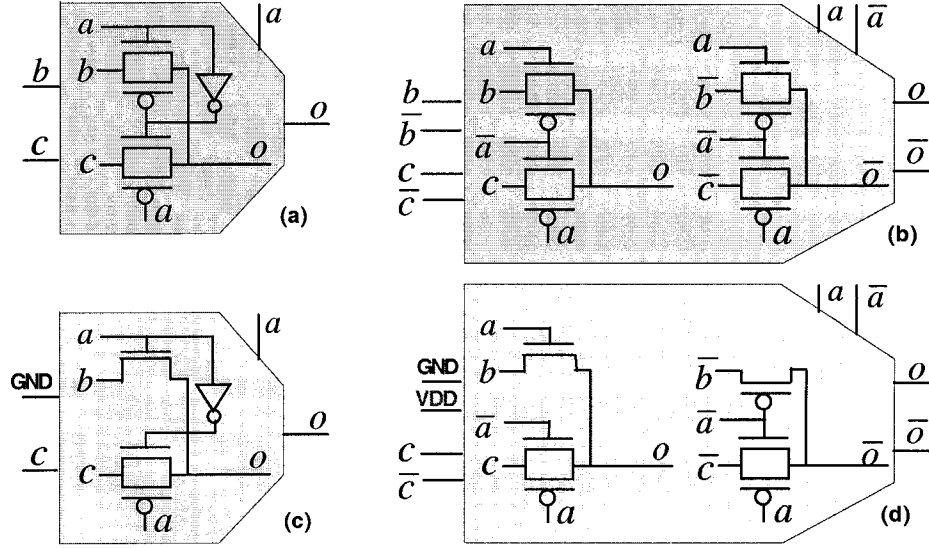


Figure 2.9: Single-rail and dual-rail PTL+ Type1 cells and their variations.

all gate output in dual-rail mixed PTL circuits generate a signal and the one with the opposite polarity, the dual-rail PTL+ cells do not have inverters which are used in single-rail cells to generate opposite polarity of gate control signals. Figure 2.9 (b) and (d) show the dual-rail PTL+ type1 cell and its a variation when the input signal b ($\bar{b}$) is connected to Gnd (Vdd), respectively.

2.3.4 PTL Cells for Floating-Body PDSOI

The SOI technology is potentially more attractive for mixed PTL circuits due to its reduced junction capacitance. Therefore, experiments presented in Chapter 6 were carried out using a $0.13\mu m$ partially depleted SOI (PDSOI) technology. SOI PTL cells have the same structure as that of bulk CMOS. In addition to the fact that the delay of PTL cells in SOI are expected to switch faster than bulk CMOS due to its lower junction capacitance, the decreased threshold voltage of a floating-body SOI device due to capacitive coupling between body and device terminals, impact ionization, and junction leakage also makes SOI device faster than bulk CMOS devices. The decrease in capacitance in SOI positively impacts on the power consumption.

2.4 SOI versus DTMOS

In this section, silicon-on-insulator (SOI) and dynamic threshold CMOS (DTMOS) will be reviewed, and experimental results of basic cells using SOI and DTMOS³ in two different $0.13\mu m$ SOI technologies will be presented. Then a proper SOI type will be chosen for SOI mixed PTL circuits.

2.4.1 SOI

Silicon on insulator (SOI) CMOS is not much different from bulk CMOS. The main difference is that an insulator layer, buried oxide layer (BOX), is inserted between the substrate and body of the device in SOI CMOS. Therefore, the body of a transistor is floated and is not connected to the device body anymore. The main advantage of the SOI device is that it has a smaller junction capacitance than bulk CMOS devices. As a result, with SOI CMOS, lower power can be achieved than with bulk CMOS devices.

Floating body SOI devices can be classified into two categories: partially depleted SOI (PD SOI) and fully depleted SOI (FD SOI) devices. The main difference between PD SOI and FD SOI is that silicon thickness of the transistor (T_{si}) of PD SOI ($200nm - 50nm$) is thicker than that of FD SOI ($50nm - 10nm$). It is known that the inverse subthreshold slope (ISS) in the FD SOI device is lower than that in the PD SOI device[39]. However, there are some manufacturability issues in FD SOI, while the PD-SOI process is almost compatible to the modern bulk CMOS process. The manufacturability issues come mainly from the fact that the V_t sensitivity to T_{si} and thin-film doping density (N_{si}) of FD-SOI is higher than that of PD SOI. Because

³DTMOS can be realized in bulk CMOS process. However, DTMOS based on bulk CMOS[38] is not dealt in this dissertation.

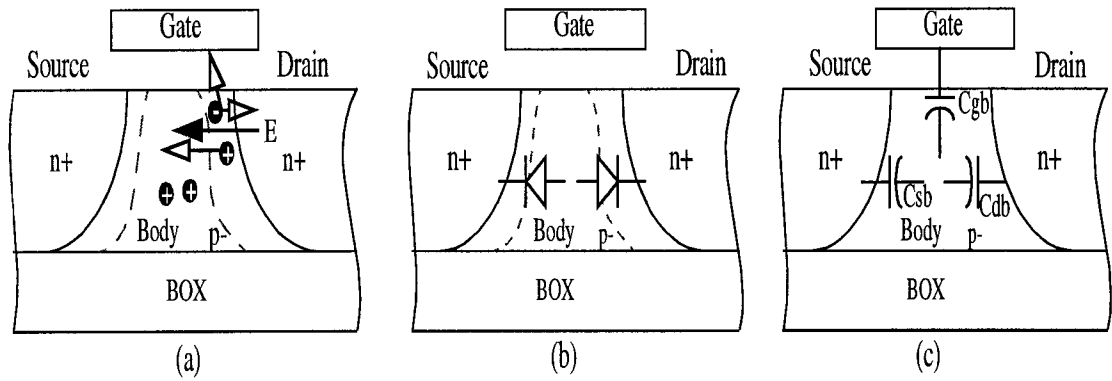


Figure 2.10: Sources of body charging mechanism in partially depleted floating body SOI. (a) impact ionization. (b) junction leakage. (c) capacitive coupling.

FD SOI devices are unpopular due to the fact described above, FD SOI will not be examined in this dissertation.

Due to the fact that the body is floated in the floating body PD SOI device, the device characteristic is no longer constant and depends on the amount of charge in the body. Therefore, it is necessary to understand body charging and discharging mechanisms in order to understand *floating body effects*[40].

Body charging is mainly caused by impact ionization, junction leakage, and capacitive coupling between body and gate, source, and drain. Figure 2.10 shows the sources of the body charging mechanism in the floating body SOI nMOS device. Impact ionization[41], shown in Figure 2.10(a), is caused by lateral electric field when the drain voltage is large. Electrons traveling from the source to the drain get high energy in the high electric field region near the drain. The high energy electrons collide with the lattice in the depletion region and generate electron/hole pairs. The newly generated electron collides again with the lattice to produce more electron/hole pairs. Most of the electrons move toward the drain, and holes move toward the body of the device. Some of the holes are accumulated at the buried oxide boundary near the source. These accumulated holes raise the body potential.

The second source of body charging is junction leakage. As shown in Figure 2.10(b), the source and/or drain junction are reverse biased when the source and/or drain potential are higher than the body potential, and there is current flow through the reversed biased junctions. As a result, the potential at body is increased.

Figure 2.10(c) shows the capacitive coupling between body and gate, source, and drain. When one of the terminal voltage goes higher than the body potential, the body potential is increased by the current flow through the terminal to the body.

The body discharging mechanism is related to forward biased junction diodes. When the voltage between body and source (or the voltage between body to drain), V_{bs} (or V_{bd}), exceeds a certain limit due to impact ionization, leakage current, and capacitive coupling, the diode turns on and passes body charge to source (or drain).

Due to the floating body, several phenomena that are nonexistent in conventional bulk CMOS devices can be observed in floating body SOI devices[41] such as history effect, kink effect, bipolar parasitic effect, and self-heating. The details of these effects are not described here.

2.4.2 Dynamic Threshold-Voltage CMOS

Dynamic threshold-voltage MOSFET (DTMOS) structure in SOI is shown in Figure 2.11[1, 2]. Because the gate and body of a DTMOS are connected as shown in Figure 2.11(a), the controllability of the gate over the channel can be improved for PD SOI devices. Therefore, the floating body effects that are observed in the PD SOI device can be eliminated in DTMOS. The threshold voltage, V_t , of DTMOS decreases as gate voltage increases. Threshold voltage reduction has a positive impact on carrier mobility. Therefore, the mobility in DTMOS is higher than the mobility in body-tied to ground (in the case of nMOS) device because the vertical electrical field is effectively reduced. The effective vertical electrical field is approximately represented

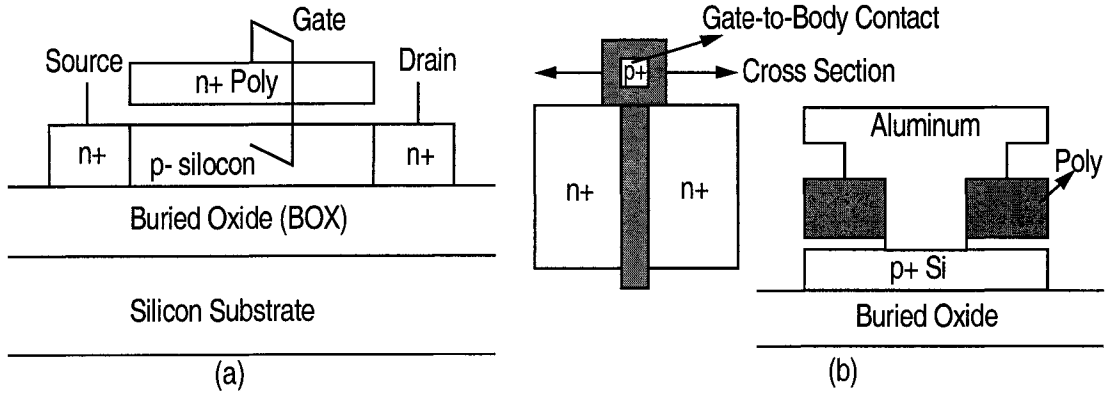


Figure 2.11: DTMOS structure[1, 2]. (a)Cross section of an SOI nMOS with body and gate tied together. (b)Gate-to-body connection using aluminum to short the poly gate to the p+ region.

by the following equation[2].

$$E_{eff} \approx \frac{V_{gs} + V_t}{6t_{ox}} \quad (2.9)$$

Because of the connection of the gate to the body, the effective capacitance seen at the gate includes front gate oxide capacitance, the equivalent capacitance accounting for the depletion region under the channel, and junction capacitance of the body-source and body-drain junction. Therefore, the effective gate capacitance of DTMOS is greater than that of conventional bulk CMOS where the body is grounded. However, the increased mobility portion dominates the increased gate capacitance, the drive current in DTMOS is much higher than that of regular CMOS, and DTMOS gates are expected to switch faster than regular CMOS. The main disadvantage of DTMOS is that DTMOS is only suitable for low-power applications, meaning a supply voltage less than $0.7V$, because the device current increases significantly when a supply voltage is greater than $0.7V$.

Since DTMOS suffers from a significant amount of current when the supply voltage is higher than diode turn-on voltage, which is about $0.7V$, a great deal of effort has been made to overcome the drawback of DTMOS[1, 2, 42, 43]. In [1] and [2], a

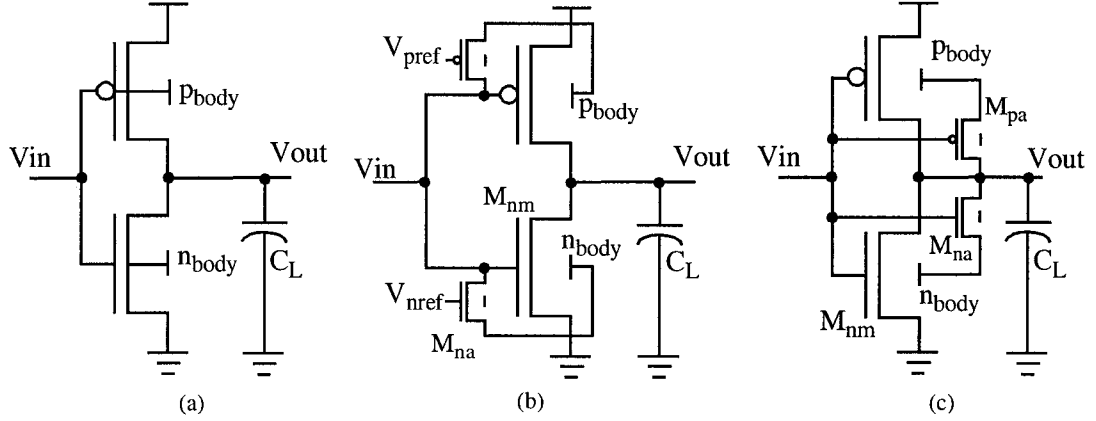


Figure 2.12: Different types of DTMOS Inverter. (a) Conventional DTMOS. (b) and (c) DTMOS with limiting transistors.

small auxiliary transistor was used to limit V_{bs} just below the diode turn-on voltage. Figure 2.12(a) and (b) show inverters, conventional DTMOS and DTMOS with a limiting transistor proposed in [1, 2], respectively. Usually the limiting transistor is very small compared to the main transistor.

Assume that the reference voltage V_{nref} (V_{pref}) in Figure 2.12(b) is connected to V_{dd} (Gnd). The limiting transistor M_{na} is on-state when input V_{in} is low, and the voltage at the body of M_{nm} is the same as the input voltage. In this situation, the threshold of M_{nm} is high because the body voltage of the transistor is low. As input voltage rises, the body voltage of M_{nm} also rises until the input voltage exceeds $V_{dd} - V_{t_{M_{na}}}$. When the input voltage is greater than $V_{dd} - V_{t_{M_{na}}}$, the transistor M_{na} goes to off-state and the body voltage stays at the voltage it reaches just before entering the off-state of M_{na} . The operation of pMOS side is very similar to the operation of nMOS. Thus, DTMOS with a limiting transistor can be operated above the diode turn-on voltage by clamping the body voltage below the diode turn-on voltage by a limiting transistor.

Requiring the additional voltage sources for the gates of limiting transistors is the main drawback of the DTMOS configuration in Figure 2.12(b). In Figure 2.12(c)[44],

because the gates of the limiting transistors are connected to the input, which is also connected to the gates of the main transistors. This configuration also does not require extra contact for the limiters. Thus, this DT MOS configuration is more desirable than Figure 2.12(b). As input changes from low to high, both main transistor (M_{nm}) and limiting transistor (M_{na}) turn on, the pull-down process begins. At the beginning of the pull-down process, the potential at the body of main transistor M_{nm} rises, and the threshold voltage of the main transistor is lowered. The threshold voltage drop speeds up the pull-down process. When the drain potential goes below the body potential, the charge at the body is injected to the drain through the limiting transistor M_{na} , then the body potential returns to zero level and the device threshold voltage returns to a high value.

2.4.3 Pass-transistor Logic in PD SOI and DT MOS

With power being an increasingly limiting factor in high density and high-performance VLSI designs, PTL circuits have received a great deal of attention as an alternative high-speed and low-power circuit style.

The applications of DT MOS technique to PTL have also been proposed in [45, 42, 46]. However, because DT MOS PTL with a body biasing circuit is an emerging circuit style, not enough work has been carried out to further examine its effectiveness, particularly in deep submicron technologies. The basic SOI PTL structure in [45, 42, 46] uses nMOS-only pass transistor trees to reduce cell size. The full rail-to-rail swing of the output signal is restored by an extra level restoring logic at the output of a SOI PTL gate. The existence of level-restoring logic at the output of PTL gates not only slows down the PTL gates due to potential drive-fights, but also increases their power consumption.

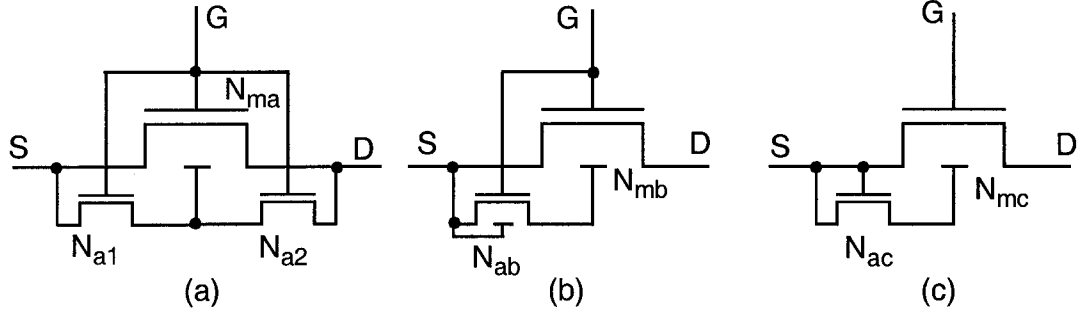


Figure 2.13: Different body bias control circuits for DT MOS.

This section presents a new SOI PTL gate where an active body control circuit is used to allow increase in supply voltages and compares the proposed SOI PTL gate with other SOI DT MOS PTL types. The experimental results using two different $0.13\mu m$ SOI technologies show that the proposed SOI PTL gate with one auxiliary transistor for each nMOS and pMOS transistor shows a significant performance improvement and power reduction over the other SOI DT MOS PTL gates.

2.4.3.1 Body Bias Control Techniques for Pass-Transistor

The nMOS pass-transistor circuit with active body bias circuit shown in Figure 2.13(a)[42] uses two auxiliary transistors, N_{a1} and N_{a2} , to clamp main transistor body voltage. However, this DT MOS configuration requires a significant amount of extra silicon area due to two auxiliary transistors. In addition, since the body potential of the main transistor is divided between two auxiliary transistors, this potential may not rise high enough to speed up the signal transfer from source to drain. The asymmetrical dynamic threshold pass-transistor (ADTPT)[46] scheme shown in Figure 2.13(b) can achieve faster signal transfer from source to drain because the main transistor body potential can be increased more than that of the main transistor body in Figure 2.13(a). The main drawback of these two DT MOS schemes using auxiliary transistor(s) to allow increase in supply voltages is that the capacitance at the gate terminal increases because the gate of the auxiliary transistor is connected to the gate

of the main transistor. The DTMOS pass-transistor in Figure 2.13(a) sees more gate capacitance than that in Figure 2.13(b). Therefore, the circuit in Figure 2.13(b) is expected to be faster than the circuit in Figure 2.13(a).

Figure 2.13(c) shows one of the body bias control circuits proposed in [43] for SOI DTMOS static gates. As shown in the figure, the gate of the auxiliary transistor N_{ac} is connected to the source of the main transistor N_{mc} , and the source of N_{ac} is also connected to the source of N_{mc} . Therefore, the capacitance seen at the gate of the main transistor N_{mc} is reduced compared to both DTMOS pass-transistor techniques in [42, 46]. The main advantage of this body biasing technique can be explained as follows.

When V_G and V_S are both high, the auxiliary transistor N_{ac} is off. Since N_{ac} is off by the fact that the voltage difference gate and drain connected to the main body is less than V_t , the capacitive coupling between the body and the drain of the auxiliary transistor N_{ac} is not shielded by a channel. When V_S goes to low, the capacitive coupling between the main transistor N_{mc} body and the gate/source of N_{ac} quickly discharges the body charge, and the body voltage drops to the voltage above zero level because of capacitive coupling. However, the DTMOS schemes in Figure 2.13 (a) and (b) pull the body voltage down to zero because the auxiliary transistors, N_{a1} and N_{ab} , are turned on during the discharge process. When V_S in Figure 2.13(c) goes to high, the initial body potential is increased by the capacitive coupling. When the gate voltage is increased above the V_t of N_{ac} , N_{ac} is on and the body charging process is accelerated. This body charging process is also faster than those of in Figure 2.13 (a) and (b).

Furthermore, the performance of pass-transistor in [46] is determined only for the delay from source to drain when the input signal at source switches from low to high. However, in a PTL circuit, a signal path can be from source to drain

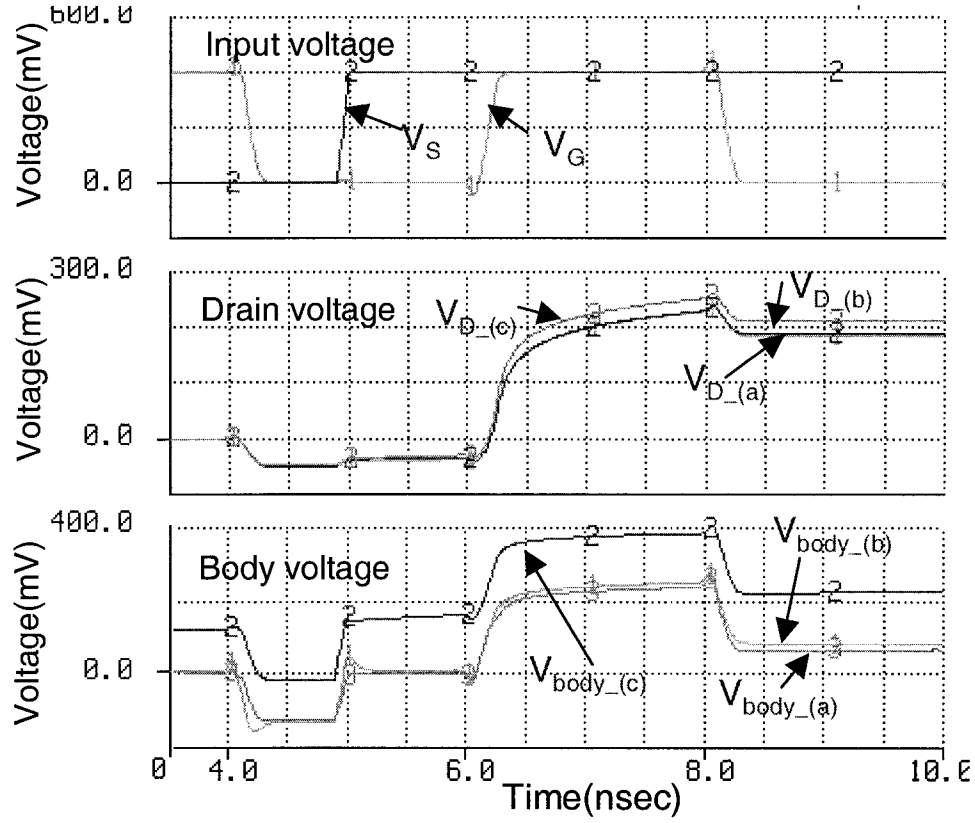


Figure 2.14: The body voltage of main transistors in Figure 2.13

when the gate is in high-state or from gate to drain when the source is low or high. Assuming that the source voltage V_S is high and the gate signal (V_G) changes from low to high, Figure 2.13(c) can take the full advantage of dynamic threshold voltage. When the source voltage V_S is high and the gate voltage V_G is low, the body of the main transistor N_{mc} in Figure 2.13(c) stays a level higher than the main body of Figure 2.13 (a) and (b), as shown in Figure 2.14. This is because the capacitive coupling through N_{ac} charges the main body of Figure 2.13(c), while the main body of Figure 2.13 (a) and (b) stays low because the auxiliary transistors are off. As V_G goes to high, the body voltage of the N_{mc} is further increased. As shown in Figure 2.14, the body voltage of Figure 2.13(c), $V_{body(c)}$, is higher than the body voltage of the main transistors in Figure 2.13 (a) and (b). Therefore, the DTMOS pass-transistor

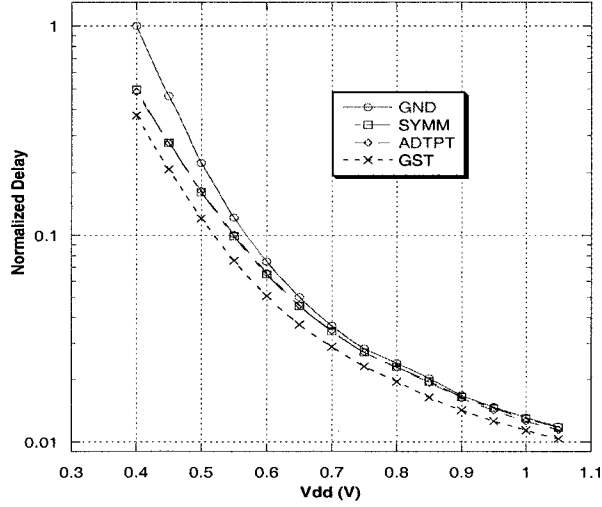


Figure 2.15: Normalized average delay of pass-transistors with different body control scheme.

using active body biasing technique in Figure 2.13(c) is faster than the methods in [42] and [46], as shown in Figure 2.15. In Figure 2.15, SYMM, ADTPT, GST, GND represent the average delay of Figure 2.15(a), Figure 2.15(b), Figure 2.15(c), and body-grounded pass transistor, respectively.

2.4.3.2 Dynamic Threshold PTL Gates

The dynamic threshold transistor structure shown in Figure 2.13(c) in [43] was intended for and experimented in the context of static CMOS circuits only. Its suitability and performance characteristics for PTL logic have not been fully understood. The effect of threshold voltage drop can significantly impact performance in DTMOS because DTMOS is usually operated at lower voltage for low-power applications. The previous methods of SOI PTL[42, 46, 47] using only nMOS for the signal path of non-control inputs suffer from the threshold voltage drop and require a level restoring logic at the output of the pass-transistor, resulting in a significant increase in power and delay. Therefore, the SOI PTL types with nMOS- or pMOS-only circuits for the non-control signal path will not be considered in this chapter. Instead, an SOI PTL type

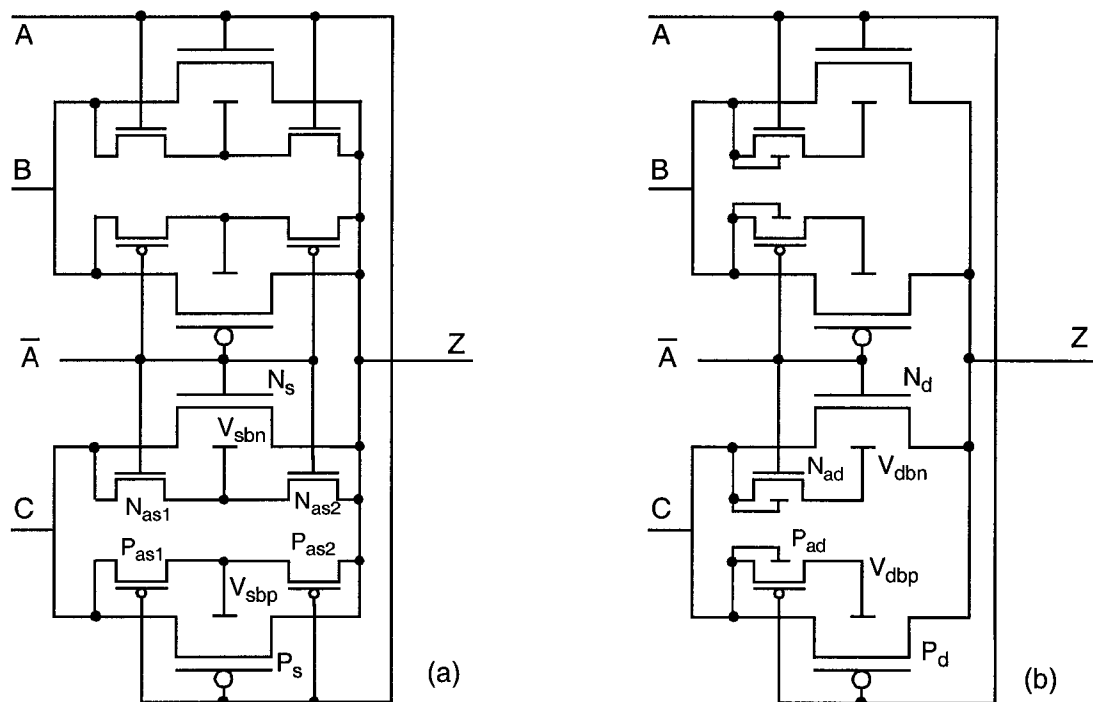


Figure 2.16: DT MOS PTL using symmetrical and ADTPT structure. (a) Symmetrical structure. (b) ADTPT structure.

that uses both nMOS and pMOS transistors will be considered. Figure 2.16 (a) and (b) shows the SOI DTMOS PTL implementations that performs the XOR function using symmetrical and ADTPT techniques, respectively. As shown in Figure 2.16, the body of each nMOS (pMOS) main transistor in symmetrical DTMOS PTL is connected to the source/drain of two auxiliary nMOS (pMOS) transistors, while the body of each nMOS (pMOS) main transistor of ADTPT is connected to the drain (source) of only one nMOS (pMOS) auxiliary transistor.

Let's examine the switching characteristics of these circuit structures in PTL logic. When V_A is low and V_C is high, the pMOS main transistors, P_s and P_d , of symmetrical and ADTPT are on while the nMOS main transistors (N_s and N_d) are off. The body voltage of P_s and P_d is high. As V_C changes from high to low, the initial discharge path from output Z to input C is formed by the main transistors P_s and P_d

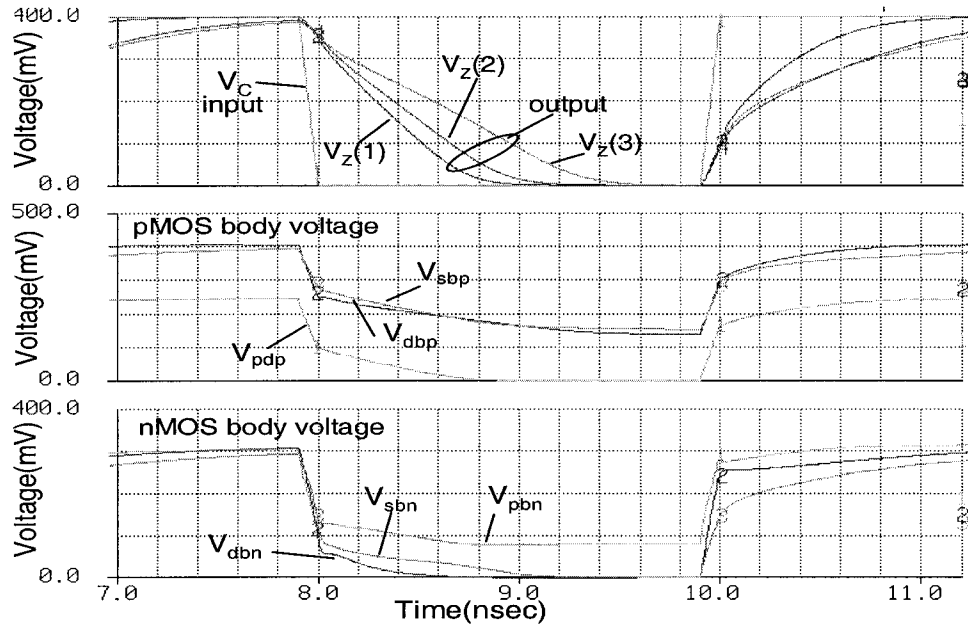


Figure 2.17: The body voltages of the pMOS and nMOS main transistors. $V_z(1)$, $V_z(2)$, and $V_z(3)$ represent the output voltage of DTPTL+, Symmetrical, and ADTPT PTL gates, respectively.

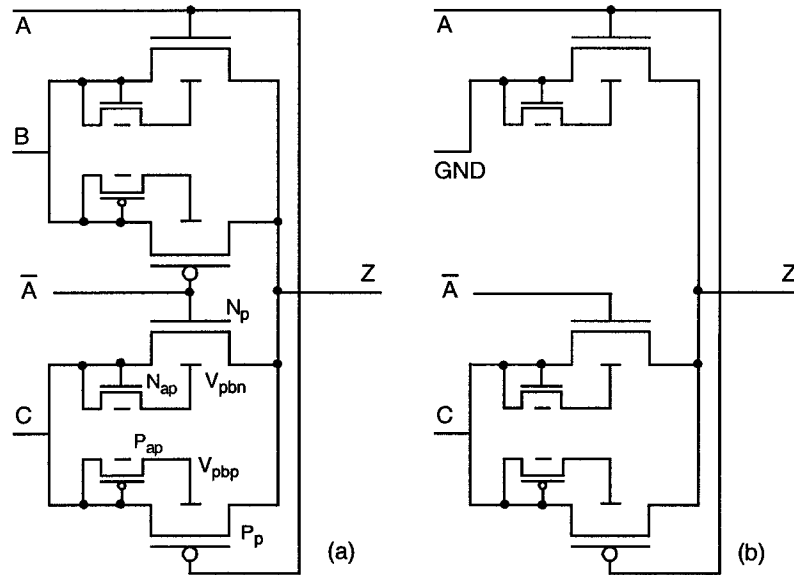


Figure 2.18: (a) DTPTL+ structure. (b) A variation of DTPTL+ when signal B is connected to ground.

for symmetrical and ADTPT structure, respectively. Because both V_{sbp} and V_{dbp} are high, the discharging speed through pMOS for both circuits is almost the same. At the same time, the body voltage of nMOS main transistors, V_{sbn} and V_{dbn} , are pulled down by the capacitive coupling of the main transistors (N_s and N_d) which are in off-state. When input V_C goes down below device threshold voltage V_t , the discharging path changes from the pMOS main transistor to the nMOS main transistor. From this time, the discharging speed of the body voltage of N_d is quicker than that of N_s . This is because the body of N_d is pulled down by the auxiliary transistor N_{ad} which is in on-state while the body voltage of N_s is pulled down slowly by the effect of the auxiliary transistor N_{as2} . Because the decreasing speed of V_{sbn} is slower than that of V_{dbn} , the symmetrical structure is faster than the ADTPT structure, as shown in Figure 2.17.

Figure 2.18(a) shows another DTMOS PTL structure referred to as DTPTL+. In the DTPTL+ case, when V_A is low and V_C is high, the body voltage of pMOS main transistor, P_p , is much lower than that of symmetrical and ADTPT because a small amount of charge is injected to the body of P_p by off-state auxiliary transistor P_{ap} during the body charging process. During discharging when V_C goes down, the auxiliary transistor P_{ap} is on and the body of P_p quickly discharges to zero, as shown in Figure 2.17. On the other hand, the body of nMOS main transistor N_p is not pulled down to ground level because N_{ap} is in off-state during the discharge. The voltage discharging speed through nMOS main transistor is also faster than symmetrical and ADTPT structures. Therefore, DTPTL+ is faster than the other two DTMOS PTL structures.

When the PTL structures described above are used in SOC applications, the load capacitance of the driving gate can be reduced by removing some of the main and auxiliary transistors as shown in Figure 2.18(b).

2.4.3.3 Experimental Results

The SOI DTMOS PTL gates described in Section 2.4.3.2 were implemented in two different $0.13\mu m$ SOI CMOS technologies (process A and process B) in various supply voltages. The main difference between process A and process B is that the channel doping density of process B is lower than that of process A. Therefore, the threshold voltage of process B is slightly lower than that of process A, and the transistors in process B are leakier than those in process A. As such, the body effect in process B will not be as pronounced as that in process A.

Most of the existing researches on DTMOS used partially depleted SOI (PDSOI) process models with a great amount of body effect. Process A resembles the PDSOI characteristics of most of the PDSOI processes on which most of the DTMOS experiments were based. Process B, however, is the result of tuning process parameters for better performance and power consumption of standard CMOS circuits.

The goal of investigating DTMOS performance and power consumption on two different PDSOI processes is to better understand the impact of PDSOI parameters on DTMOS. Because the signal paths for the gates can be from non-control inputs to output as well as from control-input to output, all possible signal paths were chosen to compare delay and power consumption. The simulation results of the DTMOS PTL were also compared to the body-grounded CMOS style PTL gate. The simulation was done by a SPICE simulator, which uses the BSIM3SOI model.

Figure 2.19 shows that the average delay of the gate using the proposed method is lower than that of the symmetrical and ADTPT structures because the threshold voltage of the main transistor in the proposed gate is lower than that of the symmetrical and ADTPT structures. In Figure 2.19, bgnd, sym, adtpt, and dtptl+ represent body-grounded, symmetrical, ADTPT, and DTPTL+ PTL gates, respectively. The delay of the body-grounded PTL gate is higher than that of the PTL gate with body

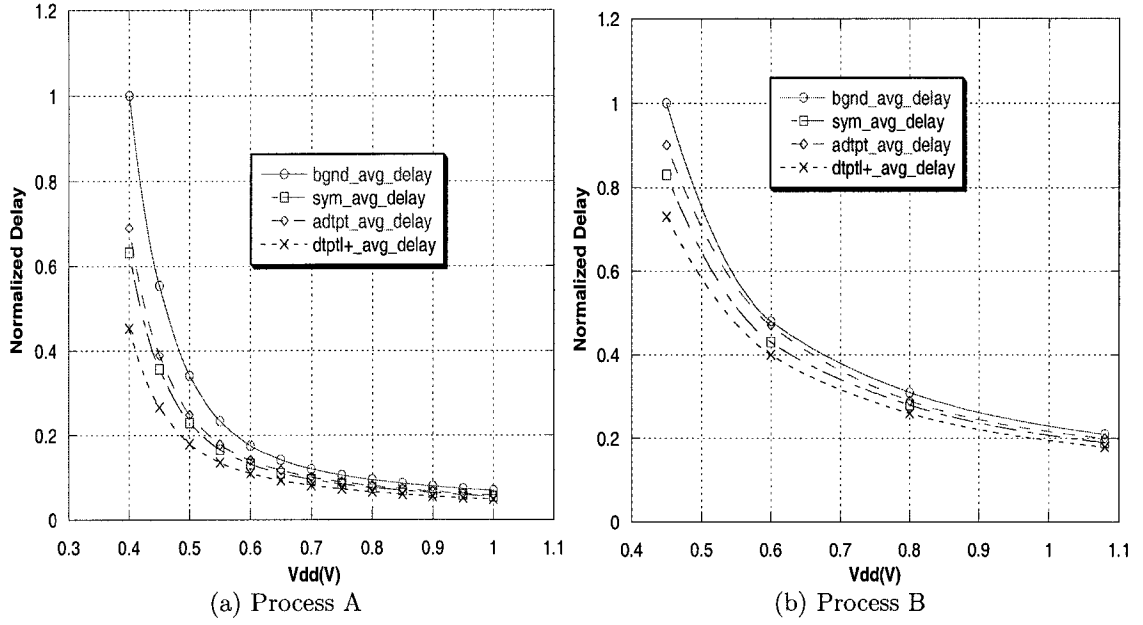


Figure 2.19: Normalized average delay vs. supply voltage.

bias control circuit. The performance gain of DTPTL+ can be seen for both processes over the whole voltage range. Therefore, the DTPTL+ structure is desirable for the high-performance SOC design using PTL logic.

The average power consumption, power-delay-product, and energy-delay product are shown in Figure 2.20, Figure 2.21, and Figure 2.22, respectively. The average power consumption of DTPTL+ circuits is lower than that of the symmetrical and ADTPT structures, and the power consumption of ADTPT structure is higher than that of symmetrical DTMOS PTL structure. The power-delay-product and energy-delay-product of DTPTL+ circuits are also better than that of symmetrical and ADTPT structure. Therefore, DTPTL+ is preferable for lower power applications using PTL logic.

When the average power consumption of the circuits implemented in process B is compared to that in process A, the former has a higher average power consumption than the latter. This is consistent with the characteristics of process A and B as discussed earlier. As shown in Figure 2.21, the average power-delay-products in

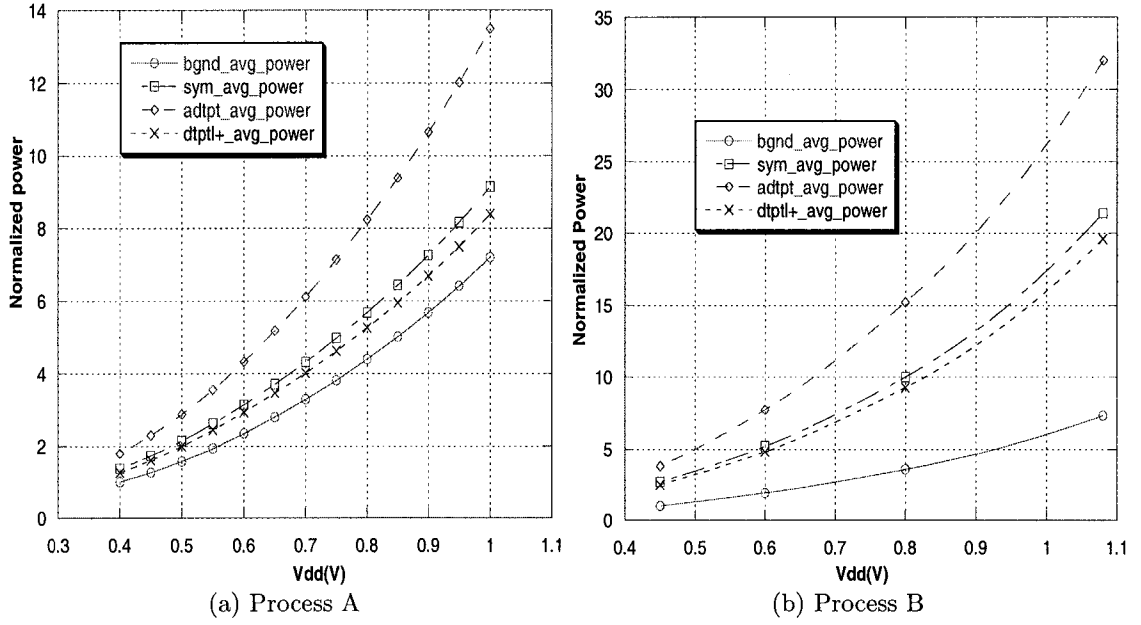


Figure 2.20: Normalized average power vs. supply voltage.

process A increase significantly when supply voltage is less than 0.5V, and this trend is very similar to the power-delay-product curves reported in [42]⁴, while the average power-delay-products for circuits in process B decrease monotonically as the supply voltage decreases. These results indicate that with leakier transistors in SOI, the advantage in performance and power consumption due to the floating body effect diminishes. However, among different DTMOS structures, the proposed DTPTL+ structure performs consistently better than others in both power consumption and performance.

2.4.4 Summary on SOI Pass-Transistor Logic

In Section 2.4.3.2, an SOI PTL structure with adaptive body-bias is presented and compared with different DTMOS PTL structures. All the SOI PTL gates were

⁴In [42], body-grounded and symmetrical SOI PTL structures were reported.

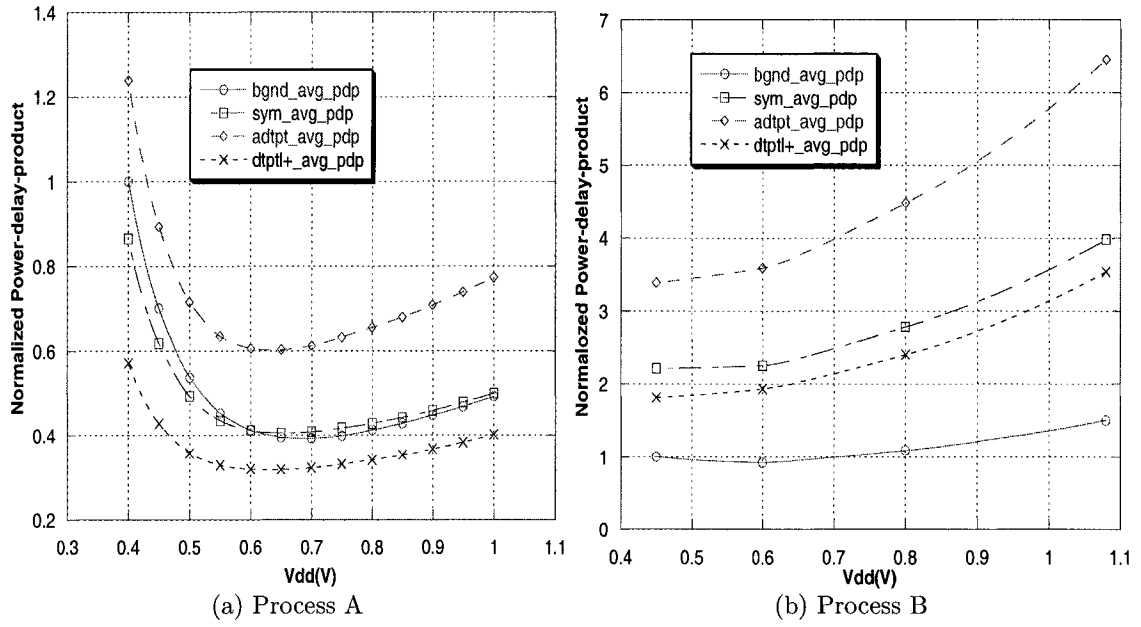


Figure 2.21: Normalized average power-delay-product vs. supply voltage.

implemented in two different $0.13\mu m$ SOI CMOS technologies. The experimental results show that the DTPTL+ structure with adaptive body-bias by one auxiliary transistor can achieve better performance than other SOI PTL gate structures. However, DTMOS PTL structures consume more power than body-grounded PTL structure in the same technology because of the forward-biased diode current of the DTMOS structures. Such a power consumption disadvantage is further exacerbated if the floating body effect is reduced by making the transistor leakier. However, DTMOS, and the DTPTL+ structure in particular, maintain a significant performance advantage over non-DTMOS structures. The tradeoffs between performance and power depends on details of process parameters.

2.5 Conclusions

In this Chapter, a new pass-transistor scheme in which conventional PTL stages are mixed with static logic gates has been presented. The static logic gates not only act as buffers between PTL stages, but also perform part of the logic functions

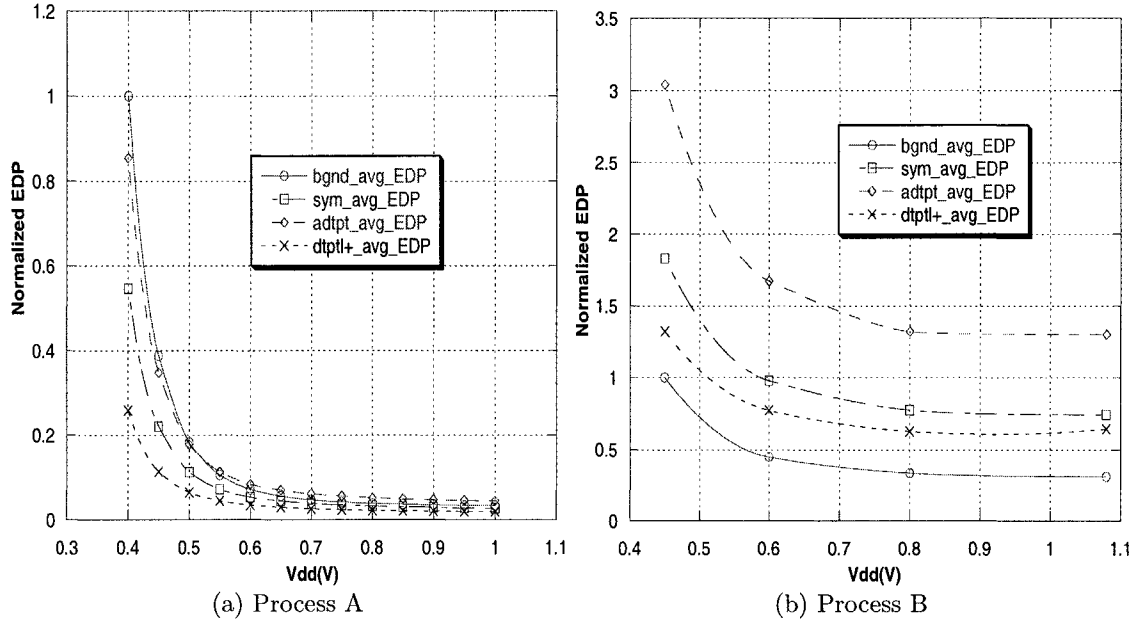


Figure 2.22: Normalized average energy-delay-product vs. supply voltage.

of the given block. Compared to the conventional PTL, the elimination of explicit buffering between PTL stages significantly improved both circuit performance and power consumption.

A new SOI DT MOS PTL structure is also proposed and compared with existing SOI DT MOS PTL structures. The experimental results show that the proposed structure with adaptive body-bias by one auxiliary transistor can achieve better performance than other SOI PTL gate structures. However, the power-delay-product and energy-delay-product trend in submicron SOI design highly depend on the technology used to implement circuits.

Chapter 3

Boolean Matching

3.1 Introduction

With logic synthesis flow, after a technology independent optimization of logic network is done, the results generally need to be mapped using library cells. Equivalence between a subnetwork and a library cell has to be defined before replacing the logic network with library cells. The process of defining the equivalence is called *matching*. This chapter describes the matching technique for mixed PTL/Static logic synthesis used in this proposal. Before describing the matching techniques, the overview of the proposed mixed PTL/Static synthesis method is reviewed in section 3.2. The rest of this chapter is organized as follows: In section 3.3, the advantages and disadvantages of structural and Boolean matching are described. In section 3.4, the logic and graph decomposition methods are described. The concept of cluster and its function is described in section 3.5. The equivalence checking method and symmetry set finding methods are described in section 3.6 and section 3.7, respectively. The details of the pin assignment process between a subnetwork and a library cell are presented in section 3.8. The Boolean matching algorithm used in the proposed mixed PTL/Static logic synthesis method is presented in section 3.8. Finally, the concluding remarks are given in section 3.10.

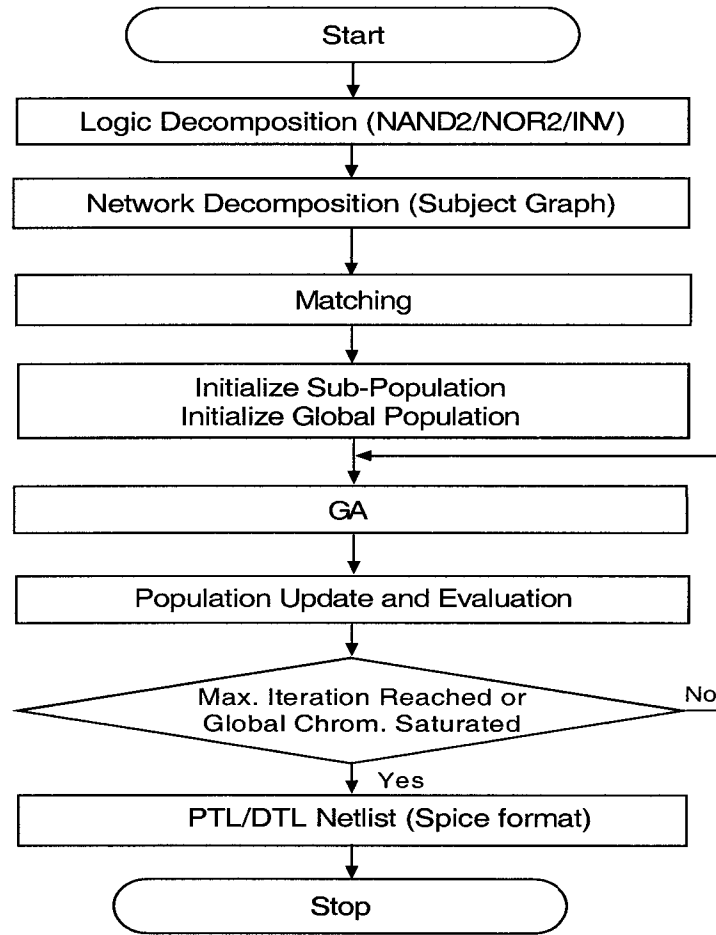


Figure 3.1: Top level flow of the proposed synthesis method. PTL and DTL represent single-rail and dual-rail mixed PTL circuits, respectively.

3.2 Overview of the proposed Logic Synthesis Flow

Figure 3.1 shows the top level flow of the proposed synthesis method. As shown in Figure 3.1, the logic synthesis process begins by decomposing a given Boolean function into a graph containing only simple gates, such as two-input NANDs, NORs and inverters. This large graph is then partitioned into subject graphs[19, 34] to make the rest of the synthesis process tractable. The third step consists of matching and covering. The matching process is to find matches between pattern trees of library

cells and a portion of the subject tree¹. The portion of the subject tree is selected from a set of functions generated at a node n_i with a fixed depth constraint in the subject graph. The covering process is to select the best matches that minimize a given cost function from the matches found in the matching stage. The final step is to generate the netlist. The synthesized mixed PTL circuit can be single-rail or dual-rail. When the synthesis mode is dual-rail, the dual-rail mixed PTL circuits were generated by replacing the single-rail PTL and static cells with corresponding dual-rail cells. There are additional steps used to manipulate buffers at the output stage. The following sections describe our approach for matching in detail. The details of the proposed covering process based on genetic algorithms will be described in Chapter 4.

3.3 Structural and Boolean Matching

Several matching techniques have been published in the past in the areas of structural matching[19, 34] and Boolean matching[48, 22, 49]. There are two methods in structural matching, *tree matching* and *graph matching*. Tree matching determines whether a *pattern tree* is isomorphic to the subgraph of the *subject tree*. This is performed by matching the root of the pattern tree to a vertex of the subject tree and visiting recursively their children. Isomorphism can be detected by comparing the degrees of pair vertices in both subject and pattern trees, starting from the root of the vertices and processing top down to the leaves. The comparing process continues until it reaches the leaf vertex of the pattern tree. Graph matching methods[19, 50] also try to detect the isomorphism between a pattern graph and a portion of the subject graph without decomposing a graph. However, this method creates other multiple fanout points because of subgraph duplication resulting in negative impact on

¹In this proposal, we use the terms *subject tree* and *subject graph* interchangeably.

performance and power consumption of mapped network. Both matching techniques decompose the subject graph and library cells in base functions such as NAND2 and INV. The drawback of these methods, however, is that the number of pattern graphs (or trees) increases dramatically because the decomposition for a cell is not unique. A new structural pattern matching method proposed in [51] is based on the fact that matches of a current node can be represented by the matches of its children. Although this method is considered as a fast method, the matching quality is poor because it cannot be applied to a directed acyclic graph (DAG).

The Boolean matching[52, 22] method involves an equivalence checking and input polarity assignment. Boolean matching can be defined as follows[48, 22] : Given a portion of subject graph function $F(x_1, x_2, \dots, x_n)$ and a library element $G(y_1, y_2, \dots, y_n)$, find an ordering $\{i, \dots, j\}$ and a polarity $\{\phi_1, \phi_2, \dots, \phi_n\}$ such that the following equation is true:

$$F(x_i^{\phi_i}, \dots, x_j^{\phi_j}) = G(y_1, y_2, \dots, y_n) \quad (3.1)$$

The Boolean matching method, however, is more flexible than the graph matching method, and can detect a match that may not be detected by the graph matching method. For example, consider two functions that are logically equivalent, $f_1 = ab + a'b' + b'c$ and $f_2 = ab + a'b' + ac$. Boolean matching can detect a match between functions f_1 and f_2 , while graph matching may not detect the match because the graph of f_1 differs from that of f_2 . Figure 3.2 shows the difference between graph representation and binary decision diagram (BDD)[25, 53, 26] representation. For this reason, Boolean matching is chosen. Boolean matching using BDDs has been widely studied in the past.

3.4 Logic and Graph Decomposition

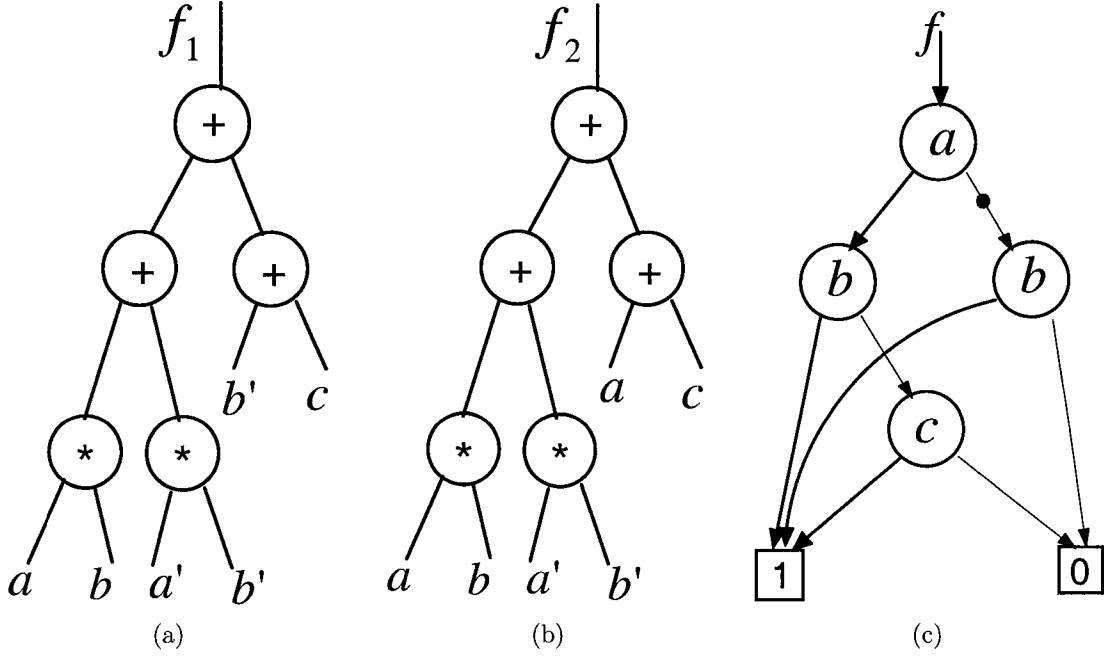


Figure 3.2: The difference between graph matching and Boolean matching. (a) graph representation of f_1 . (b) graph representation of f_2 . (c) BDD representation of both f_1 and f_2 .

The goal of logic decomposition is to express all local functions in a given logic network as simple functions (base functions) such as two-input NAND-, NOR-gate, and inverter. This decomposition is required to guarantee a solution to the network covering process, which is described in Chapter 4, by ensuring that each vertex is covered by at least one match. The library must includes cells implementing the base functions to ensure the existence of a solution. The choice of base functions affects the covering process because the number of pattern graphs depends on the base functions[32]. In [19, 30, 31], a two-input NAND-gate and an inverter were used as the base functions and showed that this base function set is good in terms of optimization potential. This is because increased base function has a potential that can increase the number of pattern graphs for some library cells. This increase of pattern graphs is a major drawback structural matching technique. On the other hand, Boolean matching does not require pattern graph because it depends on the

checking of functionality between a portion of subject graph and library cells. The base functions used in our approach, therefore, are two-input AND-, NAND-, OR-, NOR-gates, and inverters. By using this set of base functions, a logic network can easily be converted to a decomposed network while maintaining fine granularity of the decomposed network.

Once a logic network is represented by base functions, the network is needed to decompose. The process of converting an entire network to a set of smaller networks is called *graph decomposition*. The purpose of the process is to reduce problem size and to make the problem tractable because the covering process described in Chapter 4 is NP-complete[33]. As a result of the decomposition, the entire network reduces to many multi-input single output subnetworks, each subnetwork is called the *subject graph*. The common methods for generating subject graphs is to use multiple fanout points (MFP) as a boundary of breaking points. The approach used in this proposal is similar to the method used in [19] where MFP was used as the boundary of subject graphs. Figure 3.3 shows an example of subject graphs. As shown in Figure 3.3, the subject graphs are generated based on multiple fanout points, and five subject graphs were generated because there are three multiple fanout points and two primary outputs (POs).

3.5 Cluster and Its Function

A subject graph can have more input nodes than those of a library gate. In this case, a match cannot be detected because the number of variables, which is called *support set*, differs for both functions of the subject graph and the library gate. To examine all possible matches at the root node of a subject graph, it is desirable for the matching stage to define possible sub-graphs of a subject graph. A connected sub-graph of a subject graph that rooted at a root node v_r is called a *cluster*[34, 22].

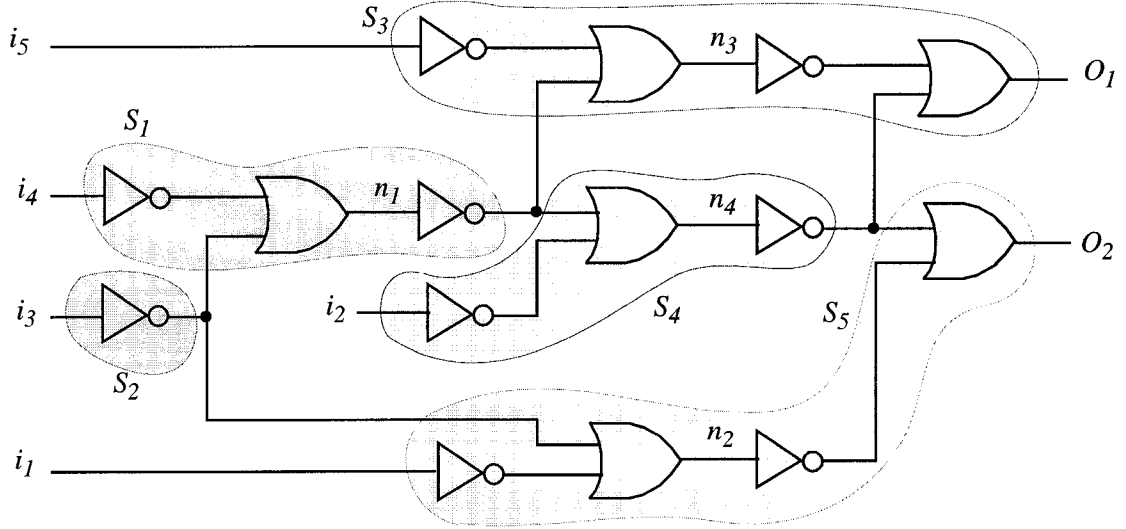


Figure 3.3: Example of subject graphs generated based on multiple fanout points. S_i represents a subject graph.

A cluster is characterized by its depth and the number of inputs, and a Boolean function associated with a cluster is called *cluster function*[34, 22]. The matching algorithm described in this chapter tries to detect a match between a cluster function and a library function (or pattern function).

3.6 Equivalence Checking

Boolean matching problem described above requires an equivalence check between a cluster function and a library function. The equivalence can be detected using reduced ordered binary decision diagrams (ROBDD)[26]. A binary decision diagram (BDD) representation of a function is based on Shannon's expansion theorem[54]. The reduced ordered binary decision diagram is a canonical representation of switching functions[26]. In other words, the ROBDD does not contain any redundancies in its representation. Therefore, it can represent a function uniquely.

Our matching step uses BDDs to represent a function. Once a cluster function and a library function are represented by ROBDDs, the equivalence checking routine

illustrated in Algorithm 3.6.1 is then applied to detect a match between the two functions. To detect a match using ROBDD, the order of variables of one BDD is arbitrary fixed, then the variables ordering of the other function is changed until a match is found.

Algorithm 3.6.1: $BDD_{EQCHECK}(bdd(f), bdd(g))$

```

1. if ( $bdd(f)$  is constant and  $bdd(g)$  is constant)
   then { if ( $f_{val} \neq g_{val}$ )
         then return ( $FALSE$ )
         else return ( $TRUE$ )
       }
   else return ( $BDD_{EQCHECK}(f_T, g_T) \& BDD_{EQCHECK}(f_E, g_E)$ )

```

Algorithm 3.6.1 shows the equivalence checking routine used in our implementation. As shown in Algorithm 3.6.1, the equivalence between two functions is detected by checking recursively to determine whether f_T and g_T have the same value. In the algorithm, f_T (g_T) and f_E (g_E) represent the positive cofactor and the negative cofactor of the cluster function (pattern function), respectively.

3.7 Symmetry Set Finding

Boolean matching requires solving the factorial number of equivalence checks between a cluster function and a pattern function. Assuming that the number of inputs of a function is n , the required number of input comparisons is $n! \cdot 2^n$ because $n!$ comes from the permutation of inputs, and there are 2^n possible input polarities. The number of polarities and input permutations can be reduced by using *unateness* and *symmetry* information [34, 22], respectively :

- *Unateness Property*: Any input permutation must associate a unate (binate) variable in the pattern function.

- *Symmetry Property*: Variables or groups of variables that are interchangeable in the cluster function must be interchangeable in the pattern function.

The unateness property implies that the cluster and pattern function must have the same number of unate and binate variables to have a match. If there are b binate variables, the required number of input permutations is $b!(n - b)!$. For example, the function of the PTL+ type3 cell described in chapter 2 is represented as follows:

$$g = abc + ab'd + a'ef + a'e'g \quad (3.2)$$

In equation 3.2, there are 4 unate variables and 3 binate variables. For the cluster f with $n = 7$, f must have 4 unate and 3 binate variables to have a match. If this is the case, only $3!4! = 144$ variable orders and corresponding OBDDs need to be considered instead of $7! = 5040$.

The symmetry property[55, 56, 57] allows us to reduce search time by exploiting variable groups that give us the same result when we change the variable order in a group. A *symmetry set* is a set of variables that are pairwise interchangeable without affecting the logic functionality. A *symmetry class* is a set of symmetry sets with the same cardinality. If we denote symmetry class by S_i when its elements have cardinality $|S_i|$, where $i = 1, 2, \dots, n$, the number of permutations required to detect a match is $\prod_{i=1}^n |S_i|!$. If we have the following function,

$$f = x_1x_2x_3 + x_4x_5 + x_6x_7 \quad (3.3)$$

there are 3 symmetry sets, $\{x_1, x_2, x_3\}$, $\{x_4, x_5\}$, and $\{x_6, x_7\}$, and two symmetry classes, $S_2 = \{\{x_4, x_5\}, \{x_6, x_7\}\}$ and $S_3 = \{\{x_1, x_2, x_3\}\}$. Therefore, when considering pattern function, only functions with $|S_2| = 2$ and $|S_3| = 1$ are valid to detect a match. In this case, the required number of permutations is only $2!$, i.e., if the

initial variable order is $(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$, the required variable orders of g are $(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ and $(x_1, x_2, x_3, x_6, x_7, x_4, x_5)$.

Unateness and symmetry property can be used together to achieve the number of non-redundant permutations. Unate and binate symmetry sets are disjoint, since both unateness and symmetry properties must be the same for two variables to be interchangeable. Thus, we can write $S_i = S_i^b \cup S_i^u, i = 1, 2, \dots, n$, and $|S_i| = |S_i^b| + |S_i^u|$, where b and u represent binate and unate, respectively. The non-redundant variable permutations are at most $\pi = \prod_{i=1}^n |S_i^b|! \cdot |S_i^u|! = \prod_{i=1}^n |S_i^b|! \cdot (|S_i| - |S_i^b|)!$. Thus, in this case, the total number of comparisons required to find a match is at most $\prod_{i=1}^n |S_i^b|! \cdot (|S_i| - |S_i^b|)! \cdot 2^{i \cdot (|S_i| - |S_i^b|)}$ [22].

Algorithm 3.7.1: GETSYMMSET($bdd(f), sup(f)$)

```

procedure ISSYMMETRY( $bdd(f), v_i, v_j$ )
  if ( $BddEqCheck(f_{v_i \bar{v}_j}, f_{\bar{v}_i v_j})$  or  $BddEqCheck(f_{v_i v_j}, f_{\bar{v}_i \bar{v}_j})$ )
    then return ( $TRUE$ )
    else return ( $FALSE$ )

main
  output ( $SymmSet$ )
  1.  $m \leftarrow -1$ 
  2. for  $n \leftarrow 0$  to  $N_{|sup(g)|} - 1$ 
    do  $\left\{ \begin{array}{l} v_1 \leftarrow sup_f^n \\ \text{if (foreach } i \leq m \text{ } SymmSet[i] \cap v_1 == 0) \\ \text{then } \left\{ \begin{array}{l} m \leftarrow m + 1 \\ SymmSet[m] \leftarrow SymmSet[m] \cup v_1 \end{array} \right. \\ \text{else continue} \end{array} \right.$ 
     $\left\{ \begin{array}{l} \text{for } k \leftarrow 0 \text{ to } N_{|sup(g)|} - 1 \\ \text{do } \left\{ \begin{array}{l} v_2 \leftarrow sup_f^k \\ \text{if } (v_1 \neq v_2 \text{ and } IsSymmetry(bdd(f), v_1, v_2)) \\ SymmSet[m] \leftarrow SymmSet[m] \cup v_2 \end{array} \right. \end{array} \right.$ 
  return ( $SymmSet$ )

```

Algorithm 3.7.1 illustrates the symmetry set finding method used in this proposal. The symmetry between any two variables is detected by comparing two BDDs of a

function. The procedure in Algorithm 3.7.1 performs the symmetry checking between two input variables of function f . There are two different types of symmetries [58, 59]. The first is positive symmetry. This type of symmetry is detected when $f_{v_i v_j} = f_{\bar{v}_i v_j}$. This property implies that the change of order of the variables v_i and v_j does not affect the BDD structure of f . The second is negative symmetry that is detected when the condition $f_{v_i v_j} = f_{\bar{v}_i \bar{v}_j}$ is satisfied. The variables in negative symmetry are interchangeable, for example, $\{\bar{v}_i, v_j\}$ or $\{v_j, \bar{v}_i\}$, without affecting the BDD structure of f . The main routine of the algorithm groups the variables in the same symmetry. The index m in the algorithm represents different symmetry sets. The last step of the algorithm is to return the symmetry sets to the caller.

3.8 Pin Assignment Based On Signal Depth

Once a match is detected between a cluster function and a library cell, the pins of the matched library cells must be mapped to the input signals of the cluster function with proper polarity of the input of the cluster function. Let's denote λ is the mapping function, and $I_{cluster}$ and I_{cell} is the set of input of the cluster function and the set of inputs of a library cell, respectively. The mapping can be represented by $\lambda : I_{cluster} \rightarrow I_{cell}$. This section describes the pin mapping method from a cluster function to a library cell.

3.8.1 Data structure of a match

Before describing the pin mapping method, let's look at the data structure of a match used in this proposal. Figure 3.4 shows the data structure of a match associated with a node structure. In a node of a subject graph, there are two lists of matches. One is a static match list, *smatch*, and the other is a PTL match list, *pmatch*. The static and PTL match list contains static matches and PTL matches, respectively.

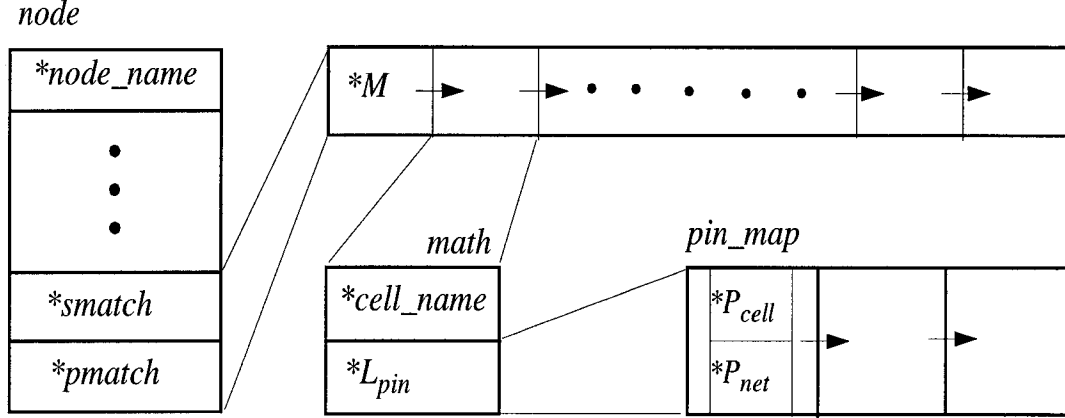


Figure 3.4: Data structure of a match and static and PTL match list.

Each match list is a linked list that consists of a library cell name, $cell_name$, and a linked list of pin information, L_{pin} . The data part of the linked list L_{pin} consists of two character strings, P_{net} and P_{cell} . P_{net} and P_{cell} represent the matched input names of a cluster function and a pin name of a library cell. The polarity information of a pin is contained in P_{net} by examining the polarities of both inputs of the cluster function and the library cell.

3.8.2 Pin mapping algorithm

Algorithm 3.8.1 illustrates the pin mapping method between the inputs of a cluster function and a library cell. In the argument list of the algorithm, n_i represents a node that is the root node of a cluster, f is the cluster function, g is a matched library function, and $\pi(f)$, $\pi(g)$, and $\mathcal{C}$ represent the ordered support set of f , g , and the matched library cell, respectively. The support sets $\pi(f)$ and $\pi(g)$ are ordered based on the depth of inputs of the cluster function f .

At the beginning of the algorithm, a pin map, $\mathcal{M}.P$, and a linked list that contain the pin map, L_{pin} , are initialized. If the matched cell type is a static cell, the first part of the *if* statement is executed. Each support set of f and g are examined

simultaneously. For each $f_i \in f$ and $g_i \in g$, P_{cell} gets g_i without modification of its polarity, and P_{net} gets f_i after modifying its polarity based on the input polarities of the cluster and library functions, $\phi(f_i)$ and $\phi(g_i)$ (statement (i) and (ii)). When the pin map between a cluster input and a input of the library cell is determined, it is added to the pin map list, L_{pin} . After examining all the support variables of f and g , the linked list is inserted in the match structure of the node (statement (iii)).

Algorithm 3.8.1: PINASSIGN($n_i, f, g, \pi(f), \pi(g), C$)

```

1.  $\mathcal{M}.P \leftarrow 0$  ;  $L_{pin} \leftarrow 0$ 
2. if ( $C == STATIC$ )
    then {
        for each ( $f_i \in \pi(f), g_i \in \pi(g)$ )
            do {
                 $P_{cell} \leftarrow g_i$ 
                if ( $\mathcal{M}.cell == inv$ )  $P_{net} \leftarrow f_i \otimes \phi(f_i)$  (i)
                else  $P_{net} \leftarrow f_i \otimes \phi(g_i)$  (ii)
                 $L_{pin} \leftarrow L_{pin} \cup P$ 
            }
         $\mathcal{M}.P \leftarrow L_{pin}$  (iii)
    }
else ( $C == PTL$ )
    {
         $L_{inv} \leftarrow IBinateVars(\pi(g), g, C)$  (iv)
        if ( $\mathcal{M}.cell == ptl\_type1$ )
            for each ( $f_i \in \pi(f), g_i \in \pi(g)$ )
                do {
                     $P_{cell} \leftarrow g_i$ 
                    if ( $g_i == a$ )
                         $P_{net} \leftarrow f_i$  (v)
                    else  $P_{net} \leftarrow f_i \otimes \phi(g_i)$ 
                    if ( $g_i \in L_{inv}$ ) switch pin b and c (vi)
                     $L_{pin} \leftarrow L_{pin} \cup P$ 
                }
            else if ( $\mathcal{M}.cell == ptl\_type2$ )
                for each ( $f_i \in \pi(f), g_i \in \pi(g)$ )
                    do {
                         $P_{cell} \leftarrow g_i$ 
                        if ( $g_i$  is binate var)
                             $P_{net} \leftarrow f_i$  (vii)
                        else {
                             $P_{net} \leftarrow f_i \otimes \phi(g_i)$ 
                            if (b  $\in L_{inv}$ ) switch pin c and d (viii)
                        }
                         $L_{pin} \leftarrow L_{pin} \cup P$ 
                    }
            else if ( $\mathcal{M}.cell == ptl\_type3$ )
                 $L_{pin} \leftarrow PinMap3(\pi(f), \pi(g), L_{inv})$ 
             $\mathcal{M}.P \leftarrow L_{pin}$ 
    }

```

When the matched cell is a PTL cell, the binate variables in the pattern function are checked if their polarities are modified during the detection of the match. All the

binate variables whose polarities are modified are listed in a linked list L_{inv} (statement (iv)). Then the matched PTL cell type is checked. If the matched cell is a PTL cell type 1 or type 2, each support variable g_i of function g is inserted to P_{cell} . The corresponding support variable f_i of f is inserted to P_{net} gets when g_i is a binate variable, otherwise the polarity of f_i is modified based on the phase of g_i , $\phi(g_i)$ and it is inserted into P_{net} (statement (v) and (vii)). If the polarity of binate variables of PTL type 1 and type 2 are inverted, the assigned pins to non-control inputs (b and c for PTL type 1, c and d for PTL type 2) must be modified. This modification of pin assignment is done by switching the names in P_{cell} for each non-control input of a PTL cell (statement (vi) and (viii)).

Algorithm 3.8.2: PINMAP3($\pi(f), \pi(g), L_{inv}$)

```

1.  $L_{pin} \leftarrow 0$ 
2. for each ( $f_i \in \pi(f), g_i \in \pi(g)$ )
    {
         $P_{cell} \leftarrow g_i$ 
        if ( $|\pi(g)| == 3$ )
            then {
                if ( $g_i$  is a or  $g_i$  is b)  $P_{net} = f_i$  (i)
                else  $P_{net} \leftarrow f_i \otimes \phi(g_i)$ 
                if ( $g_i$  is c and b  $\in L_{inv}$ ) (ii)
                    then {
                         $P_{cell} \leftarrow d$ 
                        switch pin d to c
                        switch pin g and f
                    }
            }
        do {
            else if ( $|\pi(g)| > 3$ )
                if ( $g_i$  is binate var)
                     $P_{net} \leftarrow f_i$ 
                else
                    then {
                         $P_{net} \leftarrow f_i \otimes \phi(g_i)$  (iii)
                        if (b  $\in L_{inv}$ ) then switch pin d and c (iv)
                        else if (e  $\in L_{inv}$ )
                            then switch pin f and g (v)
                    }
             $L_{pin} \leftarrow L_{pin} \cup P$ 
        }
    }
3. return ( $L_{pin}$ )

```

The pin assignment for PTL type 3 is more complicated than those of PTL type 1 and type 2. Algorithm 3.8.2 shows the pin assignment procedure for PTL type 3 matches. Each variable g_i in the support set $\pi(g)$ of a pattern function is inserted

into a P_{cell} like PTL type 1 and type 2. The assignment of variables in the cluster function consists of two parts, one is for the cluster function whose cardinality of the support set is three, and the other is for the cluster function whose cardinality of the support set is greater than three. The first condition is for three input exclusive-OR (XOR) matches, and the second is for other matches with PTL type 3.

Let's consider the case when the cardinality of $\pi(f)$ equals three. Each support variable f_i in $\pi(f)$ is assigned to P_{net} when the support variables g_i of $\pi(g)$ are a or b ; otherwise, the variable f_i with polarity modification is assigned to P_{net} (statement (i)). If the support variable g_i of function g is c and L_{inv} contains b , d is assigned to P_{cell} (statement (ii)), the support variables of the cluster function assigned to c and d are interchanged, and the variables assigned to g and f are also interchanged.

When the cardinality of the matched cluster function is greater than three, the support variable f_i is assigned to P_{net} if the current support variable g_i of function g is a binate variable; otherwise, polarity modified variable $f_i \otimes \phi(g_i)$ is assigned to P_{net} . When g_i is not a binate variable, the contents of L_{inv} are examined (statement (iv) and (v)). The support variables of the cluster function assigned to c and d are interchanged if b is in L_{inv} , and the support variables of the cluster function assigned to f and g is interchanged if e is in L_{inv} . The last step of the pin assignment procedure of PTL type 3 returns a pin map list L_{pin} to Algorithm 3.8.1. As in the case of static match, the pin assignment procedure for PTL cells update the pin map list of the match structure, $\mathcal{M}.P$, to L_{pin} .

3.9 Matching Algorithm

Algorithm 3.9.1 illustrates the matching process. The algorithm is applied to find both static and PTL cell matches. In the argument list, n_i is a node whose function is f , g is the function of a library cell $\mathcal{C}$, and $sup(f)$ and $sup(g)$ represent

the support set of function f and g , respectively. Step 1 of Algorithm 3.9.1 is to check whether both function's support sets have the same cardinality. Step 2 finds symmetry sets[22, 27] after generating BDDs[25, 26] for both functions. The number of binate and unate variables of function f are found in step 3. Step 4 performs the actual matching, and step 5 assigns the support variables of f to the pins on cell C with appropriate polarity.

Finding a possible match between f and g begins by modifying the polarity of variables in the cell function g (statement (i) in step 4). Since function g only describes a basic function of a cell, the polarity of some variables in g need to be modified to allow more thorough exploration for matching. To check this possibility, the modified function $\tilde{g}$ is calculated by multiplying a phase function, $\phi(k)$, which is a binary mask. For example, if the number of support variables of g is 4, $\text{sup}(g) = \{a, b, c, d\}$, and the value of k is 0111 (7), then the polarity of the variables $\{b, c, d\}$ in g is reversed.

Algorithm 3.9.1: MATCH($n_i, f, g, \text{sup}(f), \text{sup}(g), C$)

```

1. if ( $|\text{sup}(f)| \neq |\text{sup}(g)|$ ) return
2.  $f_{\text{sym}} \leftarrow \text{GetSymmSet}(\text{bdd}(f), \text{sup}(f))$ 
3.  $\{b_f, u_f\} \leftarrow \text{GetNumBinateUnateVars}(f)$ 
   /* perform matching */
4. for  $k \leftarrow 0$  to  $N_{|\text{sup}(g)|} - 1$ 
   do
     (i)  $\tilde{g} \leftarrow g \otimes \phi(k)$ 
     (ii)  $\{b_{\tilde{g}}, u_{\tilde{g}}\} \leftarrow \text{GetNumBinateUnateVars}(\tilde{g})$ 
     (iii) if  $(b_{\tilde{g}} \neq b_f \mid u_{\tilde{g}} \neq u_f)$  continue
     (iv)  $\tilde{g}_{\text{sym}} \leftarrow \text{GetSymmSet}(\text{bdd}(\tilde{g}), \text{sup}(g))$ 
     (v) if  $(|f_{\text{sym}}| \neq |\tilde{g}_{\text{sym}}|)$  continue
     for each  $\pi_i \in \pi(\tilde{g}_{\text{sym}})$ 
       (vi)  $\text{bdd}(\tilde{g}_{\pi_i}) \leftarrow \text{GetBdd}(\tilde{g}, \pi_i)$ 
       do
         (vii)  $m \leftarrow \text{BddEqCheck}(\text{bdd}(f), \text{bdd}(\tilde{g}_{\pi_i}))$ 
         if  $(m)$  break
       if  $(m)$  break
5. if  $(m)$ 
   then PinAssign( $n_i, f, \tilde{g}_{\pi_i}, \pi(f), \pi(\tilde{g}_{\pi_i}), C$ )
6. return

```

The number of binate and unate variables, $b_{\tilde{g}}$ and $u_{\tilde{g}}$, in function $\tilde{g}$ is calculated in statement (ii), and $b_{\tilde{g}}$ and $u_{\tilde{g}}$ are compared to the binate and unate variables in function f . If $b_{\tilde{g}}$ and $u_{\tilde{g}}$ are not equal to those of function f , there is no match and the next phase function $\phi(k+1)$ is examined (statement (iii)). The number of symmetry sets of $\tilde{g}$, $|\tilde{g}_{sym}|$, is also calculated and compared to that of function f , $|f_{sym}|$. If they are not the same, there is no match (statement (iv) and (v)). The reason for finding symmetry sets and the number of unate and binate variables in both the cluster function and pattern function is to reduce the number of comparisons as described in section 3.7.

The inner *for* loop in step 4 of Algorithm 3.9.1 checks the equivalence between two BDDs, $bdd(f)$ and $bdd(\tilde{g}_{\pi_i})$. Due to different variable ordering, two BDDs representing the same function may not be the same [26, 59]. Since reordering variables of a function in its symmetry set doesn't affect the functionality and the BDD structure, an equivalence check can be carried out on the reordered BDDs. $bdd(\tilde{g}_{\pi_i})$ in the statement (vi) represents the BDD of $\tilde{g}$ after reordering the symmetry set $\tilde{g}_{sym}$. If two BDDs, $bdd(f)$ and $bdd(\tilde{g}_{\pi_i})$, are detected to be equivalent, the algorithm exits from both inner and outer *for* loops after setting the flag m . *PinAssign()* (step 5) assigns support variables of f to the pins of cell C with appropriate polarity when m is set. This information is stored in the internal data structure of node n_i for use in the covering stage.

3.10 Conclusions

In this chapter, an overview of the proposed mixed PTL/Static logic synthesis method is presented. The proposed logic synthesis flow begins by partitioning a logic network and decomposing a subject graph. The subject graph was generated based on multiple fanout points to reduce problem size and make matching and covering

problem tractable. The Boolean matching method is selected because of its quality, and a matching algorithm based on an existing Boolean matching theory is presented. This matching algorithm builds a match list for both static and PTL cells to use in the covering stage described in Chapter 4. The pin assignment between the inputs of a cluster function and a library cell function is also presented because it is very important for the covering stage to keep the correct network functionality of mapped circuits.

Chapter 4

Covering Using Genetic Algorithms

4.1 Introduction

Technology mapping is the last step in logic synthesis flow and finalizes the mapped network structure by selecting gates from a given library. Technology mapping consists of two steps. The first is the matching step described in the previous chapter. The second step is to select actual cells from a given library based on the cost associated with the cells. This step is called graph covering (or network covering) because selection of the best cells among the matches found in the matching stage is done in order to find the best covers by selecting the matches. Graph covering is an important step in technology mapping because it determines the final mapped circuit by using the library based on the cell's functionality and cost values.

In this chapter, a new graph covering algorithm will be presented. This proposed algorithm differs from existing algorithms mainly because it optimizes the network on a global level based on genetic algorithms, while the existing methods[13, 16] for mixed PTL do not consider global level optimization. The proposed algorithm can also be applied to technology mapping using static CMOS technology mapping.

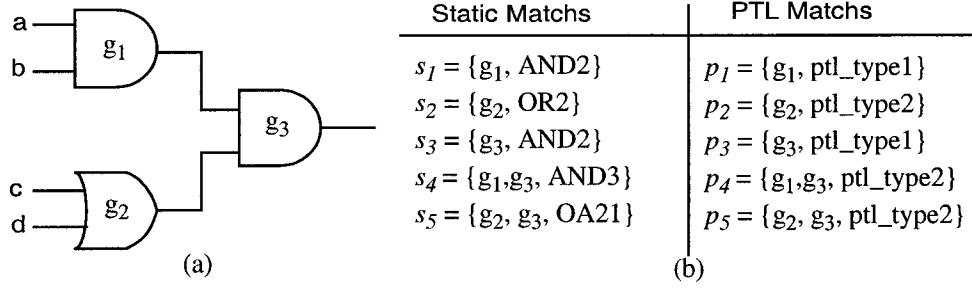


Figure 4.1: Logic network and matches. (a)Logic network. (b)Static and PTL matches.

This chapter is organized as follows: In Section 4.2, the graph covering problem is formulated as a satisfiability problem that is an NP-complete[33] problem. In Section 4.3, previously proposed problem solving techniques for the problem will be reviewed. The overview of the proposed covering algorithm using genetic algorithms is presented in Section 4.4. In Section 4.5, the encoding method for the proposed algorithm is described. The initialization methods of sub-population and global population are described in Section 4.6. In Section 4.7, the details of genetic operators used in the proposed algorithm are presented. The overall proposed algorithm description is given in Section 4.8. The convergence pattern of the algorithm and the impact of genetic parameters on the proposed algorithm are examined in Section 4.9. In Section 4.10, the performance enhancing methods for the mapped circuits are also presented. Finally, concluding remarks are given in Section 4.12.

4.2 Graph Covering Problem

The graph covering problem for mixed PTL/Static circuits can be described as follows. Given a graph $G = (V, E)$ and two sets of matches, static matches $S = \{s_i, 1 \leq i \leq n\}$ and PTL matches $P = \{p_i, 1 \leq i \leq m\}$, select the best matches that generate a legal network such that cost is minimized while given constraints are met. In other words, the covering problem can be modeled by selecting enough matches

in S and P that cover all nodes of the network in the best way. For each selected match, we must ensure that all the inputs of the matches are either primary inputs or outputs of other matches. Therefore, the choice of a match implies the choice of some other matches. This network covering problem is also known as a binate covering problem[34]. In order to have valid solution, each internal node in the solution must be covered by at least one match. This condition can usually be met by decomposing the network using base functions in the library before the matching step is applied so that all the nodes in the network can have at least one match in the library.

Consider the situation shown in Figure 4.1. This figure shows a logic network and possible static and PTL matches for the logic network. In order to select best matches among all possible static and PTL matches, each static and PTL match, s_i and p_i , can be associated with a binary value. The requirement that the gate g_1 be covered by at least one library gate is

$$(s_1 + s_4 + p_1 + p_4) = 1 \quad (4.1)$$

Similarly, the covering requirements for g_2 and g_3 can be represented by clauses $(s_2 + s_5 + p_2 + p_5)$ and $(s_3 + s_4 + s_5 + p_3 + p_4 + p_5)$, respectively. In addition to these requirements, to ensure that selected covers lead to a valid circuit, both static and PTL matches that cover the same gate(s) cannot be selected simultaneously, and additional clauses need to be generated. These conditions lead to a clause of XOR term, $s_i \oplus p_i$, for each static and PTL match. And, assuming that a static (PTL) gate cannot drive a static (PTL) gate, selecting s_3 at g_3 requires that p_1 and p_2 be chosen at g_1 and g_2 , respectively. Therefore, to get a valid solution, the following clause must hold:

$$\begin{aligned} & (s_1 + s_4 + p_1 + p_4)(s_2 + s_5 + p_2 + p_5)(s_3 + s_4 + s_5 + p_3 + p_4 + p_5) \\ & (s_1 \oplus p_1)(s_2 \oplus p_2)(s_3 \oplus p_3)(s_4 \oplus p_4)(s_5 \oplus p_5)(\overline{s_3} + p_1)(\overline{s_3} + p_2) \\ & (\overline{p_3} + s_1)(\overline{p_3} + s_2)(\overline{s_4} + p_2)(\overline{s_5} + p_1)(\overline{p_4} + s_2)(\overline{p_5} + s_1) = 1 \end{aligned} \quad (4.2)$$

This is an satisfiability problem that is known to be an NP-complete[33] problem. For this reason, many heuristic algorithms have previously been developed to approximate the solution of the network covering problem.

4.3 Previous Works and Limitations

Most of the previous methods for covering are based on greedy or dynamic programming[17] methods. However, not many works have been reported for mixed PTL logic synthesis because those works were intended for static use only. This section reviews the previous works to solve the problem for both static and mixed PTL/Static technology mapping.

A greedy search algorithm was proposed in [13] to determine the best mappings for PTL and static cells. Although a greedy search method is simple and can be easily applied to an optimization problem, it cannot guarantee the optimality of the solution because it is prone to getting stuck at local minima. Therefore, it is hard to find a global optimum for the graph covering problem described above.

The dynamic programming method is an algorithmic method that solves an optimization problem by decomposing it into a sequence of smaller decisions, and it relies on the *principle of optimality*¹[17]. Because the problem needs to be decomposed into a sequence of subproblems in order to apply dynamic programming, it is very effective to use tree structure instead of directed acyclic graph (DAG). Thus, this method has been widely used to solve covering problems by approximating the problem into a forest of trees. The covering methods using dynamic programming[18, 19, 20, 21],

¹The principle of optimality can be described as follows. An optimal sequence of decisions has the property that whatever the initial state and decisions are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

however, have the main drawback that dynamic programming does not guarantee optimality for DAG covering in terms timing or power because the load capacitance of a node is not determined yet when a match is considered. Therefore, the delay cannot be calculated at the time of the mapping.

In [19], an improved method for static CMOS logic mapping, has been proposed to estimate delay during the mapping process. First, n load bins are created at each node, and delay is calculated in the postorder tree traversal. For each bin, the match that gives minimum arrival time at a node output is stored. The final selection is made during preorder traversal by selecting matches depending on the load seen at each node.

In [20, 21], the author proposed an area-delay trade-off curve that utilizes only non-inferior area-delay points, but the load capacitances are still not determined during the postorder traversal. Additionally, the search for optimal mappings in [19, 20, 21] was applied only within a sub-tree of a given circuit represented by a DAG. Sub-trees are separated by multiple fanout points in the DAG. The cell interactions between sub-trees were not considered, and no attempt was made to optimize the overall circuit at the global level. The existing optimization techniques for mixed PTL logic[14, 16] using dynamic programming for the covering are also sub-optimal because the optimization is only restricted to within a sub-tree separated by multiple fanout points of a given circuit.

To overcome these drawbacks, in this dissertation, the genetic algorithms[60, 61] (GA) will be used to construct the best mapped circuit from the matches generated in the matching stage. The basic philosophy of the proposed approach is to consider the cell interactions between sub-graphs and attempt to find the optimum solution at the global level. Starting the next section, the details of the proposed algorithm will be described.

4.4 Overview of the proposed covering algorithm

The flowchart of the proposed covering algorithm is shown in Figure 4.2. The graph in the diagram represents the logic network that contains static and PTL matches found in the matching stage and is the target to be optimized. Sub-populations for each sub-graph are initialized the first time; then the global population is formed by selecting sub-chromosomes from the populations based on their fitness. The cost of each global chromosome is then evaluated once before genetic operators are applied to the populations.

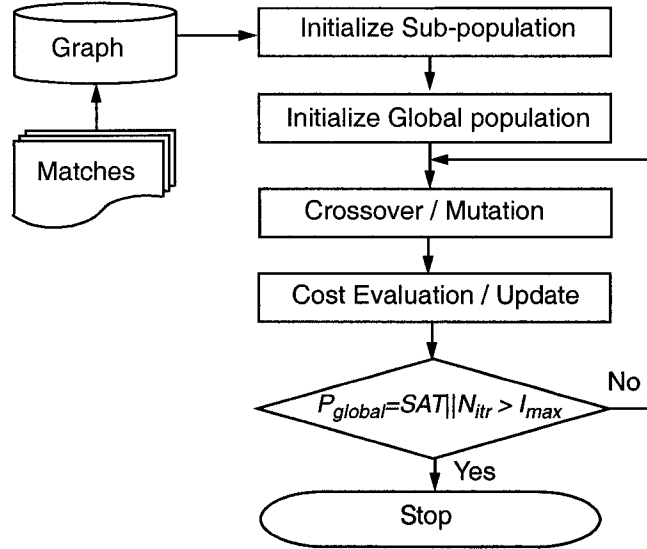


Figure 4.2: Flow chart of the proposed covering algorithm.

Once the populations are initialized, crossover operations are applied to individuals that are randomly selected from the sub-populations. Mutation operations are also applied to some individuals when the algorithm triggers the mutation based on the overall population fitness. Cost evaluation and update need to be done after the genetic operators are applied in order to make the populations evolved. The stopping criteria of the proposed algorithm are that either the global population is saturated

or the pre-specified maximum number of iterations for the crossover/mutation and cost evaluation/update routines are reached.

4.5 GA Encoding

Before the genetic operators are applied to the population, the information of the logic network and the matches need to somehow be encoded. Generally genetic algorithms (GAs) use fixed-length binary strings to represent a solution. However, binary strings are not efficient to represent the mapped circuit, because each node in the graph can have multiple matches. In this proposed method, a hierarchical GA structure is adopted reflecting the existence of subject graph. Each solution for an entire circuit, referred to as a global chromosome, is represented by a graph structure. As the entire circuit is divided into sub-trees by multiple fanout points (MFP), a sub-population is generated for each sub-tree section, and GAs are applied to each sub-population. When a new sub-population of chromosomes is formed, a chromosome from each sub-population is then selected based on its *fitness* to form a new global chromosome for the new global population.

Figure 4.3 illustrates the proposed encoding method. As shown in Figure 4.3(a), all the matches found in the matching stage are mapped to *genes*, and each gene represents either a static gate match or a PTL match. Figure 4.3(b) shows two sub-chromosomes for each sub-graph. A *sub-chromosome* represents a feasible mapping solution for a sub-graph, and it is formed by selecting possible matches at each node as shown in Figure 4.3(b). A *global chromosome* is a mapping solution of the entire graph, as shown in Figure 4.3(c). Figure 4.3(c) shows a mapping solution of an entire graph that is formed by selecting one of the two sub-chromosomes. Therefore, a hierarchical GA structure is used for graph covering.

Figure 4.4 shows the hierarchy of the GA structure used for the proposed covering algorithm. Each sub-population contains a number of sub-chromosomes, each of which represents a solution of a subject graph. In Figure 4.4, sub-population and global population represent a set of sub-chromosomes and a set of global chromosomes, respectively.

To avoid having long pass-transistor chain during the generation of chromosomes, the following rules were applied in the covering process in order to keep a maximum of three pass-transistors in series. In order to increase the number of PTL gates, rule number 5 was included.

1. PTL+ Type1, Type2 and Type3 can always drive static gates.
2. PTL+ Type1 can drive PTL+ Type2 if the inputs of Type1 are not driven by Type2 or Type3.
3. PTL+ Type1 can drive PTL+ Type3 if the inputs of Type1 are not driven by a PTL cell.
4. PTL+ Type2 and Type3 can drive Type1 if the inputs of Type2 and Type3 are not driven by a PTL+ cell.
5. A static gate cannot drive any other static gate.

Any global chromosome that violates the rules is an invalid chromosome; invalid global chromosomes are discarded. Update of the global population is synchronized with the population evolution in each sub-population until the design goal is achieved or the maximum number of iterations has been reached.

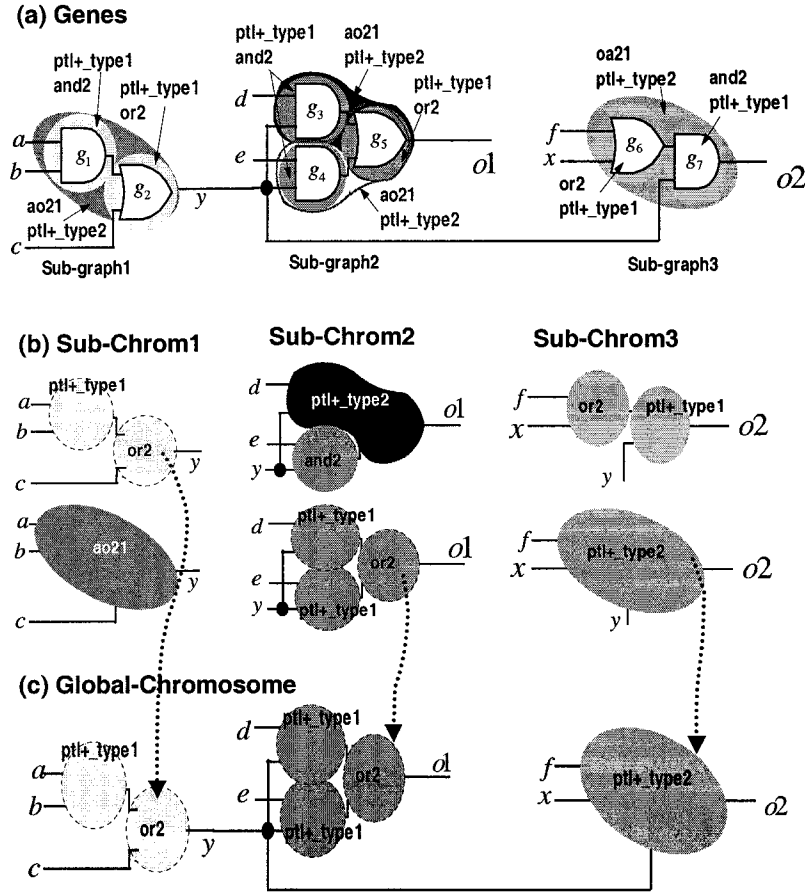


Figure 4.3: An example of Graph Encoding for GA. (a)Genes. (b)Sub-chromosomes. (c)A global-chromosome.

4.6 Initializing sub/global populations

This section describes the method of sub-populations and global population. Because the graph are decomposed based on multiple fanout points, there can be a number of sub-populations while there is only one global population.

4.6.1 Sub-Chromosome and Sub-population

Sub-chromosomes are generated by randomly selecting matches among the static and PTL matches. This is done for each sub-graph. First, the type of match (static

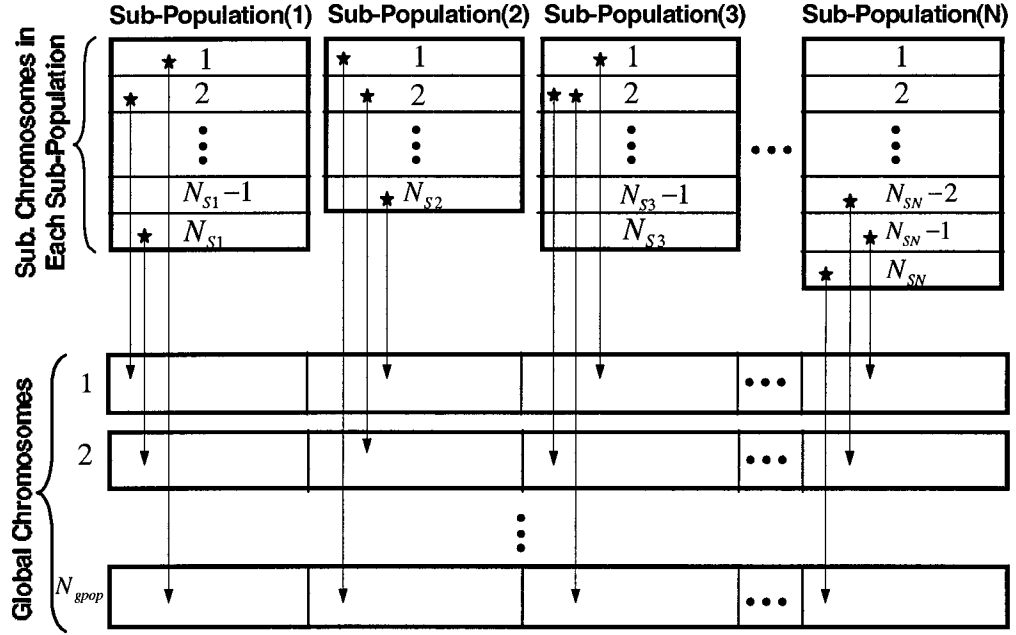


Figure 4.4: The hierarchical relationship between sub-chromosome and global chromosome.

or PTL) is selected at the root node of a sub-graph by generating a random number, γ_t^{root} , in $[0,1]$. If γ_t^{root} is 0, the static match type at root node is assigned; otherwise, the PTL match type is assigned. Once the root type is determined, a match, referred to as n_{map}^{root} , is selected among static or PTL matches at the root node. Then, the sub-graph is traversed from the root node to the leaves in the pattern of breadth-first-search (BFS). For each input node of the selected match for the root node, a new type of match is determined. Then, another match is randomly selected from the selected type of matches at the node, and validity is checked if the selection meets the rules described above. The process will check another match at the node by generating a different random number from the previous random number when the validity check fails. A legal sub-chromosome is stored into the sub-population, and the sub-population size, N_{subpop} , is a parameter that is provided as an external input.

4.6.2 Global Chromosome and Global Population

A global chromosome is formed by selecting different sub-chromosomes from each sub-population. The roulette wheel[60] method is used to determine the selection of a sub-chromosome in a sub-population. Algorithm 4.6.1 shows the details of how a global chromosome is constructed. The random number for the roulette is generated by the following equation:

$$N_{rand} = C_{subpop}^t * RAND \quad (4.3)$$

where RAND is a random number between 0 and 1, and C_{subpop}^t is the total sum of each chromosome cost in the population. Then, for each sub-chromosome, ω_i^j , the cost is calculated and is accumulated in the variable, F_{subpop}^{par} , referred to as the fitness of the partial sub-chromosome. When F_{subpop}^{par} exceeds N_{rand} after the current sub-chromosome's cost is reflected, the sub-chromosome is selected as a member of current global chromosome.

Algorithm 4.6.1: GETGLCHROMOSOME($\mathcal{G}, \Omega$)

```

output ( $\zeta$ , mapped network)
/* initialize global population */
1. for each  $\omega_i \in \Omega$ 
    do {
         $C_{subpop}^t \leftarrow CalculateTotalCost(\omega_i)$ 
         $N_{rand} \leftarrow C_{subpop}^t * RAND$ 
        for each  $\omega_i^j \in \omega_i$ 
            do {
                 $f_{\omega_i^j} \leftarrow 1/C_{\omega_i^j}$ 
                 $F_{subpop}^{par} \leftarrow F_{subpop}^{par} + f_{\omega_i^j}$ 
                if  $F_{subpop}^{par} > N_{rand}$ 
                    then  $\zeta \leftarrow \zeta \cup \omega_i^j$ 
            }
    }
/* determine the validity of global chromosome */
2. if ( $CheckValidity(\zeta)$ )
    then return ( $\zeta$ )
    else return ( $NULL$ )

```

Because all the inputs of a sub-chromosome must be connected properly to the output of another sub-chromosomes in a global chromosome, a global chromosome must be legal. All the sub-chromosomes, therefore, are sorted based on their output depth, referred to as *global depth*, in a global chromosome before checking the legality of connection. The global depth for each sub-chromosome of a global chromosome is determined by traversing the global chromosome from primary inputs to primary outputs. After a global chromosome is generated, its validity is checked. Any global chromosome that violates the rules is an invalid chromosome, and any invalid global chromosome will be discarded.

4.7 GA Operators

The proposed algorithm uses two genetic operators called *crossover* and *mutation*. The following two subsections describe the details of the two genetic operators.

4.7.1 Crossover Operation

The main purpose of the crossover operation is to introduce new mapped solutions that differ from the solutions in the current sub-population so that the cost of the children moves toward the desirable direction. In this proposed synthesis flow, the objective is to minimize the cost defined as the total transistor area under delay constraints. The crossover operation is applied to two parents (sub-chromosomes), *parent1* and *parent2*, of a sub-population, and these parents are randomly selected from the sub-population in such a way that the sub-chromosomes with lower cost are most likely to be selected. The reason for applying crossover to the sub-chromosomes is that the entire circuit was decomposed based on multiple fanout points. Two new sub-chromosomes that are created after applying crossover are called children, *child1* and *child2*.

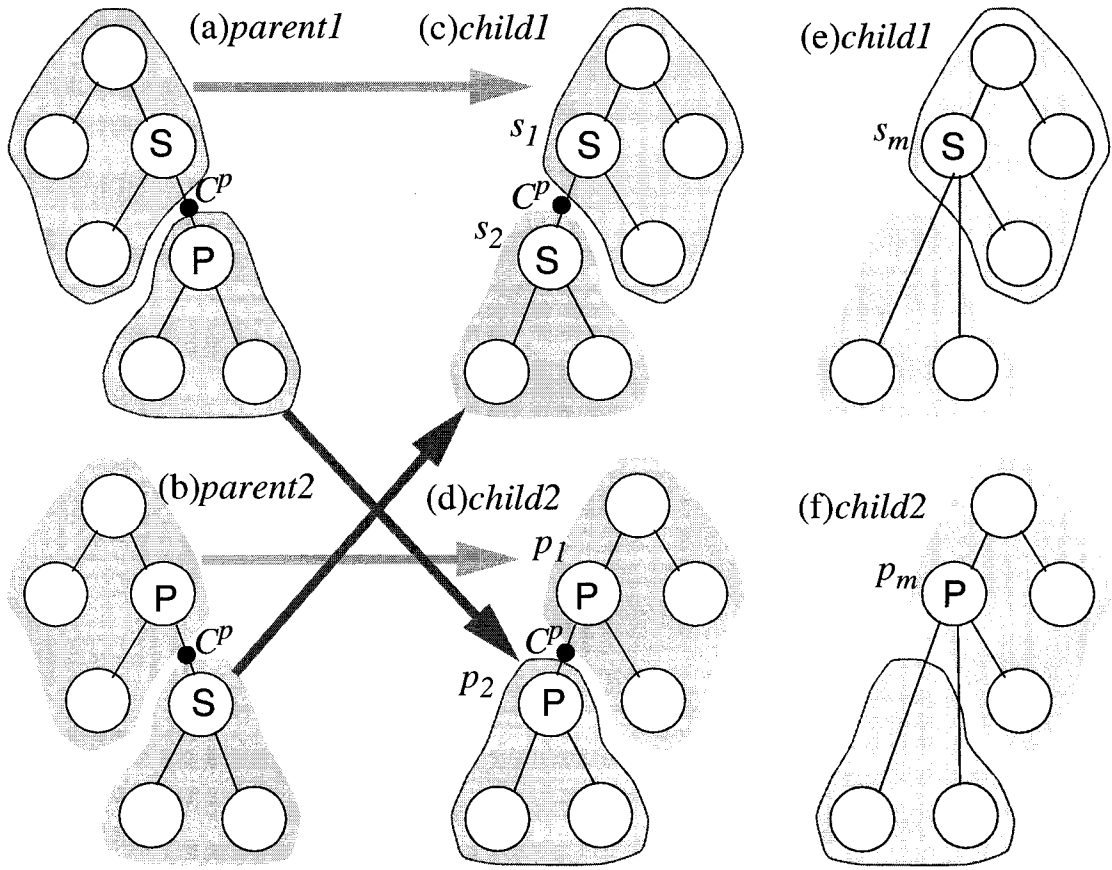


Figure 4.5: Crossover operation.

Figure 4.5 shows the concept of the crossover operation. Figures 4.5 (a) and (b) are the two parents, *parent1* and *parent2*, respectively. A crossover point, c^p , for each parent must be selected before the crossover operation. The crossover point must be the same in each parent; otherwise, the validity of each child after crossover cannot be guaranteed because nodes around c^p can be duplicated.

Once the crossover point is determined, the crossover operation is performed by exchanging the fanin cones rooted at c^p of *parent1* and *parent2*, as shown in Figure 4.5 (c) and (d). Any invalid child chromosome according to the covering rules will be discarded. Before deciding the validity of the children, the merging (also called remapping) process is performed by checking the two nodes around crossover point c^p .

For example, as shown in Figures 4.5 (c) and (d), the static (PTL) nodes s_1 (p_1) and s_2 (p_2) can be merged if matches at node s_1 (p_1) contain a static (PTL) match that covers both node s_1 (p_1) and node s_2 (p_2). Figures 4.5 (e) and (f) show the children after merging the nodes around the crossover point c^p (not shown in the figures). The merged nodes for static and PTL in the figure are s_m and p_m , respectively. This merge process of two different cells into one cell positively impacts power and performance because the effective area is reduced.

Algorithm 4.7.1: GACROSSOVER($parent1, parent2$)

```

/* procedure to partial copy */
procedure CPCH( $p, n_i, cpn$ )
1.  $Q \leftarrow n_i$ 
2. while ( $Q \neq \phi$ )
    {
         $n_c^p \leftarrow Q.Dequeue()$ 
         $n_c^c \leftarrow CreateNode(n_c^p)$ 
         $G_{ch} \leftarrow G_{ch} \cup n_c^c$ 
        for each  $n_{in} \in n_c^p$ 
            do {
                 $n_{in}^c \leftarrow CreateNode(n_{in})$ 
                 $G_{ch} \leftarrow Fanin(n_c^c, n_{in}^c)$ 
                 $G_{ch} \leftarrow Fanout(n_{in}^c, n_c^c)$ 
                do {
                    if ( $cpn \neq NULL$  and  $n_{in}^c \neq c^p$  and  $nFUNC(n_{in}^c) \neq PI$ )
                        then  $Q \leftarrow n_{in}^c$ 
                    else if ( $nFUNC(n_{in}^c) \neq PI$ )
                        then  $Q \leftarrow n_{in}^c$ 
                }
            }
    }
3. return ( $G_{ch}$ )

main
1.  $C^p \leftarrow getCrossPoint(parent1, parent2)$ 
/* perform crossover */
2.  $child1 \leftarrow CPCH(parent1, n_{p1}^{root}, c^p)$ 
3.  $child2 \leftarrow CPCH(parent2, n_{p2}^{root}, c^p)$ 
4.  $child1 \leftarrow CPCH(parent2, c^p, NULL)$ 
5.  $child2 \leftarrow CPCH(parent1, c^p, NULL)$ 
6. CheckValidity_Merge_Update( $child1, child2$ )

```

Algorithm 4.7.1 illustrates the crossover operation. The procedure $CPCH()$ in Algorithm 4.7.1 partially copies nodes from a parent to a child based on BFS search. The first step of the crossover process, step 1 of the main procedure, is to determine the crossover point c^p . As shown in step 2 and 3 in the main procedure of Algorithm 4.7.1, the construction of *child1* and *child2* is started by partially duplicating *parent1* and *parent2* rooted at the nodes n_{p1}^{root} and n_{p2}^{root} , respectively. During the partial chromosome duplication process, the procedure $CPCH()$ does not insert into the queue for further processing when it meets c^p or primary inputs (PIs).

The actual crossover takes place in step 4 and 5 of the main procedure. Step 4 copies all the nodes in the fanin cone of *parent2* rooted at c^p to *child1*, while step 5 copies all the nodes in the fanin cone of the *parent1* rooted at c^p to *child2*. After the crossover operation is performed, the validity of *child1* and *child2* is checked; then the merging process is performed when the merge is possible as described above. The last step of the crossover operation is to update the sub-population. Any child will be replaced with a sub-chromosome in the sub-population when the cost of the child is smaller than that of a sub-chromosome that has the biggest cost.

4.7.2 Mutation Operation

Mutation is another way to introduce diversity into the population. After several generations of a population, it is possible that the crossover will drive some genes in a certain position on chromosomes to have the same kind of static or PTL matches. In this case, then, the population cost will converge prematurely. Without the mutation operator, it is hard to reintroduce alternative solutions. The mutation operator, therefore, is used to reintroduce alternative solutions and is used as a hill-climbing mechanism.

The mutation operation is initiated when the average cost difference between the current global population and the previous global population is within a certain

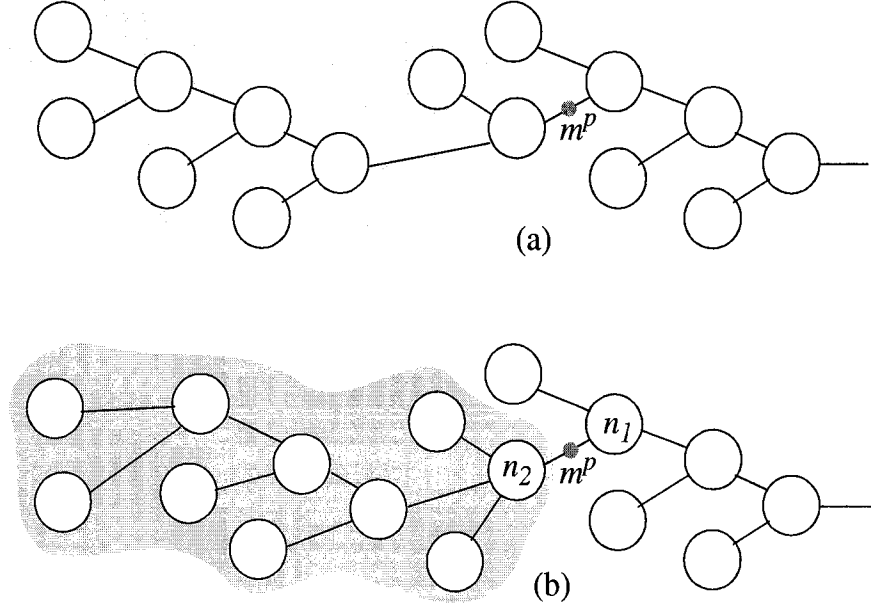


Figure 4.6: Mutation Process. (a) before mutation. (b) after mutation.

range, δ , which is referred to as the *mutation threshold*. The amount of mutation for a population is governed by a variable called *mutation rate*, μ . To maximize the preservation of the characteristics of selected and evolved samples, the disruptive effects should be minimum. This suggests that mutation should not be applied to all the populations[62, 63, 64]. In other words, μ need to be much lower than 1. Figure 4.6 shows the conceptual process of the mutation, in which m^p represents the mutation point that is the root of the cone where the mutation is applied. As shown in Figure 4.6, the cone rooted at m^p is deleted and replaced with a new cone that is a newly mapped solution from the mutation point m^p down to the leaves.

Algorithm 4.7.2 shows the detailed process of mutation for a sub-chromosome. This algorithm takes a sub-chromosome ω^s as an input and returns a mutated sub-chromosome ω_μ^s . At the beginning of the process, a node, n_{map} , that is the mutation point in a chromosome is randomly selected. Then, in step 2, the current match type, f_{mat}^{cur} , and the match list of the node, L_{mat} , is obtained by a routine *GetMatchList()*.

Step 3 of the algorithm finds a new match, f_{mat}^{new} , whose match type is the same as f_{mat}^{cur} , but the cover by this match is different from that of the match f_{mat}^{cur} . The new match is indexed by a random number between 1 and N_{mat} that represents the number of matches at the node n_{map} . If the covers of f_{mat}^{cur} and f_{mat}^{new} are different, the while-loop is exited by the process. A new sub-chromosome, ω_{μ}^s , is built by the *BuildNewCone()* routine after deleting all nodes of the fanin cone rooted at n_{map} .

Algorithm 4.7.2: GAMUTATION(ω^s)

```

output ( $\omega_{\mu}^s$ , mutated network)
1.  $n_{map} \leftarrow SelectNode(\omega^s)$ 
2.  $(f_{mat}^{cur}, L_{mat}) \leftarrow GetMatchList(n_{map})$ 
3. while (1)
   do  $\begin{cases} n_{pos} \leftarrow Rand(N_{mat}) \\ f_{mat}^{new} \leftarrow GetMatch(L_{mat}, n_{pos}) \\ \text{if } (f_{mat}^{cur} \neq f_{mat}^{new}) \\ \quad \text{then break;} \end{cases}$ 
4.  $\omega_{\mu}^s \leftarrow DeleteCone(\omega^s, n_{map})$ 
5.  $\omega_{\mu}^s \leftarrow BuildNewCone(\omega^s, n_{map}, f_{mat}^{new})$ 
6. return ( $\omega_{\mu}^s$ )

```

4.8 Covering Algorithm

This section describes the proposed graph covering that combines all the genetic operators described above. Algorithm 4.8.1 illustrates the proposed covering process using the genetic algorithms. In the argument list of *GAMAP()*, $\mathcal{G}$ represents the set of subject graphs that contain a set of possible matches found in the matching stage at every node, N_{gpop} is the size of the global population, δ is a threshold value that enables the *gaMutation()* routine, μ is the mutation rate, and I_{max} is the maximum number of allowed iterations.

The GA begins by generating sub-populations that represent the mixed PTL/CMOS mapping solutions for each subject graph g_i (step 1). *Generate_Sub_Population()* takes a subject graph g_i as input and returns a sub-graph population ω_i that contains a finite number of sub-chromosomes. Ω represents a set of sub-populations. The size of a sub-population varies with the number of nodes in graph g_i .

The next step is to initialize the global population $\mathcal{P}_g$ (step 2). A global chromosome ζ_i is formed by selecting sub-chromosomes in the routine *GetGlChromosome()*. In step 3, genetic operators such as crossover and mutation are applied to the sub-population, the cost of new chromosomes is calculated and the sub-population is updated.

Algorithm 4.8.1: GAMAP($\mathcal{G}, N_{gpop}, \delta, \mu, I_{max}$)

```

output ( $\zeta$ , mapped network)
/* initialize sub_population */
1. for each  $g_i \in \mathcal{G}$ 
   do  $\begin{cases} \omega_i \leftarrow \text{Generate\_Sub\_Population}(g_i) \\ \Omega \leftarrow \Omega \cup \omega_i \end{cases}$ 
/* initialize global population */
2. for  $i \leftarrow 1$  to  $N_{gpop}$ 
   do  $\begin{cases} \zeta_i \leftarrow \text{Get\_Global\_Chromosome}(\Omega) \\ \mathcal{P}_g \leftarrow \mathcal{P}_g \cup \zeta_i \end{cases}$ 
/* perform optimization */
3. for each  $\omega_i \in \Omega$ 
    $\begin{cases} \omega_i^t \leftarrow \text{gaCrossover}(\omega_i) & \text{(i)} \\ \text{if } ((\text{cost}_{avg}(\omega_i^t) - \text{cost}_{avg}(\omega_i^{t-1})) < \delta) & \\ \quad \text{then } \text{gaMutation}(\omega_i^t, \mu) & \text{(ii)} \end{cases}$ 
   do  $\begin{cases} \Omega^t \leftarrow \{\Omega - \omega_i\} \cup \omega_i^t \\ \zeta_i^t \leftarrow \text{Get\_Global\_Chromosome}(\Omega^t) \\ \text{if } (\text{cost}(\zeta_i^t) < \text{cost}(\zeta_{max}) \text{ and meets const.}) & \\ \quad \text{then } \mathcal{P}_g^t \leftarrow \{\mathcal{P}_g - \zeta_{max}\} \cup \zeta_i^t \end{cases}$ 
4.  $N_{itr} \leftarrow N_{itr} + 1$ 
5. Repeat step 3 until  $N_{itr} \geq I_{max}$  or  $\mathcal{P}_g^t = SAT$ 
6.  $\zeta = \min(\zeta_i)$ , where  $\zeta_i \in \mathcal{P}_g^t$ 

```

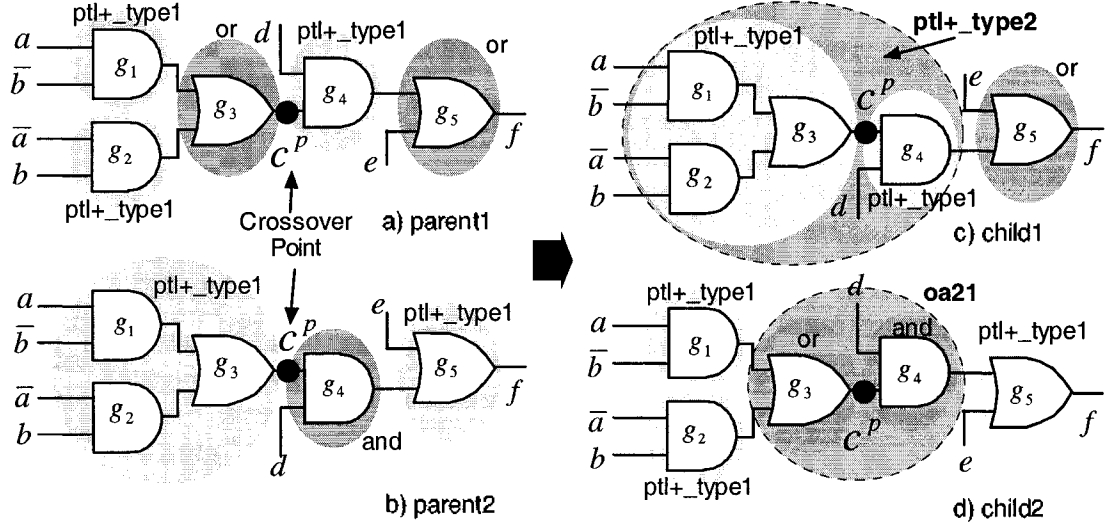


Figure 4.7: Example of mapped graph crossover.

An example of crossover on the subject graph is illustrated in Figure 4.7. Figures 4.7 (a) and (b) represent the two parents, $parent_1$ and $parent_2$, which are randomly selected from sub-population ω_i to perform the crossover operation. These parents show the mapping solutions for a sub-graph before crossover, and the PTL+ cell assignments are also shown in Figure 4.7. The crossover operation begins by selecting a crossover point c^p from the parents. After the crossover point is selected, the two sub-trees of $parent_1$ and $parent_2$ rooted at c^p are crossed over to form two new child chromosomes, as shown in Figures 4.7 (c) and (d).

Validity of each child is checked. For any invalid child chromosome, an attempt is made to re-map the invalid part to make it valid. Re-mapping is also attempted to improve the *fitness* (the total transistor gate area) of the child chromosomes as shown in Figure 4.7(b) for $child1$, where merging two Type1 cells into a Type2 cell resulted in an overall reduction in sizes. After re-mapping, each valid child will survive if its fitness is higher than the worst chromosome in the existing sub-population ω_i .

The routine $gaMutate()$, statement (ii), performs the mutation operation when the average cost difference between previous generation ω_i^{t-1} and current generation

ω_i^t is less than a threshold δ . Figure 4.8 illustrates an example of the mutation process. Mutation begins by randomly selecting a node, m^p , from a sub-chromosome, as described above. All the mapped nodes in the fanin cone rooted at m^p are deleted. We then build a new mapped circuit for the removed fanin cone. Figure 4.8 shows an example of the mutation process. In this example, the mappings for the gates g_1 to g_6 are deleted first and a new mapping solution is formed, as shown in Figure 4.8 (b). Thus, a new combination of covering is introduced in the sub-population.

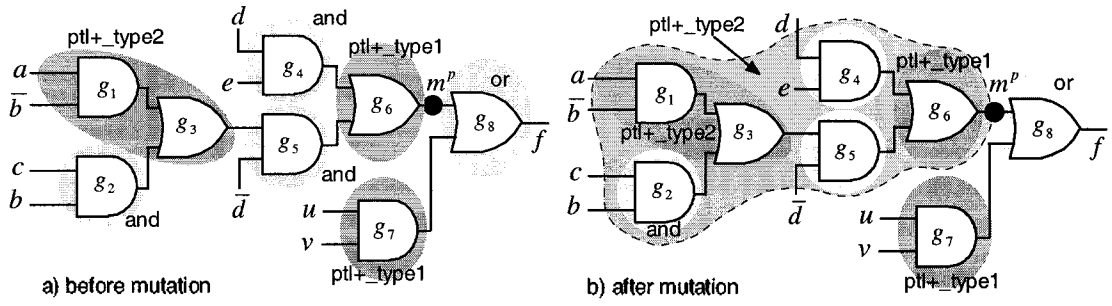


Figure 4.8: Example of a mapped graph mutation.

After crossovers and mutations are done, the sub-population list Ω is updated (Ω^t), and a new global chromosome ζ_i^t is formed to update the global population $\mathcal{P}_g$. $\mathcal{P}_g^t$ represents the updated global population in each iteration. As described in Section 4.5, any global chromosome that violates the rules is an invalid chromosome, invalid global chromosomes are discarded. Update of the global population is synchronized with the population evolution in each sub-population until the design goal is achieved or the maximum number of iterations has been reached.

Steps 3 and 4 are repeated until the N_{itr} reaches I_{max} or the global population is saturated. The last step is to select a global chromosome that has the minimum cost as the final solution. The cost function (or fitness function) in GA can be specified. Because the average power of a given circuit is also linearly proportional to the transistor size given a fixed activity factor at each node, the overall size of transistors

is used as the cost (*fitness*) throughout the experiments performed in this dissertation.

4.9 Evaluation of GA Parameters

Parameters used in GA, such as population size, mutation rate, and mutation threshold can affect the performance of the optimization algorithm and the quality of the final results. Figure 4.9 shows the number of iterations to converge with respect to mutation rate (μ) and percent threshold value that triggers mutation (δ). The maximum number of iterations to converge is relatively flat. Nevertheless, when $\delta = 0.85\%$ and $\mu = 0.35$, the required number of iterations is the smallest compared to other μ and δ settings.

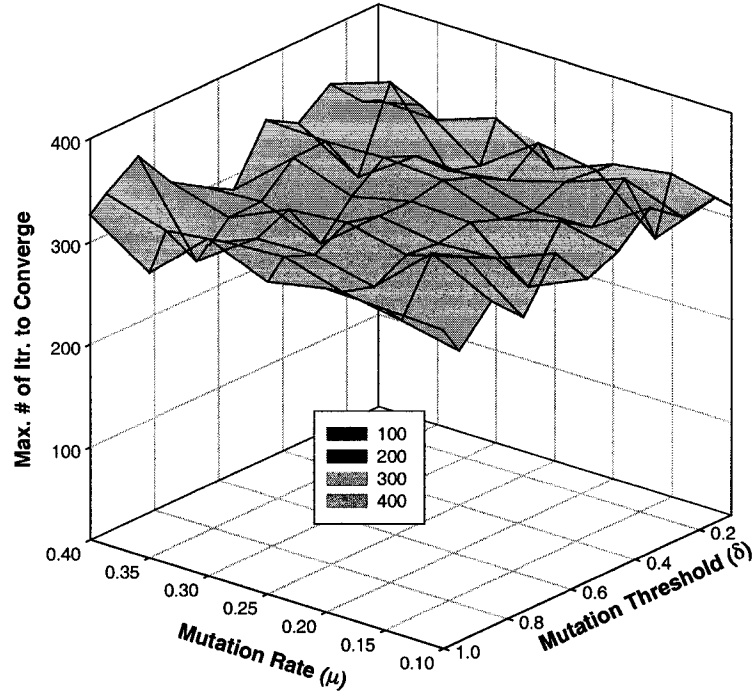


Figure 4.9: Number of iterations to converge (C499, $N_{subpop} = 25$, $N_{gpop} = 15$).

The final converged cost as a function of maximum sub-population size is shown in Figure 4.10 (a). The results show that the saturation cost (i.e., quality of the solu-

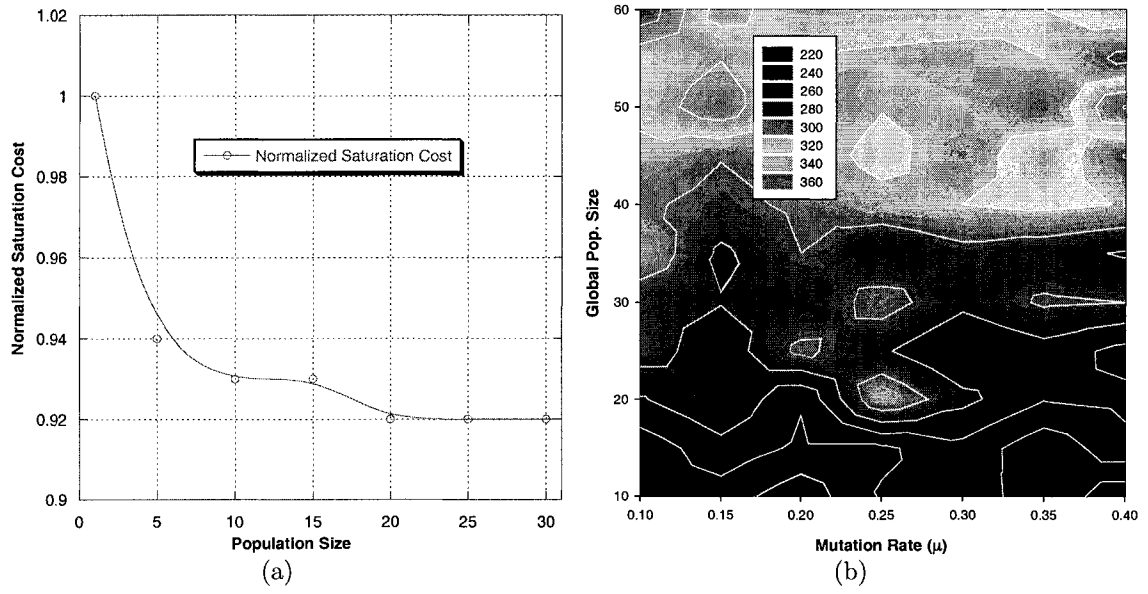


Figure 4.10: Convergence value and number of iterations to converge with respect to various genetic parameters. (a) Normalized convergence value with respect to sub-population size (C499, $\delta = 0.85$, $\mu = 0.35$, $N_{gpop} = 15$). (b) Number of iterations to converge with various mutation rate and global population size (C499, $\delta = 0.85$, $N_{subpop} = 25$).

tion) highly depends on the maximum sub-population size and that the convergence performance is relatively insensitive to the choice of the mutation rate and mutation threshold. Figure 4.10 (b) shows the maximum number of iterations to converge with respect to μ and global-population size. Based on the experimental results and the above observations, the GA parameters of global population size, sub-population size, mutation rate, and mutation threshold for the experiments presented in Chapter 5 were chosen to be 15, 25, 0.35, and 0.85%, respectively.

The experimental results also show that the GA-based mapping algorithm converges well. Most of the circuits obtained a significant amount of improvement on cost with only fewer 500 iterations, and the average cost improvements were 18%. Figure 4.11 shows the normalized cost convergence pattern for C6288, which is one of the largest ISCAS85 benchmark circuits, and C1355.

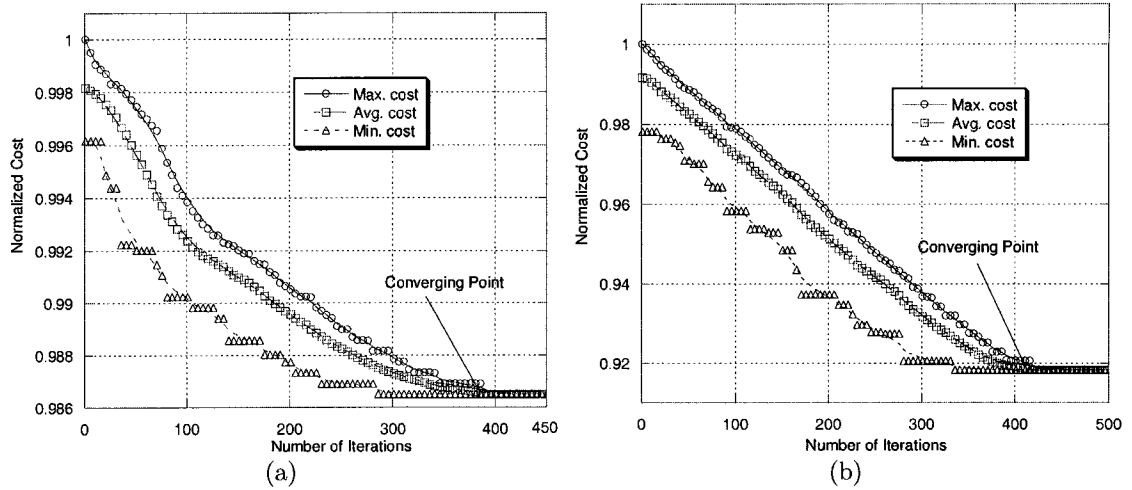


Figure 4.11: Convergence result of the proposed method. (a) Convergence result of C6288. (b) Convergence result of C1355.

4.10 Performance Enhancement

Fanout optimization is required during technology mapping (after covering) to meet certain delay constraints. Fanout optimization is a fundamental problem in timing optimization. Because this problem is known as an NP-complete problem[65], many of the previous works[66, 67, 68, 69, 70] use heuristics to solve the fanout optimization problem. These methods are widely used for timing optimization for static CMOS circuits.

In this section, a new delay reduction method for PTL synthesis by performing an automatic redundant buffer identification and removal is presented. The algorithm also performs fanout optimization similar to the one described in [65, 67, 71]. The impact of these features during synthesis on dual-rail mixed PTL/CMOS circuits using an experimental $0.1\mu m$ CMOS technology was examined.

4.10.1 Delay Model

In general, a logic network can be represented by a directed acyclic graph (DAG) where each node represents a gate and each arc represents a connection from one gate

to another gate. Delay of a gate can be expressed as follows:

$$t_d = \max_{i \in \text{inputs}} \{D_{i,int} + D_{i,load}C_L\} \quad (4.4)$$

where t_d is a gate delay, $D_{i,int}$ is the load independent intrinsic delay of the gate, $D_{i,load}$ is the drive resistance of the gate from input i to output, and C_L is the load capacitance at the gate output.

4.10.2 Redundant Buffer Removal

Assuming that the dual-rail mixed PTL circuits were generated by replacing the single-rail cells with dual-rail cells, all gate outputs in dual-rail mixed PTL circuits generate a signal of one polarity and another signal with the opposite polarity. Therefore, unlike in the case of single-rail circuits, only inverting buffers are required, and some of the inverting buffers are redundant in the case of dual-rail circuits². However, simple conversion from non-inverting buffers to inverting buffers by removing one of the inverters in the non-inverting buffers may have a negative impact on delay because an amount of fanout at a node may require multi-stage buffering. Therefore, the use of inverting and non-inverting buffering is determined during the synthesis process by examining the benefit of using the single stage inverting buffers in the context of fanin/fanout conditions of a node.

$$d_0 = D_{i,int}^{g_s} + D_{i,load}^{g_s}C_0 + D_{i,int}^{g_i} + D_{i,load}^{g_i}C_1 \quad (4.5)$$

Figure 4.12(a) shows a situation in which a dual-rail gate g_s drives a dual-rail inverter g_i , and g_i drives two sink nodes g_k^1 and g_k^2 . The delay from source node g_s to the sink nodes is the sum of delay g_s to g_i and the delay from g_i to the sink nodes as

²In this case, the problem is basically different from a buffer insertion problem.

shown in the equation (4.5), where d_0 is the delay from the source node to the sink nodes, $D_{i,int}^{g_s}$ is the load independent intrinsic delay of the source gate, $D_{i,load}^{g_s}$ is the drive resistance of the source gate, $D_{i,int}^{g_i}$ is the load independent intrinsic delay of g_i , $D_{i,load}^{g_i}$ is the drive resistance of g_i , and C_L^0 and C_L^1 are the load capacitance seen at the nodes g_s and g_i , respectively. We assumed that all the parameters for both polarities of the gate output are the same.

Algorithm 4.10.1: RBR($\mathcal{G}$)

```

1. for each  $n_i \in \mathcal{G}$ 
    do  $\left\{ \begin{array}{l} \text{if } n_i \text{ is not buffer or inverter } \text{continue} \\ F_I^{n_i} \leftarrow \text{fanin node's fanout nodes of } n_i \\ F_O \leftarrow \text{fanout nodes of } n_i \\ \Delta_d \leftarrow \text{Delay\_Difference}() \\ \text{if } \Delta_d > 0 \\ \quad \text{do } \left\{ \begin{array}{l} F_I^{n_i} \leftarrow 0 \\ F_I^{n_i} \leftarrow F_I^{n_i} \cup F_O \\ L_r \leftarrow L_r \cup n_i \end{array} \right. \end{array} \right.$ 
2.  $\mathcal{G}_i \leftarrow \{\mathcal{G} - L_r\}$ 
3. return ( $\mathcal{G}_i$ )

```

As shown in Figure 4.12(b), after removing the inverter, the source node's output signals x and $\bar{x}$ are connected to the input signal of the sink nodes $\bar{y}$ and y , respectively. The delay from the source node to the sink nodes d_1 is

$$d_1 = D_{i,int}^{g_s} + D_{i,load}^{g_s} C_1 \quad (4.6)$$

The delay difference before and after the removal of an inverter buffer, equation (4.7), need to be great than zero in order to get delay benefit after the removal of the inverter buffer.

$$\Delta_d = D_{i,load}^{g_s}(C_0 - C_1) + D_{i,int}^{g_i} + D_{i,load}^{g_i} C_1 \quad (4.7)$$

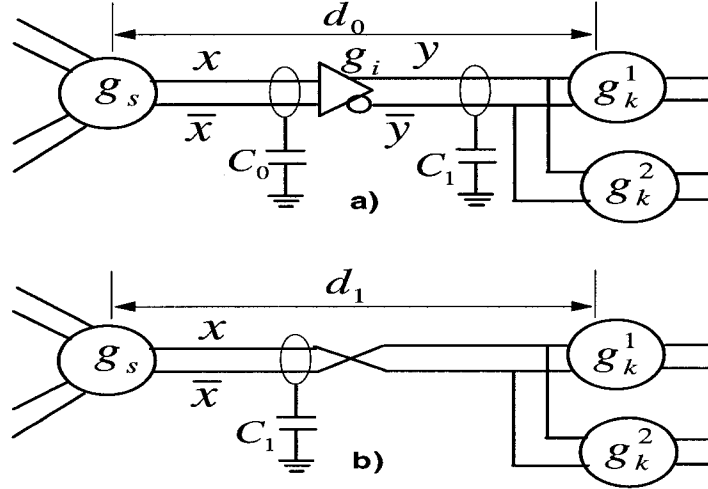


Figure 4.12: Delay reducing method. (a) before reduction. (b) after reduction.

The pseudo code of the redundant buffer removal process is shown in Algorithm 4.10.1. A given dual-rail mixed PTL circuit is converted to a graph $\mathcal{G}$; the algorithm traverses the graph. For all inverter buffer nodes during graph traversal, the delay difference Δ_d is calculated. If the delay difference Δ_d is greater than zero, the graph is modified so that the fanout nodes of node n_i are the fanout nodes of the fanin node of node n_i and the inverter node is inserted to the redundant buffer list L_r . After traversing all the nodes in the graph, the nodes in the list L_r are deleted from the graph.

4.10.3 Fanout optimization

The conventional two level tree fanout optimization method generates a buffer tree where rooted at the source node. This method inserts the buffer tree regardless of the critical path between the source node and the sink node. Because critical path isolation[65, 67, 71] can avoid the drawback of the two level tree fanout optimization method, we will use this method instead of using more sophisticated fanout algorithms such as LT-trees[72]. Although this method has been widely used in the past, its adaption in mixed PTL synthesis has not been reported so far.

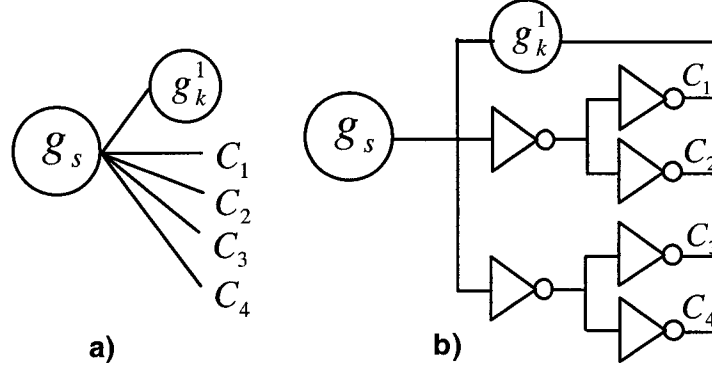


Figure 4.13: Critical path isolation fanout optimization method. (a) before optimization. (b) after optimization.

Figure 4.13(a) shows a situation in which a source node g_s drives several capacitive loads (sink nodes) including node g_k^1 . If a path that passes node g_k^1 is a critical path, this node should be excluded from the buffer tree and keep the connection to the source node g_s while sink nodes that are not in a critical path are distributed to the leaves of the tree. Algorithm 4.10.2 illustrates the fanout optimization process. The inputs of this algorithm are source node n_i , list of fanout nodes $Q_F^{n_i}$, and critical path data $\mathcal{P}_c$. The output of this algorithm is a fanout tree Q_F^t .

At the beginning of the process, the required number of sink nodes, N_{L2} , for each second level buffer is calculated such that the loads are balanced and the delay from the source node to all sinks is reduced based on the following equation.

$$D_{i,int}^{g_s} + D_{i,load}^{g_s} C_L - (A + B) > 0 \quad (4.8)$$

where A is the sum of the load independent delays in the paths from source to sinks and B is the maximum load dependent delay from the source to the sinks. Then, the fanout tree building is started by inserting the first level buffer into Q_F^t (step 4). Step 5 of the algorithm generates the fanout buffer tree by examining the fanout nodes of the source node n_i .

Algorithm 4.10.2: FANOUTOPT($n_i, Q_F^{n_i}, \mathcal{P}_c$)

```

1. Initialize  $Q_F^t$ 
2.  $N_{L2} \leftarrow \text{Calculate Sinks}$ 
3.  $j \leftarrow 0$ ;  $k \leftarrow 0$ 
4.  $Q_F^t \leftarrow I_{L1}^j$ 
5. for each  $n_q \in Q_F^{n_i}$ 
    { if  $n_q \in \mathcal{P}_c$  continue
      / * build fanout buffer tree * /
       $Q_F^{n_i} \leftarrow \{Q_F^{n_i} - n_q\}$ 
      if  $k > N_{L2}$ 
        do {  $k \leftarrow 0$ 
              do {  $Q_F^t \leftarrow I_{L1}^{j+1}$ 
                     $Q_F^t \leftarrow I_{L2}^k$ ;  $k \leftarrow k + 1$ 
                  }
              else
                do {  $Q_F^t \leftarrow I_{L2}^k$ ;  $k \leftarrow k + 1$ 
              }
        }
6. return ( $Q_F^t$ )

```

For each node n_q in the fanout list, if n_q is in the critical path $\mathcal{P}_c$, the node is not inserted into the fanout tree. A level-two buffer is inserted into the fanout buffer tree if n_q is not in the critical path. If the level-two buffer count exceeds a limit, N_{L2} , a new level is created in the tree Q_F^t . The fanout tree Q_F^t is completed after considering all the fanout nodes of the source node n_i . Figure 4.13(b) shows the generated fanout tree after fanout optimization using the algorithm 4.10.2. As shown in Figure 4.13(b), the critical path is isolated from the fanout tree in order to impose a minimum impact on the critical path by the fanout tree.

4.10.4 Experimental Results of Performance Enhancement

The characteristics of the benchmark circuits used for this evaluation are shown in Table 4.1. These circuits are small, but they are representative of both control-type

Table 4.1: Characteristics of benchmark circuits.

Circuit	# Trans.	# Inputs	# Outputs	Logic Depth
Ckt 1	524	32	26	11
Ckt 2	642	82	8	8
Ckt 3	64	18	2	7
Ckt 4	60	16	2	7
Ckt 5	894	60	36	12
Ckt 6	488	35	2	11
Ckt 7	278	25	4	14
Ckt 8	156	9	2	9
Ckt 9	246	16	25	12
Ckt 10	242	20	14	9

and data-path-type circuits used in microprocessor design. Three ISCAS85 benchmark circuits are also used in our experiment. All the PTL and static CMOS cells used in our synthesis were created using an experimental $0.1\mu m$ CMOS technology. The supply voltage in our experiments was set to $1.1V$.

Table 4.2: Experimental results of applying the redundant buffer removal method with a fanout optimization method.

Circuit	Synthesis results w/o proposed methods				Synthesis results w/ proposed methods			
	Area (μm)	Delay (ps)	Power (mW)	PDP	Area (μm)	Delay (ps)	Power (mW)	PDP
Ckt 1	39.4	93	0.014	1.31E-15	40.5	87	0.015	1.33E-15
Ckt 2	595.7	151	0.131	1.98E-14	415.4	140	0.077	1.08E-14
Ckt 3	2131.5	144	0.440	6.33E-14	1734.6	111	0.191	2.12E-14
Ckt 4	1002.1	129	0.319	4.12E-14	809.1	118	0.245	2.89E-14
Ckt 5	238.5	77	0.091	7.04E-15	126.9	74	0.066	4.85E-15
Ckt 6	209.0	90	0.076	6.88E-15	194.2	48	0.062	2.96E-15
Ckt 7	256.7	60	0.042	2.53E-15	208.8	51	0.021	1.08E-15
Ckt 8	67.3	83	0.017	1.44E-15	59.5	63	0.012	7.27E-16
Ckt 9	257.5	121	0.368	4.46E-14	266.4	90	0.053	4.81E-15
Ckt 10	723.5	192	0.258	4.95E-14	676.8	178	0.222	3.94E-14
Average	552.1	114.0	0.176	2.38E-14	453.2	96.0	0.096	1.16E-14

To see the impacts of the redundant buffer removal and fanout optimization process using PTL based cells, the results between two dual-rail mixed PTL circuits

before and after the fanout optimization and redundant buffer removal were compared in terms of delay, size, and power consumption. The results are shown in Table 4.2. The simulation results using ISCAS85 benchmark circuits are shown in Table 4.3. The delay and power measures were obtained using *PowrMill* and *PathMill*[73], respectively.

Table 4.3: Experimental results of applying the redundant buffer removal method with a fanout optimization method(ISCAS85 benchmark circuits.)

Circuit	Synthesis results w/o proposed methods				Synthesis results w/ proposed methods			
	Area (um)	Delay (ps)	Power (mW)	PDP	Area (um)	Delay (ps)	Power (mW)	PDP
C432	2710.3	477	1.093	5.21E-13	3204.5	420	1.640	6.89E-13
C499	4130.1	339	1.856	6.29E-13	3493.5	256	1.987	5.09E-13
C1355	6115.3	406	3.873	1.57E-12	3417.0	255	1.988	5.07E-13
Average	4318.6	407.3	2.274	9.08E-13	3371.7	310.3	1.872	5.68E-13

Table 4.2 shows that the dual-rail mixed PTL circuit where the fanout optimization and redundant buffer removal were applied outperforms the circuit that does not apply the methods described above. On average, the dual-rail mixed PTL circuits with fanout optimization and redundant buffer removal outperforms the results of the circuits without fanout optimization and redundant buffer removal by up to 16%, 18%, and 45% in delay, area, and power, respectively.

4.11 Miscellaneous

The proposed mixed PTL/Static logic synthesis method described in Chapters 3 and 4 deals only with combinational circuits. Since most of the modern VLSI designs include sequential parts as well combinational parts, it is necessary to include

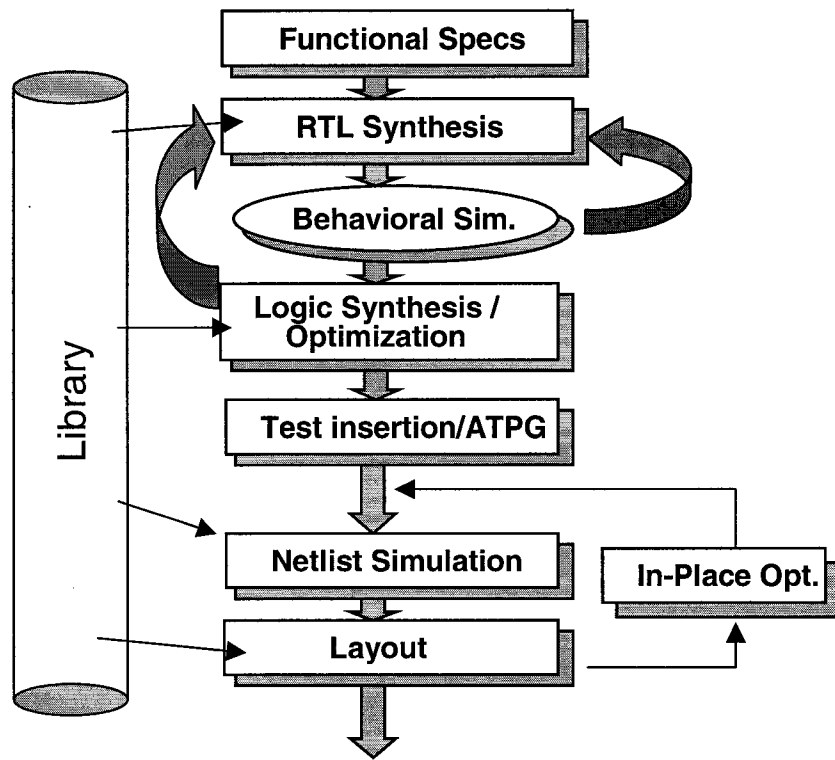


Figure 4.14: Synthesis based ASIC design flow.

a sequential circuit handling method to the proposed mixed PTL/static logic synthesis flow shown in the previous chapter. Therefore, this section describes how to incorporate the mix PTL synthesis into a commercial logic synthesis flow.

Figure 4.14 shows general ASIC design flow. The first step of the design flow is to determine functional specifications followed by RTL (Register Transfer Level) coding using Verilog[74] or VHDL[75] hardware description language (HDL). After the RTL coding, the next step involves RTL synthesis using RTL synthesis tools such as Synopsys design compiler[76] and behavioral simulation to check the functionality described in the functional specifications. If the behavioral simulation results meet the specification, the netlist from the RTL synthesis is optimized and mapped to a circuit using library cells. This step is called logic synthesis and optimization. On performing the functional simulation after the logic synthesis, the functionality may

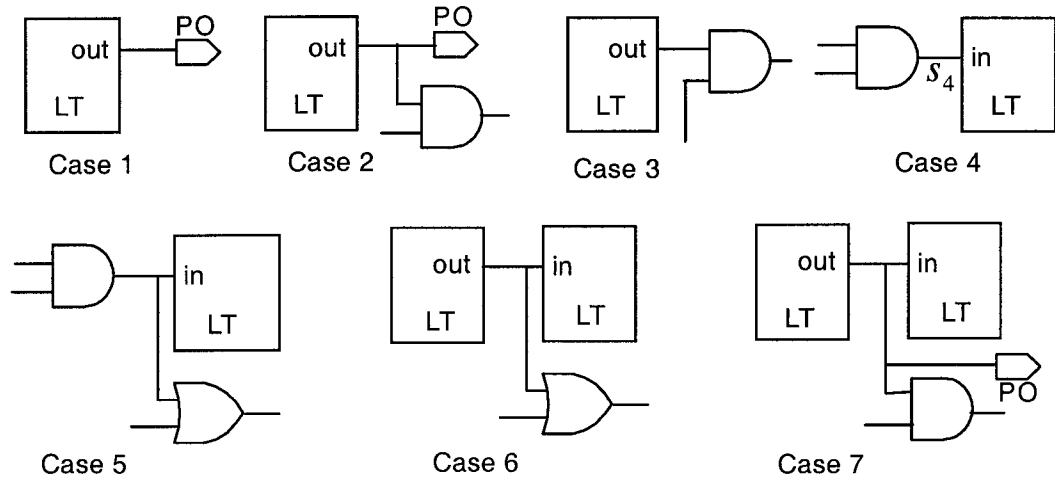


Figure 4.15: Possible signal connection patterns around latch. In each case, the signal connected to latch is the candidate of the primary input or output of the new combinational block.

not exactly match the desired functional behavior. In this case, we need to return to modify the RTL source code. Once the logic synthesis is done, the next step is to insert test circuitry and to generate test patterns for the fabricated chip. The netlist generated after the test insertion will be simulated before the physical layout as shown in Figure 4.14.

In order to incorporate the proposed mixed PTL/Static synthesis method into the ASIC design flow described above, the following steps need to be carried out:

1. Sequential Parts Separation
2. Convert combinational logic to equations and rearrangement of the block input/output ports
3. Mixed PTL logic synthesis
4. Combine the mixed PTL circuit with sequential parts

The details of the steps are as follows. First, after determining the boundary of the sequential and combinational parts, all the sequential gates are written to a file

to use later. Then, the separated combinational gates are transformed into the list of equations called SLIF (Stanford Logic Interchange Format).

However, there can be problems with signal connectivity when we simply separate combinational gates from sequential parts. This is because signals connected to input or output of a latch or flip-flop may not primary inputs nor primary outputs. Therefore, the primary input and output pins of the transformed combinational parts need to be rearranged before performing the mixed PTL synthesis. There are several possible signal connection patterns around one or more latches as shown in Figure 4.15. Therefore, the input and output pins of the transformed block need to be rearranged based on the signal connection patterns.

Table 4.4: Primary input and output relationship between original and sequential part removed block.

Case	Signal Property				Original Block I/O		New Block I/O	
	Latch	Combinational	PI	PO	PI	PO	pi	po
1	Out	-	-	Yes	-	Delete	-	-
2	Out	In	-	Yes	-	Delete	Insert	-
3	Out	In	-	-	-	-	Insert	-
4	In	Out	-	-	-	-	-	Insert
5	In	In/Out	-	-	-	-	-	Insert
6	In/Out	In	-	-	-	-	Insert	-
7	In/Out	In	-	Yes	-	Delete	Insert	-

Table 4.4 shows the signal property (connection pattern) and primary input/output relationship between original and sequential part removed block. The first column represent the possible cases in Figure 4.15, the second column shows the signal connection patterns for each case in Figure 4.15, and the third and fourth columns show the actions that need to be done for the newly transformed combinational block. For

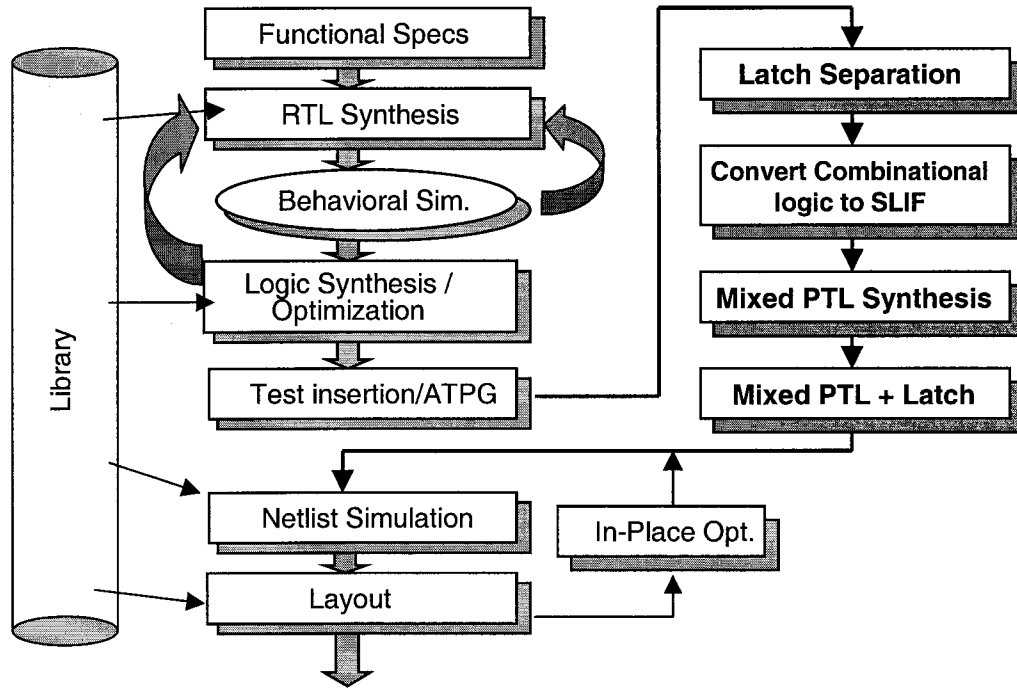


Figure 4.16: Proposed mixed PTL design flow that is incorporated into a general ASIC design flow.

example, when the signal, s_4 , is the input of a latch as well as the output of combinational gate in the original block (case 4), this signal needs to be inserted to a primary output of the combinational only block as shown in Table 4.4. However, the primary input/output list of the original block has not been changed because this signal was not primary input nor primary output of the original block.

Once the combinational gates are transformed into a set of logic equations (SLIF format) and the primary inputs and outputs are rearranged, the next step is to synthesis the set of equations, which represents all the combinational parts of the original block, using the proposed mixed PTL/Static synthesis method.

Finally, the separated sequential part need to be merged with the mixed PTL combinational parts to restore the original block after the mixed PTL circuit is acquired. At the same time, all the inputs and outputs that are temporarily inserted to primary input/output are removed from the primary input/output list, and the

original primary inputs and outputs are restored. Then, we can follow the rest of the ASIC design flow.

Figure 4.16 shows the proposed mixed PTL design flow that are incorporated into a modern ASIC design flow. As shown in Figure 4.16, the added flow starts from separating the netlist where sequential and combinational exist. After restoring sequential parts to the mixed PTL circuits, we can go through the standard ASIC design flow.

4.12 Conclusions

A new graph covering method and performance enhancing features were proposed in this chapter. The covering method was developed based on genetic algorithms that optimize the covering problem on a global level. The crossover and mutation were performed on graph structure chromosomes that are the mapped solutions of a given circuit. The proposed covering algorithm converged well. Also, a performance enhancing method through redundant buffer removal for dual-rail mixed PTL/Static logic was developed. It was also shown that the use of the redundant buffer removal method with a fanout optimization method significantly improves power and performance. Finally, we proposed a method that incorporates the mixed PTL synthesis method into a commercial ASIC design flow.

Chapter 5

Technology Scaling Impact on mixed PTL Circuits

5.1 Introduction

Power is an increasingly limiting factor in high-density and high-performance VLSI designs. As a result of that, a great deal of effort have been made to explore low-power design options without sacrificing performance.

Technology scaling[77, 78, 79], which is one of the approaches that can reduce power, has been a vehicle that makes it possible to achieve high density, high performance, and low-power VLSI designs. As physical dimension scales down, the delay and dynamic power of CMOS circuits are decreasing because of reduced gate capacitance and reduced supply voltage. However, the sub-threshold leakage exponentially increases as CMOS technology scales down due to scaling of the threshold voltage. This is further exacerbated by the increasing short-channel effect and DIBL (Drain Induced Barrier Lowering) effect. The increased sub-threshold leakage is an increasing concern and considered as one of the bottlenecks of future low-power VLSI systems[36].

Since the conventional PTL circuit styles[10, 11] can cause a large amount of power consumption and performance degradation because of excessive number of

buffers used, mixing PTL with static CMOS has been proposed[12, 13, 16] as an alternative low-power circuit style. This is referred to as a mixed PTL/Static CMOS circuit. Automated synthesis of an exclusive set of mixed PTL circuits[16, 80] shows significant advantages of mixed PTL circuits over static CMOS and dynamic circuits in terms of power and performance. However, there are concerns about the ability of PTL to scale well with technology scaling, particularly with V_{dd} scaling. The scalability of mixed PTL, including performance, dynamic power consumption, and leakage power consumption, is not well understood. The existing published results did not address these concerns, because almost all of the previous work on PTL was performed on the $0.25\mu m$ technology generation or older. With current process generation moving beyond the $0.25\mu m$ generation and into the $0.18\mu m$, $0.13\mu m$, and $0.1\mu m$ generations, this paper attempts to directly address the issues of PTL scalability by projecting the trends of performance, dynamic power consumption, and leakage power consumption on future CMOS technologies beyond $0.1\mu m$, and presenting new comparative results using more up-to-date $0.18\mu m$, $0.13\mu m$, and $0.1\mu m$ CMOS technologies. The supply voltage for each technology generation from $0.18\mu m$, $0.13\mu m$, to $0.1\mu m$ was also scaled to 1.5V, 1.3V, and 1.0V, respectively.

This comparative study also takes into account the different transistor sizing requirements for different circuit styles and their impact on final circuit performance and power consumption. In [81], PTL circuits were optimized using a commercial circuit optimizer to take advantages of each pass-transistor logic style. But the circuits used in the comparison were limited to simple XOR/XNOR gates in a $0.18\mu m$ technology.

We used a set of 10 circuits extracted from a modern 64-bit microprocessor design. The experimental results show that the mixed PTL circuits scale well down to the $0.1\mu m$ technology node compared to their dynamic and conventional static coun-

terparts. Theoretical projections for performance and power consumption provides further evidence of scalability of mixed PTL circuits.

The remaining part of this chapter consists of six sections. Section 5.2 reviews modern scaling trends and general scaling theory. To project delay and power consumption trends of different circuit styles, delay and power models for the different circuit styles will be described in Sections 5.3 and 5.4, respectively. In Section 5.5, the projected trends of delay and power consumption will be presented. Concluding remarks for this chapter are given in Section 5.6.

5.2 Technology Scaling

5.2.1 ITRS Road Map

For over 35 years, IC industry has been following Moore's Law[82, 83] in rapidly and constantly improving VLSI technology by scaling physical dimensions of transistors as well as interconnects. The physical dimension reduction factor has been around $1/\sqrt{2}$ for every two to three years. In order to help the IC industry in continuing to follow Moore's Law, ITRS (International Technology Roadmap for Semiconductors) was published to project the progression of the IC technology generations over next 15 years. Table 5.1 shows the ITRS roadmap 2001[36] for some parameters that are most relevant to discussions in this paper.

5.2.2 Technology Scaling Theory

This section briefly reviews the general technology scaling theory. There are three types of technology scaling: full scaling, fixed voltage scaling, and general scaling[84,

Table 5.1: Key Overall Chip parameters for High-performance Logic(data from ITRS 2001 road map).

		Near Term							Long Term		
Calendar Year		2001	2002	2003	2004	2005	2006	2007	2010	2013	2016
Physical Gate Length (L_{eff})	nm	65	53	45	37	32	28	25	18	13	9
Power Supply Voltage,VDD	V	1.2	1.1	1.0	1.0	0.9	0.9	0.7	0.6	0.5	0.4
Tox Electrical Equivalent (T_{ox}^{eff})	nm	2.25	2.12	2.01	1.95	1.87	1.87	1.35	1.15	1.03	0.93
Sub-Threshold Current (NMOS)	$\mu A/\mu m$	1E-2	3E-2	7E-2	1E-2	3E-1	7E-1	1E0	3E0	7E0	1E1
Effective Transconductance g_m	$\mu S/\mu m$	1000	1055	1107	1137	1181	1181	1584	2378	3434	4430
Maximum On-chip Clock Frequency	GHz	1.7	2.3	3.1	4.0	5.2	5.6	6.7	11.5	19.4	28.8
Allowable Max. Power	Watts	130	140	150	160	170	180	190	218	215	288
Number of Trs per chip	$\times 10^6$ Trs	276	348	439	553	697	878	1106	2212	4424	8848

Table 5.2: Scaling relationship for devices.

Parameters	Relation	Full Scaling	General Scaling	Fixed Voltage Scaling
W, L, T_{ox}	-	$1/\alpha$	$1/\alpha$	$1/\alpha$
V_{DD}, V_t	-	$1/\alpha$	$1/\beta$	1
N_{SUB}	V/W_{depl}^2	α	α^2/β	α^2
Area/Device	WL	$1/\alpha^2$	$1/\alpha^2$	$1/\alpha^2$
C_{ox}	$1/t_{ox}$	α	α	α
C_L	$C_{ox}WL$	$1/\alpha$	$1/\alpha$	$1/\alpha$
k_p, k_n	$C_{ox}W/L$	α	α	α
I_{av}	$k_{n,p}V^2$	$1/\alpha$	α/β^2	α
$t_p(\text{intrinsic})$	$C_L V/I_{av}$	$1/\alpha$	β/α^2	$1/\alpha^2$
P_{av}	$C_L V^2/t_p$	$1/\alpha^2$	α/β^3	α
PDP	$C_L V^2$	$1/\alpha^3$	$1/\alpha\beta^2$	$1/\alpha$

85, 77, 86]. First, the full scaling, which is also called constant electric field scaling, is an ideal model. In this scaling method, physical dimensions and supply voltage are scaled together by the same scaling factor α as shown in Table 5.2 (adapted from [77]). The purpose of this scaling is to keep the electrical field patterns in a scaled device identical to those in the previous technology generations to prevent the secondary effects such as breakdown. In this method, the substrate doping density need to be increased by a factor α in order to maintain same electrical patterns in the device. Using this scaling method, the delay and power consumption can be reduced by factors $1/\alpha$ and $1/\alpha^2$, respectively, as shown in Table 5.2.

The fixed voltage scaling method was most commonly used scaling method until early 90's. This scaling method only scales physical dimensions by a factor $1/\alpha$ while maintaining the supply and threshold voltages identical to the previous technology generation. This scaling method is adequate for the situation where the designer cannot change the supply voltage arbitrarily because of the compatibility to the older version of designs. The scaling method can speed up the device by a factor $1/\alpha^2$. However, the system power consumption is increased by a factor α . Therefore, this method is not appropriate when the power reduction is required in the scaled system.

Finally, the general scaling scales the device dimension and voltage using different scaling factors, $1/\alpha$ and $1/\beta$, respectively, as shown in Table 5.2. This scaling method is most realistic method for today's situation because there may be different restrictions for scaling device dimension and voltage. The scaling methods used in ITRS roadmap shown in Table 5.1 falls into this scaling category.

5.3 Delay Models for Different Circuit Styles

The drain-source current of MOS transistor in square-law model[28, 87] is given by

$$I_{DS} = \begin{cases} 0 & , V_{GS} \leq V_{TH} \\ K\{(V_{GS} - V_{TH})V_{DS} - 0.5V_{DS}^2\} & , V_{DS} \leq V_{GS} - V_{TH} \\ K(V_{GS} - V_{TH})^2/2 & , V_{DS} \geq V_{GS} - V_{TH} \end{cases} \quad (5.1)$$

where V_{TH} is threshold voltage, V_{GS} is gate-source voltage, and V_{DS} is drain-source voltage. K is a drivability factor and equals $\mu(\epsilon_{ox}/t_{ox})(W/L_{eff})$, where μ is an effective mobility, ϵ_{ox} is dielectric constant of a gate oxide, t_{ox} is a gate oxide thickness, W is a device width, and L_{eff} is the effective channel length. The $V_{DS} - I_D$ characteristic of this model is fairly accurate where the channel length is long. However, this model is not accurate enough in DSM (deep-submicron) devices.

For DSM CMOS devices, the α -power law model has been proposed[35]. The α -power law is given by

$$I_{DS} = \begin{cases} 0 & , V_{GS} \leq V_{TH} \\ (I'_{D0}/V'_{D0})V_{DS} & , V_{DS} \leq V_{GS} - V_{TH} \\ I'_{D0} & , V_{DS} \geq V_{GS} - V_{TH} \end{cases} \quad (5.2)$$

In equation (5.2), I'_{D0} and V'_{D0} are given by

$$I'_{D0} = \frac{WP_c}{L}(V_{GS} - V_{TH})^\alpha \quad (5.3)$$

$$V'_{D0} = P_v(V_{GS} - V_{TH})^{\alpha/2} \quad (5.4)$$

where α is dependent on the degree of velocity saturation, which varies from 1 to 2. The exact value of α is derived empirically. P_c and P_v are the technology parameters

to be determined by experimentally. This model is fairly accurate in DSM MOS devices. Therefore, in this dissertation, we used α -power law model to model drain-source current for a MOS transistor.

5.3.1 Static and Dynamic Logic Delay

For an inverter, when the input goes from low to high and initially the output is at V_{DD} , the nMOS of the inverter is in saturation region. The fall time delay of an inverter can be computed from the following differential equation[28, 87]

$$C_L \frac{dV_{out}}{dt} = -I_{DSn} \quad (5.5)$$

where C_L is load capacitance, $I_{DSn} = K_n(V_{GSn} - V_{Tn})^\alpha$ is the drain-source current through the nMOS transistor, and V_{out} is the voltage across the load capacitance. Assuming a step function at the input, the propagation delay of output falling is the output transition time from V_{DD} to $V_{DD}/2$. Integrating the Equation (5.5) from $t = t_1$ ($V_{out} = V_{DD}$) to $t = t_2$ ($V_{out} = V_{DD}/2$), the delay time of high-to-low transition of an inverter is given by

$$t_{HL} = \frac{C_L V_{DD}}{2K_n(V_{GSn} - V_{Tn})^\alpha} \quad (5.6)$$

where K_n is a technology parameter[35]. Similarly, high-to-low delay time of an inverter, t_{LH} , is given by

$$t_{LH} = \frac{C_L V_{DD}}{2K_p(V_{SGp} - |V_{Tp}|)^\alpha} \quad (5.7)$$

The transconductance of an nMOS device is defined by

$$\begin{aligned} g_m^n &= \frac{\partial I_{DSn}}{\partial V_{GSn}} \\ &= \alpha K_n (V_{GSn} - V_{Tn})^{\alpha-1} \end{aligned} \quad (5.8)$$

Thus, the drain current of a nMOS device in saturation mode can be represented by

$$\begin{aligned} I_{DSn} &= \alpha K_n (V_{GSn} - V_{Tn})^{\alpha-1} \frac{K_n (V_{GSn} - V_{Tn})^\alpha}{\alpha K_n (V_{GSn} - V_{Tn})^{\alpha-1}} \\ &= g_m^n \alpha^{-1} (V_{GSn} - V_{Tn}) \end{aligned} \quad (5.9)$$

For finite rise and fall time at the input, an additive modifying factor, δ , can be added to the delay models in Equations (5.6) and (5.7). The modifying factor δ is given by[35, 88]

$$\delta = \left(\frac{1}{2} - \frac{1 - V_T/V_{DD}}{1 + \alpha} \right) t_T \quad (5.10)$$

where α is the velocity saturation index which is the same as that in Equation (5.6), V_T is the threshold voltage of a nMOS or pMOS transistor, and t_T is the rise or fall time of the input waveform. This input slope dependent delay (δ) increases the delay in Equations (5.6) and (5.7). The question is whether the load dependent delay t_{HL} (or t_{LH}) dominates the input slope dependent delay δ . Using the parameters defined in ITRS[36] for $0.1\mu m$ CMOS technology, the ratio of V_T to V_{DD} is 0.14 and the transconductance (g_m) is $810\mu S/\mu m$. Assuming that $C_L = 50fF$, $\alpha = 1.2$ as in[35], and $t_T = 50ps$, the input slope dependent delay δ will be $0.11t_T(5.5ps)$. On the other hand, the load dependent delay $t_{HL}(ps)$ is $43.1/W_n$ where the unit for W_n is μm . For internal gates, it is reasonable to expect that the typical W_n of an inverter will be in the range of $0.2\mu m \sim 1.0\mu m$, using the Equation (5.6), the resulting load dependent fall-time delay is in the range of $215.5ps \sim 43.1ps$. It is clear that the load dependent delay dominates the input slope dependent delay. The similar conclusion can be reached for the rise-time delay. In addition, it is known[35, 88] that, for a fixed V_T/V_{DD} , the delay t_{HL} (or t_{LH}) becomes less sensitive to input slope when the carrier velocity saturation effect gets severe and α becomes smaller due to the fact that the factor δ decreases monotonically as α decreases. Acceptable value of V_T/V_{DD} is between $1/4$ and $1/5$ [42] in modern CMOS process and it is projected to be

even smaller for future CMOS technologies[36]. This implies that the load dependent delay becomes more dominant than the input slope dependent delay as V_T/V_{DD} and α become smaller. Based on the above analysis, the remaining discussions on trend of circuit delays will be focused on load dependent delay only.

The overall delay (or propagation delay time) of an inverter is given by

$$t_{pINV} = \frac{1}{2}(t_{LH} + t_{LH}) \quad (5.11)$$

Therefore, the propagation delay time of an inverter is given by

$$t_{pINV} = \frac{C_L V_{DD}}{2} \left(\frac{0.5}{g_m^n \alpha^{-1} (V_{GSn} - V_{Tn})} + \frac{0.5}{g_m^p \alpha^{-1} (V_{SGp} - |V_{Tp}|)} \right) \quad (5.12)$$

For n-input NAND gate, assuming that n identical nMOS are connected in series and all the transistors are in saturation node, the high-to-low and high-to-low propagation delay time can be approximated by the following equations[89].

$$t_{HL}^{nand} = \frac{n C_L V_{DD}}{2 g_m^n \alpha^{-1} (V_{GSn} - V_{Tn})} \quad (5.13)$$

$$t_{LH}^{nand} = \frac{C_L V_{DD}}{2 g_m^p \alpha^{-1} (V_{SGp} - |V_{Tp}|)} \quad (5.14)$$

Therefore, the delay time of a NAND gate is given by

$$t_{pNAND} = \frac{C_L V_{DD}}{2} \left(\frac{0.5n}{g_m^n \alpha^{-1} (V_{GSn} - V_{Tn})} + \frac{0.5}{g_m^p \alpha^{-1} (V_{SGp} - |V_{Tp}|)} \right) \quad (5.15)$$

Similarly, in case of a NOR gate, assuming that n identical pMOS are connected in series and there in saturation node when the load capacitance is charged, the propagation delay time of a n-input NOR gate is given by

$$t_{pNOR} = \frac{C_L V_{DD}}{2} \left(\frac{0.5}{g_m^n \alpha^{-1} (V_{GSn} - V_{Tn})} + \frac{0.5n}{g_m^p \alpha^{-1} (V_{SGp} - |V_{Tp}|)} \right) \quad (5.16)$$

For a domino logic gate [90, 28, 77, 87], there are three delay components: pre-charge, evaluation delay, and the delay through the inverter at the end of the gate.

For example, shown in Figure 5.2 (b), the worst case propagation delay of a domino gate is given by [91]

$$\begin{aligned} t_{pDyn} &= t_{eval} + t_{pINV} \\ &= \frac{C_y V_{DD}}{2} \frac{n}{g_m^n \alpha^{-1} (V_{GSn} - V_{Tn})} + t_{pINV} \end{aligned} \quad (5.17)$$

where t_{eval} is the evaluation delay, C_y is the capacitance at node y in Figure 5.2 (b). t_{pINV} denotes the delay caused by the inverter in Figure 5.2 (b), and n in Equation (5.17) represents the number of transistors connected in series in the nMOS tree.

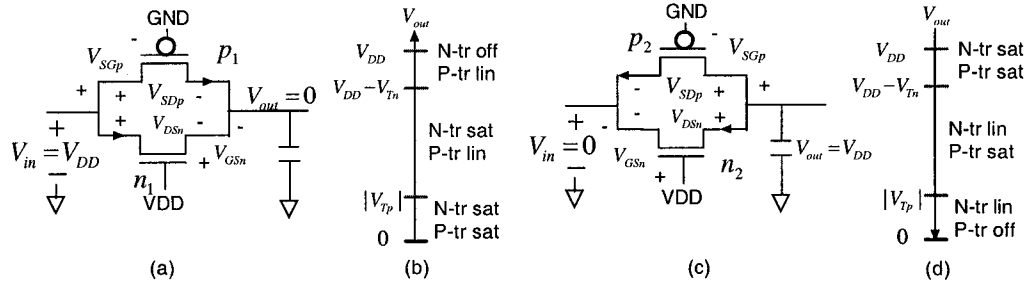


Figure 5.1: Pass-transistor logic operational modes. (a) and (c) are voltage and current relationship for logic 1 and logic 0 transfer, respectively. (b) and (d) are the operating regions for logic 1 and logic 0 transfer, respectively.

5.3.2 Delay Models for Pass-Transistor Logic Gates

In order to derive delay models of PTL+ cells, we need to examine the operational modes of each transistors in PTL+ cell. Figures 5.1 (a) and (c) show the voltage and current relationship of a part of a PTL+ cell. When logic 1 is transferred from input to output as shown in Figure 5.1 (a), the nMOS transistor, n_1 , is in saturation mode when $V_{out} \leq V_{DD} - V_{tn}$ and n_1 is in cutoff region when $V_{out} > V_{DD} - V_{tn}$. The pMOS transistor, p_1 , is in saturation mode when $V_{out} \leq |V_{tp}|$ and it is in non-saturation mode when $V_{out} > |V_{tp}|$. Figure 5.1 (b) illustrates the operating regions of the n-channel and the p-channel transistors for transferring a logic 1. Since V_{DD} is several

times higher than the device threshold voltage in sub-180nm technology nodes[36], for most of the switching time, nMOS and pMOS transistors are in saturation and non-saturation mode, respectively. Therefore, the low-to-high delay of Figure 5.1 (a) can be approximated as

$$\begin{aligned} t_{LH}^{PTL+} &= \frac{C_L V_{DD}}{2(I_{DSn}^{sat} + I_{SDp}^{lin})} \\ &= \frac{C_L V_{DD}}{2(g_m^{ns} \alpha^{-1}(V_{GSn} - V_{Tn}) + g_m^{pl} 2\alpha^{-1}(V_{SGp} - |V_{Tp}|))} \end{aligned} \quad (5.18)$$

where g_m^{ns} (g_m^{pl}) is the transconductance of nMOS n_1 (pMOS p_1) in saturation (non-saturation) mode, and I_{DSn}^{sat} and I_{SDp}^{lin} are the saturation current of n_1 and non-saturation current of p_1 , respectively.

When logic 0 is transferred from input to output, the nMOS transistor, n_2 , is in non-saturation mode when $V_{out} \leq V_{DD} - V_{tn}$ and n_2 is in saturation region when $V_{out} > V_{DD} - V_{tn}$. The pMOS transistor, p_2 , is in saturation mode when $V_{out} > |V_{tp}|$ and it is in cutoff mode when $V_{out} \leq |V_{tp}|$. Figure 5.1 (d) illustrates the operating regions of the n-channel and the p-channel transistors for transferring a logic 0. The high-to-low transition delay of Figure 5.1 (c) can be approximated as

$$\begin{aligned} t_{HL}^{PTL+} &= \frac{C_L V_{DD}}{2(I_{DSn}^{lin} + I_{SDp}^{sat})} \\ &= \frac{C_L V_{DD}}{2(g_m^{nl} 2\alpha^{-1}(V_{GSn} - V_{Tn}) + g_m^{ps} \alpha^{-1}(V_{SGp} - |V_{Tp}|))} \end{aligned} \quad (5.19)$$

where g_m^{nl} and g_m^{ps} are the linear mode and saturation mode transconductance of nMOS n_2 and pMOS p_2 , respectively. Since the PTL+ type1 cell is composed by connecting the circuits shown in Figure 5.1 (a) or (c) in parallel, the delay of the PTL+ type1 cell is proportional to the delay of the circuit in Figure 5.1 (a) or (c). Therefore, the overall propagation delay of PTL+ type 1 cell, shown in Figure 2.8 is given by

$$t_{pPTL+} = (t_{LH}^{PTL+} + t_{HL}^{PTL+})/2 \quad (5.20)$$

The worst case delay of PTL+ type2 and type3 is the sum of the delay of two sections of the PTL+ type1 in series while the load capacitances for each section are different.

5.4 Power Consumption Model

In CMOS logic, power consumption of a circuit is given by

$$P_{tot} = C_L V_{DD} V_{swing} f A_i + I_{leak} V_{DD} + I_{tunnel} V_{DD} \quad (5.21)$$

where V_{swing} is the voltage swing at an output node, f is the frequency of the input signal, A_i is activity factor, and I_{leak} and I_{tunnel} are total sub-threshold current and gate tunneling current of the circuit. The first term in Equation (5.21) represents dynamic power consumption of CMOS circuits. The second and third terms are the power consumption caused by sub-threshold leakage and gate tunneling currents. The following two subsections describe these currents in detail.

5.4.1 Leakage Current Model

5.4.1.1 Sub-threshold Current

The sub-threshold leakage current of a MOSFET is given by[92]

$$I_{sub} = A e^{\frac{q}{3kT}(V_{GS}-V_{SB}-V_{th0}-\gamma V_{SB}+\eta V_{DS})} (1 - e^{\frac{-qV_{DS}}{kT}}) \quad (5.22)$$

where V_{GS} , V_{SB} , and V_{DS} are the gate-source, source-substrate, and drain-source voltages, respectively, γ , S , and η represent linearized body-effect coefficient, sub-threshold swing, and DIBL coefficient that is representing the effect of V_{DS} on threshold voltage, respectively. The coefficient A is given by

$$A = \mu_0 C_{ox} \frac{W_{eff}}{L_{eff}} (kT/q)^2 e^{1.8} \quad (5.23)$$

where μ_0 and C_{ox} are the zero-biased carrier mobility and the gate-oxide capacitance per unit area, respectively. When transistors are connected in parallel and all the transistors are in off-state, the V_{DS} each transistor are the same. Therefore, the

leakage current is the sum of the currents that flow through each transistor. If the transistors are connected in series and all of them are turned off the V_{DS} of all transistors are not the same because the source voltages between transistors are different, while the currents through each transistor are the same. In this situation, assuming that the source voltage of the first transistor (the top transistor in the stack) V_{S1} is much smaller than V_{DD} ($V_{S1} \ll V_{DD}$), V_{DS2} can be represented in terms of V_{DD}

$$V_{DS2} = \frac{SkT}{q(1+2\eta+\gamma)} \ln\left(1 + \frac{A_1}{A_2}(1 - e^{\frac{-q}{kT}V_{DD}})\right) \quad (5.24)$$

Similarly, the drain-source voltage of the i^{th} transistor, The drain-source voltage of a transistor in a transistor stack, V_{DSi} , in Equation (5.22) can be calculated using the following equation[93]

$$V_{DSi} = \frac{SkT}{q(1+\gamma)} \ln\left(1 + \frac{A_{i-1}}{A_i}(1 - e^{\frac{-q}{kT}V_{DSi-1}})\right) \quad (5.25)$$

This equation can be iteratively used to find V_{DS} of each transistor.

The leakage current model in Equation (5.22) is reasonably accurate and considers all the possible sources of leakage currents. In order to use this equation, the drain-source voltage of each transistor needs to be determined. For a large chip, this process can be very expensive. To overcome this limitation, the following empirical model for leakage current has been proposed in [94]

$$I_{leak} = I_{off} \frac{W_{tot}}{S_f} C_t \quad (5.26)$$

where I_{off} is the leakage current of a transistor measured from an actual silicon, W_{tot} is the sum of all nMOS and pMOS transistor areas in a circuit, and S_f and C_t represent stacking factor and temperature factor. The value S_f is calculated based on the fact that the leakage current in transistor stacks is less than a single transistor device because of self-reverse bias of V_{GS} of a transistor in the stack. The coefficient C_t

is used to scale I_{off} for an appropriate junction temperature of the devices. Assuming that it is fixed, the leakage current is highly depends on W_{tot}/S_f that is given by

$$\frac{W_{tot}}{S_f} = \sum_{i \in cell} \frac{W_i}{S_i} \quad (5.27)$$

where W_i and S_i are the total width and the stacking factor of cell i . To get the leakage current of different circuit styles, we need to calculate W_i/S_i first for each cell i in the circuits, then all the W_i/S_i are summed to get W_{tot}/S_f . This method can be easily applied when a circuit is designed with ASIC style. When a circuit is not designed with ASIC style, all different sub-circuit styles whose stacking factors are the same need to be classified into different groups, then we can calculate W_{tot}/S_f by summing the ratio of total area to stacking factor of all different groups.

5.4.1.2 Tunneling Current

Due to high electric field at gate oxide, direct tunneling current from gate material to the substrate increases exponentially as the gate oxide thickness decrease. As technology scaling continues this current cannot be ignored any more. The tunneling current can be modeled by [95]

$$J_n = A(V_{ox}/t_{ox})^2 e^{\frac{-B(1-(1-V_{ox}/\Phi_B)^{3/2})}{V_{ox}/t_{ox}}} \quad (5.28)$$

where A and B are physical parameters. Since V_{ox} depends on gate-source voltage V_{gs} and the surface potential ψ_s ($V_{ox} = V_{gs} - V_{fb} - \psi_s$), it is necessary to calculate ψ_s to calculate tunneling current using Equation (5.28). However, ψ_s can only be calculated accurately using an iterative numerical method which is expensive[96]. Ignoring some second-order effects, a simpler model can be used[36] here as:

$$I_{ox} = 16.2L_{eff}e^{(3.45V_{DD}-10.6t_{ox})}[\mu A/\mu m] \quad (5.29)$$

The tunneling current of a circuit was calculated in this work by multiplying total transistor width of the circuit with I_{ox} .

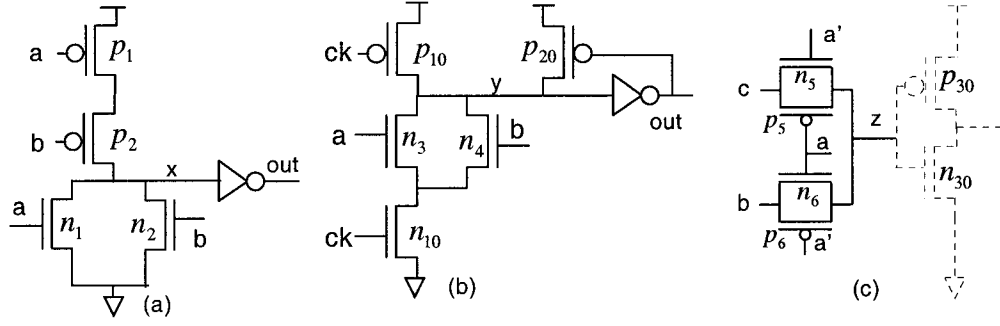


Figure 5.2: Static, Dynamic, and PTL logic implementation of a OR gate. (a) , (b), and (c) represent static, dynamic, and PTL, respectively. For PTL, it is assumed that the input b is connected to V_{dd} .

5.4.2 Stacking Effects of Different Logic Styles

In order to determine the leakage current of a given gate, the stacking factor needs to be determined. Similar to the approach used in [94], the stacking factor of a gate was calculated based on the amount of current through turned-off transistor stacks compared to the current through a single turned-off transistor using Equation (5.30).

$$S_f = I_0 / \left(\frac{1}{N} \sum_i I_i^{leak} \right) \quad (5.30)$$

where I_0 represents the amount of current through a single turned-off transistor, N represents the number of possible input combinations for the given stack, i and I_i^{leak} represent a input combination and the corresponding sub-threshold leakage current, respectively. I_0 and I_i^{leak} were determined based on a SPICE simulation. Based on a $0.1\mu m$ CMOS process, our experimental simulation results show that the leakage currents in a stack with 2, 3, and 4 turned-off transistors are $1/6$, $1/7$, and $1/7$ of that in a single transistor stack. Figures 5.2 (a) and (b) show an OR gate implementations using static and dynamic logic styles, respectively. We will use the OR gate implementations in Figure 5.2 to illustrate how stacking factor for each type logic gate is calculated.

Table 5.3: The leakage current flow of static and dynamic gates in Figures 5.2 (a) and (b), respectively, with different input combinations.

Inputs		Static gate leakage	Dynamic gate leakage		
a	b		pre.	eval.	total
Gnd	Gnd	$2I_0$	$\frac{I_0}{6}(2I_0)$	$2I_0(2I_0)$	$\frac{13I_0}{6}(4I_0)$
Gnd	Vdd	I_0	$I_0(\text{na})$	$2I_0(2I_0)$	$3I_0(2I_0)$
Vdd	Gnd	I_0	$I_0(\text{na})$	$2I_0(2I_0)$	$3I_0(2I_0)$
Vdd	Vdd	$I_0/6$	$I_0(\text{na})$	$2I_0(2I_0)$	$3I_0(2I_0)$
Average		$1.04I_0$	-	-	$1.4I_0(2I_0)$

Table 5.3 shows the input combinations and corresponding leakage current for the static and dynamic gates. Let's consider the static gate in Figure 5.2(a) first, when inputs (a,b)=(11), the leakage current flows from V_{dd} to node x through pMOS p_1 and p_2 . In this case, the current is six times less than that of a single device. When inputs (a,b)=(00), the leakage flows through both nMOS n_1 and n_2 in parallel, and the amount of current is twice of a single turned-off device current. When inputs (a,b)=(01) or (a,b)=(10), the leakage current flows through pMOS p_2 or p_1 , but there is no leakage through nMOS because node x is grounded, resulting in the same current as a single turned-off device current for both cases. For all these inputs combinations, the stacking factor of the gate is 0.96 using Equation (5.30). The effective stacking-area of the static gate in Figure 5.2 (a) is

$$(W_{tot}/S_f)_{static} = W_{nand}/S_{nor} + W_{inv}/S_{inv} \quad (5.31)$$

where $S_{nor} = 0.96$ and $S_{inv} = 1$.

For dynamic logic in Figure 5.2 (b) with nMOS n_{10} , during the evaluation phase, when the inputs (a,b)=(11), the leakage flows through pMOS p_{10} and p_{20} in parallel, when the inputs (a,b)=(00), the leakage flows through nMOS n_3 and n_4 in parallel; when the inputs (a,b)=(01) or (10), the leakage flows through pMOS p_{10} and p_{20}

in parallel. In the pre-charge period, when the inputs (a,b)=(11), the leakage flows only through nMOS n_{10} ; when the inputs (a,b)=(00), the leakage flows through all nMOS transistors, n_3 , n_4 , and n_{10} ; when the inputs (a,b)=(01) or (10), the leakage flows through nMOS n_{10} . By applying the similar method used for the static case, the total amount of leakage current can be calculated, and the resulting stacking factor for this gate is 0.72. When speed is critical in a dynamic gate, transistor n_{10} in Figure 5.2 (b) is often removed. The values in the parentheses in Table 5.3 represent the leakage current for different input combinations without n_{10} . In this case, the stacking factor is reduced to 0.5. The effective stacking-area of the dynamic gate in Figure 5.2 (b) can be found by similar way as in static case.

Table 5.4: The leakage current flow of a PTL cell in Figure 5.2 (c) with different input and output combinations.

Inputs and Output				Current Flow	Leakage
a	b	c	z		
Vdd	Gnd	Vdd	Gnd	$c \rightarrow z$	$1.56I_0$
Vdd	Gnd	Gnd	Gnd	No leakage	0
Gnd	Gnd	Vdd	Vdd	$z \rightarrow b$	$1.56I_0$
Gnd	Gnd	Gnd	Gnd	No leakage	0
Vdd	Vdd	Vdd	Vdd	No leakage	0
Vdd	Vdd	Gnd	Vdd	$z \rightarrow c$	$1.56I_0$
Gnd	Vdd	Vdd	Vdd	No leakage	0
Gnd	Vdd	Gnd	Gnd	$b \rightarrow z$	$1.56I_0$
Average				-	$0.79I_0$

For the PTL gate as shown in Figure 5.2 (c), there is no leakage path through the PTL gate for a certain input/output combinations because the potentials at both input and output are the same. Table 5.4 shows the possible input and output combinations and corresponding leakage current flow. Since there are some input combinations that do not result in any leakage current, the stacking factor of a PTL

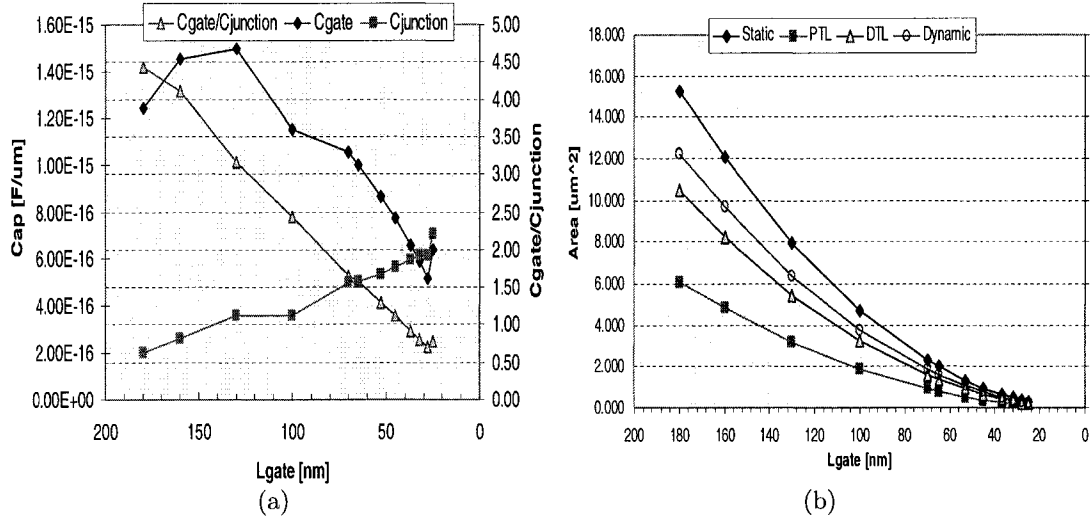


Figure 5.3: Gate capacitance, junction capacitance, and area projection results.

gate is higher than that of static as well as dynamic gates. By applying the same method used for the static and dynamic gates, the stacking factor is 1.27.

5.5 Projection Results

To understand the trend of delay, dynamic power and leakage power consumption on future technologies, we scaled the benchmark circuits to future CMOS technologies using the parameters in the ITRS roadmap shown in Table 5.1. The benchmark circuits used for the projection were implemented in two different groups of four different circuits, which are pure static CMOS, dynamic CMOS, and single-rail and dual-rail mixed PTL circuits. The two different groups are implementing two logic functions, $f_1 = a'b + cd$ and $f_2 = y(a(bc + b'd') + a'e)$, respectively. The sizes of transistors for each different logic style are determined so that the rising and falling time at the each node of each gate was in the range of 50ps to 100ps in $0.1\mu m$ process with a fanout of five at the output node.

Figures 5.3 (a) and (b) show the projection results of gate and junction capacitance[28] of a transistor per micron and the areas of different circuit styles, respectively. In

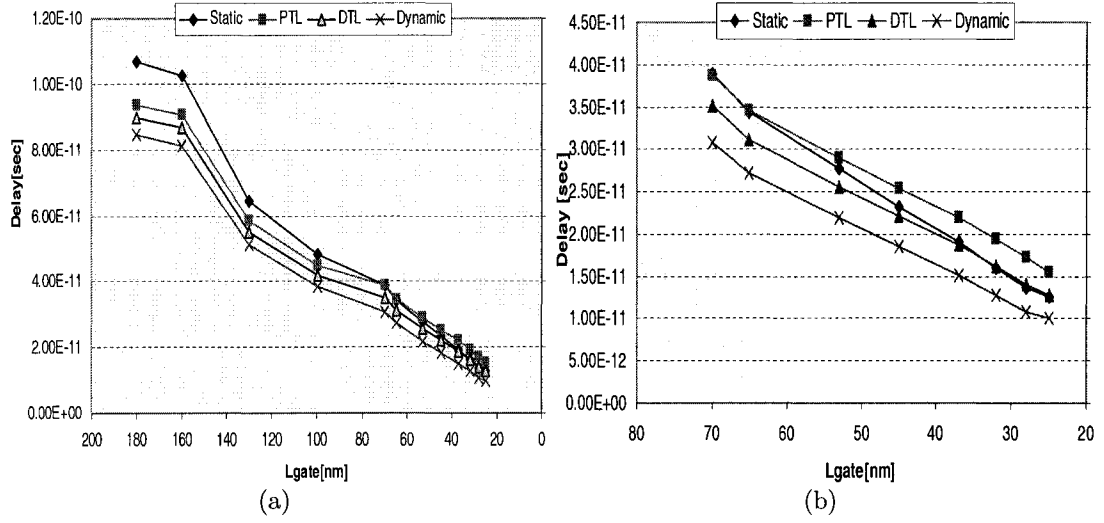


Figure 5.4: Delay projection results.

these figures, PTL and DTL represent single-rail and dual-rail mixed PTL/Static circuits, respectively. The general trend is that the gate capacitance per micron is decreasing as technology scales because the decreasing rate of effective channel length is larger than that of gate oxide. On the other hand, the junction capacitance is increasing as technology scales because of increasing substrate doping density. As shown in Figure 5.3 (a), the ratio of gate capacitance to junction capacitance, C_g/C_j , is also decreasing because both two capacitance values are moving in opposite directions. The junction capacitance is greater than the gate capacitance below 37nm technology nodes. For the areas of different circuit styles shown in Figure 5.3 (b), single-rail and dual-rail mixed PTL are smaller than those of static and dynamic circuit styles. The area of static circuit is larger than that of dynamic because p-tree of dynamic circuit consists of only pre-charge and keeper transistors.

The projection results on the delay are shown in Figures 5.4 (a) and (b). Figure 5.4 (b) magnifies Figure 5.4 (a) from 70nm to 25nm technology nodes. The delay were averaged the critical path delays for each benchmark circuit. As shown in Figure 5.4, the dynamic circuit is the fastest circuit style, and single-rail and dual-rail mixed PTL circuits are faster than static circuits and are slower than dynamic circuit

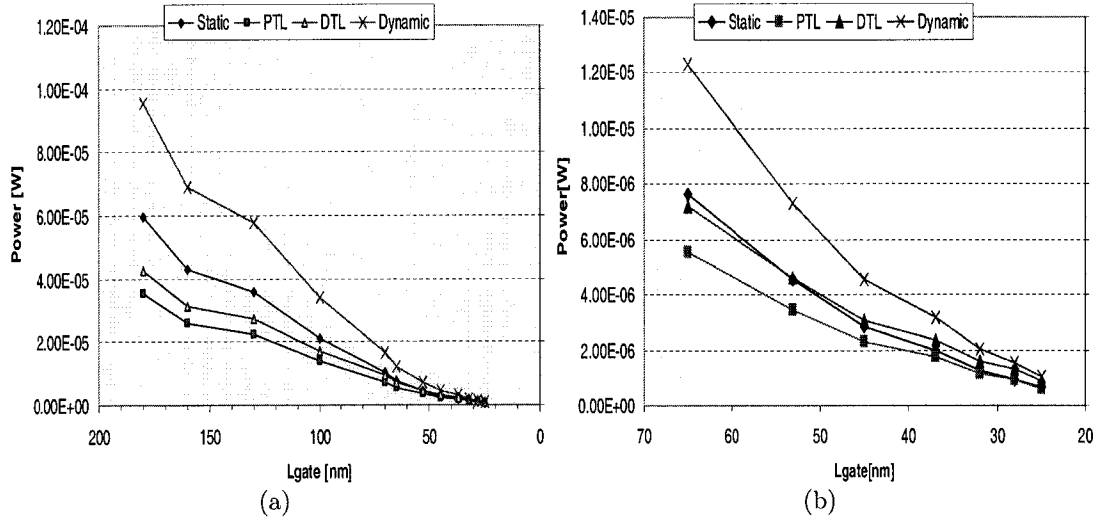


Figure 5.5: Dynamic power projection results.

style. As technology scales down, the delay of mixed PTL circuits become worse than static circuit because mixed PTL circuit sees increasing load capacitance as technology scales down. According to the delay projection, single-rail and dual-rail mixed PTL become worse than static from 65nm and 25nm technology nodes, respectively.

Projections for the dynamic power consumption are shown in Figure 5.5. Dynamic circuits consume more power than other circuit styles. This is because dynamic logic use continuously running clock signal(s), and this continuously running clock signal(s) is one of the major sources of dynamic power consumption in dynamic circuits. The dynamic power consumption of dual-rail mixed PTL are lower than that of static CMOS when the gate length is longer than 50nm. The power consumption of single-rail mixed PTL becomes more than that of static CMOS below 25nm technology node.

Figures 5.6 (a) and (b) show the sub-threshold and the tunneling leakage power projection results using the benchmark circuits. The results indicate that the sub-threshold leakage power consumption increases dramatically beyond the 65nm generation for all circuit styles. However, the mixed PTL style keeps a 2-5X advantage

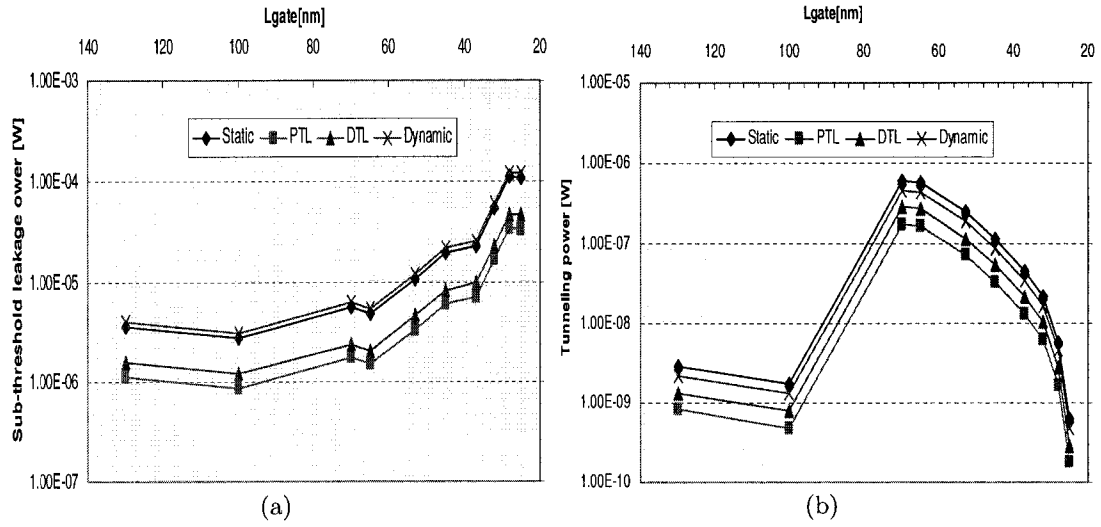


Figure 5.6: Sub-threshold leakage and tunneling leakage power projection results.

over the conventional static CMOS style and the dynamic circuit style throughout different generations of CMOS technology.

The tunneling leakage power is orders of magnitude lower than the sub-threshold leakage power. For generations beyond the 65nm generation, the tunneling leakage power is projected to be decreasing due to the projected introduction of high-k gate dielectric material.

Since the tunneling leakage power plays a lesser important role relative to the sub-threshold leakage power, the overall total leakage power follows the trend of that of the sub-threshold leakage power as shown in Figure 5.7 (a). The results in Figure 5.7 (a) illustrate the overall advantage of the mixed PTL circuit style over other circuit styles in terms of overall leakage power consumption.

The trend for total power consumption, including dynamic, sub-threshold, and tunneling power, is shown in Figure 5.7 (b). Mixed PTL also keeps 2-5x advantage over the conventional static CMOS style and dynamic CMOS style throughout different generations of technologies. As shown in Figure 5.7 (b), the leakage power consumption as the percentage of total power consumption exceeds that of the dynamic power consumption in the technology generations below 50nm gate length.

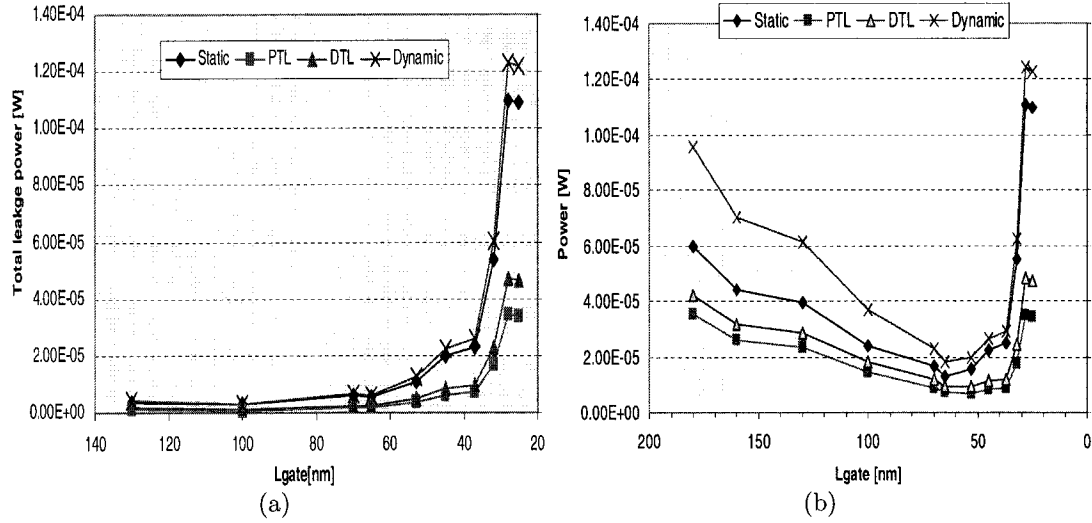


Figure 5.7: Overall power projection results. (a) Total leakage power. (b) Total power including dynamic, sub-threshold leakage, and tunneling power.

Therefore, leakage power reduction is more important than dynamic power reduction for future technology generations.

The power-delay-product and the energy-delay-product projection results are shown in Figures 5.8 and 5.9, respectively. These figures show that the mixed PTL/Static circuits have advantage over the conventional static and dynamic CMOS styles in terms of power-delay-product and energy-delay-product by a large margin.

5.6 Conclusions

In this Chapter, the scalability of mixed PTL/Static circuits was examined by projecting the trends of performance and power consumption on the future technology generations. The projection results show that the mixed PTL circuits scales well down to 25nm technology node. Due to increased junction capacitance, the delay of single-rail and dual-mixed PTL are getting worse below 65nm and 25nm technology generations, respectively, and dual-rail mixed PTL consumes more dynamic power than static CMOS circuit styles. However, mixed PTL circuit keeps advantage over static and dynamic circuit styles throughout the technology generations in the total

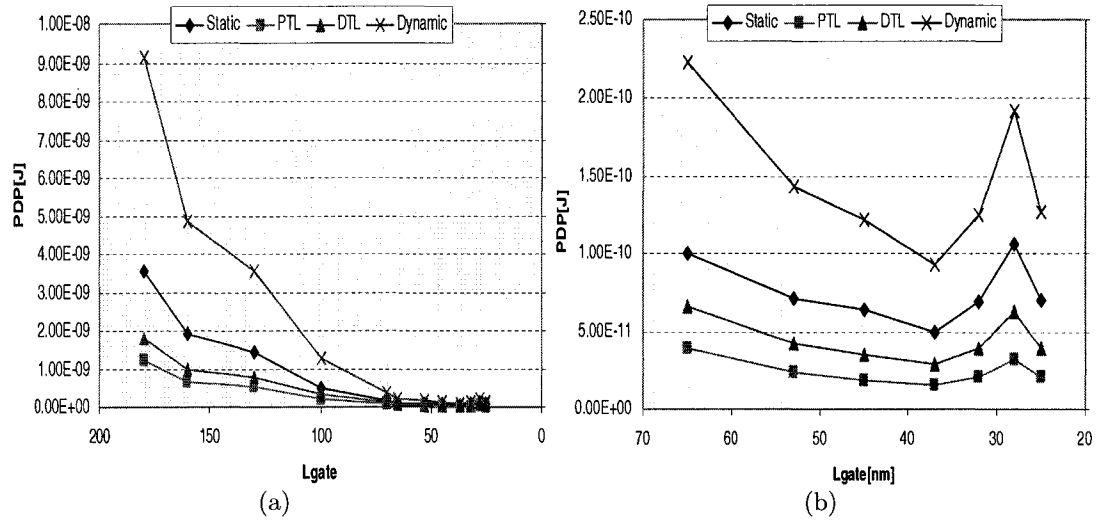


Figure 5.8: Power-delay-product projection results.

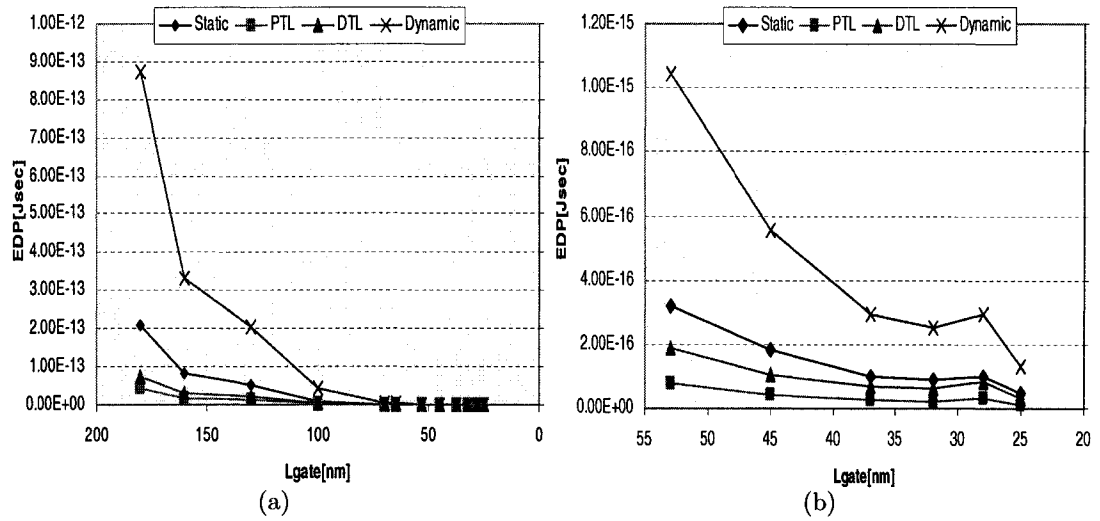


Figure 5.9: Energy-delay-product projection results.

power and leakage power consumption. The projection results give more evidence that the mixed PTL/Static circuit style is a promising alternative to static and domino circuits styles for high performance and low-power applications in the future.

The experimental results of technology scaling effect of the mixed PTL/Static circuit styles and comparisons to that of the domino and the conventional static CMOS will be presented in Chapter 6 in three different state-of-the-art modern CMOS technologies.

Chapter 6

Experimental Results

The Boolean matching and the covering method using genetic algorithms (GAs) presented in this dissertation were implemented in C++ on a Pentium III Linux platform. The implemented program consists of three major parts. The first part of the program performs logic and graph decomposition. The second part performs Boolean matching between cluster functions and pattern functions after building binary decision diagrams (BDDs). We used Colorado University Decision Diagram (CUDD) package[97] to build the BDDs. The last part of the program performs the covering using genetic algorithms (GAs) to generate mapped mixed PTL/Static circuits.

This chapter, first, presents the experimental results of single-rail and dual-rail mixed PTL/Static circuits using the proposed synthesis method and compares these circuits to their static counterparts using SIS[19, 30, 31] and a commercial logic synthesis tool from Synopsys[76] in terms of delay, size, and power consumption. Although comparing our results with that of other mapping algorithms[13, 14] using mixed PTL circuits would be desirable, the difficulty lies in the fact that the results presented in [13, 14] were from mappings to a technology that was not mentioned in their papers. Second, the experimental results of technology scaling impact on the mixed PTL circuits using three different bulk CMOS technologies will be presented. We also present the experimental results of leakage power consumption on ISCAS85

benchmark circuits as well as the benchmark circuits from a 64-bit microprocessor design.

This chapter is organized as follows: Section 6.1 presents the experimental results on the mixed PTL/Static and static CMOS circuits in a $0.25\mu m$ CMOS, a $0.1\mu m$ CMOS, and a $0.13\mu m$ SOI technologies. In Section 6.2, the impact of technology scaling on mixed PTL circuits will be presented by comparing with the results of static and dynamic CMOS circuit styles. The experimental results on leakage power consumption are also presented in Section 6.2. Concluding remarks are given in Section 6.3.

6.1 Experimental Results on Mixed PTL and Static CMOS

6.1.1 Experimental Setup

We synthesized mixed PTL/Static circuits for all the ISCAS85 circuits[98] (10 circuits), some of the MCNC91 circuits (18 circuits)[98], and ten benchmark circuits from a 64-bit microprocessor design using the proposed approach. The characteristics of the ten benchmark circuits from the microprocessor used in this experiment were shown in Table 4.1. Dual-rail mixed PTL/Static circuits were also synthesized using the proposed method. The main difference between single-rail and dual-rail mixed PTL is that, in dual-rail mixed PTL, all single-rail PTL and static cells were replaced with those of dual-rail cells. In addition, the proposed redundant buffer removal method was applied to all dual-rail mixed PTL/Static circuits.

All the PTL cells were locally created using a $0.25\mu m$ CMOS process from TSMC (Taiwan Semiconductor Manufacturing Company), a $0.1\mu m$, a $0.13\mu m$ CMOS process, and a $0.13\mu m$ PDSOI process. The supply voltages for $0.25\mu m$, $0.1\mu m$, $0.13\mu m$

CMOS, and $0.13\mu m$ PDSOI technologies were set to $2.5V$, $1.1V$, $1.3V$, and $1.1V$, respectively. All static cells used in the experiments using the $0.25\mu m$ CMOS process were from a commercial TSMC $0.25\mu m$ standard cell library. For other technologies, all the static cells were from a company referred to as company A.

6.1.2 Experiment Results in a $0.25\mu m$ Bulk CMOS Process

We compared the results to the SIS1.2 technology mapper [31] in terms of delay, size, and power consumption in a $0.25\mu m$ CMOS technology. The results are shown in Table 6.1. The delay and power measures were obtained using *PowerMill* and *PathMill*[73], respectively. To get the estimated power consumption results for circuits from the proposed method and SIS, we used randomly generated test vectors. It is known that the estimated power results depend on the number of applied test vectors and that a large number of random vectors are required. Statistically, however, we can get a good estimate of power with a reasonable number of input vectors[99, 100]. In this experiment, 1000 vectors were used to estimate power consumption.

Table 6.1 shows that our proposed method outperforms the SIS mapping method in both area and power consumption. Out of 28 benchmark circuits, 17 circuits outperform the delay of SIS mapping results. On average, our proposed method outperforms the SIS results by up to 7%, 25%, and 40% in delay, area, and power, respectively.

The amount of PTL cells used in the final solution as a percentage is also shown in Table 6.1. More than half of the gates in the final solution were mapped to PTL cells. It is widely perceived that PTL cells are not good candidates for implementing random logic with few MUX-type functions. However, our results show that the majority of a given logic function can be mapped to PTL to achieve lower power.

Table 6.1: Synthesis results using TSMC 0.25 μm CMOS process, $V_{dd} = 2.5V$.

Circuit	SIS			Single-rail Mixed PTL/CMOS				
	Area (μm^2)	Power (W)	Delay (ns)	Area (μm^2)	Power (W)	Delay (ns)	PTL (%)	CPU (sec)
C17	34.56	2.71248E-06	0.371	17.68	1.87606E-06	0.333	62	0.2
z4ml	266.88	2.71843E-05	0.982	187.75	1.13619E-05	0.913	75	9.1
cm85a	163.20	8.81638E-06	0.754	122.03	4.97350E-06	0.792	46	5.8
pm1	240.00	1.62004E-05	0.742	166.09	9.79278E-06	0.729	49	7.0
cmb	222.72	1.22988E-05	0.614	147.03	4.81478E-06	0.592	63	7.0
tcon	90.24	1.12456E-05	0.354	46.56	3.49415E-06	0.321	88	2.9
cm162a	193.92	1.69276E-05	1.202	101.61	8.52815E-06	0.790	54	8.0
cc	266.88	1.98248E-05	0.765	169.98	1.02557E-05	0.885	59	11.5
cu	311.04	2.34724E-05	0.795	178.57	6.58060E-06	0.521	75	9.1
pcle	274.56	2.00098E-05	1.070	168.45	1.39336E-05	0.695	59	8.0
cm150a	236.16	1.52917E-05	0.800	102.01	3.14843E-06	0.762	66	3.3
mux	193.92	1.79837E-05	0.612	97.18	3.58940E-06	0.588	73	6.2
sct	420.48	3.56385E-05	0.777	289.53	1.71265E-05	0.865	61	6.7
comp	650.88	4.72740E-05	1.469	418.78	3.15643E-05	1.314	43	23.2
C432	1038.72	1.00732E-04	2.498	739.64	4.27678E-05	0.825	55	154.1
c8	614.40	5.46348E-05	0.791	447.60	2.36047E-05	0.851	60	21.7
ttt2	1061.76	7.24873E-05	0.924	696.05	3.64845E-05	1.101	58	61.1
C499	2002.56	1.89635E-04	1.770	1397.40	1.25112E-04	1.776	49	192.9
cht	583.68	6.57218E-05	0.899	410.51	2.90478E-05	0.535	58	35.1
x2	289.92	2.37871E-05	0.736	180.08	1.23180E-05	0.635	57	8.9
C880	1619.04	1.28600E-04	1.841	1192.13	8.80903E-05	1.877	43	248.4
C1355	3050.88	2.46173E-04	2.257	1897.24	1.33281E-04	2.087	41	206.9
C1908	2776.32	2.70395E-04	2.464	2186.09	2.16138E-04	2.790	52	338.2
C2670	3663.60	2.73823E-04	2.397	2921.78	1.83649E-04	2.161	55	1654.9
C3540	5197.44	4.60475E-04	3.376	3758.62	2.48132E-04	3.507	53	4737.0
C5315	8443.20	7.74245E-04	2.777	6649.81	6.23143E-04	3.234	61	151320.0
C6288	12842.88	8.03638E-03	10.757	9804.80	4.58620E-03	9.826	28	7617.0
C7552	11695.20	1.31515E-03	2.627	9314.35	1.01056E-03	2.892	55	63254.0
Avg.	2087.32	4.38825E-04	1.694	1564.62	2.67484E-04	1.578	57	-

Table 6.2: The number of iterations to reach convergence and cost improvement.

Circuit	Initial Cost	No. of Iteration	Final Cost	% Cost Improv.
C17	31.29	51	28.92	7.57
z4ml	299.26	400	199.44	33.36
cm85a	183.22	264	154.5	15.68
pm1	244.20	302	205.38	15.90
cmb	223.32	400	160.52	28.12
tcon	110.84	308	77.28	30.28
cm162a	177.38	314	144.98	18.27
cc	255.96	370	220.43	13.88
cu	277.18	167	227.38	17.97
pcl	254.34	300	213.88	15.91
cm150a	180.30	193	86.98	51.76
mux	173.94	422	80.88	53.50
sct	397.48	147	327.86	17.52
comp	515.66	217	476.06	7.68
C432	980.92	418	847.40	13.61
c8	599.22	314	463.74	22.61
ttt2	973.82	399	811.94	16.62
C499	1860.70	444	1656.08	11.00
cht	696.44	414	484.42	30.44
x2	265.90	299	218.70	17.75
C880	1621.52	381	1398.52	13.75
C1355	2397.90	418	2202.00	8.17
C1908	2776.90	437	2488.26	10.39
C2670	3808.40	503	3104.58	18.48
C3540	5002.82	551	4523.98	9.57
C5315	8965.40	107	7777.96	13.24
C6288	14016.10	390	13826.90	1.35
C7552	11894.50	170	11083.50	6.82
Average	2113.75	325	1910.45	18.37

The CPU time in Table 6.1 was measured on a 733MHz Pentium III PC running Linux with 256MB memory. The CPU time grows dramatically when the circuit size approaches several thousand gates. This is mainly due to the overwhelming number of sub-chromosomes the genetic algorithm has to deal with, for example, the number of sub-chromosomes in C5315 is around one thousand, and the increasing probability of getting invalid global chromosomes during generating global population. One way to improve the run time in the future may involve better choice of crossover and mutation points in the GA based on the type of neighboring cells to improve its efficiency, rather than randomly choose the crossover and mutation points. Furthermore, recent advances in computing technology can also potentially cut the runtime significantly.

Table 6.2 lists the number of iterations to reach convergence for all the circuits, and the corresponding amount of improvement on cost. Most circuits obtained a significant amount of improvement on cost with fewer than 500 iterations.

6.1.3 Experiment Results in a $0.1\mu m$ Bulk CMOS Process

The results of the conventional static CMOS, single-rail and dual-rail mixed PTL circuit styles of the ISCSA85 benchmark circuits and the circuits from a microprocessor in the $0.1\mu m$ CMOS process are shown in Tables 6.3 and 6.4, respectively. The delay and power measures were obtained using *PowrMill* and *PathMill*[73], respectively. To get the estimated power consumption results for circuits from the proposed method and that from Synopsys, we used randomly generated test vectors. It is known that the estimated power results are depends on the number of applied test vectors and a large number of random vectors are required. Statistically, however, we

Table 6.3: Experimental results on ISCAS85 benchmark circuits in $0.1\mu m$ CMOS process.

Circuit	Static using Synopsis synthesis tool				Single-Rail				Dual-Rail			
	Area (μm)	Delay (ps)	Power (W)	PDP (J)	Area (μm)	Delay (ps)	Power (W)	PDP (J)	Area (μm)	Delay (ps)	Power (W)	PDP (J)
C432	4136.2	321.0	2.38E-03	7.63E-13	1976.2	392.0	1.57E-03	6.14E-13	3204.5	419.0	1.64E-03	6.87E-13
C499	8953.6	260.0	4.10E-03	1.07E-12	2367.0	285.0	1.40E-03	3.98E-13	3493.5	256.0	1.99E-03	5.09E-13
C880	6943.0	329.0	3.92E-03	1.29E-12	2052.4	350.0	9.17E-04	3.21E-13	3292.4	319.0	1.24E-03	3.95E-13
C1355	9544.1	258.0	5.04E-03	1.30E-12	2398.1	296.0	1.54E-03	4.55E-13	3417.0	255.0	1.99E-03	5.07E-13
C1908	8861.5	404.0	6.67E-03	2.69E-12	2350.7	432.0	1.72E-03	7.43E-13	3966.6	406.0	2.91E-03	1.18E-12
C2670	9857.9	341.0	7.41E-03	2.53E-12	3005.4	319.0	1.97E-03	6.29E-13	4768.3	292.0	2.68E-03	7.83E-13
C3540	22593.7	542.0	1.51E-02	8.17E-12	8022.7	630.0	4.85E-03	3.05E-12	15585.9	626.0	8.51E-03	5.32E-12
C5315	30042.9	495.0	2.46E-02	1.22E-11	8884.0	561.0	6.59E-03	3.70E-12	13787.9	525.0	9.00E-03	4.72E-12
C6288	84501.2	1634.0	8.40E-02	1.37E-10	13000.6	1554.0	2.05E-02	3.19E-11	29520.6	1551.0	4.78E-02	7.41E-11
C7552	25245.4	1033.0	1.76E-02	1.82E-11	8428.3	513.0	5.42E-03	2.78E-12	14424.5	424.0	7.78E-03	3.30E-12
Avg.	21068.0	561.7	1.71E-02	1.85E-11	5248.5	533.2	4.65E-03	4.46E-12	9546.1	507.3	8.55E-03	9.15E-12

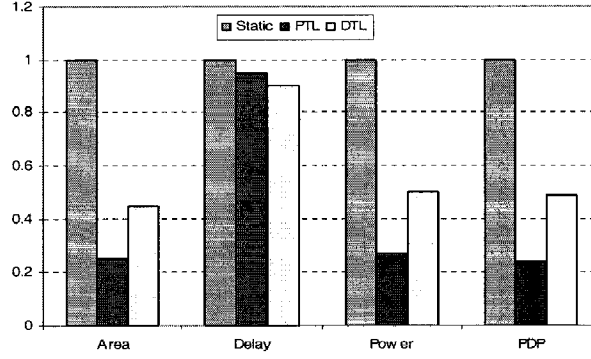


Figure 6.1: Normalized area, delay, and power on ISCAS85 benchmark circuits in $0.1\mu m$ CMOS process. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.

can get a good estimate of power with a reasonable number of input vectors[99, 100]. In this experiment 1000 vectors were used to estimate power consumption.

Tables 6.3 and Figure 6.1 show that the single-rail and dual-rail mixed PTL styles of the ISCAS85 benchmark circuits using the proposed method are 73% and 50% better than their static counterparts in power consumption with performance gain of 5% and 10%, respectively. The area and power overhead of dual-rail circuits over single-rail circuits were 82% and 84%, respectively. However, the area of single-rail and dual-rail circuits using proposed method were 75% and 55% better than that of static counterparts, respectively.

This subsection presents the experimental results of the benchmark circuits from a commercial microprocessor in single-rail, dual-rail mixed PTL/Static circuits using the proposed method, and their static counterparts. The static circuits were synthesized using a commercial synthesis tool from Synopsys.

Table 6.4 and Figure 6.2 show that the delay of the single-rail and the dual-rail mixed PTL styles of the benchmark circuits from the microprocessor are 11% and 35% better than that their static counterparts, respectively. Power savings of the single-rail and dual-rail mixed PTL circuits over their static counterparts were up to

Table 6.4: Experimental results on the benchmark circuits from a microprocessor in $0.1\mu m$ CMOS process.

Circuit	Static using Synopsis synthesis tool				Single-Rail				Dual-Rail			
	Area (μm)	Delay (ps)	Power (W)	PDP (J)	Area (μm)	Delay (ps)	Power (W)	PDP (J)	Area (μm)	Delay (ps)	Power (W)	PDP (J)
Ckt1	310.6	150.0	1.02E-04	1.52E-14	36.4	122.0	1.73E-05	2.11E-15	40.5	87.0	1.53E-05	1.33E-15
Ckt2	1543.4	170.0	3.15E-04	5.35E-14	284.7	185.0	7.05E-05	1.30E-14	415.5	140.0	7.74E-05	1.08E-14
Ckt3	1624.6	165.0	3.32E-04	5.48E-14	862.0	145.0	1.43E-04	2.07E-14	1734.6	111.0	1.91E-04	2.12E-14
Ckt4	1082.9	135.0	3.60E-04	4.86E-14	498.7	153.0	2.04E-04	3.13E-14	809.1	118.0	2.45E-04	2.89E-14
Ckt5	1727.5	73.0	2.97E-04	2.17E-14	209.8	116.0	6.74E-05	7.81E-15	126.9	73.0	6.60E-05	4.82E-15
Ckt6	637.6	116.0	1.65E-04	1.92E-14	133.6	72.0	4.86E-05	3.50E-15	194.2	48.0	6.17E-05	2.96E-15
Ckt7	123.7	137.0	7.58E-06	1.04E-15	45.8	78.0	1.84E-05	1.43E-15	52.9	51.0	2.11E-05	1.08E-15
Ckt8	116.6	120.0	1.42E-05	1.71E-15	37.4	86.0	1.07E-05	9.23E-16	59.5	63.0	1.15E-05	7.27E-16
Ckt9	678.2	236.0	1.99E-04	4.69E-14	148.9	160.0	5.00E-05	8.00E-15	263.1	90.0	5.35E-05	4.81E-15
Ckt10	2890.7	179.0	8.82E-04	1.58E-13	471.4	196.0	1.84E-04	3.61E-14	675.3	178.0	2.22E-04	3.94E-14
Avg.	1073.6	148.1	2.67E-04	4.20E-14	272.9	131.3	8.14E-05	1.25E-14	437.14	95.9	9.64E-05	1.16E-14

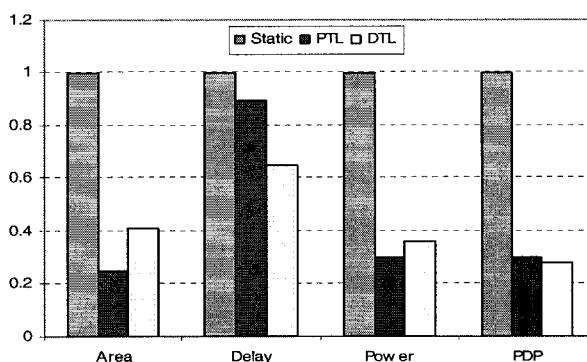


Figure 6.2: Normalized area, delay, and power on the circuits from a microprocessor in $0.1\mu m$ CMOS process. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.

70% and 64%, respectively. The power-delay-product (PDP) of single-rail and dual-rail mixed PTL circuits using the proposed synthesis method are also significantly better than that of their static counterparts.

6.1.4 Experiment Results in a $0.13\mu m$ SOI Process

This section presents the experimental results of static, single-rail mixed PTL, and dual-rail mixed PTL circuits on ISCAS85 benchmark circuits and the benchmark circuits from a microprocessor design (see Table 4.1) in an $0.13\mu m$ SOI CMOS process. In this experiment, the body of SOI devices was floated. The static circuits were also synthesized using a commercial logic synthesis tool from Synopsys. Single-rail and dual-rail mixed PTL/Static circuits were synthesized using the proposed synthesis method.

The results of conventional static CMOS, single-rail and dual-rail mixed PTL in both a $0.13\mu m$ bulk CMOS process and a $0.13\mu m$ PDSOI process using the ISCAS85 and the microprocessor benchmark circuits are shown in Tables 6.5, 6.6, and 6.7, respectively. Figures 6.3 and 6.4 show the normalized area, delay, and power consumption of the benchmark circuits from the microprocessor in the bulk CMOS and

Table 6.5: Experimental results on the benchmark circuits from a microprocessor in $0.13\mu m$ bulk CMOS process.

Circuit	Static				Single-Rail				Dual-Rail			
	Area (μm)	Delay (ps)	Power (mW)	PDP (pJ)	Area (μm)	Delay (ps)	Power (mW)	PDP (pJ)	Area (μm)	Delay (ps)	Power (mW)	PDP (pJ)
Ckt1	403.8	179.3	7.55E-04	1.35E-01	38.2	168.9	7.39E-05	1.25E-02	42.5	112.3	7.95E-05	8.92E-03
Ckt2	2006.4	260.0	1.18E-03	3.07E-01	298.9	257.1	2.11E-04	5.42E-02	436.2	202.7	2.57E-04	5.21E-02
Ckt3	2112.0	105.4	1.32E-03	1.40E-01	905.1	211.6	4.08E-04	8.63E-02	1821.3	162.6	3.51E-04	5.71E-02
Ckt4	1407.8	190.9	1.32E-03	2.52E-01	523.6	202.8	6.77E-04	1.37E-01	849.5	127.4	7.02E-04	8.94E-02
Ckt5	2245.8	187.6	1.19E-03	2.24E-01	220.3	206.6	1.82E-04	3.77E-02	133.3	144.4	1.93E-04	2.79E-02
Ckt6	828.9	124.7	4.48E-04	5.58E-02	140.2	105.6	1.13E-04	1.19E-02	203.9	63.6	1.16E-04	7.37E-03
Ckt7	160.8	155.1	4.59E-05	7.12E-03	48.0	108.2	4.54E-05	4.92E-03	55.5	81.2	2.02E-05	1.64E-03
Ckt8	151.6	182.4	8.66E-05	1.58E-02	39.3	131.5	4.52E-05	5.95E-03	62.5	103.5	4.94E-05	5.11E-03
Ckt9	881.7	172.7	8.76E-04	1.51E-01	156.3	148.2	1.87E-04	2.76E-02	276.2	125.9	1.90E-04	2.39E-02
Ckt10	3757.9	215.8	3.33E-03	7.20E-01	494.9	243.6	5.29E-04	1.29E-01	709.0	181.4	5.85E-04	1.06E-01
Avg.	1395.7	177.4	1.06E-03	2.01E-01	286.5	178.4	2.47E-04	5.07E-02	459.0	130.5	2.54E-04	3.80E-02

Table 6.6: Experimental results on the benchmark circuits from a microprocessor in $0.13\mu m$ partially depleted SOI process.

Circuit	Static				Single-Rail				Dual-Rail			
	Area (μm)	Delay (ps)	Power (mW)	PDP (pJ)	Area (μm)	Delay (ps)	Power (mW)	PDP (pJ)	Area (μm)	Delay (ps)	Power (mW)	PDP (pJ)
Ckt1	1217.4	114.3	3.50E-04	4.00E-02	38.2	133.4	4.58E-05	6.11E-03	42.5	86.0	5.13E-05	4.41E-03
Ckt2	6272.8	171.4	1.53E-03	2.62E-01	298.9	231.0	1.34E-04	3.09E-02	436.2	153.2	1.55E-04	2.38E-02
Ckt3	4698.2	89.9	1.98E-03	1.78E-0	905.1	124.8	4.08E-04	5.09E-02	1821.3	119.0	2.71E-04	3.23E-02
Ckt4	3495.1	116.2	1.11E-03	1.29E-01	523.6	148.6	4.64E-04	6.90E-02	849.5	89.8	5.03E-04	4.52E-02
Ckt5	2136.8	134.3	5.62E-04	7.55E-02	220.3	170.0	1.49E-04	2.53E-02	133.3	120.2	1.55E-04	1.87E-02
Ckt6	1458.6	108.6	3.61E-04	3.92E-02	140.2	74.0	8.52E-05	6.31E-03	203.9	50.5	9.61E-05	4.86E-03
Ckt7	257.0	86.0	4.16E-05	3.58E-03	48.0	83.2	3.05E-05	2.54E-03	55.5	67.2	1.76E-05	1.18E-03
Ckt8	207.2	97.6	4.30E-05	4.20E-03	39.3	112.9	2.73E-05	3.09E-03	62.5	87.7	3.45E-05	3.03E-03
Ckt9	1401.6	151.9	5.19E-04	7.89E-02	156.3	133.1	1.09E-04	1.45E-02	276.2	98.6	1.47E-04	1.45E-02
Ckt10	7721.3	149.4	2.65E-03	3.97E-01	494.9	234.8	3.59E-04	8.43E-02	709.0	141.9	4.38E-04	6.21E-02
Avg.	2886.6	122.0	9.16E-04	1.21E-01	286.5	144.6	1.81E-04	2.93E-02	459.0	101.4	1.87E-04	2.10E-02

Table 6.7: Experimental results on ISCAS85 benchmark circuits in $0.13\mu m$ partially depleted SOI process.

Circuit	Static using Synopsis synthesis tool				Single-Rail				Dual-Rail			
	Area (μm)	Delay (ps)	Power (W)	PDP (J)	Area (μm)	Delay (ps)	Power (W)	PDP (J)	Area (μm)	Delay (ps)	Power (W)	PDP (J)
C432	40118.7	426.0	2.52E-03	1.07E-12	11672.6	524.0	1.16E-03	6.10E-13	23875.6	460.0	2.28E-03	1.05E-12
C499	62759.9	369.0	8.18E-03	3.02E-12	17029.2	356.0	2.08E-03	7.40E-13	33050.2	256.0	2.98E-03	7.62E-13
C880	70480.3	344.0	4.05E-03	1.39E-12	16564.2	455.0	1.41E-03	6.40E-13	35047.1	403.0	2.51E-03	1.01E-12
C1355	62862.3	364.0	7.45E-03	2.71E-12	26709.9	349.0	3.46E-03	1.21E-12	31238.3	310.0	6.55E-03	2.03E-12
C1908	57974.1	555.0	7.13E-03	3.96E-12	30607.3	548.0	4.23E-03	2.32E-12	35263.2	527.0	4.71E-03	2.48E-12
C2670	122830.6	329.0	9.33E-03	3.07E-12	27645.0	360.0	3.66E-03	1.32E-12	47296.4	342.0	4.86E-03	1.66E-12
C3540	154295.0	673.0	1.28E-02	8.60E-12	50749.4	794.0	5.88E-03	4.67E-12	96127.5	794.0	9.40E-03	7.46E-12
C5315	255406.8	633.0	2.64E-02	1.67E-11	55172.9	720.0	8.01E-03	5.76E-12	89886.6	659.0	1.13E-02	7.44E-12
C6288	541259.4	2310.0	7.03E-02	1.62E-10	95371.2	2201.0	2.87E-02	6.31E-11	244380.4	2078.0	6.67E-02	1.39E-10
C7552	253881.7	856.0	2.05E-02	1.75E-11	57041.0	713.0	6.68E-03	4.76E-12	120313.6	517.0	1.30E-02	6.73E-12
Avg.	162186.9	685.9	1.69E-02	2.21E-11	38856.3	702.0	6.52E-03	8.51E-12	75647.9	634.6	1.24E-02	1.69E-11

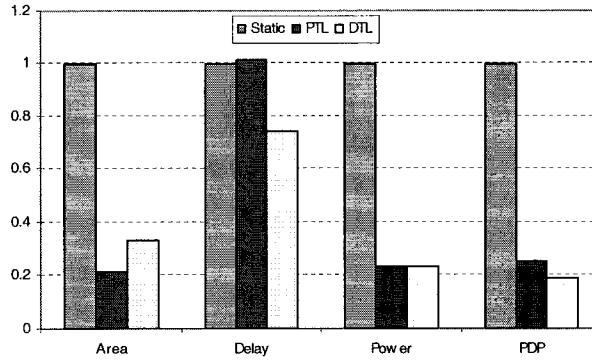


Figure 6.3: Normalized area, delay, and power of the benchmark circuits from a microprocessor in $0.13\mu m$ bulk CMOS process. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.

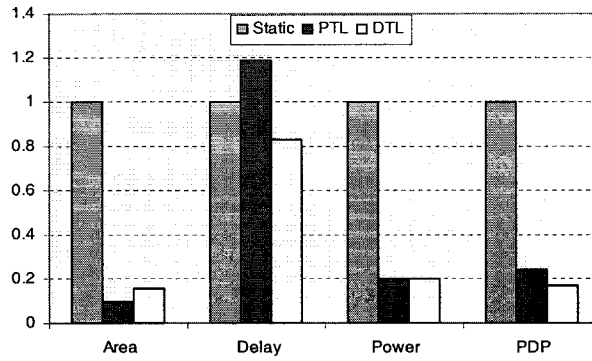


Figure 6.4: Normalized area, delay, and power of the benchmark circuits from a microprocessor in $0.13\mu m$ SOI process. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.

SOI processes. Figure 6.5 shows the normalized area, delay, and power consumption of ISCAS85 circuits in the $0.13\mu m$ PDSOI process. The delay and power measures of the benchmark circuits from the microprocessor were obtained using SPICE circuit simulators that use BSIM3 and BSIM3SOI model for the bulk CMOS and PDSOI processes, respectively. The delay and power measures of ISCAS85 circuits were obtained using *PathMill* and *PowrMill*, respectively.

Figure 6.3 and Table 6.5 show that the average delay of single-rail mixed PTL/Static circuits on the benchmark circuits using the proposed method in bulk CMOS is slightly higher (1%) than that of their static counterparts. Figure 6.4 and Table 6.6

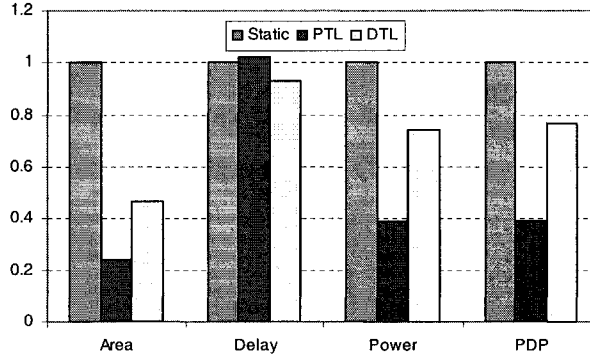


Figure 6.5: Normalized area, delay, and power of ISCAS85 benchmark circuits in $0.13\mu m$ SOI process. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.

show that the average delay of single-rail mixed PTL/Static circuits on the benchmark circuits using the proposed method in partially depleted SOI is 19% higher than that of their static counterparts. However, the average delay of dual-rail in both bulk CMOS and partially depleted SOI technologies are 26% and 17% better than their static counterparts, respectively. As shown in Figure 6.5 and Table 6.7, the average delay of single-rail mixed PTL on ISCAS85 circuits is also higher (2%) than that of static circuits while the delay of dual-rail mixed PTL circuits is 7% better than its static counterpart. These results indicate that the performance gain of mixed PTL circuits over their static counterparts in SOI is less than that of the bulk CMOS process. This decreased performance gain in SOI is mainly due to the fact that the delay of static circuits in SOI improved more dramatically compared to their bulk CMOS counterparts.

The average power of single-rail and dual-mixed mixed PTL style of the benchmark circuits from the microprocessor in bulk CMOS process are approximately four times smaller than that of their static counterparts. In the SOI process, the average power consumptions of single-rail and dual-rail circuits using the benchmark circuits from the microprocessor are 80% and 79% better than their static counterparts, respectively. For ISCAS85 benchmark circuits, the average power consumptions of

single-rail and dual-rail circuits are 61% and 26% better than their static counterparts, respectively. In bulk CMOS, the average areas of single-rail and dual-rail mixed PTL style of the benchmark circuits from the microprocessor using proposed method were 79% and 67% better than that of static counterparts, respectively. In SOI, the average areas of single-rail and dual-rail mixed PTL style of ISCAS85 circuits are 76% and 53% better than their static counterparts, respectively. The average areas of single-rail and dual-rail mixed PTL style of the benchmark circuits from the microprocessor were nine and five times smaller than that of their static counterparts, respectively. One obvious question to these results is that why the area and power consumption of the dual-rail mixed PTL circuits are not doubled, especially for the benchmark circuits from the microprocessor, compared to those of single-rail mixed PTL circuits. This is because we applied the redundant buffer removal technique[101] to the dual-rail mixed PTL circuits. Removing redundant buffers in dual-rail mixed PTL circuits has a significant impact on the delay and power.

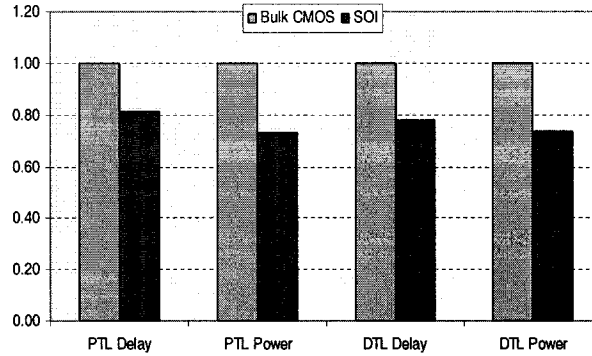


Figure 6.6: Normalized delay and power in $0.13\mu m$ SOI and bulk CMOS processes. PTL and DTL represents single-rail and dual-rail mixed PTL, respectively.

Figure 6.6 compares the average delay and power consumption of mixed PTL style of the benchmark circuits from the microprocessor in the $0.13\mu m$ bulk CMOS with those in the $0.13\mu m$ SOI process. The average delay of single-rail and dual-rail mixed PTL circuits in SOI process are 19% and 22% better than their counterparts

in the bulk CMOS process, respectively. Power consumption of single-rail and dual-rail mixed PTL circuits using the proposed methods in the SOI process are 27% and 26% better compared to those circuits in the bulk CMOS process. This further confirms that increased driving capability and decreased capacitance in SOI improve the performance and power consumption of CMOS circuits compared to those in the bulk CMOS process. In addition, these results also indicate that mixed PTL circuits using the proposed method are good candidates for future high-performance and low-power digital system designs in both bulk CMOS and partially depleted SOI process.

6.2 Experimental Results on Technology Scaling and Leakage

6.2.1 Experimental Setup

Ten circuits with varying complexities were chosen from a 64-bit microprocessor design as a set of benchmark circuits. These circuits can be categorized into three different groups: NAND/NOR gate intensive datapath logic, XOR/XNOR intensive datapath logic, and random logic. Each circuit was implemented in domino, conventional static CMOS, single-rail, and dual-rail mixed PTL circuits. These circuits were mapped to $0.18\mu m$, $0.13\mu m$, and $0.1\mu m$ CMOS technologies, and the supply voltages for the CMOS technology were 1.5V, 1.3V, and 1.0V, respectively. The characteristics of the benchmark circuits, which are taken from a microprocessor, for this experiment are shown in Table 4.1.

All circuits were optimized under various power/delay tradeoff conditions using a commercial circuit optimization tool. Figure 6.7 shows the power-delay curve of a circuit in three different implementations using the $0.18\mu m$ CMOS technology. Every

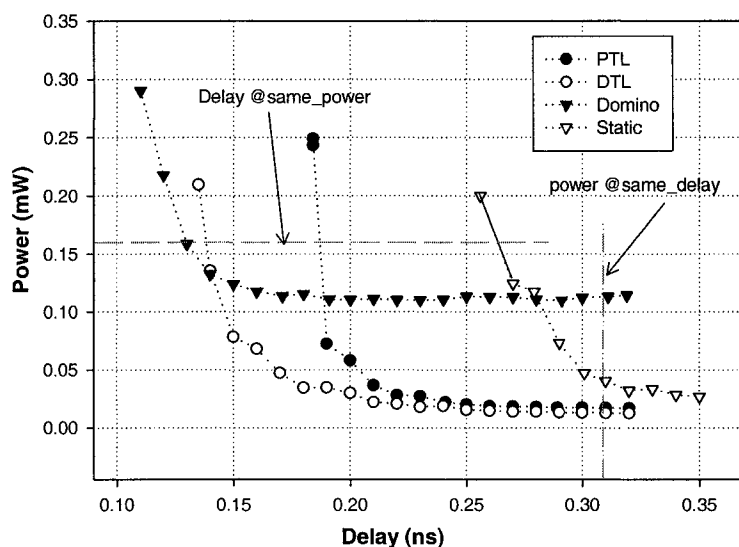


Figure 6.7: Power-Delay curve of four different implementations of a circuit in $0.18\mu m$.

single point in the power-delay curve represents different power vs. delay tradeoff of the circuit. Activity factors for power simulation were set to 30% for all circuits and technologies.

As we can see from Figure 6.7, the power-delay curves are in a well-defined convex shape. Because normal design points of a circuit in a practical situation should be located somewhere around the kneeling points of the power-delay curve, it is reasonable that the delay and the power of three different implementations need to be chosen at the kneeling points for proper comparison.

The leakage current for all circuits were obtained by averaging leakage current measurement from SPICE using a set of input patterns in a $0.1\mu m$ CMOS technology. The measured leakage current for each benchmark circuit is compared to that estimated by the models in the previous sections. The stacking factor for a cell, S_i , were calculated by averaging the leakage currents of all possible input combinations as described in Section 5.4.2.

6.2.2 Delay and Dynamic Power

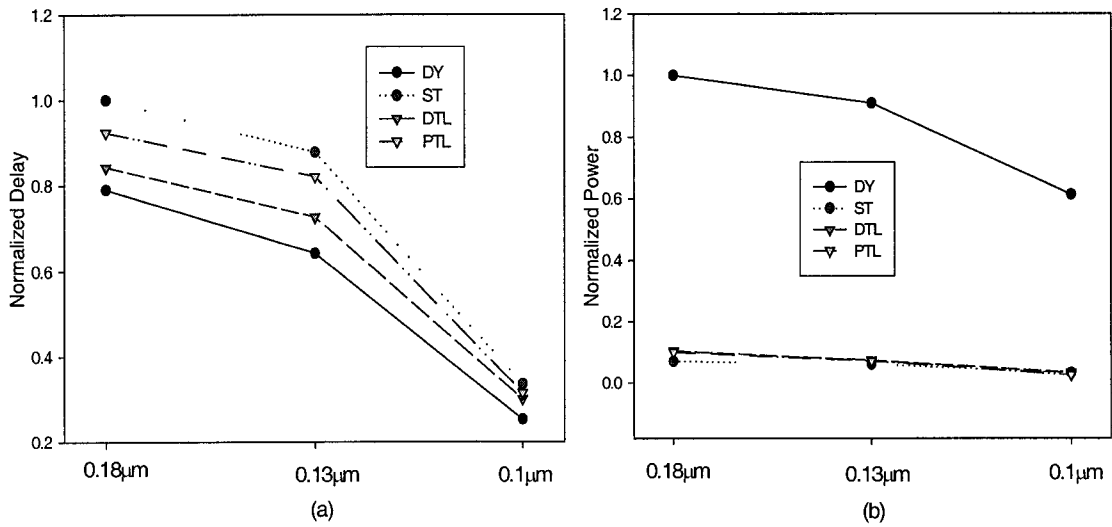


Figure 6.8: Group 1 normalized average (a) delay and (b) power.

The normalized average power and delay curves for Group 1 in each technology are shown in Figure 6.8, where DY, ST, PTL, and DTL represent domino, static, single-rail, and dual-rail mixed PTL/Static circuits, respectively. The delay of each circuit shows almost the same trend as the technology scales down. The delay of domino circuits is fastest in this group. Because domino use only n-tree to implement certain logic functions, this circuit style can realize NOR/NAND functions very efficiently. Dual-rail PTL showed better delay performance than that of the single-rail PTL. This is because single-rail PTL circuits contain inverters to generate opposite polarity of control-inputs. These inverters can cause more delay than DTL circuits. The static logic shows the worst delay performance among all the technologies. This indicates that mixed PTL/Static circuits can implement NOR/NAND intensive logic in a more efficient way than the conventional static CMOS logic.

In the case of power dissipation, the domino logic showed the highest power as expected. The rest of the circuit styles, static CMOS logic, single-rail, and dual-rail PTL circuits, show almost the same power dissipation.

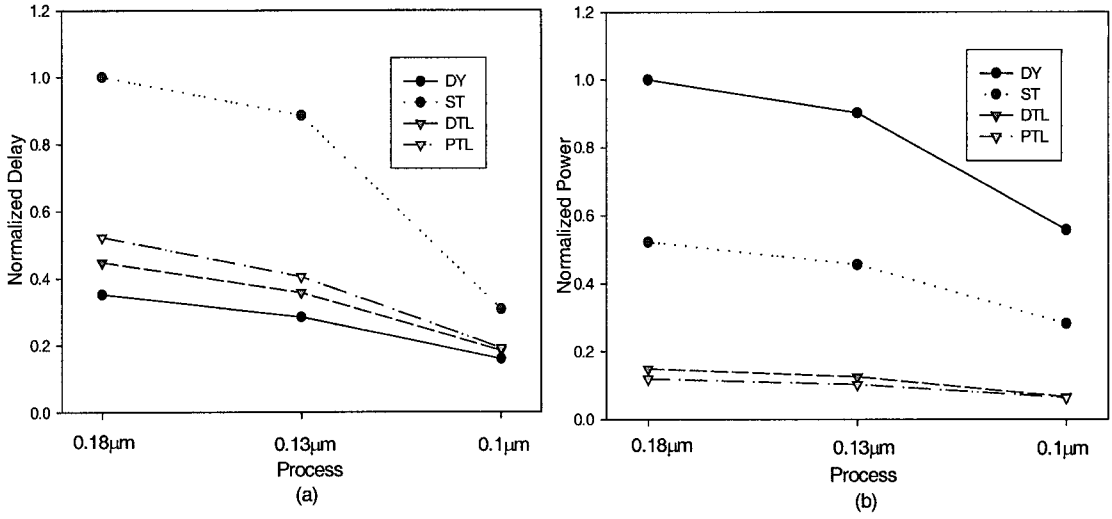


Figure 6.9: Group 2 normalized average (a) delay and (b) power.

Figure 6.9 shows the normalized average delay and power trend of Group 2 in each of three technologies. In this group, the domino circuit style also shows the best delay, but the gap between the mixed PTL circuits and the domino circuits is less than that of the circuits in Group 1 due to the fact that mixed PTL/Static logic can implement XOR/XNOR functions more efficiently. The power results for this group of circuits show significant advantage of mixed PTL logic. Not only is the gap between the mixed PTL logic and domino logic maintained, but the gap between the mixed PTL logic and the conventional static CMOS logic has widened significantly. This can be attributed to structural efficiency of using PTL logic to implement XOR/XNOR intensive logic.

The trends of normalized average delay and power in Group 3 are shown in Figure 6.10. It is interesting to note that mixed PTL circuits show the best delay in Group 3. This result is due to the way the circuits were synthesized using the mixed PTL synthesis tool[80]. The mixed PTL synthesis tool performs global searches using genetic algorithms to maximize the coverage of logic cones for all PTL gates in the circuit. It also strategically determines the best spots for using static gates

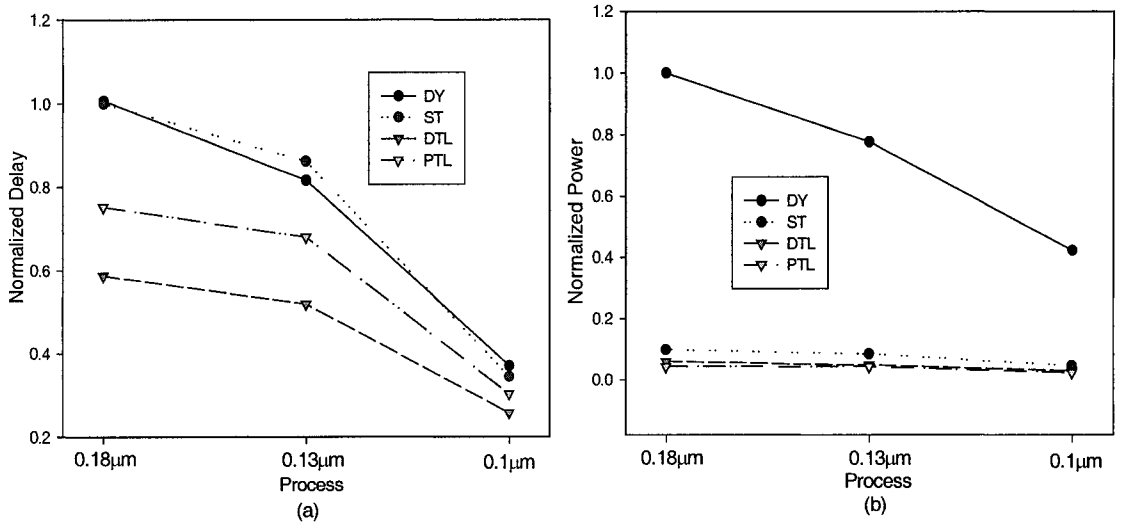


Figure 6.10: Group 3 normalized average (a) delay and (b) power.

to perform buffering functions. Consequently, using the mixed PTL logic can result in simpler implementations in terms of the number of gates used, leading to better performance and power consumption. The results also reaffirm the well-known fact that the dynamic logic styles are not suitable for implementing random logic.

Figure 6.11 shows overall normalized average delay and power trends for all circuits in each of the three technologies. On average, the domino circuits showed the best delay characteristics, with the dual-rail mixed PTL logic coming in a close second. We can also see that the delay gap between mixed PTL logic and domino logic is small relative to that between the static logic and the domino logic. The experimental results of the relative advantages of different logic design styles in terms of delay and power shown in Figure 6.11 matches those projected shown in Figures 5.4 and 5.7.

The gap between dynamic logic and mixed PTL logic in terms of delay shrinks with technology scaling. This shrinkage results from the fact that dynamic logic becomes less and less effective in improving speed when leakage current become more pronounced with technology scaling. This illustrates the potential viability of the

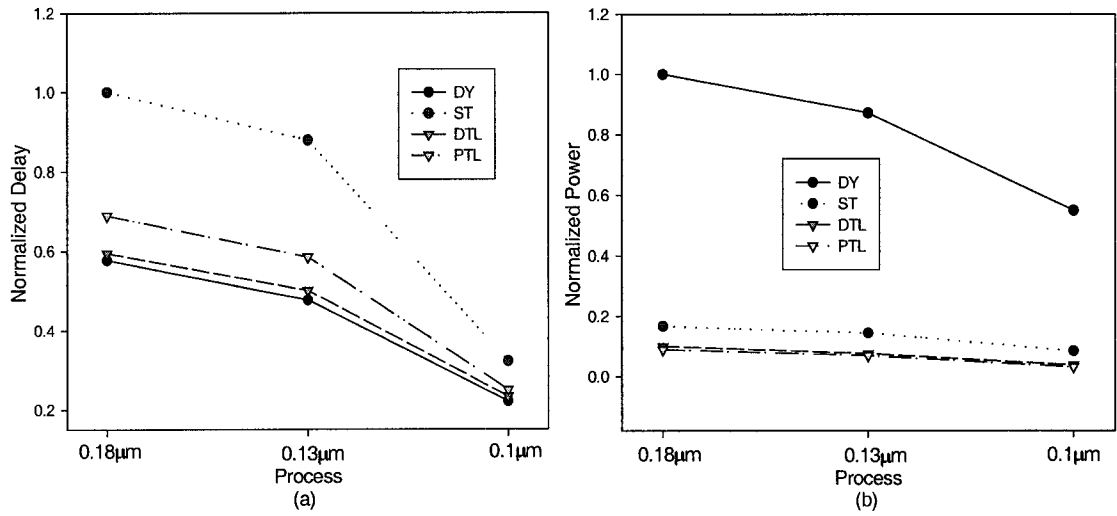


Figure 6.11: Overall normalized average (a) delay and (b) power.

mixed PTL logic family for future high performance and low power applications. As for power, the mixed PTL logic maintains an advantage over the conventional static logic and a much larger advantage over the dynamic logic.

The normalized average power-delay-product and energy-delay-product of ten benchmark circuits are shown in Figures 6.12 (a) and (b), respectively. The single-rail and the dual-rail mixed PTL/Static logic are the lowest energy consumers in all technologies. Power-delay-product and energy-delay-product projection results shown in Figures 5.8 and 5.9 show the similar power-delay-product and energy-delay-product trends shown Figures 6.12 (a) and (b).

6.2.3 Experimental Results on Leakage Power Consumption

In order to compare the amount of leakage current of different logic styles, three different logic styles were implemented using pure static, single-rail and dual-rail mixed PTL/Static logic styles for ISCAS85 circuits. And ten circuits with varying complexities were chosen from a 64-bit microprocessor design as another set of benchmark circuits. The benchmark circuits from a microprocessor, shown in Table 4.1,

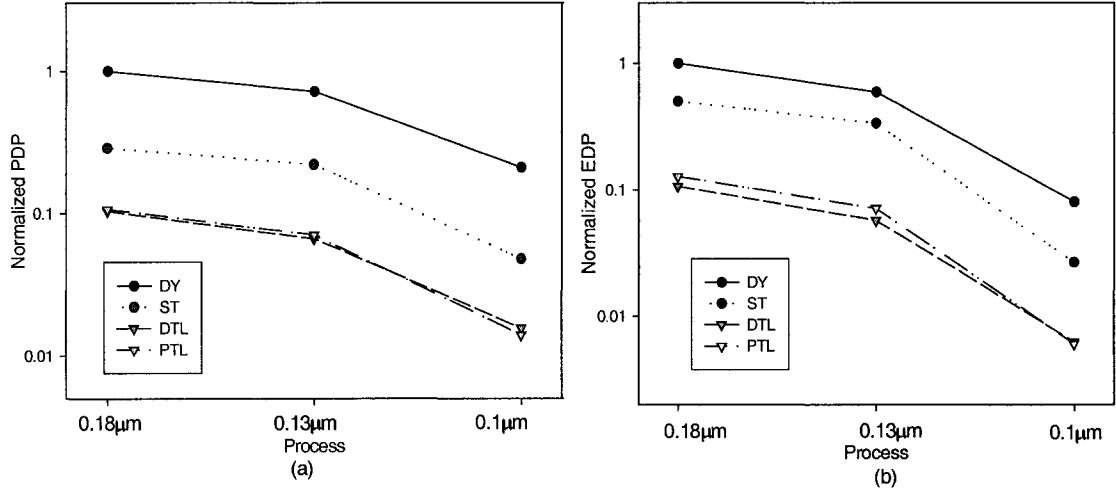


Figure 6.12: Overall normalized average (a) power-delay-product and (b) energy-delay-product in each technology node.

were implemented in four different circuits, pure static, single-rail and dual-rail mixed PTL/Static, and dynamic logic styles, respectively. The size of each circuit styles for both benchmark circuits were determined so that the delay of each circuit style for the same benchmark circuits in the range of 10% difference. The technology used in this experiments was a $0.1\mu m$ CMOS process. The leakage current for all different circuits was obtained by averaging leakage current measurement from SPICE using a set of input patterns. The measured leakage current for each benchmark circuit is compared to that estimated by the models in the previous sections. The stacking factor for a cell, S_i , were calculated by averaging the leakage currents of all possible input combinations as described in Section 5.4.2.

The measured and estimated leakage current of different circuits for ISCAS85 and the benchmark circuits from a microprocessor are shown in Figures 6.13 and 6.14, respectively. In Figures 6.13 and 6.14, PTL and DTL represent single-rail and dual-rail mixed PTL/Static circuits, respectively. Both the measured and estimated leakage currents include both the sub-threshold current and the gate tunneling current.

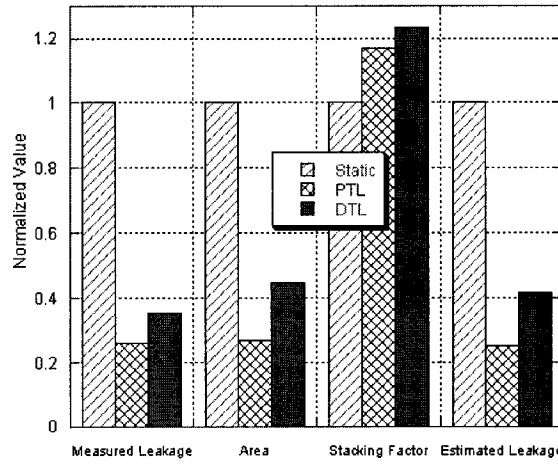


Figure 6.13: Normalized measured and estimated leakage current of ISCAS85 benchmark circuits.

For the ISCAS85 benchmark circuits, the mixed PTL circuits, on average, consume about a third of the leakage power compared to that of the conventional static CMOS circuits. This can be attributed mainly to two reasons:

1. the mixed PTL circuits has, on average, higher stacking factors than that of the conventional static CMOS circuits,
2. the overall gate area of the mixed PTL circuits are much smaller than that of the conventional static CMOS circuits in terms of relative relationships.

The estimated leakage power consumption using the results from Section 5.4.2 matches the experimental results very well. The dual-rail mixed PTL circuits consume slightly higher leakage power compared to their single-rail counterpart purely due to the fact that they are bigger. In terms of stacking factor, the dual-rail mixed PTL circuits, on average, are actually slightly higher than that of the single-rail PTL circuits because of the elimination of inverters in the dual-rail PTL circuits.

The benchmark circuits from the 64-bit microprocessor design consist of mainly datapath circuits. The results shown in Figure 6.14 follow the same trend as that

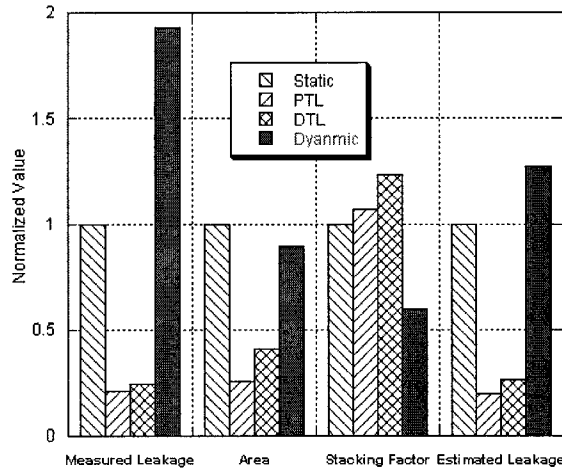


Figure 6.14: Normalized measured and estimated leakage current of the benchmark circuits from a microprocessor.

shown in Figure 6.13 for the ISCAS85 circuits for the same reason mentioned above. One interesting and yet predictable fact is the highly elevated leakage currents associated with the dynamic circuits. Although the gate area of the dynamic circuits is smaller than that of conventional static CMOS circuits, the highly elevated leakage current comes from the much reduced stacking factor due to

1. pull-up keeper circuits,
2. pre-charging circuits to reduce charge sharing, and
3. elimination of p-tree in the logic.

The estimated leakage power consumption using the results from Section 5.4.2 matches the trends of the experimental results except for the dynamic circuits. The difference between estimated leakage and measured leakage for the dynamic circuits may come from the inaccuracy of calculating the stacking factors due to complex structures related to keeper circuits and pre-charging circuits to reduce charge sharing.

6.3 Conclusions

In this chapter, the experimental results of static circuits were presented in terms of delay, power, and power-delay-product. The results are compared with single-rail and dual-rail mixed PTL/Static circuits that were synthesized using the proposed method.

The experimental results of technology scaling effect of the mixed PTL/Static circuit styles are also presented and compared to that of the domino and the conventional static CMOS in three different state-of-the-art modern CMOS technologies. All the circuits used in this section were optimized under various power/delay tradeoff conditions to take the full advantage of each circuit style. The experimental results show that the mixed PTL circuits scale well down to the level of $0.1\mu m$ technology.

The experimental results show that the mixed PTL/Static circuits generated by the proposed method have advantage over the conventional static CMOS and dynamic circuit styles in terms of performance and power consumption including leakage power.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this dissertation, a new mixed PTL/Static logic synthesis method using genetic algorithms (GAs) was presented.

Pass-transistor cell structure used in mixed PTL/static logic significantly impacts mixed PTL/Static circuits in terms of performance and power consumption. A new pass-transistor cell structure for mixed PTL/Static circuits based on the comparison of performance/power consumption and analysis of different pass-transistor logic structures was presented in Chapter 2. There is a slight area overhead in the proposed cell. The proposed pass-transistor cell structure, however, showed the best power and performance results compared to the other three cell structures because the proposed cell structure exhibits full rail-to-rail output voltage swing and reduces the amount of short circuit current at static stage. These benefits significantly outweigh the slightly bigger area in terms of performance and power consumption.

In Chapter 3, existing matching techniques were described. Since Boolean matching method is more efficient than graph matching method, we have chosen Boolean matching and implemented to create static and PTL match lists.

A new covering method for mixed PTL/Static logic was proposed in Chapter 4. The proposed GA-based mapping and covering method allows global optimization to explore the best combinations of PTL cells and static CMOS cells for a given circuit in terms of area, delay, and power consumption. The crossover and mutation for GA were performed on graph structure chromosomes that are the mapped solutions of a given circuit. The parameters used for the GAs were evaluated, and the proposed covering algorithm converged well. Also, a performance enhancing method through redundant buffer removal for dual-rail mixed PTL/Static logic was developed. It was shown that the use of the redundant buffer removal method with a fanout optimization method significantly improves power and performance. We also presented a sequential circuit handling method to the proposed mixed PTL/static logic synthesis flow that can incorporate the mix PTL synthesis into a commercial logic synthesis flow.

In Chapter 5, the scalability of PTL was examined by projecting the delay and power consumption trends for the future technology generations. The projection results show that the leakage power exceeds technology scales down because of decreasing transistor threshold voltage. The projection results, however, show that the mixed PTL circuits scales well down to 25nm technology node.

In Chapter 6, various experimental results of the mixed PTL/Static circuits and their static counterparts were presented. The single-rail and dual-rail mixed PTL/Static circuits synthesized using the proposed method were compared to their static counterparts which are synthesized using SIS and a commercial logic synthesis tool from Synopsys as well. The experiments were performed in $0.1\mu m$ CMOS and $0.13\mu m$ PDSOI CMOS technologies. The experimental results on technology scaling show that the mixed PTL/Static circuits scale well down to the $0.1\mu m$ technology node level compared to their static and dynamic counterparts. For the leakage power, the mixed PTL circuits consume less amount of leakage power than that of their static

and dynamic counterparts because of high stacking factor as well as small area of the mixed PTL circuits. This scalability gives more evidence that the mixed PTL/Static circuit style is a promising alternative to static and domino circuits styles for high performance and low-power applications in the future. The overall experimental results also confirm that the mixed PTL/Static logic using the proposed synthesis method is a promising alternative to conventional static CMOS circuits for future high-performance and low-power VLSI designs.

7.2 Future Work

Our research has been focused on the mixed PTL/Static logic synthesis method using genetic algorithms, and the generated circuits were optimized on a global level. The mixed PTL circuits using the proposed mixed PTL synthesis method have advantages in terms of power and/or performance over their static and dynamic counterparts. The scalability of the mixed PTL/Static circuits has also been presented by means of experiment and projection for future CMOS technologies, and it has been shown that the mixed PTL circuits well scales for both modern and future technology generations. However, the CPU time of the proposed synthesis method, as shown in Chapter 6, grows dramatically when the circuit size approaches several thousand gates. This is mainly due to the the increasing probability of getting invalid global chromosomes during global population generation. The run time in the future may involve a better choice of crossover and mutation points in the GA based on the type of neighboring cells to improve its efficiency, rather than randomly choosing the crossover and mutation points.

Appendix A

Abbreviations and Acronyms

ASIC	Application-Specific Integrated Circuits
ADTPT	Asymmetrical Dynamic Threshold Pass-Transistor
BDD	Binary Decision Diagram
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CUDD	Colorado University Decision Diagram
DAG	Directed Acyclic Graph
DSM	Deep-Submicron
DTMOS	Dynamic Threshold MOSFET
GA	Genetic Algorithm
FA	Full Adder
FDSOI	Fully Depleted SOI
HDL	Hardware Description Language
ITRS	International Technology Roadmap for Semiconductors
MFP	Multiple Fanout Point
MOS	Metal Oxide Semiconductor
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
OBDD	Ordered Binary Decision Diagram
PDSOI	Partially Depleted SOI
PTL	Pass-Transistor Logic
RTL	Register Transfer Level
SOI	Silicon-On-Insulator
TSMC	Taiwan Semiconductor Manufacturing Company
VHDL	Very High Speed Integrated Circuits (VHSIC) Hardware Description Language
XOR	Exclusive-OR
XNOR	Exclusive-NOR

Appendix B

Notation

$\mathcal{G}$	A set of mapped subject graph
$\mathcal{M}$	Match List
f_{sym}	Symmetry set of a Boolean function f
N_{gpop}	Number of Global-Populations
N_{subpop}	Number of Sub-Populations
μ	Mutation rate
δ	Threshold value that triggers mutation
ξ_i	A global chromosome
ω_i	A sub-population
Ω	A set sub-populations
c^p	Cross-over point
m^p	Mutation point
N_{itr}	Number of iterations
$\mathcal{P}_g$	A set of global chromosomes
V_{dd} or V_{DD}	Power supply voltage
V_{th}	Threshold voltage of a MOS transsitor
S_f	Stacking factor
ψ_s	Surface potential
L_{eff}	Effecting channel length of a MOS device
t_{ox} or T_{ox}	Gate Oxide Thickness
ϵ_{ox}	Dielectric constant of a gate oxide
$sup(f)$	Support set of a Boolean function f
$\pi(f)$	Ordered support set of a Boolean function f
g_m	Transconductance of a MOS transistor
I_{ox}	Tunneling current through gate oxide
I_{leak}	Sub-threshold leakage current of a MOS device
I_{DS}	Drain-Source current of a MOS device
t_p or t_d	Propagation delay time of a gate
t_{HL}/t_{LH}	High-to-low / Low-to-high delay time

Appendix C

Source Codes

Table C.1: Summary of source codes for mixed PTL/Static logic synthesis.

Code Name	Lines	Descriptions
LinkedList.h	549	Class definition of double linked list and its methods
lsElem.h	117	Class definition of the elements of linked-list
Node.h	364	Class definition of Node (a gate in un-mapped circuit)
Graph.h	688	Class definition of Graph and its method
eqn_opr.h	196	Class definition of equation operators
eqn_stack.h	123	Class definition of the fixed-length array stack
Queue.h	111	Class definition of Queue using linked list
Stack.h	83	Class definition of Stack using linked list
LibCell_new.h	182	Class definition of a cell in the cell library
PinInfo_new.h	177	Class definition of input and output pins of a cell
Read_eqn.cc	1693	Reads SLIF equations which is equivalent to netnist
Read_lib.cc	601	Reads mixed PTL and Static Cell library
Subgraph_Bdd_new.cc	1225	Graph partitioning and BDD generation
Permute.cc	94	Subroutines for BDD variable re-ordering
Match_depth.cc	3019	Check and build match between cluster and cell and
MapGraph.h	573	Class definition of mapped circuit and methods
MapNode.h	241	Class definition of mapped circuit and methods
Delay_cal_new.cc	1733	Delay and Cost calculation
Gacover_sub_dPTL.cc	2974	Subroutines for GA covering
Spice_net.cc	1180	Generates ELDO circuit netlist
main.cc	327	Top level program for mixed PTL synthesis
Total	16250	-

```

/*****
 * LinkedList.h
 * Class definition of double linked list and its methods
 *****/
#ifndef LINKEDLIST_CLASS
#define LINKEDLIST_CLASS

#include <fstream.h>
#include <string.h>
#include <stdlib.h>

#ifndef NULL
const int NULL = 0;
#endif

#include "lsElem.h"

template <class T>
class LinkedList
{
protected:
    // Dummy nodes for the ends of the Linked List
    lsElem<T> headdummy, taildummy;

    // pointers maintain access to front and rear of list
    lsElem<T> *front, *rear;

    // used for data retrieval, insertion and deletion
    lsElem<T> *prevPtr, *currPtr;

    // number of elements in the list
    int size;

    // position in list. used by Reset method
    int position;

    // private methods to allocate and deallocate nodes
    //lsElem<T> *GetlsElem(const T& item, lsElem<T> *ptrNext=NULL);

    lsElem<T> *GetlsElem(const T& item);
    void FreelsElem(lsElem<T> *p);

    // copies list L to current list
    void CopyList(const LinkedList<T>& L);

public:
    // constructors
    LinkedList(void);
    LinkedList(const LinkedList<T>& L); // Not yet

    // destructor
    ~LinkedList(void); // Not yet

    // assignment operator
    LinkedList<T>& operator=(const LinkedList<T>& L); // Not yet

    // methods to check list status
    int ListSize(void) const;
    int isEmpty(void) const;

    // Traversal methods
    void Reset(int pos = 0);
    void Next(void);

    int CurrentPosition(void) const;
    int isEndOfList(void) const;
    int EndOfList(void) const;

    // Insertion methods
    void InsertFront(const T& item);
    void InsertRear(const T& item);
    //void InsertAt(const T& item);
    void InsertRight(const T& item);

    // Deletion methods
    T DeleteFront(void);
    T DeleteRear(void);
    T DeleteCurrent(void);

    //void DeleteAt(int pos);
    lsElem<T> *DeleteCurrentElem(void);
    lsElem<T> *DeleteFrontElem(void);
    lsElem<T> *DeleteRearElem(void);

    // Data retrieval/modification
    T& Data(void); // To get currPtr->data.
    T& Data(int pos); // To get currPtr->data.

    // method to clear the list
    void ClearList(void);

    friend class Graph;
    friend class MapGraph;
    friend class LibCell;
};

template <class T>
lsElem<T> *LinkedList<T>::GetlsElem(const T& item)
{
    lsElem<T> *p;
    p = new lsElem<T>(item);
    if (p == NULL)
    {
        cout
        << "Memory allocation failure LinkedList::GetlsElem(T& item)!\n";
        exit(1);
    }
    return p;
}

template <class T>
void LinkedList<T>::FreelsElem(lsElem<T> *p)
{
    delete p;
}

// copy L to the current list, which is assumed to be empty
template <class T>
void LinkedList<T>::CopyList(const LinkedList<T>& L)
{
    // use p to chain through L
    lsElem<T> *p = L.front;
    int pos;

    // insert each element in L at the rear of current object
    while (p != NULL)
    {
        InsertRear(p->data);
        p->Next = p->NextlsElemRight();
        p->Prev = p->NextlsElemLeft();
    }

    // if list is empty return
    if (position == -1)
        return;

    // reset prevPtr and currPtr in the new list
    currPtr = front;
    for (pos = 0; pos != position; pos++)
    {
        currPtr->Next = currPtr->NextlsElemRight();
        currPtr->Prev = currPtr->NextlsElemLeft();
    }
}

// create empty list by setting pointers to NULL, size to 0
// and list position to -1
template <class T>
LinkedList<T>::LinkedList(void)
{
    front = &headdummy;
    rear = &taildummy;
    front->Next = rear;
    front->Prev = NULL;
    rear->Next = NULL;
    rear->Prev = front;
    prevPtr = NULL;
    currPtr = front;
    size = 0;
    position = -1;
}

template <class T>
LinkedList<T>::LinkedList(const LinkedList<T>& L)
{
    front = rear = NULL;
    prevPtr = currPtr = NULL;
    size = 0;
    position = -1;
    CopyList(L);
}

template <class T>
void LinkedList<T>::ClearList(void)
{
    lsElem<T> *currPosition, *nextPosition;

    currPosition = front->Next;
    while (currPosition != rear)
    {
        // get address of next node and delete current node
        nextPosition = currPosition->Next;
        FreelsElem(currPosition);
        currPosition = nextPosition; // Move to next node
    }
    currPtr = front->Next;
    size = 0;
    position = -1;
}

template <class T>
LinkedList<T>::~LinkedList(void)
{
    if (size != 0)
    {
        ClearList();
    }
}

template <class T>
LinkedList<T>& LinkedList<T>::operator=(const LinkedList<T>& L)
{
    if (this == &L) // Can't assign list to itself
        return *this;

    ClearList();
    CopyList(L);
    return *this;
}

template <class T>
int LinkedList<T>::ListSize(void) const
{
    return size;
}

template <class T>
int LinkedList<T>::isEmpty(void) const
{
    return size == 0;
}

// move prevPtr and currPtr forward one node
template <class T>
void LinkedList<T>::Next(void)
{
    // if traversal has reached the end of the list or
    // the list is empty, just return
    if (currPtr != NULL)
    {
        // advance the two pointers one node forward
    }
}

```

```

        currPtr = currPtr->Next;
        position++;
    }
}

// True if the client has traversed the list (for while loop)
template <class T>
int LinkedList<T>::isEndOfList(void) const
{
    return currPtr->Next == rear;
}

// To use in for loop
template <class T>
int LinkedList<T>::EndOfList(void) const
{
    return currPtr == rear;
}

// return the position of the current node
template <class T>
int LinkedList<T>::CurrentPosition(void) const
{
    return position;
}

// reset the list position to pos
template <class T>
void LinkedList<T>::Reset(int pos)
{
    int startPos;

    // if the list is empty, return
    if (front == NULL)
        return;

    // if the position is invalid, terminate the program
    if (pos < 0 || pos > size-1)
    {
        cerr << "Reset: Invalid list position: " << pos << "\n";
        cout << "Reset: Invalid list position: " << pos << "\n";
        return;
    }

    // move list traversal mechanism to node pos
    if (pos == 0)
    {
        // reset to front of the list
        currPtr = front->Next;
        position = 0;
    }
    else
    {
        // reset currPtr, prevPtr, and position
        {
            currPtr = front->Next->Next;
            startPos = 1;

            // move right until position == pos
            for (position = startPos; position != pos; position++) {
                currPtr = currPtr->Next;
            }
        }
    }
}

// return a reference to the data value in the current node
template <class T> T& LinkedList<T>::Data(void)
{
    // error if list is empty or traversal completed
    if (size == 0)
    {
        cerr << "Data: invalid reference!" << endl;
    }
    return currPtr->data;
}

template <class T> T& LinkedList<T>::Data(int pos)
{
    // error if list is empty or traversal completed
    if (size == 0)
    {
        cerr << "Data: invalid reference!" << endl;
        //exit(1);
    }
    Reset(pos);
    return currPtr->data;
}

// Insert item at front of list
template <class T> void LinkedList<T>::InsertFront(const T& item)
{
    lsElem<T> *newlsElem;
    newlsElem = GetlsElem(item);    // create the new node

    newlsElem->Next = front->Next;
    newlsElem->Prev = front;
    front->Next = newlsElem;
    newlsElem->Next->Prev = newlsElem;

    currPtr = newlsElem;
    position = 0;
    size++;
}

// Insert item at rear of list
template <class T> void LinkedList<T>::InsertRear(const T& item)
{
    lsElem<T> *newlsElem;
    newlsElem = GetlsElem(item);    // create the new node

    newlsElem->Next = rear;
    newlsElem->Prev = rear->Prev;
    rear->Prev = newlsElem;
    newlsElem->Prev->Next = newlsElem;

    prevPtr = rear->Prev->Prev;
    currPtr = newlsElem;
    position = size;
    size++;
}

// Insert item after the current list position
template <class T>
void LinkedList<T>::InsertRight(const T& item)
{
    lsElem<T> *p;

    if (currPtr == NULL) {
        // First time insertion after object initialized
        cout << "Trying to insert an object without CURRENT Object" << endl;
        cout << "Forced to InsertRear()" << endl;
        InsertRear(item);
    }
    else {
        p = GetlsElem(item);
        p->Prev = currPtr;
        p->Next = currPtr->Next;
        currPtr->Next->Prev = p;
        currPtr->Next = p;

        currPtr = p; // Move Current Object Pointer;
        position++;
        size++;
    }
}

// Delete the node at the front of list
template <class T> T LinkedList<T>::DeleteFront(void)
{
    lsElem<T> *templElem;
    T rdata;

    if (size) {
        size--;
        templElem = front->Next;
        front->Next = templElem->Next;
        templElem->Next->Prev = front;
        templElem->Next = NULL;
        templElem->Prev = NULL;

        currPtr = front->Next; // Move the current object position
        position = 0;
        rdata = templElem->data;
        delete templElem;
        return rdata;
    }
    else {
        cout << "Invalid deletion!" << endl;
        return NULL;
    }
}

// Delete the node at the rear of list
template <class T> T LinkedList<T>::DeleteRear(void)
{
    lsElem<T> *templElem;
    T rdata;

    if (size) {
        size--;
        templElem = rear->Prev;
        rear->Prev = templElem->Prev;
        templElem->Prev->Next = rear;
        templElem->Prev = NULL;
        templElem->Next = NULL;

        currPtr = rear->Prev; // Move the current object position
        position = ListSize()-1;

        rdata = templElem->data;
        delete templElem;
        return rdata;
    }
    else {
        cout << "In DeleteRear(), Invalid deletion (List Size = 0)!" << endl;
        return NULL;
    }
}

// Delete the node at the current list position
template <class T> T LinkedList<T>::DeleteCurrent(void)
{
    lsElem<T> *p;
    T item;

    // deletion must occur at front->Next node
    // or rear->Prev node or inside the list
    if (currPtr->Prev == front && size) { // delete "front-Next"
        item = DeleteFront();
        return item;
    }
    else if (currPtr->Next == rear && size) {
        item = DeleteRear();
        return item;
    }
    else {
        size--;
        p = currPtr;
        p->Prev->Next = p->Next;
        p->Next->Prev = p->Prev;
        p->Next = NULL;
        p->Prev = NULL;

        currPtr = p->Prev;

        item = p->data;
        delete p; // deleting current lsElem
        position--;

        return item;
    }
}

```

```

}

// Delete the node at the front of list
template <class T> lsElem<T> *LinkedList<T>::DeleteFrontElem(void)
{
    //T item;

    lsElem<T> *templElem;

    if (size) {
        size--;
        templElem = front->Next;
        front->Next = templElem->Next;
        templElem->Next->Prev = front;
        templElem->Next = NULL;
        templElem->Prev = NULL;

        currPtr = front->Next; // Move the current object position
        position = 0;
        //return templElem->data;
        return templElem;
    }
    else {
        cout << "Invalid deletion!" << endl;
        return NULL;
    }
}

// Delete the node at the rear of list
template <class T>
lsElem<T> *LinkedList<T>::DeleteRearElem(void)
{
    lsElem<T> *templElem;

    if (size) {
        size--;
        templElem = rear->Prev;
        rear->Prev = templElem->Prev;
        templElem->Prev->Next = rear;
        templElem->Prev = NULL;
        templElem->Next = NULL;

        currPtr = rear->Prev; // Move the current object position
        position = ListSize()-1;
        //return templElem->data;
        return templElem;
    }
    else {
        cout << "In DeleteRear(), Invalid deletion (List Size = 0)!" << endl;
        return NULL;
    }
}

template <class T>
lsElem<T> *LinkedList<T>::DeleteCurrentElem(void)
{
    lsElem<T> *p;

    if (currPtr->Prev == front && size) { // delete "front-Next"
        p = DeleteFrontElem();
        return p;
    }
    else if (currPtr->Next == rear && size) {
        p = DeleteRearElem();
        return p;
    }
    else {
        if (size > 0) {
            size--;
            p = currPtr;
            p->Prev->Next = p->Next;
            p->Next->Prev = p->Prev;
            p->Next = NULL;
            p->Prev = NULL;

            currPtr = p->Prev;
            position--;
            return p;
        }
        else {
            return NULL;
        }
    }
}
#endif // LINKEDLIST_CLASS

/*****
* lsElem.h
* Class definition of the elements of linked-list
*****/
#ifndef DLL_LSELEM_CLASS
#define DLL_LSELEM_CLASS

template <class T>
class lsElem
{
private:
public:
    // circular links to the left and right
    lsElem<T> *Prev;
    lsElem<T> *Next;
public:
    // data is public
    T data;

// constructors
lsElem(void);
lsElem(const T& item);

// default destructor
~lsElem();

// list modification methods
void InsertRight(lsElem<T> *p);
void InsertLeft(lsElem<T> *p);
lsElem<T> *DeleteElem(void);
T* Data(void);

// obtain address of the next node to the left or right
lsElem<T> *NextlsElemRight(void) const;
lsElem<T> *NextlsElemLeft(void) const;
};

// constructor that creates an empty list and
// leaves the data uninitialized. use for header
template <class T>
lsElem<T>::lsElem(void)
{
    // initialize the node so it points to itself
    Prev = Next = NULL;
}

// default destructor
template <class T>
lsElem<T>::~lsElem()
{
};

// constructor that creates an empty list and initializes data
template <class T>
lsElem<T>::lsElem(const T& item)
{
    // set node to point to itself and initialize data
    Prev = Next = NULL;
    data = item;
}

// insert a node p to the right of current node
template <class T>
void lsElem<T>::InsertRight(lsElem<T> *p)
{
    // link p to its successor on the right
    p->Next = Next;
    Next->Prev = p;

    // link p to the current node on its left
    p->Prev = this;
    Next = p;
}

// insert a node p to the left of current node
template <class T>
void lsElem<T>::InsertLeft(lsElem<T> *p)
{
    // link p to its successor on the left
    p->Prev = Prev;
    Prev->Next = p;

    // link p to the current node on its right
    p->Next = this;
    Prev = p;
}

// unlink the current node from the list and return its address
template <class T>
lsElem<T> *lsElem<T>::DeleteElem(void)
{
    // node to the left must be linked to current node's right
    Prev->Next = Next;

    // node to the right must be linked to current node's left
    Next->Prev = Prev;

    // return the address of the current node
    return this;
}

// return pointer to the next node on the right
template <class T>
lsElem<T> *lsElem<T>::NextlsElemRight(void) const
{
    return Next;
}

// return pointer to the next node on the left
template <class T>
lsElem<T> *lsElem<T>::NextlsElemLeft(void) const
{
    return Prev;
}
#endif // DLL_LSELEM_CLASS

/*****
* Node.h
* Class definition of Node, which is a gate in un-mapped circuit
*****/
#ifndef NODE_CLASS
#define NODE_CLASS

#include "LinkedList.h"

//typedef enum node_type_enum NodeType;
enum NodeType {
    UNASSIGNED, PRIMARY_INPUT, PRIMARY_OUTPUT, INTERNAL, DUMMY
};

//typedef enum node_function_enum NodeFunc;
enum NodeFunc {
    NODE_UNDEFINED, NODE_PT, NODE_PD, NODE_O, NODE_I, NODE_BUF,
    NODE_INV, NODE_AND, NODE_OR, NODE_NAND, NODE_NOR, NODE_COMPLEX,
    NODE_PTL1, NODE_PTL2, NODE_PTL3
};

enum CellType { STATIC, PTL, UNKNOWN}; // also used in LibCell.h

class pinMap
{
public:
    char *cell_pin;
};

```

```

    char *net_node;
    pinMap() {
        cell_pin = new char[3]; // since name_len = 1;
        net_node = new char[63];
    }
};

class Match
{
public:
    char *cell_name;
    LinkedList<pinMap *> *pinmap;
    Match();
    ~Match();
};

// constructor
Match::Match()
{
    pinmap = new LinkedList<pinMap *>;
};

// destructor
Match::~Match()
{
    free(cell_name);
    delete pinmap;
}

class Node
{
private:
    char *name; // node name
    NodeType type; // node type
    NodeFunc func; // node function
    CellType mattype; // matched cell type (ST, PTL) during
                    // covering */

    int id; // node id

    int numfanin; // Number of Inputs
    LinkedList<Node *> *fanin; // List of fanin nodes

    int numfanout; // Number of fanout nodes
    LinkedList<Node *> *fanout; // List of fanin nodes

    LinkedList<Match *> *smap; // Static mapped cell list
    LinkedList<Match *> *pmap; // PTL mapped cell list

    double areaP; // Area
    double powerP; // Power for a cell
    double fDelayP; // Falling delay
    double rDelayP; // Rising delay

    double areaS; // Area
    double powerS; // Power for a cell
    double fDelayS; // Falling delay
    double rDelayS; // Rising delay

    int visited; // flag for visiting
    int level;
    int glevel;

public:
    Node(void); // Default constructor
    Node(char *name, NodeType type, NodeFunc func, int id);
    ~Node(void); // Destructor

    void SetName(char *nname);
    char * GetName(void);

    void SetNodeType(NodeType gtype);
    NodeType GetNodeType(void);

    void SetNodeFunc(NodeFunc gfunc);
    NodeFunc GetNodeFunc(void);

    void SetMatchType(CellType match_type); // for covering
    CellType GetMatchType(void); // for covering

    void SetNodeId(int gid);
    int GetNodeId(void);

    void MarkVisit(int mark=1);
    int GetVisit(void);

    // local level
    void SetLevel(int level);
    int GetLevel(void);

    // Global level
    void SetGLevel(int glevel);
    int GetGLevel(void);

    // Ptl
    void SetAreaP(double narea);
    void SetPowerP(double npower);
    void SetFallDelayP(double nrDelay);
    void SetRiseDelayP(double nrDelay);

    double GetAreaP(void);
    double GetFallDelayP(void);
    double GetRiseDelayP(void);
    double GetPowerP(void);

    // Static
    void SetAreaS(double narea);
    void SetPowerS(double npower);
    void SetFallDelayS(double nrDelay);
    void SetRiseDelayS(double nrDelay);

    double GetAreaS(void);
    double GetFallDelayS(void);

    double GetRiseDelayS(void);
    double GetPowerS(void);

    int GetNumFanin(void);
    int GetNumFanout(void);
    void SetNumFanin(int nfanin);
    void SetNumFanout(int nfanout);

    void InsertFanin(Node *&N);
    void InsertFanout(Node *&N);
    LinkedList<Node *> * GetFaninList(void);
    LinkedList<Node *> * GetFanoutList(void);

    LinkedList<Match *> * GetSMatchList(void);
    LinkedList<Match *> * GetPMatchList(void);

    friend class Graph;
};

// Default Constructor
Node::Node()
{
    name = (char *)calloc(2, sizeof(char));
    fanin = new LinkedList<Node *>;
    fanout = new LinkedList<Node *>;

    smap = new LinkedList<Match *>;
    pmap = new LinkedList<Match *>;

    numfanin = 0;
    numfanout = 0;
    visited = -1;
    level = -1;
    glevel = -1;
};

// Constructor
Node::Node(char *gname, NodeType gtype, NodeFunc gfunc, int gid)
{
    int name_len=0;
    if(name_len>0){
        name = (char *)calloc(name_len+1, sizeof(char));
        name = gname;
    }
    type = gtype;
    func = gfunc;
    id = gid;

    fanin = new LinkedList<Node *>;
    fanout = new LinkedList<Node *>;

    smap = new LinkedList<Match *>;
    pmap = new LinkedList<Match *>;

    numfanin = 0;
    numfanout = 0;

    areaP = powerP = 0.0;
    fDelayP = rDelayP = 0.0;

    areaS = powerS = 0.0;
    fDelayS = rDelayS = 0.0;

    visited = -1;
    level = -1;
    glevel = -1;
};

// Default destructor
Node::~Node(void)
{
    free(name);
    delete fanin;
    delete fanout;
    delete smap;
    delete pmap;
};

// Set node name
void Node::SetName(char *nname)
{
    int name_len, old_len;

    old_len = strlen(name);
    name_len = strlen(nname);
    if(old_len != name_len){
        name = (char *)realloc(name, name_len+2);
        strcpy(name, nname);
        strcat(name, "\0");
    } else {
        strcpy(name, nname);
    }
};

// Get node name
char * Node::GetName(void) { return name; }

// Set node type and Get node type
void Node::SetNodeType(NodeType gtype) { type = gtype; }
NodeType Node::GetNodeType(void) { return type; }

// Set node func and Get node func
void Node::SetNodeFunc(NodeFunc gfunc) { func = gfunc; }
NodeFunc Node::GetNodeFunc(void) { return func; }

void Node::SetMatchType(CellType match_type) //for covering
{
    mattype = match_type;
}

CellType Node::GetMatchType(void) // for covering
{
    return mattype;
}

```

```

// set node id (number)
void Node::SetNodeId(int gid) { id = gid; }
// get node id
int Node::GetNodeId(void) { return id; }

// set local level
void Node::SetLevel(int nlevel) { level = nlevel; }
// get local level
int Node::GetLevel(void) { return level; }

// set global level
void Node::SetGLevel(int nlevel) { glevel = nlevel; }
// get global level
int Node::GetGLevel(void) { return glevel; }

void Node::MarkVisit(int mark) {visited = mark;}
int Node::GetVisit(void) {return visited;}

void Node::SetAreaP(double narea)
{
    if (narea >= 0) areaP = narea;
}

void Node::SetPowerP(double npower)
{
    if (npower >= 0) powerP = npower;
}

void Node::SetFallDelayP(double nfdelay)
{
    if(nfdelay >= 0) fDelayP = nfdelay;
}

void Node::SetRiseDelayP(double nrdelay)
{
    if(nrdelay >= 0) rDelayP = nrdelay;
}

double Node::GetAreaP(void) { return areaP; }
double Node::GetFallDelayP(void) { return fDelayP; }
double Node::GetRiseDelayP(void) { return rDelayP; }
double Node::GetPowerP(void) { return powerP; }

void Node::SetAreaS(double narea)
{
    if (narea >= 0) areaS = narea;
}
void Node::SetPowerS(double npower)
{
    if (npower >= 0) powerS = npower;
}

void Node::SetFallDelayS(double nfdelay)
{
    if(nfdelay >= 0) fDelayS = nfdelay;
}
void Node::SetRiseDelayS(double nrdelay)
{
    if(nrdelay >= 0) rDelayS = nrdelay;
}

double Node::GetAreaS(void) { return areaS; }
double Node::GetFallDelayS(void) { return fDelayS; }
double Node::GetRiseDelayS(void) { return rDelayS; }
double Node::GetPowerS(void) { return powerS; }

int Node::GetNumFanin(void) { return numfanin; }
int Node::GetNumFanout(void) { return numfanout; }

void Node::SetNumFanin(int nfanin) { numfanin = nfanin; }
void Node::SetNumFanout(int nfanout) { numfanout = nfanout; }

void Node::InsertFanin(Node *g)
{
    fanin->InsertRear(g);
    numfanin++;
}

void Node::InsertFanout(Node *g)
{
    fanout->InsertRear(g);
    numfanout++;
}

// Get fanin list of a node
LinkedList<Node*> * Node::GetFaninList(void)
{
    return fanin;
}

// Get fanout list of a node
LinkedList<Node*> * Node::GetFanoutList(void)
{
    return fanout;
}

// Get Static Cell map List
LinkedList<Match*> * Node::GetSMatchList(void)
{
    return smap;
}

// Get PTL Cell map List
LinkedList<Match*> * Node::GetPMatchList(void)
{
    return pmap;
}
#endif // NODE_CLASS

/*
 * Graph.h
 * Class definition of Graph and its methods
 */

#ifdef GRAPH_CLASS
#define GRAPH_CLASS

#include <iostream.h>
#include <fstream.h>
#include "Node.h"
#include "lsElem.h"
#include "LinkedList.h"

// Definitoin of Graph class
class Graph
{
private:
    char * network_name;

    LinkedList<Node*> *pilist; // Lists of all primary inputs
    LinkedList<Node*> *polist; // Lists of all primary outputs
    LinkedList<Node*> *allnodes; // Lists of all pi/po/gates
    LinkedList<Edge*> *edgelist; // Lists of all nets

    int numgate, numpi, numpo, numedge;
    int size, unique_id;

    float area_given; // Required area
    float area;
    float delay_given; // Required delay
    float delay;
    float power_given; // Required power
    float power;

public:
    Graph();
    ~Graph(void); // Destructor

    void SetName(char *name);
    char *GetName();
    int GetSize();
    void SetSize(int new_size);

    int GetNumGate();
    void SetNumGate(int ngate);

    void InsertPiNode(Node *g);
    void InsertPoNode(Node *g);
    void InsertGate(Node *g);

    LinkedList<Node*> * GetPiList(void);
    LinkedList<Node*> * GetPoList(void);
    LinkedList<Node*> * GetGateList(void);
    Node * GetGate(char *gate_name, NodeType &ntype, NodeFunc &nfunc);
    int DeleteGate(char *gate_name);

    int FindNode(LinkedList<Node*> *L, char *item_name);
    Node * FindOrCreateNode(char *node_name, int &position,
        NodeType ntype=UNASSIGNED,
        NodeFunc nfunc=NODE_UNDEFINED);

    int FaninOutRearrange(void);

    // DeleteEdge, returns positoin of deleted edge in the edgelist
    int DeleteEdge(Node *g, Node *tonode);

    void PrintPiNodes(void);
    void PrintPoNodes(void);
    void PrintAllNodes(void);
    void Print_Node_Type(int type);
    void Print_Node_Func(int type);
    void PrintNodesOfList(LinkedList<Node*> *nodelist);
};

// Constructor
Graph::Graph()
{
    network_name = (char *)calloc(2, sizeof(char));
    allnodes = new LinkedList<Node*>;
    pilist = new LinkedList<Node*>;
    polist = new LinkedList<Node*>;

    area_given = area = 0.0;
    delay_given = delay = 0.0;
    power_given = power = 0.0;

    size = numgate = numpi = numpo = 0;
    unique_id = 1;
    numedge=0;
};

// Destructor
Graph::~Graph()
{
    free(network_name);
    if(allnodes->ListSize()!=0){
        allnodes->ClearList();
        delete allnodes;
    }
    if(pilist->ListSize()!=0){
        pilist->ClearList();
        delete pilist;
    }
    if(polist->ListSize()!=0){
        polist->ClearList();
        delete polist;
    }
};

void Graph::SetName(char *name)
{
    int name_len, net_name_len;

    name_len = strlen(name);
    net_name_len = strlen(network_name);

    if(name_len != net_name_len){
        network_name = (char *)realloc(network_name, name_len+1);
        strcpy(network_name, name);
    }
}
#endif

```

```

    } else {
        strcpy(network_name, name);
    }
}

char * Graph::GetName() { return network_name; }

int Graph::GetSize() { return size; }
void Graph::SetSize(int new_size) { size = new_size; }

int Graph::GetNumGate()
{
    return numgate;
}

void Graph::SetNumGate(int ngate)
{
    numgate = ngate;
}

LinkedList<Node *> * Graph::GetGateList(void) {return allnodes;}
LinkedList<Node *> * Graph::GetPiList(void) {return pilist;}
LinkedList<Node *> * Graph::GetPoList(void) {return polist;}

void Graph::InsertPiNode(Node *& item)
{
    item->SetNodeId(size);
    pilist->InsertRear(item);
    allnodes->InsertRear(item);
    numpi++; numgate++;
    size++;
}

void Graph::InsertPoNode(Node *& item)
{
    item->SetNodeId(size);
    polist->InsertRear(item);
    allnodes->InsertRear(item);
    numpo++; numgate++;
    size++;
}

void Graph::InsertGate(Node *& gate_name)
{
    gate_name->SetNodeId(size);
    allnodes->InsertRear(gate_name);
    numgate++;
    size++;
}

// Retrieve Node class by searching the node_name
Node * Graph::GetGate(char *& gate_name, NodeType &ntype, NodeFunc &nfunc)
{
    int position;
    Node *tempNode;

    if ((position = FindNode(allnodes, gate_name)) != -1) {
        allnodes->Reset(position);
        tempNode = allnodes->Data();
        ntype = tempNode->GetNodeType();
        nfunc = tempNode->GetNodeFunc();

        return tempNode;
    }
    else {
        return tempNode = NULL;
    }
}

// Delete a Node (pi, po, gates) searching the node name
// Returns 0 if it exist and deleted, -1 otherwise
int Graph::DeleteGate(char *& gate_name)
{
    int position;
    char *temp_name;
    Node *tempNode;
    NodeType type;

    if ((position = FindNode(allnodes, gate_name)) != -1) {
        allnodes->Reset(position);
        tempNode = allnodes->Data();
        temp_name = tempNode->GetName();
        type = tempNode->GetNodeType();

        // Delete all edges that are connected to this Node
        // 1. find all nodes (fanins and fanouts)
        // 2. find all edges (returns position in the edgelist)
        // 3. delete those edges from the list
        tempNode = allnodes->DeleteCurrent();
        numgate--;
        delete tempNode;

        if (type == PRIMARY_INPUT) {
            position = FindNode(pilist, temp_name);
            if (position == -1) {
                cout << " Graph::DeleteGate("<< gate_name << ")" << endl;
                cout << " Node is in GateLists but not in PiList !!";
                exit(1);
            }
            pilist->Reset(position);
            tempNode = pilist->DeleteCurrent();
            numpi--;
            delete tempNode;
            return 0;
        }
        else if (type == PRIMARY_OUTPUT) {
            position = FindNode(polist, temp_name);
            if (position == -1) {
                cout << " Graph::DeleteNode("<< gate_name << ")" << endl;
                cout << " Node is in GateLists but not in PoList !!";
                exit(1);
            }
            polist->Reset(position);
            tempNode = polist->DeleteCurrent();
            numpo--;
            delete tempNode;
            return 0;
        }
    }

    return 0;
}

int Graph::FindNode(LinkedList<Node *> *L, char *& item_name)
{
    int item_position = -1;
    lsElem<Node *> *temp_node=NULL;

    if (item_name == NULL) return -1;
    if (L->front->Next != L->rear) {
        //lsElem<Node *> *temp_node = L->front->Next;
        temp_node = L->front->Next;
        //while(temp_node->Next != L->rear) {
        while(temp_node->Next != NULL) {
            item_position++;
            if (strcmp(temp_node->data->name, item_name)==0) {
                return item_position;
            }
            temp_node = temp_node->Next;
        }
    }
    return -1;
}

// Find a node, if not exist create a node and return it's pointer and
// position in allnodes. If it is created, position=-1 is returned.
Node * Graph::FindOrCreateNode(char *& node_name, int &position,
    NodeType ntype, NodeFunc nfunc)
{
    position = -1;
    if ((position = FindNode(allnodes, node_name)) != -1) {
        Node *tempNode;
        allnodes->Reset(position);
        tempNode = allnodes->Data();
        return tempNode;
    }
    else {
        Node *tempNode = new Node;
        tempNode->SetName(node_name);

        if (ntype == PRIMARY_INPUT) {
            tempNode->SetNodeType(ntype);
            tempNode->SetNodeFunc(nfunc);
            InsertPiNode(tempNode);
        }
        else if (ntype == PRIMARY_OUTPUT) {
            tempNode->SetNodeType(ntype);
            tempNode->SetNodeFunc(nfunc);
            InsertPoNode(tempNode);
        }
        else {
            tempNode->SetNodeType(ntype);
            tempNode->SetNodeFunc(nfunc);
            InsertGate(tempNode);
        }
        return tempNode;
    }
}

/* The purpose of this routine is to rearrange fanin and fanouts related
* to FuncType=NODE_UNDEFINED after reading the given logic network.
* This happens after after processing of assignment operation
* like n = $12;..
* Example)
* ---Before using this routine
* 10 : $10 INTERNAL(3) NODE_AND 10 fanin : $9 a fanout : $12
* 11 : b' INTERNAL(3) NODE_INV 11 fanin : b fanout : $12 $16 $22
* 12 : $12 INTERNAL(3) NODE_AND 12 fanin : b' $10 fanout : n
* 13 : n INTERNAL(3) NODE_UNDEFINED 13 fanin : $12 fanout : n' $18
* 14 : n' INTERNAL(3) NODE_INV 14 fanin : n fanout : e0 $24
*
* 18 : $18 INTERNAL(3) NODE_OR 18 fanin : n $17 fanout : n1
* 19 : n1 INTERNAL(3) NODE_UNDEFINED 19 fanin : $18 fanout : n1
*
* 24 : $24 INTERNAL(3) NODE_OR 24 fanin : n' $23 fanout : p
* 25 : p INTERNAL(3) NODE_UNDEFINED 25 fanin : $24 fanout : p'
*
* ---After using this routine
* 10 : $10 INTERNAL(3) NODE_AND 10 fanin : $9 a fanout : n
* 11 : b' INTERNAL(3) NODE_INV 11 fanin : b fanout : n $16 $22
* 12 : n INTERNAL(3) NODE_AND 12 fanin : b' $10 fanout : n' n1
* 13 : n' INTERNAL(3) NODE_INV 14 fanin : n fanout : e0 p
*
* 17 : n1 INTERNAL(3) NODE_OR 18 fanin : n $17
* 22 : p INTERNAL(3) NODE_OR 24 fanin : n' $23 fanout : p'
*/
int Graph::FaninOutRearrange(void)
{
    Node * undefNode, *fNode, *tNode;
    LinkedList<Node *> *undefInList, *undefOutList, *undefInOutList;
    char *undef_name, *tNode_name;
    int index = -1, num_deleted = 0, position = -1, num_fanout;

    int un_pos;
    LinkedList<Node *> *fNodeInList;

```

```

LinkedList<char *> *delList = new LinkedList<char *>;
NodeFunc node_func;

for(allnodes->Reset(); !allnodes->EndOfList(); allnodes->Next()){
    undefNode = allnodes->Data();
    if(undefNode->GetNodeFunc() == NODE_UNDEFINED){
        undef_name = undefNode->GetName();
        delList->InsertRear(undef_name);

        undefInList = undefNode->GetFaninList();
        // Undef_Node always has only one input node.
        if(!undefInList->IsEmpty() && undefInList->ListSize() == 1){
            undefInList->Reset();
            tNode = undefInList->Data(); //Target Node
            tNode->SetName(undef_name); //replace node with undef node.
            tNode_name = tNode->GetName();
        }

        undefInOutList = tNode->GetFanoutList();
        if(!undefInOutList->IsEmpty() && undefInOutList->ListSize() == 1){
            undefInOutList->Reset();
            (void)undefInOutList->DeleteCurrent();
            tNode->SetNumFanout(tNode->GetNumFanout()-1); // numfanout--

            /* The #fanout(s) of the undefined_node can be more
             * than one nodes
             */
            position = -1;
            undefOutList = undefNode->GetFanoutList();
            if((num_fanout = undefOutList->ListSize()) != 0){
                for(undefOutList->Reset(); !undefOutList->EndOfList();
                    undefOutList->Next()){
                    fNode = undefOutList->Data(); // undefNode's fanouts
                    // change fNode(fanout of undefNode)'s fanin
                    fNodeInList = fNode->GetFaninList();
                    // find undef node position and replace it with new in
                    un_pos = FindNode(fNodeInList, tNode_name);
                    fNodeInList->Reset(un_pos);
                    fNodeInList->DeleteCurrent();
                    fNode->SetNumFanin(fNode->GetNumFanin()-1); // numfain--
                    fNode->InsertFanin(tNode);

                    tNode->InsertFanout(fNode);
                }
            }
            num_deleted++;
        }
    }

    // Deleting node in delList from allnodes in Graph
    position = -1;
    for(index = 0; index < num_deleted; index++){
        for(allnodes->Reset(); !allnodes->EndOfList(); allnodes->Next()){
            undefNode = allnodes->Data();
            undefInList = undefNode->GetFaninList();
            node_func = undefNode->GetNodeFunc();
            if((node_func == NODE_UNDEFINED) && (undefInList->ListSize() == 1)){
                (void)allnodes->DeleteCurrent();
                numgate--;
                break;
            }
        }
        delete delList;
        return num_deleted;
    }

    // Get from/to node name of an edge
    char *Graph::GetFromNodeName(Edge *& edgePtr)
    {
        return edgePtr->from->name;
    }

    char *Graph::GetToNodeName(Edge *& edgePtr)
    {
        return edgePtr->to->name;
    }

    // Print all Primary Input Nodes and it's fanin/fanout nodes
    void Graph::PrintPiNodes(void)
    {
        int position = -1;
        int fanin_pos = -1;
        int fanout_pos = -1;

        char *node_name;
        NodeType node_type;
        NodeFunc node_func;

        cout << "Printing Input Lists of Graph !!!" << endl;
        cout << "Number of inputs = " << numpi << endl;

        if (!pelist->IsEmpty()) {
            lsElem<Node *> *tempPI = pelist->front->Next;
            while (tempPI->Next != NULL) {
                position++;
                pelist->Reset(position);
                node_name = pelist->Data()->name;
                node_type = pelist->Data()->GetNodeType();
                node_func = pelist->Data()->GetNodeFunc();
                printf("%s %d %d ", node_name, node_type, node_func);

                if (!pelist->Data()->fanin->IsEmpty()) {
                    lsElem<Node *> *tempFI = pelist->Data()->fanin->front->Next;
                    fanin_pos = -1;
                    cout << "fanin : ";
                    while (tempFI->Next != NULL) {
                        fanin_pos++;
                        pelist->Data()->fanin->Reset(fanin_pos);
                        cout << pelist->Data()->fanin->Data()->name << " ";
                        tempFI = tempFI->Next;
                    }
                }
            }
        }

        // Print all Primary Output Nodes and it's fanin/fanout nodes
        void Graph::PrintPoNodes(void)
        {
            int position = -1;
            int fanin_pos = -1;
            int fanout_pos = -1;

            char *node_name;
            NodeType node_type;
            NodeFunc node_func;

            cout << "Printing Output Lists of Graph !!!" << endl;
            cout << "Number of outputs = " << numpo << endl;

            if (!polist->IsEmpty()) {
                lsElem<Node *> *tempPO = polist->front->Next;
                while (tempPO->Next != NULL) {
                    position++;
                    polist->Reset(position);
                    node_name = polist->Data()->name;
                    node_type = polist->Data()->GetNodeType();
                    node_func = polist->Data()->GetNodeFunc();
                    printf("%s %d %d ", node_name, node_type, node_func);

                    if (!polist->Data()->fanin->IsEmpty()) {
                        lsElem<Node *> *tempFI = polist->Data()->fanin->front->Next;
                        fanin_pos = -1;
                        cout << "fanin : ";
                        while (tempFI->Next != NULL) {
                            fanin_pos++;
                            polist->Data()->fanin->Reset(fanin_pos);
                            cout << polist->Data()->fanin->Data()->name << " ";
                            tempFI = tempFI->Next;
                        }
                    }

                    if (!polist->Data()->fanout->IsEmpty()) { //is_polist_empty
                        lsElem<Node *> *tempFO = polist->Data()->fanout->front->Next;
                        fanout_pos = -1;
                        cout << "fanout : ";
                        while (tempFO->Next != NULL) {
                            fanout_pos++;
                            polist->Data()->fanout->Reset(fanout_pos);
                            cout << polist->Data()->fanout->Data()->name << " ";
                            tempFO = tempFO->Next;
                        }
                    }

                    tempPO = tempPO->Next;
                    cout << endl;
                }
            }

            // Print all Nodes and it's fanin/fanout nodes
            void Graph::PrintAllNodes(void)
            {
                Node *aNode, *inNode, *outNode;
                LinkedList<Node *> *inList, *outList;

                cout << "Printing All Lists of Graph !!!" << endl;
                cout << "Number of allnodes = " << numgate << endl;

                if (!allnodes->IsEmpty()) {
                    for(allnodes->Reset(); !allnodes->EndOfList(); allnodes->Next()){
                        aNode = allnodes->Data();
                        printf("%3d%2d%2d : %6s ", allnodes->position,
                            aNode->GetLevel(), aNode->GetLevel(), \
                                aNode->GetName());
                        Print_Node_Type(allnodes->Data()->type);
                        Print_Node_Func(allnodes->Data()->func);
                        cout << allnodes->Data()->id << " ";

                        inList = aNode->GetFaninList();
                        if (!inList->IsEmpty()) {
                            cout << "fanin : ";
                            for(inList->Reset(); !inList->EndOfList(); inList->Next()){
                                inNode = inList->Data();
                                cout << inNode->GetName() << " ";
                            }
                            cout << "\t";
                        }

                        outList = aNode->GetFanoutList();
                        if (!outList->IsEmpty()) { //is_polist_empty
                            cout << "fanout : ";
                            for(outList->Reset(); !outList->EndOfList(); outList->Next()){
                                outNode = outList->Data();
                                cout << outNode->GetName() << " ";
                            }
                            cout << endl;
                        } // for loop
                    }
                }

                void Graph::Print_Node_Type(int type)

```

```

{
    switch (type) {
        case 0: cout << "UNASSIGNED(0) ";
                break;
        case 1: cout << "RY_INPUT(1) ";
                break;
        case 2: cout << "Y_OUTPUT(2) ";
                break;
        case 3: cout << "INTERNAL(3) ";
                break;
        default: cout << "Invalid Node Type (<<type<<) ";
    }
}

void Graph::Print_Node_Func(int type)
{
    switch (type) {
        case 0: cout << "NODE_UNDEFINED ";
                break;
        case 1: cout << "NODE_PI ";
                break;
        case 2: cout << "NODE_PO ";
                break;
        case 3: cout << "NODE_O ";
                break;
        case 4: cout << "NODE_1 ";
                break;
        case 5: cout << "NODE_BUF ";
                break;
        case 6: cout << "NODE_INV ";
                break;
        case 7: cout << "NODE_AND ";
                break;
        case 8: cout << "NODE_OR ";
                break;
        case 9: cout << "NODE_NAND ";
                break;
        case 10: cout << "NODE_NOR ";
                break;
        case 11: cout << "NODE_CPLX ";
                break;
        case 12: cout << "NODE_PTL1 ";
                break;
        case 13: cout << "NODE_PTL2 ";
                break;
        case 14: cout << "NODE_PTL3 ";
                break;
        default: cout << "Invalid Node Function(<<type<<)";
    }
}

// Print all Nodes and it's fanin/fanout nodes
void Graph::PrintNodesOfList(LinkedList<Node *> *nodeList)
{
    int position = -1;
    int fanin_pos = -1;
    int fanout_pos = -1;
    int p_detail = 1;

    cout << "Printing All Nodes In a Lists !!!!"<<endl;
    cout << "Size of List = "<< nodeList->ListSize() << endl;

    if (!nodeList->isEmpty()) {
        lsElem<Node *> *tempAll = nodeList->front->Next;
        while (tempAll->Next != NULL) {
            position++;
            nodeList->Reset(position);
            printf("%3d : %6s ", position, nodeList->Data()->name);
            if (p_detail == 1) {
                cout << "("<<nodeList->Data()->visited<<") ("<<endl;
                cout <<nodeList->Data()->glevel<<") ";
            }
            Print_Node_Type(nodeList->Data()->type);
            Print_Node_Func(nodeList->Data()->func);

            cout << nodeList->Data()->id << " ";

            if (!nodeList->Data()->fanin->isEmpty()) {
                lsElem<Node *> *tempFI = nodeList->Data()->fanin->front->Next;
                fanin_pos = -1;
                cout << "fanin : ";
                while (tempFI->Next != NULL) {
                    fanin_pos++;
                    nodeList->Data()->fanin->Reset(fanin_pos);
                    if (p_detail == 1) {
                        cout<<nodeList->Data()->fanin->Data()->name << "("<<endl;
                        cout<<nodeList->Data()->fanin->Data()->visited<<") ("<<endl;
                        cout<<nodeList->Data()->fanin->Data()->glevel<<") ";
                    } else {
                        cout<<nodeList->Data()->fanin->Data()->name << " ";
                    }
                    tempFI = tempFI->Next;
                }
            }
            cout << "\t";

            if (!nodeList->Data()->fanout->isEmpty()) { //is_polist_empty
                lsElem<Node *> *tempFO = nodeList->Data()->fanout->front->Next;
                fanout_pos = -1;
                cout << "fanout : ";
                while (tempFO->Next != NULL) {
                    fanout_pos++;
                    nodeList->Data()->fanout->Reset(fanout_pos);
                    if (p_detail == 1) {
                        cout<<nodeList->Data()->fanout->Data()->name << "("<<endl;
                        cout<<nodeList->Data()->fanout->Data()->visited<<") ("<<endl;
                        cout<<nodeList->Data()->fanout->Data()->glevel<<") ";
                    } else {
                        cout<<nodeList->Data()->fanout->Data()->name << " ";
                    }
                    tempFO = tempFO->Next;
                }
            }
            tempAll = tempAll->Next;
        }
        cout<<endl;
    }
}

}
}

#endif

/*****
 * eqn_ior.h
 * Class definition of equation operators, such as +. -. *. (, )...
 *****/
#ifndef EQN_OPR_H
#define EQN_OPR_H

#include "Graph.h"

// list of constants specifying specific error messages
const int OperatorExpected = 0,
        OperandExpected = 1,
        MissingLeftParenthesis = 2,
        MissingRightParenthesis = 3,
        InvalidInput = 4;

// labels designating the parentheses characters
const char leftparenthesis = '(',
        rightparenthesis = ')';

// a class that handles operators on the operator stack
class EqnOperator
{
private:
    // operator and its two precedence values
    char op;
    int inputprecedence;
    int stackprecedence;

public:
    // constructors; includes default constructor and a
    // constructor that initializes the object
    EqnOperator(void);
    EqnOperator(char ch);

    // member functions handling operator on the stack
    int operator>= (EqnOperator a) const;
    //void Evaluate (Stack<float> &OperandStack);
    void Evaluate (Graph &Ckt, Stack<char *> &OperandStack);
    char GetOp(void);
};

// default constructor
EqnOperator::EqnOperator(void)
{
}

// constructor that assigns operator and precedences values
EqnOperator::EqnOperator(char ch)
{
    op = ch; // assign operator
    switch(op)
    {
        // '+' have input/stack precedence 1
        case '+': inputprecedence = 1;
                stackprecedence = 1;
                break;

        // '*' and '/' have input/stack precedence 2
        case '*': inputprecedence = 2;
                stackprecedence = 2;
                break;

        // '(' has input precedence 3, stack precedence -1
        case '(': inputprecedence = 3;
                stackprecedence = -1;
                break;

        // ')' has input/stack precedence 0
        case ')': inputprecedence = 0;
                stackprecedence = 0;
                break;

        // '=' has input precedence 4 / stack precedence -2
        case '=': inputprecedence = 4;
                stackprecedence = -2;
                break;
    }
}

// overload the >= operator by comparing stackprecedence of
// the current object with inputprecedence of a. used when
// reading an operator to determine whether operators
// on the stack should be evaluated before pushing the new
// operator on the stack.
int EqnOperator::operator>= (EqnOperator a) const
{
    return stackprecedence >= a.inputprecedence;
}

// evaluate operator for the current object. First pop two
// operands from the operand stack, then execute operator and
// push the result back onto the operand stack.
void EqnOperator::Evaluate (Graph &Ckt, Stack<char *> &OperandStack)
{
    char * operand1 = OperandStack.Pop(); // get right operand
    char * operand2 = OperandStack.Pop(); // get left operand

    char *temp_str = new char[65];
    char *out_name;
    int position1, position2, position3, size, temp_len;

    Node *oprNode1, *oprNode2, *outputNode;

    // Create new node_name -> must be reconsidered.

```

```

size = Ckt.GetSize();
sprintf(temp_str, "%d", size);
temp_len = strlen(temp_str);

if(temp_len > 64){
    cout<<"Node "<<temp_str<<"'s name is too long.\n";
    cout<<"Program is terminating ..... \n";
    exit(0);
}
out_name = (char *)calloc(temp_len + 1, sizeof(char));
strcpy(out_name, temp_str);

// evaluate operator and push result back on the stack
switch (op) // select operation
{
    case '+':
        //printf("Or operator --> CreateEqnNode(or, opr1, opr2)\n");
        oprNode1 = Ckt.FindOrCreateNode(operand1, position1);
        oprNode2 = Ckt.FindOrCreateNode(operand2, position2);
        outputNode = Ckt.FindOrCreateNode(out_name, position3, INTERNAL,
                                           NODE_OR);
        if (position3 != -1)
            cout << "WARNING !! ("<<out_name <<" ) is already used ..."<<endl;
        // Fanins for OR_Gate
        outputNode->InsertFanin(oprNode1);
        outputNode->InsertFanin(oprNode2);
        // Fanout for each input of the OR_Gate
        //if (position3 == -1) {
        oprNode1->InsertFanout(outputNode);
        oprNode2->InsertFanout(outputNode);
        //}
        OperandStack.Push(out_name);
        break;

    case '*':
        //==printf("And operator -->CreateEqnNode(and, opr1, opr2)\n");
        //search linked list to find out operand1 and operand2 nodes,
        //if they are not exist, insert their name in the list, and
        //assign node id [i] and return output node $ad[i].
        oprNode1 = Ckt.FindOrCreateNode(operand1, position1);
        oprNode2 = Ckt.FindOrCreateNode(operand2, position2);
        outputNode = Ckt.FindOrCreateNode(out_name, position3, INTERNAL,
                                           NODE_AND);
        if (position3 != -1)
            cout << "WARNING !! ("<<out_name <<" ) is already used ..."<<endl;
        // Fanins for OR_Gate
        outputNode->InsertFanin(oprNode1);
        outputNode->InsertFanin(oprNode2);
        // Fanout for each input of the OR_Gate
        oprNode1->InsertFanout(outputNode);
        oprNode2->InsertFanout(outputNode);
        //OperandStack.Push(operand1);
        //OperandStack.Push(out_name);
        break;

    case '=':
        oprNode1 = Ckt.FindOrCreateNode(operand1, position1);
        if(oprNode1->GetNodeType() == UNASSIGNED) {
            oprNode1->SetNodeType(INTERNAL);
        }
        oprNode2 = Ckt.FindOrCreateNode(operand2, position2);
        if(oprNode2->GetNodeType() == UNASSIGNED) {
            oprNode2->SetNodeType(INTERNAL);
        }
        oprNode2->InsertFanin(oprNode1);
        // Insert oprNode2 into fanout_list of oprNode1
        oprNode1->InsertFanout(oprNode2);
        break;
}
delete [] temp_str;
}

// return operator associated with current object
char EqnOperator::GetOp(void)
{
    return op;
}
#endif //EQN_OPR_H

/*****
 * eqn_stack.h
 * Class definition of the fixed-length array stack
 *****/
#ifndef TEMPLATE_STACK_CLASS
#define TEMPLATE_STACK_CLASS

#include <fstream.h>
#include <stdlib.h>

const int MaxStackSize = 50;

template <class T>
class Stack
{
private:
    // private data members. stack array and top
    T stacklist[MaxStackSize];
    int top;
public:
    // constructor
    Stack (void); // initialize top

    // stack modification operations
    void Push (const T& item);
    T Pop (void);
    void ClearStack(void);

    // stack access
    T Peek (void) const;

    // stack test methods
    int StackEmpty(void) const;
    int StackFull(void) const;
    int GetTop(void) const;
};

// initialize stack top
template <class T>
Stack<T>::Stack (void): top(-1)
{
}

// push item on the the stack
template <class T>
void Stack<T>::Push (const T& item)
{
    if (top == MaxStackSize-1)
    {
        cerr << "Eqn. Stack overflow!" << endl;
        exit(1);
    }

    // increment top and copy item to stacklist
    top++;
    stacklist[top] = item;
}

// pop the stack and return the top element
template <class T>
T Stack<T>::Pop (void)
{
    T temp;

    // if stack is empty, terminate the program
    if (top == -1) {
        cerr << "Attempt to pop an empty eqn. stack!" << endl;
        exit(1);
    }

    // record the top element
    temp = stacklist[top];

    // Free the memory
    top--;
    return temp;
}

// return the value at the top of the stack
template <class T>
T Stack<T>::Peek (void) const
{
    // if the stack is empty, terminate the program
    if (top == -1)
    {
        cerr << "Attempt to peek at an empty stack!" << endl;
        exit(1);
    }
    return stacklist[top];
}

// test for an empty stack
template <class T>
int Stack<T>::StackEmpty(void) const
{
    // return the logical value top == -1
    return top == -1;
}

// test for a full stack
template <class T>
int Stack<T>::StackFull(void) const
{
    // return logical condition top == MaxStackSize-1
    return top == MaxStackSize-1;
}

// test for a full stack
template <class T>
int Stack<T>::GetTop(void) const
{
    return top;
}

// clear stack
template <class T>
void Stack<T>::ClearStack(void)
{
    // reset top to -1
    top = -1;
}
#endif //TEMPLATE_STACK_CLASS

/*****
 * Queue.h
 * Class definition of Queue using linked list
 *****/

```

```

*****
#ifndef QUEUE_CLASS
#define QUEUE_CLASS

#include "LinkedList.h"

template <class T>
class Queue
{
private:
    // A linked list object to hold the queue items
    LinkedList<T> qList;

public:
    // constructors
    Queue(void);
    Queue(void);

    // queue modification methods
    void QInsert(const T& elt);
    T DeQueue(void);

    // Queue access
    T QFront(void);

    // queue List
    LinkedList<T> *QList(void);

    int QLength(void) const;
    int QEmpty(void) const;
    void QClear(void);

    friend class Graph;
};

// Default constructor
template <class T>
Queue<T>::Queue(void) {}

template <class T>
Queue<T>::~Queue(void)
{
    int n, listSize;

    listSize = qList.ListSize();
    for(n=0; n<listSize; n++){
        delete qList.DeleteCurrentElem();
    }
}

// LinkedList method InsertRear() inserts item at rear
template <class T>
void Queue<T>::QInsert(const T& elt)
{
    qList.InsertRear(elt);
}

// LinkedList method DeleteFront() removes item from front
template <class T>
T Queue<T>::DeQueue(void)
{
    // Test for an empty queue and terminate if true
    if(qList.IsEmpty()){
        cerr << "Calling DeQueue() for an empty queue!" << endl;
        exit(1);
    }
    return qList.DeleteFront();
}

// returns the data value from the first item in the queue
template <class T>
T Queue<T>::QFront(void)
{
    // Test for an empty queue and terminate if true
    if(qList.IsEmpty()){
        cerr << "Calling QFront() for an empty queue!" << endl;
        exit(1);
    }

    // reset to the front of the queue and return data
    qList.Reset();
    return qList.Data();
}

template <class T>
LinkedList<T> * Queue<T>::QList(void)
{
    return &qList;
}

template <class T>
int Queue<T>::QLength(void) const
{
    return qList.ListSize();
}

template <class T>
int Queue<T>::QEmpty(void) const
{
    return qList.IsEmpty();
}

template <class T>
void Queue<T>::QClear(void)
{
    qList.ClearList();
}
#endif

*****

```

```

* Stack.h
* Class definition of Stack using linked list
*****
#ifndef LStack_CLASS
#define LStack_CLASS

#include "LinkedList.h"

template <class T>
class LStack
{
private:
    // A linked list object to hold the queue items
    LinkedList<T> stList;

public:
    // constructors
    LStack(void);
    LStack(void);

    // queue modification methods
    void Push(const T& elt);
    T Pop(void);

    // stack test methods
    int Size(void) const;
    int Empty(void) const;

    // Stack Clear
    void Clear(void);

    friend class Graph;
};

// Default constructor
template <class T>
LStack<T>::LStack(void) {}

// Default Destructor
template <class T>
LStack<T>::~LStack(void)
{
    Clear();
}

// LinkedList method InsertFront() inserts item at rear
template <class T>
void LStack<T>::Push(const T& elt)
{
    stList.InsertFront(elt);
}

// LinkedList method DeleteFront() removes item from front
template <class T>
T LStack<T>::Pop(void)
{
    // Test for an empty queue and terminate if true
    if(stList.IsEmpty()){
        cerr << "Calling Pop() for an empty stack!" << endl;
        exit(1);
    }
    return stList.DeleteFront();
}

template <class T>
int LStack<T>::Size(void) const
{
    return stList.ListSize();
}

template <class T>
int LStack<T>::Empty(void) const
{
    return stList.IsEmpty();
}

template <class T>
void LStack<T>::Clear(void)
{
    stList.ClearList();
}
#endif

/*****
* LibCell_new.h
* Class definition of a cell in the cell library
*****
#ifndef LIBCELL_CLASS
#define LIBCELL_CLASS

#include <iostream.h>
#include <fstream.h>
#include "Node.h"
#include "IsElem.h"
#include "LinkedList.h"
#include "Graph.h"
#include "Pininfo_new.h"

// Definitoin of Cell class
class LibCell
{
private:
    char * cell_name;
    CellType cell_type;
    Graph *cGraph;
    LinkedList<char *> *rpn; // reverse polish notation of cell func
    LinkedList<Pin *> *pinlist; // Lists of all input pins
    int num_input;
    float cell_area;

public:
    LibCell();
    LibCell(char *k name);
    ~LibCell(void); // Destructor
}

```

```

void SetName(char *&name);
char *GetName(void);
int GetNumInput(void) { return num_input; }
void SetCellType(CellType ctype);
CellType GetCellType(void);

void SetArea(float cell_area);
float GetArea(void);

void InsertPin(Pin *pin);
int FindPin(Pin *&pin);
int FindPin(LinkedList<Pin *> *L, char *&pin_name);
Pin *FindOrCreatePin(char *&pin_name, int &position, Strap stype=NONE);

Graph *GetCellGraph(void);
LinkedList<char *> *GetFunc(void);
LinkedList<Pin *> *GetPinList(void);
};

// Constructor
LibCell::LibCell()
{
    cell_name = (char *)calloc(2, sizeof(char));
    cell_type = UNKNOWN;
    cGraph = new Graph;
    rpn = new LinkedList<char *>;
    pinlist = new LinkedList<Pin *>;
    num_input = 0;
    cell_area = 0.0;
};

// Destructor
LibCell::~LibCell()
{
    free(cell_name);
    delete cGraph;
    delete rpn;
    delete pinlist;
};

void LibCell::SetName(char *&name)
{
    int cname_len, new_len;
    cname_len = strlen(cell_name);
    new_len = strlen(name);

    if(cname_len != new_len || strcmp(cell_name, name)!=0){
        cell_name = (char *)realloc(cell_name, new_len+2);
        strcpy(cell_name, name);
    }
    strcpy(cell_name, name);
}

char * LibCell::GetName(void) { return cell_name; }

void LibCell::SetCellType(CellType ctype)
{
    cell_type = ctype;
}

CellType LibCell::GetCellType(void) {return cell_type;}

void LibCell::SetArea(float area) {cell_area = area;}
float LibCell::GetArea(void) {return cell_area;}

// Insert a Pin into pinlist and set its node id.
void LibCell::InsertPin(Pin * pin)
{
    pin->SetPinId(num_input);
    pinlist->InsertRear(pin);
    num_input++;
}

// Find a Pin using Pin pointer by searching pinlist
int LibCell::FindPin(Pin *&pin)
{
    int item_position = -1;
    char *target_pin = pin->GetPinName();
    char *search_pin;
    Pin *tmpPin;

    if(pinlist->front->Next != pinlist->rear){
        for(pinlist->Reset(); !pinlist->EndOfList(); pinlist->Next()){
            item_position++;
            tmpPin = pinlist->Data();
            search_pin = tmpPin->GetPinName();
            if(strcmp(search_pin, target_pin)==0){
                return item_position;
            }
        }
    }
    return (-1);
}

// Find a Pin using "pin_name" by searching in rpn List.
int LibCell::FindPin(LinkedList<Pin *> *L, char *&pin_name)
{
    int item_position = -1;
    char *target_pin = pin_name;
    char *search_pin;
    Pin *tmpPin;

    if(L->front->Next != L->rear){
        for(L->Reset(); !L->EndOfList(); L->Next()){
            item_position++;
            tmpPin = L->Data();
            search_pin = tmpPin->GetPinName();
            if(strcmp(search_pin, target_pin)==0){
                return item_position;
            }
        }
    }
}

}

return (-1);
}

Pin * LibCell::FindOrCreatePin(char *&pin_name, int &position, Strap stype)
{
    position = -1;
    if((position = FindPin(pinlist, pin_name)) != -1){
        pinlist->Reset(position);
        return pinlist->Data();
    }
    else {
        Pin *tmpPin = new Pin;
        tmpPin->SetPinName(pin_name);
        tmpPin->SetStrap(stype);
        InsertPin(tmpPin);
        return tmpPin;
    }
}

// returns Graph class pointer for a cell
Graph * LibCell::GetCellGraph(void)
{
    return cGraph;
}

// returns rpn list pointer
LinkedList<char *> * LibCell::GetFunc(void)
{
    return rpn;
}

// returns Pin list pointer
LinkedList<Pin *> * LibCell::GetPinList(void)
{
    return pinlist;
}
#endif

/*****
 * Read_eqn.cc
 * Reads SLIF equations which is equivalent to netlist
 *****/
#include <fstream.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>
//STL
#include <string>
#include "eqn_stack.h"
#include "eqn_opr.h"
#include "LinkedList.h"
#include "Stack.h"
#include "Node.h"
#include "Graph.h"

// checks if character is an operator or parentheses
int isoperator(char ch)
{
    if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '(' || ch == ')'){
        return 1;
    }
    else {
        return 0;
    }
}

// checks if character is a whitespace character
int iswhitespace(char ch)
{
    //if (ch == ' ' || ch == '\t' || ch == '\n')
    if (ch == '\n')
        return 1;
    else
        return 0;
}

// this routine eat consecutive new line characters
int new_line_eatup(ifstream &fin)
{
    char ch;
    // new line char eatup()
    while ((ch=fin.get()) == '\n'){
        continue;
    }
    fin.putback(ch);
    return 0;
}

// this routine eatup a line until it sees a line char.
int rdeqnLineEatup(ifstream &fin)
{
    char ch;
    // new line char eatup()
    while ((ch=fin.get()) != '\n'){
        continue;
    }
    //fin.putback(ch);
    return 0;
}

// error handling function
void error(int n)
{
    // table gives the different error messages
    static char *errmsgs[] = {
        "Operator expected",
        "Operand expected",
        "Missing left parenthesis",
    }
}

```

```

        "Missing right parenthesis",
        "Invalid input"
    );
    // the parameter n is an error message index
    // print the message and terminate the program
    cerr << errmsgs[n] << endl;
    exit(1);
}

// Copy a character string from source to destination
char *rdeqnCopyCharString(ifstream &fin, const char *src_str)
{
    int src_len;
    char ch, *dest_str;

    src_len = strlen(src_str);
    ch = fin.peek();
    if(src_len > 64){
        dest_str = (char *)calloc(65, sizeof(char));
        strncpy(dest_str, src_str, 64);
        if(src_str[src_len-1] == ';'){
            char *tmp_str = new char[src_len-1];
            strcpy(tmp_str, src_str);
            tmp_str[src_len-1] = '\0';
            cout << "String \"<<tmp_str<<\" has more than 64 characters.\n";
            cout << "It is truncated to 64 characters....\n\n";
            free(tmp_str);
        } else {
            cout << "String \"<<src_str<<\" has more than 64 characters.\n";
            cout << "It is truncated to 64 characters....\n";
        }
    } else if(ch == '\t' || ch == ' '){
        dest_str = (char *)calloc(src_len + 1, sizeof(char));
        strcpy(dest_str, src_str);
    } else { // the last char is ';'
        dest_str = (char *)calloc(src_len + 1, sizeof(char));
        strcpy(dest_str, src_str);
        dest_str[src_len-1] = '\0';
    }

    return dest_str;
}

void rdeqnSetGraphName(Graph *eGraph, ifstream &fin, const char *token)
{
    char *temp_str, ch, last_char;
    temp_str = rdeqnCopyCharString(fin, token);
    eGraph->SetName(temp_str);
    last_char = token[strlen(token)-1];
    if(last_char != ';'){
        while((ch=fin.get()) != ';'){
            if(!((ch == '\t' || ch == ' ' || ch == '\n' || ch == ';'))){
                cout << "Module name is not terminated with \t, \n, \n";
                cout << "Program is halted.....\n";
                exit(0);
            }
            continue;
        }
    }
    free(temp_str);
}

// Reads words separated by ' ', and create input nodes for a Graph
//Node *rdeqnCreateInputNodes(Graph *eGraph, ifstream &fin, char *token)
void rdeqnCreateInputNodes(Graph *eGraph, ifstream &fin, char *token)
{
    Node *tmpNode; char *temp;
    int pos, token_len;

    while(1){
        fin >> token;
        token_len = strlen(token);
        temp = (char *)calloc(2, sizeof(char));
        if (token_len != 1 && token_len > 0){
            if (token[token_len-1] == ';'){
                temp = (char *)realloc(temp, token_len+2);
                strcpy(temp, token);
                temp[token_len-1] = '\0';
            } else {
                temp = (char *)realloc(temp, token_len+3);
                strcpy(temp, token);
            }
        } else {
            strcpy(temp, token);
            if (token[token_len-1] == ';'){
                break;
            }
        }
        pos = -1;
        tmpNode = eGraph->FindOrCreateNode(temp, pos, PRIMARY_INPUT, NODE_PI);
        if(pos != -1){
            cout << "Input node "<<temp<<" has already been in the Graph.\n";
            cout <<temp<<" is a duplicated input node.\n";
            cout << "Program is halted....\n";
            exit(0);
        }
        free(temp);
        if (token[token_len-1] == ';'){
            break;
        }
    }
}

// Reads words separated by ' ', and create input nodes for a Graph
//Node *rdeqnCreateOutputNodes(Graph *eGraph, ifstream &fin, char *token)
void rdeqnCreateOutputNodes(Graph *eGraph, ifstream &fin, char *token)
{
    Node *tmpNode; char *temp=NULL;
    int pos, token_len;

    while(1){
        fin >> token;
        token_len = strlen(token);
        temp = (char *)calloc(2, sizeof(char));
        if (token_len != 1 && token_len > 0){
            if (token[token_len-1] == ';'){
                temp = (char *)realloc(temp, token_len+2);
                strcpy(temp, token);
                temp[token_len-1] = '\0';
            } else {
                temp = (char *)realloc(temp, token_len+3);
                strcpy(temp, token);
            }
        } else {
            strcpy(temp, token);
            if (token[token_len-1] == ';'){
                break;
            }
        }
        pos = -1;
        tmpNode = eGraph->FindOrCreateNode(temp, pos, PRIMARY_OUTPUT, NODE_PO);
        if(pos != -1){
            cout << "Output node '<<temp<<' has already been in the Graph.\n";
            cout << "Program is halted....\n";
            exit(0);
        }
        free(temp);
        if (token[token_len-1] == ';'){
            break;
        }
    }
}

void rdeqnProcessOperand(Stack<char *> &OperandStack, ifstream &fin, int &rank)
{
    char ch, *temp_name;
    char *node_name;
    int name_len, i = 0;

    // ';': the end of a statement (eqn, io def, comment)
    temp_name = (char *)calloc(65, sizeof(char));
    while ((ch=fin.peek())) {
        fin.get(temp_name[i++]);
        ch=fin.peek();
        if (isoperator(ch) || (ch == ';') || (ch == ')') || (ch == '\')){
            break;
        }
    }

    if(i>63){
        cout << "Node name "<<temp_name<<" is too long (>64 chars).\n";
        cout << "Program is terminating.....\n";
        free(temp_name);
        exit(0);
    } else {
        name_len = strlen(temp_name);
        node_name = (char *)calloc(name_len + 1, sizeof(char));
        strcpy(node_name, temp_name);
        free(temp_name);
    }

#ifdef READ_EQN_DEBUG
    cout << "node name = "<< node_name ;
    cout << " rank = "<<rank <<endl;
#endif

    // rank of operand is 1. accumulated rank must be 1
    rank++;

#ifdef READ_EQN_DEBUG
    cout << "node name = "<<node_name<<" rank = "<<rank<<endl;
#endif

    if (rank > 1) error(OperatorExpected);

    // push the operand on the operand stack
    OperandStack.Push(node_name);
}

// Pop-up an operand from OperandStack one time and create a new node
// by concatenating "\n", and put the new node-name into OperandStack
void rdeqnProcessInvOperator(Graph *eGraph, Stack<char *> &OperandStack)
{
    Node *tmpNode, *tmpNode1;
    char *node_name, *temp_name;
    int name_len, position, position1;

    node_name = OperandStack.Pop();

    tmpNode = eGraph->FindOrCreateNode(node_name, position);
    if (tmpNode->GetNodeType() == UNASSIGNED) {
        tmpNode->SetNodeType(INTERNAL);
    }

    name_len = strlen(node_name);
    if(name_len > 64){
        cout << "Node "<<temp_name<<"'s name is too long.\n";
        cout << "Program is terminating ..... \n";
        exit(0);
    }

    temp_name = (char *)calloc(name_len + 2, sizeof(char));
    strcpy(temp_name, node_name);
    strcat(temp_name, "\n\0");
}

```

```

tmpNode1 = eGraph->FindOrCreateNode(tmp_name, position1);
if (tmpNode1->GetNodeType() == UNASSIGNED) {
    tmpNode1->SetNodeType(INTERNAL);
}
if (tmpNode1->GetNodeFunc() != NODE_INV) {
    tmpNode1->SetNodeFunc(NODE_INV);
}
// Insert tmpNode1 into the fanin_list of tmpNode1
if (position1 == -1) {
    tmpNode1->InsertFanin(tmpNode);
}
// Insert tmpNode1 into the fanout_list of tmpNode1
if (position1 == -1) {
    tmpNode->InsertFanout(tmpNode1);
}
free(tmp_name);
free(node_name); // (I) when a node is created, SetName(char *) allocate mem
node_name = tmpNode1->GetName();
OperandStack.Push(node_name);
}

// This routine is activated by looking ' '.
void rdeqnProcessAOperator(Graph *eGraph, Stack<EqnOperator> &OperatorStack,
    Stack<char *> &OperandStack,
    ifstream &fin, char &ch, int &rank)
{
    char tmp_ch;
    EqnOperator opr1, opr2;
    if (ch == ' ') {
        tmp_ch = ch;
        while (fin.get(ch)) {
            if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
                tmp_ch = ch;
            }
            if (!isoperator(ch) && ch != '\n') {
                fin.putback(ch); // to process later
                ch = tmp_ch;
                break;
            }
            else if (ch == '(' || ch == ')') {
                if (ch == '(') {
                    fin.putback(ch);
                }
                ch = tmp_ch;
                break;
            }
        }
    }
    // if (ch != ' ') {
    //     rank of each operator other than ' ' is -1.
    //     accumulated rank should be 0
    //     if (ch != '(') // rank of ' ' is 0
    //         rank--;
    //     if (rank < 0) {
    //         cout << "Operator Processing ... \n";
    //         error(OperandExpected);
    //     }
    // }
    if (ch != ' ') {
        // Build a MathOperator object holding current operator.
        // Pop the operator stack and evaluate as long as the operator on the
        // top of the stack has a precedence >= that of the current operator.
        // Push the current operator on the operator stack
        opr1 = EqnOperator(ch);
        while (!OperatorStack.StackEmpty() &&
            (opr2 = OperatorStack.Peek()) >= opr1) {
            opr2 = OperatorStack.Pop();
            opr2.Evaluate(*eGraph, OperandStack);
        }
        OperatorStack.Push(opr1);
    }
}

// Check the circuit after equation reading if there are any nodes whose
// function are NODE_UNDEFINED due to intermediate node in the equation.
// If any, print the undefined nodes and stop running.
void rdeqnCheckUndefinedNodes(Graph *eGraph)
{
    LinkedList<Node *> *gateList;
    LinkedList<Node *> *undefList = new LinkedList<Node *>;
    Node *curNode;
    gateList = eGraph->GetGateList();
    for (gateList->Reset(); !gateList->EndOfList(); gateList->Next()) {
        curNode = gateList->Data();
        if (curNode->GetNodeFunc() == NODE_UNDEFINED) {
            undefList->InsertRear(curNode);
        }
    }
    if (undefList->ListSize() > 0) {
        cout << "\n This circuit has UNDEFINED nodes..... \n";
        cout << " Please check the input file, and remove the following "
            << "intermediate nodes(s)..... \n";
        (void)eGraph->PrintNodesOfList(undefList);
        delete undefList;
        exit(1);
    }
    delete undefList;
}

void rdeqnFaninFanoutNodeCheck(Graph *eGraph)
{
    Node *curNode;
    LinkedList<Node *> *gateList, *finList, *foutList;
    int error = 0;
    gateList = eGraph->GetGateList();
    for (gateList->Reset(); !gateList->EndOfList(); gateList->Next()) {
        curNode = gateList->Data();
        finList = curNode->GetFaninList();
        foutList = curNode->GetFanoutList();
        if (finList->IsEmpty() && (curNode->GetNodeType() == INTERNAL ||
            curNode->GetNodeType() == PRIMARY_OUTPUT)) {
            error = 1;
            cout << "Error :: Node "<< curNode->GetName()
                << " is not driven by any nodes.\n";
        }
        if (foutList->IsEmpty()) {
            if (curNode->GetNodeType() == INTERNAL) {
                error = 1;
                cout << "Error :: Node "<< curNode->GetName()
                    << " does not drive any nodes.\n";
            }
            else if (curNode->GetNodeType() == PRIMARY_INPUT) {
                cout << "Warning :: Input node "<< curNode->GetName()
                    << " does not drive any nodes.\n";
            }
        }
    }
    if (error) exit(0);
}

/* The purpose of this routine is to rearrange fanin and fanouts related
* to FuncType=NODE_UNDEFINED after reading the given logic network.
* This happens after after processing of assignment operation
* like n = $12;.
* Example)
* -----Before using this routine
* 10 : $10 INTERNAL(3) NODE_AND 10 fanin : $9 a fanout : $12
* 11 : b' INTERNAL(3) NODE_INV 11 fanin : b fanout : $12 $16 $22
* 12 : $12 INTERNAL(3) NODE_AND 12 fanin : b' $10 fanout : n
* 13 : n INTERNAL(3) NODE_UNDEFINED 13 fanin : $12 fanout : n' $18
* 14 : n' INTERNAL(3) NODE_INV 14 fanin : n fanout : e0 $24
* 18 : $18 INTERNAL(3) NODE_OR 18 fanin : n $17 fanout : n1
* 19 : n1 INTERNAL(3) NODE_UNDEFINED 19 fanin : $18
* 24 : $24 INTERNAL(3) NODE_OR 24 fanin : n' $23 fanout : p
* 25 : p INTERNAL(3) NODE_UNDEFINED 25 fanin : $24 fanout : p'
* -----After using this routine
* 10 : $10 INTERNAL(3) NODE_AND 10 fanin : $9 a fanout : n
* 11 : b' INTERNAL(3) NODE_INV 11 fanin : b fanout : n $16 $22
* 12 : n INTERNAL(3) NODE_AND 12 fanin : b' $10 fanout : n' $18
* 13 : n' INTERNAL(3) NODE_INV 14 fanin : n fanout : e0 p
* 17 : n1 INTERNAL(3) NODE_OR 18 fanin : n $17
* 22 : p INTERNAL(3) NODE_OR 24 fanin : n' $23 fanout : p'
*/
// Finds undefined nodes and rearrange the surrounding node's fanin/fanouts
int rdeqnFindUndefinedNodeAndRearrange(Graph *eGraph)
{
    Node *undefNode, *fNode, *tNode, *undefFoNode;
    Node *tInNode;
    LinkedList<Node *> *undefInList, *undefOutList, *undefInOutList;
    LinkedList<Node *> *tNodeInList;
    char *undef_name, *tNode_name, *tInNode_name;
    int index = -1, num_deleted = 0, position = -1, num_fanout;
    int undef_len;
    int un_pos, num_gate;
    LinkedList<Node *> *fNodeInList, *gateList;
    LinkedList<char *> *delList = new LinkedList<char *>;
    NodeFunc node_func;
    gateList = eGraph->GetGateList();
    (void)rdeqnFaninFanoutNodeCheck(eGraph);
    // Fanin Fanout Rearrange for the node (NODE_UNDEFINED)
    for (gateList->Reset(); !gateList->EndOfList(); gateList->Next()) {
        undefNode = gateList->Data();
        if (undefNode->GetNodeFunc() != NODE_UNDEFINED) {
            continue;
        }
        else {
            undef_name = undefNode->GetName();
            delList->InsertRear(undef_name);
            undefInList = undefNode->GetFaninList();
            undefOutList = undefNode->GetFanoutList();
            // Undef_Node always has only one input node.
            if (!undefInList->IsEmpty() && undefInList->ListSize() == 1) {
                undefInList->Reset();
                tNode = undefInList->Data(); // Target Node
                tNode->SetName(undef_name); // replace node with undef node.
                tNode_name = tNode->GetName();
            }
            // if undef_node's fanout is neither a mf nor NODE_INV
            // 041001
            if (undefOutList->ListSize() == 1) {
                undefFoNode = undefOutList->Data(0);
                node_func = undefFoNode->GetNodeFunc();
                if (node_func != NODE_INV && tNode->GetNodeFunc() == NODE_INV) {
                    tNodeInList = tNode->GetFaninList();
                    tInNode = tNodeInList->Data(0);
                    tInNode_name = tInNode->GetName();
                    undef_len = strlen(undef_name);
                    if (undef_name[undef_len-1] == '\n') {
                        tInNode_name = (char *)calloc(undef_len+1, sizeof(char));
                        strcpy(tInNode_name, undef_name);
                    }
                }
            }
        }
    }
}

```

```

        tInNodeName[undef_len-1] = '\0';
        tInNode->SetName(tInNodeName);
        free(tInNodeName);
    } else {
        tInNodeName = (char *)calloc(undef_len+2, sizeof(char));
        strcpy(tInNodeName, undef_name);
        strcat(tInNodeName, "\\0");
        tInNode->SetName(tInNodeName);
        free(tInNodeName);
    }
} //041001

// Undef_node's input's output nodeList update
undefInOutList = tNode->GetFanoutList();
if(!undefInOutList->isEmpty() && undefInOutList->ListSize()==1){
    undefInOutList->Reset();
    (void)undefInOutList->DeleteCurrentElem();
    tNode->SetNumFanout(tNode->GetNumFanout()-1); // numfanout--

    /* The #fanout(s) of the undefined_node can be more
    * than one nodes
    */
    position = -1;
    if((num_fanout = undefOutList->ListSize())!=0){
        for(undefOutList->Reset(); !undefOutList->EndOfList();
            undefOutList->Next()){
            fNode = undefOutList->Data(); // undefNode's fanouts
            // change fNode(fanout of undef_node)'s fanin
            fNodeInList = fNode->GetFaninList();
            // find undef_node position and replace it with new in
            un_pos = eGraph->FindNode(fNodeInList, tNode_name);
            if(un_pos < 0) continue;
            fNodeInList->Reset(un_pos);
            fNodeInList->DeleteCurrentElem();
            fNode->SetNumFanin(fNode->GetNumFanin()-1); // numfanin--
            fNode->InsertFanin(tNode);
            tNode->InsertFanout(fNode);
        }
    }
}

eGraph->SetNumGate(eGraph->GetNumGate()-1);
num_deleted++;
}

// Deleting node in delList from allnodes in Graph
position = -1;
for(index = 0; index < num_deleted; index++){
    for(gateList->Reset(); !gateList->EndOfList(); gateList->Next()){
        undefNode = gateList->Data();
        undefInList = undefNode->GetFaninList();
        node_func = undefNode->GetNodeFunc();
        if((node_func == NODE_UNDEFINED) && (undefInList->ListSize()==1)){
            (void)gateList->DeleteCurrentElem();
            num_gate = eGraph->GetNumGate();
            eGraph->SetNumGate(num_gate--);
            break;
        }
    }
}
delete delList;
return num_deleted;
}

// Retrieve a node's inverted node from gateList of eGraph
Node * rdeqnFindReversedPolarityNode(Graph *eGraph, Node *targetNode)
{
    LinkedList<Node> * gateList;
    Node *invNode;
    char *targetName, *tmpName;
    int tmp_name_len, num_gate, pos = -1;
    NodeFunc targetFunc;

    tmpName = targetNode->GetName();
    tmp_name_len = strlen(tmpName);
    targetFunc = targetNode->GetNodeFunc();

    if(tmpName[tmp_name_len-1] == '~'){
        targetName = (char *)calloc(tmp_name_len+1, sizeof(char));
        //strncpy(targetName, tmpName, strlen(tmpName)-1);
        strcpy(targetName, tmpName);
        targetName[tmp_name_len-1] = '\0';
    } else {
        targetName = (char *)calloc(tmp_name_len+2, sizeof(char));
        strcpy(targetName, tmpName);
        strcat(targetName, "\\0");
    }

    gateList = eGraph->GetGateList();
    num_gate = gateList->ListSize();
    pos = eGraph->FindNode(gateList, targetName);

    if(pos != -1 && pos < num_gate){
        gateList->Reset(pos);
        invNode = gateList->Data();
    } else {
        if(targetFunc == NODE_PO && targetNode->GetNumFanout()>0){
            invNode = NULL;
        } else if(targetFunc == NODE_PO && targetNode->GetNumFanout()==0){
            invNode = targetNode;
        } else {
            invNode = NULL;
        }
    }

    free(targetName);
    return invNode;
}

void rdeqnChangeNodeFunction(Node *curNode)
{
    NodeFunc curFunc;

    curFunc = curNode->GetNodeFunc();
    if(curFunc==NODE_OR){
        curNode->SetNodeFunc(NODE_NOR);
    } else if(curFunc==NODE_AND){
        curNode->SetNodeFunc(NODE_NAND);
    } else {
        cout<<"Error : In rdeqnChangeNodeFunction(), current node func is not"
        <<" OR or AND.\n Current node func is "<<(int)curNode->GetNodeFunc()
        <<"\n";
        exit(0);
    }
}

void rdeqnConvertList2Queue(Queue<Node> * *foQ, LinkedList<Node> * *inList)
{
    Node *curNode;

    for(inList->Reset(); !inList->EndOfList(); inList->Next()){
        curNode = inList->Data();
        foQ->QInsert(curNode);
    }

    // Create a INV node and rearrange curNode and newNode's fanin/fanout
    Node *rdeqnCreateINVNode(Graph *eGraph, Node *curNode, char *new_name,
        int dummy = 0)
    {
        Node *newNode = new Node, *foNode;
        LinkedList<Node> * *cur_foList, *new_foList;
        //LinkedList<Node> * *cur_foList, *new_foList;
        //int node_id, cur_foList_Size, i;
        int i;
        Queue<Node> * *foQ = new Queue<Node> *;
        char *fo_name;
        NodeFunc curNodeFunc;

        // 1) Create a new INV node
        newNode->SetName(new_name);
        newNode->SetNodeId(eGraph->GetSize());
        eGraph->SetSize(eGraph->GetSize()+1);

        if(dummy){
            newNode->SetNodeType(DUMMY); //10-01-00
        } else {
            newNode->SetNodeType(INTERNAL);
        }

        newNode->SetNodeFunc(NODE_INV); // node_id, name, fanin/out
        newNode->InsertFanin(curNode);

        // 2) Rearrange curNode and newNode's fanin/fanout
        curNodeFunc = curNode->GetNodeFunc();
        if(curNodeFunc != NODE_PO && curNodeFunc != NODE_PI){
            new_foList = newNode->GetFanoutList();
            cur_foList = curNode->GetFanoutList();

            (void)rdeqnConvertList2Queue(foQ, cur_foList);
            while(!foQ->QEmpty()){
                foNode = foQ->Dequeue();
                fo_name = foNode->GetName();
                newNode->InsertFanout(foNode);
                if(!eGraph->FindNode(cur_foList, fo_name)!=-1){
                    cur_foList->Reset(i);
                    cur_foList->DeleteCurrent();
                    curNode->SetNumFanout(curNode->GetNumFanout()-1); //numfanout--
                }
                curNode->InsertFanout(newNode);
            }
        } else if(curNodeFunc == NODE_PO || curNodeFunc==NODE_PI){
            curNode->InsertFanout(newNode);
        }

        delete foQ;
        return newNode;
    }

    // Change curNode Func (ex: OR -> NOR), create newNode, and add to gateList
    void rdeqnReverseNodeFuncAndInsertInverter(Graph *eGraph, Node *curNode)
    {
        //char *new_name, *cur_name, *cur_fo_name;
        char *new_name, *cur_name;
        char *tmp_char;
        int cur_name_len, pos = -1;
        LinkedList<Node> * *gateList, *cur_foList, *fiList;
        Queue<Node> * *foQ = new Queue<Node> *;
        Node *newNode, *foNode;
        NodeFunc curFunc;

        cur_name = curNode->GetName();
        cur_name_len = strlen(cur_name);

        curFunc = curNode->GetNodeFunc();
        if(curFunc != NODE_PO && curFunc != NODE_PI){
            cur_foList = curNode->GetFanoutList();
            // delete curNode from curNode fanout node's fanin node;
            for(cur_foList->Reset(); !cur_foList->EndOfList(); cur_foList->Next()){
                foNode = cur_foList->Data();
                fiList = foNode->GetFaninList();
                pos = eGraph->FindNode(fiList, cur_name);
                if(pos != -1){
                    fiList->Reset(pos);
                    fiList->DeleteCurrent(); // delete curNode from cur fo's fanin
                    foNode->SetNumFanin(foNode->GetNumFanin()-1); // numfanin--
                    foQ->QInsert(foNode);
                }
            }

            // Deleting current node's fanout list (in case of more than 1 fanout)
            // Happens in "cm162a", mod: 10-03-00
            //if(cur_foList->ListSize()>0){
            //for_each(cur_foList){

```

```

// cur_foList->DeleteCurrent();
// curNode->SetNumFanout(curNode->GetNumFanout()-1); // numfanout--
// }
// new name for INV node (new_name = cur_name)
new_name = (char *)calloc(cur_name_len+2, sizeof(char));
strcpy(new_name, cur_name);

// new name of curNode (cur_name = cur_name')
if(cur_name[cur_name_len-1]!='\0'){
    cur_name = (char *)realloc(cur_name, (cur_name_len+2));
    strcat(cur_name, "\0");
} else {
    tmp_char = (char *)calloc(cur_name_len+1, sizeof(char));
    strcpy(tmp_char, cur_name);
    tmp_char[cur_name_len-1] = '\0';
    strcpy(cur_name, tmp_char);
    free(tmp_char);
}
curNode->SetName(cur_name);
(void)rdeqnChangeNodeFunction(curNode);
} else {
    // new name for INV node
    new_name = (char *)calloc(cur_name_len+2, sizeof(char));
    strcpy(new_name, cur_name);
    strcat(new_name, "\0");
}

// NewNode : type -> Internal, func-->NODE_INV
if(curFunc==NODE_PO || curFunc==NODE_PI){ // 10-01-00
    newNode = rdeqnCreateINVNode(eGraph, curNode, new_name, 1);
} else {
    newNode = rdeqnCreateINVNode(eGraph, curNode, new_name);
}

// Insert new node into eGraph
gateList = eGraph->GetGateList();
pos = eGraph->FindNode(gateList, cur_name);
gateList->Reset(pos);
gateList->InsertRight(newNode);
eGraph->SetNumGate(eGraph->GetNumGate()+1); // (eGraph->numgate)++

// Insert newNode as a curNode's fanout node's fanin
while(!foQ->IsEmpty()){
    foNode = foQ->DeQueue();
    foNode->InsertFanin(newNode);
}
free(new_name); // (I, 0131)
delete foQ;

// Performs network transformation by adding inverters at the non-inverted
// nodes where their inverted signal does not exist.
void rdeqnAddInverterNodes(Graph *eGraph)
{
    LinkedList<Node*> *gateList;
    Node *curNode, *invNode;
    char *curName;
    int num_fo;
    NodeType node_type;
    NodeFunc node_func;
    Queue<Node*> *Q = new Queue<Node*>;

    gateList = eGraph->GetGateList();
    for(gateList->Reset(); !gateList->EndOfList(); gateList->Next()){
        curNode = gateList->Data();
        Q->QInsert(curNode);
    }

    while(!Q->IsEmpty()){
        curNode = Q->DeQueue();
        curName = curNode->GetName();
        node_type = curNode->GetNodeType();
        node_func = curNode->GetNodeFunc();
        num_fo = curNode->GetNumFanout();
        if(num_fo==0 && node_func==NODE_PI) continue;

        if(node_func != NODE_INV){
            invNode = rdeqnFindReversedPolarityNode(eGraph, curNode);
            if(invNode == NULL && (node_type == INTERNAL ||
                node_func==NODE_PO && num_fo > 0 || node_func==NODE_PI)){
#ifdef READ_EQN_DEBUG
                cout<<"Target Nodes : "<<curNode->GetName()<<"\n";
#endif
                (void)rdeqnReverseNodeFuncAndInsertInverter(eGraph, curNode);
#ifdef READ_EQN_DEBUG
                cout<<"curNode = "<<curNode->GetName()<<"\n";
#endif
            }
        }
        delete Q;

        // Global Depth
        /* 10-23-00 Initialize global depth for match pin assignment based on
        * depth. Here, global_depth in node CLASS is newly introduced
        */
        // returns the number of fanins that are already visited.
        int rd_depthAreAllFaninsVisitedGetMaxInDepth(Node *curNode, int &max_depth)
        {
            LinkedList<Node*> *finList;
            Node *inNode;
            int count = 0, visited, gLevel;

            max_depth = 0;
            finList = curNode->GetFaninList();
            for(finList->Reset(); !finList->EndOfList(); finList->Next()){
                inNode = finList->Data(); visited = -1;
                visited = inNode->GetVisit();
                gLevel = inNode->GetGLevel();
                if(visited==1) {
                    count++;
                    max_depth = Max(max_depth, gLevel);
                }
            }
            return count;
        }

        int rd_depthIsCurrentNodeInQueue(Queue<Node*> *Q, Node *foNode)
        {
            LinkedList<Node*> *qList;
            Node *tmpNode;
            char *tmp_name, *fo_name;
            int return_val = 0;

            if(Q->IsEmpty()) return (0);
            fo_name = foNode->GetName();
            qList = Q->QList();
            for(qList->Reset(); !qList->EndOfList(); qList->Next()){
                tmpNode = qList->Data();
                tmp_name = tmpNode->GetName();
                if(strcmp(tmp_name, fo_name)==0){
                    return_val = 1;
                    break;
                }
            }
            return return_val;
        }

        // Initialize Global depth flag from PI(gLevel = 0) to PO.
        void rd_depthInitGlobalDepth(Graph *origNet)
        {
            LinkedList<Node*> *piList, *foList;
            Queue<Node*> *Q = new Queue<Node*>;
            Node *curNode, *foNode;
            int num_visit, max_depth, visited, num_fanin, num_fanout;
            int gLevel;
            NodeFunc curFunc;
            NodeType curType;
            int itr_cnt = 0;

            piList = origNet->GetPiList();
            for(piList->Reset(); !piList->EndOfList(); piList->Next()){
                curNode = piList->Data();
                curNode->MarkVisit();
                curNode->SetGLevel(0);
                if((num_fanout = curNode->GetNumFanout())==0) continue;
                foList = curNode->GetFanoutList();
                for(foList->Reset(); !foList->EndOfList(); foList->Next()){
                    foNode = foList->Data();
                    if(!rd_depthIsCurrentNodeInQueue(Q, foNode)){
                        Q->QInsert(foNode);
                    }
                }
            }

            while(!Q->IsEmpty()){
                itr_cnt++;
                curNode = Q->DeQueue();

                num_fanin = curNode->GetFaninList()->ListSize();
                visited = curNode->GetVisit();
                num_visit=rd_depthAreAllFaninsVisitedGetMaxInDepth(curNode, max_depth);
                if(num_visit==num_fanin && num_visit > 0 && visited==1){
                    curNode->SetGLevel((gLevel = max_depth+1));
                    curNode->MarkVisit();

                    curFunc = curNode->GetNodeFunc();
                    curType = curNode->GetNodeType();
                    num_fanout = curNode->GetNumFanout();

                    if(curFunc==NODE_PO&&num_fanout==0 || curType==DUMMY) continue;
                    foList = curNode->GetFanoutList();
                    for(foList->Reset(); !foList->EndOfList(); foList->Next()){
                        foNode = foList->Data();
                        if(!rd_depthIsCurrentNodeInQueue(Q, foNode)){
                            Q->QInsert(foNode);
                        }
                    }
                } else {
                    if(!rd_depthIsCurrentNodeInQueue(Q, curNode)&&visited==1){
                        Q->QInsert(curNode);
                    }
                }

                /* //if(gLevel > 28 && DEBUG ==1) {
                cout<<"Q_length = "<<Q->QLength()<<"\n";
                LinkedList<Node*> *tmpList = new LinkedList<Node*>;
                while(!Q->IsEmpty()){
                    tmpNode = Q->DeQueue();
                    tmpList->InsertRear(tmpNode);
                }
                for(tmpList->Reset(); !tmpList->EndOfList(); tmpList->Next()){
                    tmpNode = tmpList->Data();
                    Q->QInsert(tmpNode);
                }
                origNet->PrintNodesOfList(tmpList);
                delete tmpList;
                */
            }
            delete Q;

            /* ----- Balanced Graph ----- */
            // Reset (-1) visited flag for all nodes in a sub-graph except the nodes in
            // piList of the graph. The visited flag of the nodes in piList is set to 1.
            void rd_balResetVisitFlag(Graph *ifGraph)
            {
                LinkedList<Node*> *piList, *allNodeList;
            }

```

```

Node *aNode;
char *aNode_name;

pilst = fGraph->GetPiList();
allNodeList = fGraph->GetGateList();

for(allNodeList->Reset();allNodeList->EndOfList();allNodeList->Next()){
    aNode = allNodeList->Data();
    aNode_name = aNode->GetName();
    if(fGraph->FindNode(pilst, aNode_name) != -1){
        aNode->MarkVisit(1);
    } else {
        aNode->MarkVisit(-1);
    }
}

// Returns global depth difference of fanin nodes
// if left-heavy, +2, and if right-heavy -2, else 0 returns.
//int rd_balGetFaninsGlobalDepthDifference(Node *curNode, Node *hvyNode)
Node *rd_balGetFaninsGlobalDepthDifference(Node *curNode, int &gdif)
{
    LinkedList<Node *> *finList;
    Node *lowNode, *highNode;
    NodeFunc lowFunc, highFunc;
    int low_depth, high_depth;

    finList = curNode->GetFaninList();
    if(finList->ListSize() != 2){
        cout<<"Number of fanins are not 2, in "<<__FILE__<<endl;
        exit(0);
    } else {
        lowNode = finList->Data(0); // left
        highNode = finList->Data(1); // right
        lowFunc = lowNode->GetNodeFunc();
        highFunc = highNode->GetNodeFunc();
        if(!(lowFunc == NODE_PI && highFunc == NODE_PO)){
            low_depth = lowNode->GetGLevel();
            high_depth = highNode->GetGLevel();
            gdif = low_depth - high_depth;
            if(abs(gdif) < 2 || abs(gdif) > 10){
                gdif = 0;
                return NULL;
            } else {
                if(gdif > 0 && lowFunc != NODE_PO) {
                    return lowNode;
                } else if(gdif < 0 && highFunc != NODE_PO){
                    return highNode;
                } else {
                    return NULL;
                }
            }
        } else {
            gdif = 0;
            return NULL;
        }
    }
    return (NULL);
}

void rd_balDoSingleRotation(Graph *origNet, Node *curNode, Node *hvyNode)
{
    LinkedList<Node *> *cfoList, *cfoInList, *cinList;
    LinkedList<Node *> *hvfList, *hvfInList, *hvfIn_foList;
    Node *hvinNode, *cfoNode, *tmpNode0, *tmpNode1;
    char *cur_name, *hvy_name, *hvyIn_name, *cfo_name;
    int cur_pos_in_cfoInL=-1, hvy_pos_in_cinL=-1, hvin_pos_in_hvinL=-1;
    int hvy_pos_in_hvinFol=-1; // for sct.slif
    NodeFunc cfo_Func;

    cur_name = curNode->GetName();
    hvy_name = hvyNode->GetName();

    // 1) Deleting curNode's fanout form cfoList
    cfoList = curNode->GetFanoutList();
    cfoList->Reset(); // curNode must have only one fanout
    cfoNode = cfoList->Data();
    (void)cfoList->DeleteCurrentElem();
    (void)curNode->SetNumFanout(curNode->GetNumFanout()-1);

    // 2) Deleting curNode from curNode's fanout node's faninList (cfoInList)
    cfoInList = cfoNode->GetFaninList();
    cur_pos_in_cfoInL = origNet->FindNode(cfoInList, cur_name);
    if(cur_pos_in_cfoInL != -1){
        cfoInList->Reset(cur_pos_in_cfoInL);
        (void)cfoInList->DeleteCurrentElem();
        (void)cfoNode->SetNumFanin(cfoNode->GetNumFanin()-1);
    }

    // 3) Deleting curNode from hvyNode's fanout List (hvfList)
    hvfList = hvyNode->GetFanoutList();
    hvfList->Reset(); // hvyNode must have only one fanout
    (void)hvfList->DeleteCurrentElem(); // deleting curNode from hvfList
    (void)hvyNode->SetNumFanout(hvyNode->GetNumFanout()-1);

    // 4) Deleting hvyNode from curNode's fanin List (cinList)
    cinList = curNode->GetFaninList();
    hvy_pos_in_cinL = origNet->FindNode(cinList, hvy_name);
    if(hvy_pos_in_cinL != -1){
        cinList->Reset(hvy_pos_in_cinL);
        (void)cinList->DeleteCurrentElem();
        //delete cinList->DeleteCurrentElem();
        (void)curNode->SetNumFanin(curNode->GetNumFanin()-1);
    }

    // 5) find a node (hvinNode) whose gdepth is smaller from hvyNode's fanins
    hvinList = hvyNode->GetFaninList();
    if(hvinList->ListSize() == 2){
        tmpNode0 = hvinList->Data(0);
        tmpNode1 = hvinList->Data(1);

        (tmpNode0->GetGLevel()) > tmpNode1->GetGLevel() ? hvinNode = tmpNode1 : hvinNode = tmpNode0;

        hvyIn_name = hvinNode->GetName();
    } else {
        cout<<"Error :: in "<<__FILE__<<" at line "<<__LINE__<<endl;
        exit(0);
    }

    // 6) Deleting hvinNode from hvyNode's fanin List (hvinList)
    hvin_pos_in_hvinL = origNet->FindNode(hvinList, hvyIn_name);
    if(hvin_pos_in_hvinL != -1){
        hvinList->Reset(hvin_pos_in_hvinL);
        (void)hvinList->DeleteCurrentElem();
        //delete hvinList->DeleteCurrentElem();
        (void)hvyNode->SetNumFanin(hvyNode->GetNumFanin()-1);
    }

    // 7) Deleting hvyNode from hvinNode's fanout list (hvin_foList)
    hvin_foList = hvinNode->GetFanoutList();
    hvy_pos_in_hvinFol = origNet->FindNode(hvin_foList, hvy_name);
    if(hvy_pos_in_hvinFol != -1){ // for sct.slif
        hvin_foList->Reset(hvy_pos_in_hvinFol);
        (void)hvin_foList->DeleteCurrentElem();
        (void)hvinNode->SetNumFanout(hvinNode->GetNumFanout()-1);
    }

    // 8) do rotation by inserting fanins/fanouts
    cfoNode->InsertFanin(hvyNode);
    cfo_Func = cfoNode->GetNodeFunc();
    if(cfo_Func == NODE_INV){
        // added on 11-11-00 to insure s and s' after transform
        cfo_name = cfoNode->GetName();
        cfo_name = (char *)realloc(cfo_name, strlen(hvy_name)+3);
        strcpy(cfo_name, hvy_name);
        strcat(cfo_name, "\\0");
    }
    hvyNode->InsertFanout(cfoNode);
    hvyNode->InsertFanin(curNode);
    curNode->InsertFanout(hvyNode);
    curNode->InsertFanin(hvinNode);
    hvinNode->InsertFanout(curNode);
}

Node *rd_balGetUnBalancedNode(Graph *origNet, int &gdif)
{
    LinkedList<Node *> *gateList;
    Node *curNode, *hvyNode = NULL;
    NodeFunc curFunc, hvyFunc;
    //int num_fanin, num_fanout, gdif;
    int num_fanin, num_fanout, hv_fanout;

    gateList = origNet->GetGateList();
    for(gateList->Reset();gateList->EndOfList();gateList->Next()){
        curNode = gateList->Data();
        curFunc = curNode->GetNodeFunc();
        num_fanin = curNode->GetNumFanin();
        num_fanout = curNode->GetNumFanout();
        if(num_fanout >= 2) continue;
        if(curFunc != NODE_PI && curFunc != NODE_PO && num_fanin==2){
            hvyNode = rd_balGetFaninsGlobalDepthDifference(curNode, gdif);
            if(hvyNode == NULL) continue;
            hvyFunc = hvyNode->GetNodeFunc();
            hv_fanout = hvyNode->GetNumFanout();
            if(gdif==0 && hvyNode!=NULL && curFunc==hvyFunc && hv_fanout==1){
                #ifdef MAKE_BALANCED_GRAPH_DEBUG
                cout<<"found transformation target, node= "
                <<curNode->GetName()<<"\n";
                cout<<"heavy_node = "<<hvyNode->GetName()<<endl;
                cout<<"gdif = "<<gdif<<endl;
                #endif
                break;
            } else {
                hvyNode = NULL;
            }
        } else {
            hvyNode = NULL;
        }
    }
    return hvyNode;
}

//1) Calculate global depth (glevel) at each node after resetting visit flag
//2) Check fanin node's global depth difference at each node (the fanin node
//   should be neither MFP nor PO).
//   If the difference is great than 2 and one of the fanin node type is
//   the same as current node, insert the current node into a Queue.
//3) If the Q is not empty do Transformation
//   else stop
//4) Update global depth (re-calculate global depth)
//5) Find un-balanced node (fanin's gdepth diff >2) -> Q
//5) Repeat step (3)
void rd_balMakeBalancedGraph(Graph *origNet)
{
    LinkedList<Node *> *hv_foList;
    Node *curNode, *hvyNode;
    int gdif;
    Queue<Node *> Q;

    (void)rd_balResetVisitFlag(origNet);
    (void)rd_depthInitGlobalDepth(origNet);

    (void)origNet->PrintAllNodes();

    // 1) find a node which is un balanced --> Q
    hvyNode = rd_balGetUnBalancedNode(origNet, gdif);
    if(hvyNode != NULL){
        hv_foList = hvyNode->GetFanoutList();
    }
}

```

```

        if(hv_foList->ListSize() != 1){
            cout<<"Error :: in "<<_FILE_<<" at line "<<_LINE_<<endl;
        } else {
            curNode = hv_foList->Data(0);
        }
        Q.Insert(curNode);
        Q.Insert(hvyNode);
    }

    // 2) while Q is not empty
    while(!Q.Empty()){
        // 2.1) dequeue
        curNode = Q.DeQueue();
        hvyNode = Q.DeQueue();

        // 2.2) do_rotate
        (void)rd_balDoSingleRotation(origNet, curNode, hvyNode);
        (void)rd_balResetVisitFlag(origNet);
        (void)rd_depthInitGlobalDepth(origNet);

#ifdef MAKE_BALANCED_GRAPH_DEBUG
        (void)origNet->PrintAllNodes();
#endif

        // 2.3) find a node which is unbalanced --> Q
        hvyNode = rd_balGetUnBalancedNode(origNet, gdiff);
        if(hvyNode != NULL){
            hv_foList = hvyNode->GetFanoutList();
            if(hv_foList->ListSize() != 1){
                cout<<"Error :: in "<<_FILE_<<" at line "<<_LINE_<<endl;
            } else {
                curNode = hv_foList->Data(0);
            }
            Q.Insert(curNode);
            Q.Insert(hvyNode);
        }
    }
    (void)rd_balResetVisitFlag(origNet);

void rdeqnRemove2InvsInSeries(Graph *origNet, Node *curNode, Node *foNode)
{
    LinkedList<Node*> *gateList, *finList, *in_foList, *fo_foList;
    LinkedList<Node*> *fo_foInList;
    Node *inNode, *fo_foNode;
    char *del_name, *foName;
    int in_foSize, n, pos;

    gateList = origNet->GetGateList();
    finList = curNode->GetFaninList();
    foName = foNode->GetName();

    if(finList->ListSize() == 1){
        inNode = finList->Data(0); // curNode's fanin
    } else {
        // since an inverter's fanin is always one.
        cout<<"Node "<<curNode->GetName()<<"' fanin is not one.....\n";
        cout<<"in "<<_FILE_<<" at line "<<_LINE_<<"\n";
        exit(0);
    }

    //1) get curNode fanin node's fanout list and delete curNode(clear)
    del_name = curNode->GetName();
    in_foList = inNode->GetFanoutList();
    pos = origNet->FindNode(in_foList, del_name);
    if(pos == -1){
        cout<<"Node "<<del_name<<" cannot find in cur_inNode's foList"<<"\n";
        cout<<"in "<<_FILE_<<" at line "<<_LINE_<<"\n";
        exit(0);
    }

    in_foList->Reset(pos);
    delete in_foList->DeleteCurrentElem();
    inNode->SetNumFanout(inNode->GetNumFanout()-1); // numfaout--

    //2) insert foNode's fanout into curNode's fanout list
    //3) for all fanout nodes of foNode, get faninList and find foNode, delete
    // and insert inNode into fo_inList;

    fo_foList = foNode->GetFanoutList();
    for(fo_foList->Reset(); !fo_foList->EndOfList(); fo_foList->Next()){
        fo_foNode = fo_foList->Data(0);
        inNode->InsertFanout(fo_foNode);
        fo_foInList = fo_foNode->GetFaninList();
        pos = origNet->FindNode(fo_foInList, foName);
        if(pos != -1){
            fo_foInList->Reset(pos);
            delete fo_foInList->DeleteCurrentElem();
            fo_foNode->SetNumFanin(fo_foNode->GetNumFanin()-1); // numfain--
            fo_foNode->InsertFanin(inNode);
        }
    }

    //4) delete curNode and foNode from gateList
    pos = -1;
    del_name = curNode->GetName();
    pos = origNet->FindNode(gateList, del_name);
    if(pos != -1){
        gateList->Reset(pos);
        delete gateList->DeleteCurrentElem();
    } else {
        cout<<"Node "<<del_name<<"' is not found in gateList.....\n";
        cout<<"in "<<_FILE_<<" at line "<<_LINE_<<"\n";
        exit(0);
    }
    pos = -1;
    del_name = foNode->GetName();
    pos = origNet->FindNode(gateList, del_name);
    if(pos != -1){
        gateList->Reset(pos);
    }

    delete gateList->DeleteCurrentElem();
}

void rdeqnCheckNRemove2ConsecutiveInvs(Graph *origNet)
{
    LinkedList<Node*> *gateList, *foList;
    Node *curNode, *foNode;
    Queue<Node*> *Q;

    gateList = origNet->GetGateList();
    for(gateList->Reset(); !gateList->EndOfList(); gateList->Next()){
        curNode = gateList->Data(0);
        if(curNode->GetNodeFunc() == NODE_INV){
            foList = curNode->GetFanoutList();
            if(foList->ListSize() == 1){
                foNode = foList->Data(0);
                if(foNode->GetNodeFunc() == NODE_INV){
                    cout<<"2 consecutive nodes : cur_name = "
                        <<curNode->GetName()<<endl;
                    Q.Insert(curNode);
                    Q.Insert(foNode);
                }
            }
        }
    }

    while(!Q.Empty()){
        curNode = Q.DeQueue();
        foNode = Q.DeQueue();
        (void)rdeqnRemove2InvsInSeries(origNet, curNode, foNode);
    }
}

void rdeqnChangeNamesx(Graph *origNet, Node *curNode, Node *foNode)
{
    LinkedList<Node*> *fo_foList, *fo_foInL, *curInList;
    LinkedList<Node*> *cur_foL, *gateList;
    Node *curInNode, *fo_foNode;
    char *foName, *curName;
    int pos, n;

    curName = curNode->GetName();
    foName = foNode->GetName();

    //Delete foNode from origNet's gateList
    gateList = origNet->GetGateList();
    if((pos = origNet->FindNode(gateList, foName)) != -1){
        gateList->Reset(pos);
        delete gateList->DeleteCurrentElem();
    }

    cur_foL = curNode->GetFanoutList();
    pos = origNet->FindNode(cur_foL, foName);
    if(pos != -1){
        cur_foL->Reset(pos);
        delete cur_foL->DeleteCurrentElem();
        curNode->SetNumFanout(curNode->GetNumFanout()-1); // numfain--
    }

    curInList = curNode->GetFaninList();
    curInNode = curInList->Data(0);

    curInNode->SetName(curName);
    curNode->SetName(foName);

    fo_foList = foNode->GetFanoutList();
    for_each(fo_foList){
        fo_foNode = fo_foList->Data(0);
        fo_foInL = fo_foNode->GetFaninList();
        pos = origNet->FindNode(fo_foInL, foName);
        if(pos != -1){
            fo_foInL->Reset(pos);
            delete fo_foInL->DeleteCurrentElem();
            fo_foNode->SetNumFanin(fo_foNode->GetNumFanin()-1); // numfain--
            fo_foNode->InsertFanin(curInNode);
        }
        curInNode->InsertFanout(fo_foNode);
    }

    fo_foList->Reset();
    for(n = 0; n < fo_foList->ListSize(); n++){
        delete fo_foList->DeleteCurrentElem();
    }
}

void rdeqnChangeNames(Graph *origNet, Node *curNode, Node *foNode)
{
    LinkedList<Node*> *fo_foList, *fo_foInL, *curInList;
    LinkedList<Node*> *cur_foL, *gateList;
    Node *curInNode, *fo_foNode;
    char *foName;
    int cur_name_len, pos, n;

    string cur_name, tmp_str;

    cur_name += curNode->GetName();
    foName = foNode->GetName();

    // 1) Delete foNode from origNet's gateList
    gateList = origNet->GetGateList();
    if((pos = origNet->FindNode(gateList, foName)) != -1){
        gateList->Reset(pos);
        delete gateList->DeleteCurrentElem();
    }

    // 2) delete foNode from curNode's fanoutList
    cur_foL = curNode->GetFanoutList();
}

```

```

pos = origNet->FindNode(cur_foL, foName);
if(pos != -1){
    cur_foL->Reset(pos);
    delete cur_foL->DeleteCurrentElem();
    curNode->SetNumFanout(curNode->GetNumFanout()-1); // numfan--
}

curInList = curNode->GetFaninList();
curInNode = curInList->Data();

// 3) modify curInNode's name
cur_name_len = cur_name.size();
tmp_str.erase();
if(cur_name.substr(cur_name_len-1,1)=="\\"){ //the last char
    tmp_str = cur_name.substr(0,cur_name_len-1);
    curInNode->SetName((char *)tmp_str.c_str());
} else {
    tmp_str = cur_name + "\\";
    curInNode->SetName((char *)tmp_str.c_str());
}

// 4) delete old foNode from foNode's fanList and insert new fanin
// And insert fanout of foNode into cur_in_node's fanout list
fo_foList = foNode->GetFanoutList();
for_each(fo_foList){
    fo_foNode = fo_foList->Data();
    fo_foInL = fo_foNode->GetFaninList();
    pos = origNet->FindNode(fo_foInL, foName);
    if(pos != -1){
        fo_foInL->Reset(pos);
        delete fo_foInL->DeleteCurrentElem();
        fo_foNode->SetNumFanin(fo_foNode->GetNumFanin()-1); // numfan--
        fo_foNode->InsertFanin(curInNode);
        curInNode->InsertFanout(fo_foNode);
    }
}

fo_foList->Reset();
for(n = 0; n < fo_foList->ListSize(); n++){
    delete fo_foList->DeleteCurrentElem();
}

// This routine rearranges node names for the following situations.
// for C880.slif, C1908.slif, C1355.slif, C3450.slif
// 1) If a node (name = a) is a mfp point and its fanin node name is not a', but
//    a' exist after mfp point.
// 2) If a node (name = a) is a mfp point and its fanin node name is not a'.
void rdeqnRearrangeNodeNames(Graph *origNet)
{
    LinkedList<Node> *gateList, *foList, *filist;
    Node *curNode, *foNode;
    Queue<Node> *Q;

    gateList = origNet->GetGateList();
    for(gateList->Reset(); !gateList->EndOfList(); gateList->Next()){
        curNode = gateList->Data();
        if(curNode->GetNodeFunc() == NODE_INV){
            foList = curNode->GetFanoutList();
            filist = curNode->GetFaninList();
            if(foList->ListSize() > 1){
                for(foList->Reset(); !foList->EndOfList(); foList->Next()){
                    foNode = foList->Data();
                    if(foNode->GetNodeFunc() == NODE_INV){
                        cout<<"Rearrange target : curNode = "
                             <<curNode->GetName() <<"curNode_b = "
                             <<foNode->GetName()<<endl;
                        Q.Insert(curNode);
                        Q.Insert(foNode);
                    }
                }
            }
        }
    }

    while(!Q.Empty()){
        curNode = Q.DeQueue();
        foNode = Q.DeQueue();
        (void)rdeqnChangeNames(origNet, curNode, foNode);
    }

// Top routine of equation reading
Graph *Read_eqn(char *file_name)
{
    Graph *eGraph = new Graph;
    //Node *tmpNode, *tmpNode1;;

// declaration of operator stack with MathOperator objects
Stack<EqnOperator> OperatorStack;

// declaration of the operand stack.
Stack<char> *OperandStack;

EqnOperator opr1, opr2;

char *token = new char[80];
int rank = 0;
char ch;

ifstream fin(file_name);
if (!fin) {
    cerr << "File " << file_name << " does not exist" << endl;
    exit(1);
} else {
    cout<<"\nReading File : "<<file_name<<"\n\n";
}

while ((ch=fin.peek()) != EOF) {

        if (ch == '\n') { // new line char eatup()
            (void)new_line_eatup(fin);
            continue;
        } else if (ch == ',') {
            // Process "module", "endmodule", "inputs", and "outputs"
            ch = fin.get();
            fin >> token;

            if (strcmp(token, "module") == 0){
                fin >> token;
                (void)rdeqnSetGraphName(eGraph, fin, token);
                continue;
            } else if (strcmp(token, "endmodule") == 0){
                fin >> token;
                #ifdef READ_EQN_DEBUG
                cout << "\nmodule "<< token << " has been read !!!" << endl;
                #endif
                break;
            } else if (strcmp(token, "inputs") == 0){ // input pin nodes
                (void)rdeqnCreateInputNodes(eGraph, fin, token);
                continue;
            } else if (strcmp(token, "outputs") == 0){ // output pin nodes
                //tmpNode = rdeqnCreateOutputNodes(eGraph, fin, token);
                (void)rdeqnCreateOutputNodes(eGraph, fin, token);
                continue;
            } else {
                while((ch=fin.get())!='\n'){
                    fin.get(ch);
                }
                continue;
            }
        } else {
            // process the expression until ';' is read
            if(ch=='#'){
                (void)rdeqnLineEatup(fin);
            }
            while (fin.get(ch))
            {
                // ***** process literal operands *****
                /* Ignore new_line character */
                if (!isoperator(ch) && (ch != '\n') && (ch != ';') &&
                    (ch != ',') && (ch != '\n')) {
                    fin.putback(ch); // to get node_name from start
                    (void)rdeqnProcessOperand(OperandStack, fin, rank);
                }
                else if (ch==';') {
                    break;
                }
                // process an operator
                else if (isoperator(ch)) // ' ', '+', '*', '(', and ')'
                {
                    (void)rdeqnProcessAOperator(eGraph, OperatorStack,
                        OperandStack, fin, ch, rank);
                }
                else if (ch == '(') { // Inverter operator process
                    (void)rdeqnProcessInvOperator(eGraph, OperandStack);
                }
                // ***** process a right parenthesis *****
                else if (ch == ')')
                {
                    // build a MathOperator object holding ')', which
                    // has precedence lower than the stack precedence
                    // of any operator except '('. pop the operator stack
                    // and evaluate the subexpression until '(' surfaces
                    // or the stack is empty. if the stack is empty, a
                    // '(' is missing; otherwise, delete '('.
                    opr1 = EqnOperator(ch);
                    while(!OperatorStack.StackEmpty() &&
                        (opr2 = OperatorStack.Peek()) >= opr1)
                    {
                        opr2 = OperatorStack.Pop();
                        //opr2.Evaluate(OperandStack);
                        opr2.Evaluate(*eGraph, OperandStack);
                    }
                    if(OperatorStack.StackEmpty())
                        error(MissingLeftParenthesis);
                    opr2 = OperatorStack.Pop(); // get rid of '('
                }
                // ***** have some invalid input *****
                else if (!iswhitespace(ch))
                    error(InvalidInput);
            }
        }
    } // Inner WHILE loop End */
} // Endof If

/* the rank of the complete expression must be 1 */
if (rank != 1){
    cout<<"Processing outside of while() loop...\n";
    error(OperandExpected);
}

/* flush operator stack and complete expression evaluation.
 * if find left parenthesis, a right parenthesis is missing.
 */
while (!OperatorStack.StackEmpty())
{
    opr1 = OperatorStack.Pop();
    if (opr1.GetOp() == leftparenthesis) {
        error(MissingRightParenthesis);
    }
    //opr1.Evaluate(OperandStack);
    opr1.Evaluate(*eGraph, OperandStack);
}
}

```

```

        rank = 0;
        /* value of the expression is on the top of the operand stack */
        //cout << "The value is " << OperandStack.Pop() << endl;
    } /* Outer WHILE_loop end */
    fin.close();
#ifdef READ_EQN_DEBUG
    cout << endl;
    eGraph->PrintPiNodes();
    cout << endl;
    eGraph->PrintPoNodes();
    cout << endl;
    eGraph->PrintAllNodes();
    cout << endl;
#endif
    int num_deleted;
    num_deleted = rdeqnFindUndefinedNodesAndRearrange(eGraph);
#ifdef READ_EQN_DEBUG
    cout << endl;
    eGraph->PrintAllNodes();
    cout << endl;
    cout << "Number of Nodes rearranged is : " << num_deleted << endl;
#endif
    (void)rdeqnCheckUndefinedNodes(eGraph);
    (void)rdeqnCheckNRemove2ConsecutiveInvs(eGraph);
#ifdef MAKE_BALANCED_GRAPH
    (void)rd_balMakeBalancedGraph(eGraph);
#endif
    (void)rdeqnRearrangeNodeNames(eGraph);
    cout << "After Rearrange and before Add Inverter\n";
    eGraph->PrintAllNodes();
#ifdef ADD_INVERTER
    (void)rdeqnAddInverterNodes(eGraph);
#endif
    cout << " PRINT ALL NODES After adding INVs\n";
    eGraph->PrintAllNodes();
#ifdef READ_EQN_DEBUG
    cout << endl;
    cout << "New Graph .. \n";
    eGraph->PrintAllNodes();
    cout << endl;
#endif
    return eGraph;
} /* EndOf_Read_eqn() */

/*****
 * Read_lib.cc
 * Reads mixed PTL and Static Cell library
 *****/
#include <fstream.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>

#include "eqn_stack.h"
#include "eqn_opr.h"
#include "Node.h"
#include "Graph.h"
#include "LibCell.h"

// whole line eat-up
int line_eatup(ifstream &fin)
{
    char ch;
    while((ch=fin.get()) != '\n'){
        continue;
    }
    fin.putback(ch);
    return 1;
}

// tab eat-up
int tab_eatup(ifstream &fin)
{
    char ch;
    // tab char eatup()
    while ((ch=fin.get()) == '\t'){
        continue;
    }
    fin.putback(ch);
    return (1);
}

// Create a Cell when "GATE" token is encountered.
// Sets cells_name, cGraph name, reads cell's area and power
LibCell * libCreateCell(ifstream &fin, LinkedList<LibCell *> *Cells, \
                        char *token, int index)
{
    float cell_area, cell_power;
    Graph *cGraph;
    //char *cell_name = new char[10];
    char *cell_name=NULL;
    int token_len;

    token_len = strlen(token);
    cell_name = (char *)calloc(token_len+1, sizeof(char));
    if(cell_name == NULL){

```

```

        cout<<"Cell name allocation error!!!!\n";
        exit (1);
    }
    strcpy(cell_name, token);
    LibCell *newCell = new LibCell;
    if(newCell == NULL){
        cout<<"Memory allocation error for New_Cell !!!!\n";
        return (0);
    }
    (*newCell).SetName(cell_name);
    if(!strcmp(cell_name,"ptl",3)){
        newCell->SetCellType(PTL);
    }
    else {
        newCell->SetCellType(STATIC);
    }
    fin >> cell_area; fin >> cell_power;
    newCell->SetArea(cell_area);
    newCell->SetPower(cell_power);
    cGraph = (*newCell).GetCellGraph();
    cellGraph->SetName(cell_name);

    LibCell *tmpCell;
    char *tmp_name;
    if(!Cells->isEmpty()){
        for(Cells->Reset();!Cells->EndOfList();Cells->Next()){
            tmpCell = Cells->Data();
            tmp_name = tmpCell->GetName();
            if(strcmp(cell_name, tmp_name)==0){
                cout << "Cell : "<<cell_name<<" is already exist...\n";
                exit(1);
            }
        }
    }
    Cells->InsertRear(newCell);
    free(cell_name); //(1, 0131)
    return newCell;
}

// Allocate memory for signal(input/output) name and assemble char
// into char string which is signal name
char *libGetSignalName(ifstream &fin, LinkedList<LibCell *> *Cells,
                        int index, char chr, int *rank)
{
    char *node_name;
    char *tmp_name , ch;
    node_name = (char *)calloc(11,sizeof(char));
    // for single letter string
    tmp_name = (char *)calloc(2,sizeof(char));

    // place the char back to get node_name from start
    fin.putback(chr);
    while ((ch=fin.get())) {
        *tmp_name = ch;
        strcat(node_name,tmp_name);
        ch=fin.peek();
        if (isoperator(ch) || (ch == ';' || (ch ==')') || (ch == '\')){
            *tmp_name = '\0';
            strcat(node_name,tmp_name);
            break;
        }
    }
#ifdef LIB_DEBUG
    cout << "node name = "<< node_name ;
    cout << " rank = "<<rank <<endl;
#endif
    // rank of operand is 1. accumulated rank must be 1
    *rank += 1;
#ifdef LIB_DEBUG
    cout<<"node name = "<< node_name << " rank = "<<rank<<endl;
#endif
    if (*rank > 1){
        error(OperatorExpected);
    }
    Node *tmpNode; Graph *clGraph;
    LibCell *libCell; int position;
    Cells->Reset(index);
    libCell = Cells->Data();
    clGraph = libCell->GetCellGraph();
    if(strcmp(node_name,"0")==0){
        tmpNode = clGraph->FindOrCreateNode(node_name, position,
                                            PRIMARY_OUTPUT, NODE_PO);
    }
    else {
        tmpNode = clGraph->FindOrCreateNode(node_name, position,
                                            PRIMARY_INPUT, NODE_PI);
    }
    free(tmp_name);
    return node_name;
}

// Once find an operator, check what the next character is.
void libOperatorPreProcess(ifstream &fin, char *ch, int *rank)
{
    char tmp_ch;

```

```

if (*ch == ' ') {
    tmp_ch = *ch;
    while ((*ch = fin.get())) {
        if (*ch == '+' || *ch == '-' || *ch == '*') {
            tmp_ch = *ch;
        }
        if (!isoperator(*ch) && *ch != '\n') {
            fin.putback(*ch); // to process later
            *ch = tmp_ch;
            break;
        }
        else if (*ch == '(' || *ch == '\n') {
            fin.putback(*ch);
            *ch = tmp_ch;
            break;
        }
    }
}

// rank of each operator other than '(' is -1.
// accumulated rank should be 0
if (*ch != '(') { // rank of '(' is 0
    *rank -= 1;
}
if (*rank < 0)
    error(OperandExpected);

#ifdef LIB_DEBUG
    cout << "Operator = " << *ch << endl;
#endif

}

// When ')' operator, create a inverted node of the node which is currently
// on the top of the OperandStack.
void libCreateInvertedNode(Stack<char *> *OperandStack, \
    LinkedList<LibCell *> *Cells, int index)
{
    Node *tmpNode, *tmpNode1;
    char *node_name = new char;
    int position, position1;

    Graph *clGraph;
    LibCell *libCell;

    Cells->Reset(index);
    libCell = Cells->Data();

    clGraph = libCell->GetCellGraph();

    node_name = OperandStack->Pop();
    //040501 : due to insure++'s reporting --> WIRTE_OVERFLOW
    node_name = (char *)realloc(node_name, strlen(node_name)+2); //040501

    tmpNode = clGraph->FindOrCreateNode(node_name, position);
    if (tmpNode->GetNodeType() == UNASSIGNED) {
        tmpNode->SetNodeType(INTERNAL);
    }

    strcat(node_name, "\n");
    tmpNode1 = clGraph->FindOrCreateNode(node_name, position1);
    if (tmpNode1->GetNodeType() == UNASSIGNED) {
        tmpNode1->SetNodeType(INTERNAL);
    }
    if (tmpNode1->GetNodeFunc() != NODE_INV) {
        tmpNode1->SetNodeFunc(NODE_INV);
    }

    // Insert tmpNode into the fanin_list of tmpNode1
    if (position1 == -1) {
        tmpNode1->InsertFanin(tmpNode);
    }

    // Insert tmpNode1 into the fanout_list of tmpNode
    if (position == -1) {
        tmpNode->InsertFanout(tmpNode1);
    }
}

#ifdef LIB_DEBUG
    cout << "Created Node => " << node_name << endl;
#endif
OperandStack->Push(node_name);
}

// Read pin information for each cell
int libReadPinInfo(ifstream &fin, LibCell *curCell)
{
    Graph *curCellGraph;
    LinkedList<Node *> *grPiList;
    LinkedList<Pin *> *cCpinList; // current cell pinlist
    Pin *curPin = new Pin; // current cell pinlist

    int position;
    float value;
    char *token = new char[10];

    curCellGraph = curCell->GetCellGraph();
    grPiList = curCellGraph->GetPiList();
    cCpinList = curCell->GetPinList();
    cCpinList = curCell->GetPinList();

    while(1){
        fin >> token;
        if(strcmp(token, ";") == 0 || token[strlen(token)-1] == ';'){
            break;
        }
        if(strcmp(token, "PIN") == 0){
            fin >> token;
            if(strcmp(token, "=") == 0){
                cout << "Cell Library syntax error!\n";
            }
        }
        else {
            cout << "Don't use wild card for pin names\n";
            return (0);
        }
        else {
            if(curCell->FindPin(curCell->GetPinList(), token) != -1){
                cout << "Pin name (" << curPin->GetPinName() << ") is duplicated. Check cell library file.\n";
                return (0);
            }

            // Create a pin and insert it into pinlist in curCell class
            if((curCellGraph->FindNode(grPiList, token) != -1){
                curPin = curCell->FindOrCreatePin(token, position);
            }

            fin >> token;
            if(strcmp(token, "INV") == 0){
                curPin->SetPolarity(INV);
            }
            else if (strcmp(token, "NOINV") == 0){
                curPin->SetPolarity(NOINV);
            }
            else if (strcmp(token, "UNKNOWN") == 0){
                curPin->SetPolarity(BINATE);
            }
        }
        fin >> value;
        curPin->SetInputCap(value);

        fin >> value;
        curPin->SetTdr(value);
        fin >> value;
        curPin->SetTdr_slope(value);

        fin >> value;
        curPin->SetTdf(value);
        fin >> value;
        curPin->SetTdf_slope(value);
        continue;
    }
}

delete [] token;
return (1);
}

// Rpn list for each library cells
void libGetFunctionForBdd(Graph *fGraph, Node *bNode, int &nlit, \
    LinkedList<char *> *rpnList)
{
    LStack<Node *> *S = new LStack<Node *>;

    LinkedList<Node *> *finList;
    Node *curNode, *aNode;
    char *aNode_name;
    int visited, num_visit;

    ResetVisitFlag(fGraph);
    nlit = 0;

    if((visited = bNode->GetVisit()) < 0){
        S->Push(bNode);
    }
    else {
        cout << "In GetFunctionForBdd(), starting node has already been"
            << " visited.\n";
        exit(1);
    }

    while(!S->Empty()){
        curNode = S->Pop();
        finList = curNode->GetFaninList();
        num_visit = 0;
        for(finList->Reset(); !finList->EndOfList(); finList->Next()){
            aNode = finList->Data();
            if((visited = aNode->GetVisit()) > 0 )
                num_visit++;
        }

        if(num_visit == curNode->GetNumFanin()){
            // insert node name into list
            for(finList->Reset(); finList->EndOfList(); finList->Next()){
                aNode = finList->Data();
                aNode_name = aNode->GetName();
                if(fGraph->FindNode(fGraph->GetPiList(), aNode_name) != -1){
                    rpnList->InsertRear(aNode_name);
                    nlit++;
                }
            }

            switch(curNode->GetNodeFunc()){
                case NODE_INV: rpnList->InsertRear("\n");
                    break;
                case NODE_AND: rpnList->InsertRear("&");
                    break;
                case NODE_OR: rpnList->InsertRear("+");
                    break;
                default : break;
            }

            // set visit flag of curNode;
            curNode->MarkVisit(1);
        }
        else {
            S->Push(curNode);
            for(finList->Reset(); !finList->EndOfList(); finList->Next()){
                aNode = finList->Data();
                if((visited = aNode->GetVisit()) < 0 )
                    S->Push(aNode);
            }
        }
    }

    delete S;
} // EndOf libGetFunctionForBdd()

// Top routine of reading library
LinkedList<LibCell *> *libReadLibrary(char * file_name)
{
    Graph *curCellGraph = NULL;
    LibCell *curCell = NULL;
}

```

```

// declaration of operator stack with MathOperator objects
Stack<EqnOperator> OperatorStack;

// declaration of the operand stack.
Stack<char*> OperandStack;

EqnOperator opr1,opr2;

int rank = 0;
//char *temp = new char;
char *token = new char[80];

char *node_name, ch;

LinkedList<LibCell*> *CellsList = new LinkedList<LibCell*>;
int num_cell = -1;

ifstream fin(file_name);
if (!fin) {
    cerr << "File " << file_name << " does not exist" << endl;
    exit(1);
}

while ((ch=fin.peek()) != EOF) {
    if (ch == '\n') {
        (void) new_line_eatup(fin);
        continue;
    }
    else if (ch == '\t') {
        (void) tab_eatup(fin);
        continue;
    }
    else if (ch == '#') {
        (void) line_eatup(fin);
        continue;
    }
    else if (ch == '.') {
        ch = fin.get();
        fin >> token;

        if (strcmp(token,"GATE")==0){
            num_cell++;
            fin >> token;

            if(! (curCell = libCreateCell(fin, CellsList, token, num_cell))) {
                cout << "Cell creation error !!!\n";
            }

            curCellGraph = curCell->GetCellGraph();
            continue;
        }
        /* = reserved for Latch cell
        * else if (strcmp(token,"LATCH")==0){
        * fin >> token;
        * strcpy(temp, token, strlen(token)-1);
        * cout << "module "<< temp << " has been read !!!" << endl;
        * break;
        * }
        */
    }
    else if (ch == 'P') {
        if (!libReadPinInfo(fin, curCell)){
            cout << "Cell[" << curCell->GetName() << "]: ";
            cout << " : Library read error !!!!!!!!!\n";
            exit(1);
        }
        continue;
    }
    else {
        // process the expression until ';' is read
        while (fin.get(ch)) {
            /* Process literal operands and ignore new_line character */
            if (!isoperator(ch) && (ch != '\n') && (ch != ';') &&
                (ch != '\t') && (ch != '\r')) {
                node_name=libGetSignalName(fin,CellsList,num_cell,ch,&rank);

                // push the operand on the operand stack
                OperandStack.Push(node_name);
            }
            else if (ch==';') {
                break;
            }
            // ***** process an operator *****
            else if (isoperator(ch)) {
                (void)libOperatorPreProcess(fin, &ch, &rank);
                opr1 = EqnOperator(ch);
                while(!OperatorStack.StackEmpty() &&
                    (opr2 = OperatorStack.Peek()) >= opr1)
                {
                    opr2 = OperatorStack.Pop();
                    opr2.Evaluate((*(curCellGraph), OperandStack);
                }
                OperatorStack.Push(opr1);
            }

            // ***** process a right parenthesis *****
            else if (ch == rightparenthesis) {
                opr1 = EqnOperator(ch);
                while(!OperatorStack.StackEmpty() &&
                    (opr2 = OperatorStack.Peek()) >= opr1)
                {
                    opr2 = OperatorStack.Pop();
                    opr2.Evaluate((*(curCellGraph), OperandStack);
                }
                if(OperatorStack.StackEmpty())
                    error(MissingLeftParenthesis);
                opr2 = OperatorStack.Pop(); // get rid of '('
            }
            else if (ch == '\r') {
                libCreateInvertedNode(&OperandStack, CellsList, num_cell);
            }
            // ***** have some invalid input *****
            else if (!iswhitespace(ch))
                error(InvalidInput);
        }
        /* Inner WHILE loop End */
    } // Endof If

    /* the rank of the complete expression must be 1 */
    if (rank != 1)
        error(OperandExpected);

    /* flush operator stack and complete expression evaluation.
    * if find left parenthesis, a right parenthesis is missing.
    */
    while (!OperatorStack.StackEmpty()) // At this stage,
    {
        opr1 = OperatorStack.Pop(); // normally opr = '=', assignment opr
        if (opr1.GetOp() == leftparenthesis) {
            error(MissingRightParenthesis);
        }
        opr1.Evaluate((*(curCellGraph), OperandStack);
    }

    rank = 0;

#ifdef LIB_DEBUG
    cout << endl;
    cout << "==== "<< (*curCellGraph).GetName() << endl;
    (*curCellGraph).PrintPiNodes();
    (*curCellGraph).PrintAllNodes();
#endif

    /* Outer WHILE loop end */
    fin.close();

    // Create RPN equation list for each cell
    Node *bNode = NULL, *outNode = NULL;
    char *out_name = "0";
    LinkedList<Node*> *nodeList, *finList;
    LinkedList<char*> *rpnList;
    int out_pos, num_liter;

    for(CellsList->Reset();!CellsList->EndOfList();CellsList->Next()){
        curCell = CellsList->Data();
        curCellGraph = curCell->GetCellGraph();
        rpnList = curCell->GetFunc();

        // find begin node;
        nodeList = curCellGraph->GetGateList();
        if((out_pos = curCellGraph->FindNode(nodeList, out_name)) != -1){
            nodeList->Reset(out_pos);
            outNode = nodeList->Data();
        }
        else {
            cout << "ERROR !!!\n";
            cout << "In RPN creation routine, can't find node name \"0\".\n";
        }

        finList = outNode->GetFaninList();
        if(finList->ListSize() != 1){
            cout << "ERROR !!!\n";
            cout << "In RPN creation routine for each cell, PO node has";
            cout << " more than one fanin_nodes !!!\n";
        }
        else {
            finList->Reset(0);
            bNode = finList->Data();
        }
        (void) libGetFunctionForBdd(curCellGraph, bNode, num_liter, rpnList);
    }

#ifdef LIB_DEBUG_RPN
    for(CellsList->Reset();!CellsList->EndOfList();CellsList->Next()){
        curCell = CellsList->Data();
        curCellGraph = curCell->GetCellGraph();
        rpnList = curCell->GetFunc(); // at this moment, empty function

        cout << "Cell[" << curCell->GetName() << "]: ";
        for(rpnList->Reset();!rpnList->EndOfList();rpnList->Next()){
            cout << rpnList->Data() << " ";
        }
        cout << "\n\n";
    }
#endif
    delete [] token;
    return CellsList;
}

/***** Subgraph Bdd generation *****/
* Subgraph Bdd_new.cc
* Graph partitioning and BDD generation
*****/
#include <fstream.h>
#include <stdlib.h>
#include <ctype.h> // used for function 'isdigit'
#include <string.h>
#include <stdio.h> // used for function 'isdigit'
#include <math.h>

#include "eqn_stack.h" // include template-based stack class
#include "eqn_opr.h" // defines the MathOperator class
#include "Node.h"
#include "LinkedList.h"
#include "Graph.h"

#include "Queue.h" // used for sug-graph utility
#include "Stack.h" // used for sug-graph utility

```

```

#include "cuddInt.h" // include for bdd generation

// Search Network.allnodes, when a PO or multiple-fanout node is found,
// insert it into a LinkedList<Node *> *mfpAndPo. Do this until all the
// nodes in the list Network.allnodes.
// Returns a LinkedList.
LinkedList<Node *> * GetMultiFanoutAndPoNode(Graph *origNet)
{
    LinkedList<Node *> *gateList, *mfpPoList = new LinkedList<Node *>();
    Node * node;
    int num_fanout;
    NodeFunc nodeFunc;
    char *node_name, *finName;

    LinkedList<Node *> *mfpn_fiList;
    Node *finNode;

    gateList = origNet->GetGateList();

    for(gateList->Reset(); !gateList->EndOfList(); gateList->Next()){
        node = gateList->Data();
        node_name = node->GetName();
        num_fanout = (*node).GetNumFanout();
        nodeFunc = node->GetNodeFunc();
        if(nodeFunc == NODE_PO || (num_fanout > 1 && nodeFunc != NODE_PI &&
            nodeFunc != NODE_INV && nodeFunc != NODE_PO ||
            node->GetNodeType() == DUMMY)){
            if(origNet->FindNode(mfpPoList, node_name) == -1){
                (*mfpPoList).InsertRear(node);
            }
        }
        else if(nodeFunc == NODE_INV){
            mfpn_fiList = node->GetFaninList();
            if(num_fanout > 1){
                if(origNet->FindNode(mfpPoList, node_name) == -1){
                    (*mfpPoList).InsertRear(node);
                }
                if(mfpn_fiList->ListSize() == 1){
                    mfpn_fiList->Reset();
                    finNode = mfpn_fiList->Data();
                    finName = finNode->GetName();
                    if((finNode->GetNodeFunc() != NODE_PI &&
                        (finNode->GetNumFanout() == 1))){
                        if(origNet->FindNode(mfpPoList, finName) == -1){
                            (*mfpPoList).InsertRear(finNode);
                        }
                    }
                }
            }
        }
        else if(num_fanout == 1){ // rv. code : 010901-01
            if(mfpn_fiList->ListSize() == 1){
                finNode = mfpn_fiList->Data();
                finName = finNode->GetName();
                if((finNode->GetNodeFunc() != NODE_PI ||
                    (finNode->GetNodeFunc() != NODE_PI &&
                     (finNode->GetNumFanout() > 1))){
                    if(origNet->FindNode(mfpPoList, node_name) == -1){
                        (*mfpPoList).InsertRear(node);
                    }
                }
            }
        }
    }
    return mfpPoList;
}

// Returns a graph (sub_graph of the given Network).
// Outputs of the sub_graph are multiple fanout points or primary outputs,
// and inputs are multiple fanout points or primary inputs.
// Leveling is also done during bfs traversal.
// Graph * BFS(Graph *NetWork, Node *begin_node)
Graph * GenSubGraph(Node *begin_node)
{
    Graph *sGraph = new Graph;
    Queue<Node *> *q = new Queue<Node *>;
    Node *sNode, *aNode;
    LinkedList<Node *> *finList;
    char *anode_name;
    int level = 0;
    NodeFunc sNodeFunc;

    begin_node->MarkVisit();
    begin_node->SetNodeType(PRIMARY_OUTPUT);
    begin_node->SetLevel(level);
    (*sGraph).InsertPoNode(begin_node);
    Q->QInsert(begin_node);

    while(!Q->QEmpty()){
        sNode = Q->DeQueue();
        sNodeFunc = sNode->GetNodeFunc(); //10-07
        finList = sNode->GetFaninList();
        if(finList->IsEmpty() && sNode->GetNodeFunc() != NODE_PI){
            cerr << "Error::#of fanin = 0 at node ("<<sNode->GetName()<<")\n";
            exit(1);
        }
        level = sNode->GetLevel()+1; // get fanins level of current node
        for(finList->Reset(); !finList->EndOfList(); finList->Next()){
            aNode = finList->Data();
            aNode->SetLevel(level);
            //if(aNode->GetVisit() != 1){
            if(aNode->GetNodeFunc() == NODE_PI || aNode->GetNumFanout() > 1 \
                || aNode->GetNodeFunc() == NODE_PO ||
                (sNodeFunc == NODE_INV && sNode->GetNumFanout() > 1) ){ //10-07
                aNode_name = aNode->GetName();
                //if(sGraph->FindNode(sGraph->GetGateList(), aNode_name) == -1){
                if(sGraph->FindNode(sGraph->GetPiList(), aNode_name) == -1){
                    aNode->SetNodeType(PRIMARY_INPUT);
                    (*sGraph).InsertPiNode(aNode);
                }
            }
            else {
                (*sGraph).InsertGate(aNode);
                Q->QInsert(aNode); // Insert aNode to Q to visit its fanins
            }
        }
    }
}

// Global Depth --> routines are moved to Read_eqn.cc (10-29-00)
// 10-23-00 Initialize global depth for match pin assignment based on
// depth. Here, global_depth in Node CLASS is newly introduced
// Subject Graph generation routine (at Multiple fanout point)
// Returns a LinkedList<Graph *> pointer of Graph Class.
LinkedList<Graph *> *GetSubjectGraphs(Graph *origNet)
{
    LinkedList<Graph *> *gList = new LinkedList<Graph *>;
    Node *sNode;
    Graph *sGraph;
    char *subgraph_name = (char *)calloc(65, sizeof(char));
    LinkedList<Node *> *mfpAndPo;

    (void)rd_depthInitGlobalDepth(origNet);
    mfpAndPo = GetMultiFanoutAndPoNode(origNet);
    int gid=1;

    // Subject graph generation
    for(mfpAndPo->Reset(); !mfpAndPo->EndOfList(); mfpAndPo->Next()){
        sNode = (*mfpAndPo).Data();

        sGraph = GenSubGraph(sNode);
        strcpy(subgraph_name, "");
        sprintf(subgraph_name, "Graph # [%d]", gid);
        sGraph->SetName(subgraph_name);

        (*gList).InsertRear(sGraph);
    }

#ifdef SUBG_DEBUG
    cout<<endl;
    cout<<endl;
    cout<<"**** Graph # ["<<gid<<"]\n";
    (*sGraph).PrintPoNodes();
    (*sGraph).PrintPiNodes();
    (*sGraph).PrintAllNodes();
#endif
    gid++;
    free(subgraph_name);
    return gList;
} // End of subject graph generation

// A calling function passes newGraph, node_list(tList), and root_node(bNode).
// 1) foreach gate_list(tList)'s input, if its fanin is not in gate list
// insert it into newGraph's primary input list.
// 2) Insert the root_node into newGraph's primary output list.
// 3) Insert the elements in gate_list(tList) into newGraph's GateList.
void List2Graph(Graph **newGraph, LinkedList<Node *> *tList, Node *bNode)
{
    LinkedList<Node *> *piList = new LinkedList<Node *>;
    LinkedList<Node *> *fiList;
    Node *cNode, *finNode;
    char *node_name, *bNode_name, *node_nameB;
    int name_len;

    bNode_name = bNode->GetName();
    for(tList->Reset(); !tList->EndOfList(); tList->Next()){
        cNode = tList->Data();
        finList = cNode->GetFaninList();
        for(finList->Reset(); !finList->EndOfList(); finList->Next()){
            finNode = finList->Data();
            node_name = finNode->GetName();
            if(((*newGraph)->FindNode(tList, node_name) == -1 &&
                ((*newGraph)->FindNode(piList, node_name) == -1)){
                piList->InsertRear(finNode);
            }
            else if(finNode->GetNumFanout() > 1){ //02-20-01
                if(((*newGraph)->FindNode(piList, node_name) == -1){
                    piList->InsertRear(finNode);
                }
            }
            else {
                // added for xor3 (ptl_y3) 02-20-01
                name_len = strlen(node_name);
                if(node_name[name_len-1] == '\'){
                    node_nameB = (char *)calloc(name_len+1, sizeof(char));
                    strncpy(node_nameB, node_name, name_len-1);
                    node_nameB[name_len] = '\0';
                }
                else {
                    node_nameB = (char *)calloc(name_len+2, sizeof(char));
                    strcpy(node_nameB, node_name);
                    strcat(node_nameB, "\'");
                }
                if(((*newGraph)->FindNode(piList, node_name) == -1 &&
                    //finNode->GetNodeFunc() != NODE_PO &&
                    ((*newGraph)->FindNode(piList, node_nameB) != -1){
                    piList->InsertRear(finNode);
                }
            }
            free(node_nameB);
        }
    }

    for(piList->Reset(); !piList->EndOfList(); piList->Next()){
        cNode = piList->Data();
        (*newGraph)->InsertPiNode(cNode);
    }

    for(tList->Reset(); !tList->EndOfList(); tList->Next()){

```

```

        cNode = tList->Data();
        node_name = cNode->GetName();
        if(strcmp(node_name, bNode_name)==0){
            (*newGraph)->InsertPoNode(cNode);
        } else {
            if((*newGraph)->FindNode(piList, node_name) == -1){
                (*newGraph)->InsertGate(cNode);
            }
        }
    }
    piList->ClearList();
    delete piList;
}

// Set level of each node using BFS
//void subgInitLevel(Graph *sGraph, Node *begin_node)
void subgInitLevel(Graph *sGraph)
{
    Queue<Node *> Q;
    Node *sNode, *aNode, *bNode;
    LinkedList<Node *> *finList, *allNode, *poList;
    //char *aNode_name, *bNode_name;
    char *bNode_name;
    int level = 0, count = 0;

    // find output Node
    allNode = sGraph->GetGateList();
    poList = sGraph->GetPoList();
    for(allNode->Reset(); !allNode->EndOfList(); allNode->Next()){
        bNode = allNode->Data();
        bNode_name = bNode->GetName();
        if(sGraph->FindNode(poList, bNode_name) != -1 &&
            (bNode->GetNumFanout() > 1 || bNode->GetNodeFunc() == NODE_PO)){
            Q.Insert(bNode);
            count++;
        }
    }
    if(count > 1){
        cout << "In subgInitLevel() : ";
        cout << "# of output node (assumed PO or MFP) is more than ONE!!\n";
        exit(1);
    }
    while(!Q.Empty()){
        sNode = Q.DeQueue();
        finList = sNode->GetFaninList();
        if(finList->isEmpty() && sNode->GetNodeFunc() != NODE_PI){
            cerr << "Error::#of fanin = 0 at node ("<<sNode->GetName()<<")"
                << "in "<<__FILE__<<" line "<<(__LINE__)-2<<endl;
            exit(1);
        }
        level = sNode->GetLevel()+1; // get fanins level of current node
        for(finList->Reset(); !finList->EndOfList(); finList->Next()){
            aNode = finList->Data();
            aNode->SetLevel(level);
            if(!(aNode->GetNodeFunc() == NODE_PI || aNode->GetNumFanout() > 1 ||
                aNode->GetNodeFunc() == NODE_PO)){
                Q.Insert(aNode); // Insert aNode to Q to visit its fanins
            }
        }
    }
}

#ifdef SUBG_INIT_DEBUG
cout << "Sub-graph Name = "<<sGraph->GetName()<<endl;
sGraph->PrintAllNodes();
#endif

void subgCopyNodeList(LinkedList<Node *> *srcList, LinkedList<Node *> *destList)
{
    Node *curNode;
    for(srcList->Reset(); !srcList->EndOfList(); srcList->Next()){
        curNode = srcList->Data();
        destList->InsertRear(curNode);
    }
}

char *subgCopyNodeName(Node *curNode)
{
    char *cur_name, *copied_name;
    cur_name = curNode->GetName();
    copied_name = (char *)calloc(strlen(cur_name)+1, sizeof(char));
    strcpy(copied_name, cur_name);
    return copied_name;
}

// Finds the possible function set from a subject graph based on
// depth constraints
LinkedList<Node *> * subgGetSubListRecursively(char *cur_name, int rootLvl,
    LinkedList<Node *> *sPiList,
    LinkedList<Node *> *gateList, Graph *sGraph,
    Queue<LinkedList<Node *> *> *allQ,
    LinkedList<Node *> *subList, int depth)
{
    LinkedList<Node *> *finList, *newList;
    Node *curNode, *lowNode, *hiNode;
    char *low_name, *hi_name;
    int num_fanin, pos, curLevel, lvl_diff;
    LinkedList<Node *> *subList0, *subList1, *subList2;
    int low_depth, hi_depth;

    if((pos=sGraph->FindNode(gateList, cur_name))!=-1){
        curNode = gateList->Data(pos);
        curLevel = curNode->GetLevel();
        lvl_diff = curLevel - rootLvl;

        if((pos=sGraph->FindNode(sPiList, cur_name))!=-1 || (lvl_diff > depth) ||
            curNode->GetNodeFunc() == NODE_PI){
            return subList;
        } else {
            subList->InsertRear(curNode);

            num_fanin = curNode->GetNumFanin();
            finList = curNode->GetFaninList();
            if(finList->ListSize()==1){
                lowNode = finList->Data(0);
                low_name = subgCopyNodeName(lowNode);

                subList0 = new LinkedList<Node *>;
                (void)subgCopyNodeList(subList, subList0);
                newList =
                    subgGetSubListRecursively(low_name, rootLvl, sPiList, gateList,
                        sGraph, allQ, subList0, depth);

                allQ->QInsert(subList0);
                free(low_name);
                return subList0;
            } else if(finList->ListSize()==2){
                lowNode = finList->Data(0); low_depth = lowNode->GetLevel();
                hiNode = finList->Data(1); hi_depth = hiNode->GetLevel();
                low_name = subgCopyNodeName(lowNode);
                hi_name = subgCopyNodeName(hiNode);

                if(hi_depth > low_depth){
                    subList2 = new LinkedList<Node *>;
                    (void)subgCopyNodeList(subList, subList2);

                    subList0 = new LinkedList<Node *>;
                    (void)subgCopyNodeList(subList, subList0);

                    // hi node
                    newList =
                        subgGetSubListRecursively(hi_name, rootLvl, sPiList, gateList,
                            sGraph, allQ, subList2, depth);

                    allQ->QInsert(subList2);

                    // low node
                    newList =
                        subgGetSubListRecursively(low_name, rootLvl, sPiList, gateList,
                            sGraph, allQ, subList0, depth);

                    allQ->QInsert(subList0);

                    subList1 = new LinkedList<Node *>;
                    (void)subgCopyNodeList(newList, subList1);
                    // low hi node
                    newList =
                        subgGetSubListRecursively(hi_name, rootLvl, sPiList, gateList,
                            sGraph, allQ, subList1, depth);

                    //allQ->QInsert(newList);
                    allQ->QInsert(subList1);
                    free(low_name);
                    free(hi_name);
                    return newList;
                } else {
                    subList2 = new LinkedList<Node *>;
                    (void)subgCopyNodeList(subList, subList2);

                    subList0 = new LinkedList<Node *>;
                    (void)subgCopyNodeList(subList, subList0);

                    // low node
                    newList =
                        subgGetSubListRecursively(low_name, rootLvl, sPiList, gateList,
                            sGraph, allQ, subList2, depth);

                    //allQ->QInsert(newList);
                    allQ->QInsert(subList2);

                    // hi node
                    newList =
                        subgGetSubListRecursively(hi_name, rootLvl, sPiList, gateList,
                            sGraph, allQ, subList0, depth);

                    //allQ->QInsert(newList);
                    allQ->QInsert(subList0);

                    subList1 = new LinkedList<Node *>;
                    (void)subgCopyNodeList(newList, subList1);
                    // low hi node
                    newList =
                        subgGetSubListRecursively(low_name, rootLvl, sPiList, gateList,
                            sGraph, allQ, subList1, depth);

                    //allQ->QInsert(newList);
                    allQ->QInsert(subList1);
                    free(low_name);
                    free(hi_name);
                    return newList;
                }
            } else {
                cout << "Error :: node, "<<cur_name<<" have more than 2 fanins.\n"
                    <<"In "<<__FILE__<<" at line "<<(__LINE__)-2<<endl;
            }
        }
    }
    return subList;
}

// added for pt1_y3->xor3
LinkedList<Node *> *subgGetFullList(Node *bNode, Graph *sGraph, int depth)
{
    LinkedList<Node *> *fullList = new LinkedList<Node *>;
    LinkedList<Node *> *sPiList, *gateList, *finList;
    Node *curNode, *fanNode;
    NodeFunc finFunc;
    Queue<Node *> Q;
    //char *curName, *finName;
    char *curName, *finName;
    int bNLevel, finLevel, pos, lvl_diff;

    sPiList = sGraph->GetPiList();
    gateList = sGraph->GetGateList();
    bNLevel = bNode->GetLevel();

    fullList->InsertRear(bNode);
    Q.Insert(bNode);
    while(!Q.Empty()){

```

```

    curNode = Q.DeQueue();
    curName = curNode->GetName(); // was missing, added 04-03-01
    finList = curNode->GetFaninList();
    for(finList->Reset(); !finList->EndOfList(); finList->Next()){
        finNode = finList->Data();
        finFunc = finNode->GetNodeFunc();
        finLevel = finNode->GetLevel();
        finName = finNode->GetName();
        lvl_diff = finLevel - bLevel;
        if(finFunc != NODE_PO && finFunc != NODE_PI && (lvl_diff <= depth)
            && (pos=sGraph->FindNode(sPiList, curName))!=-1){
            if(sGraph->FindNode(fullList, finName) == -1){
                fullList->InsertRear(finNode);
                Q.Insert(finNode);
            }
        }
    }
    if(fullList->ListSize() <= 30){
        return fullList;
    } else {
        delete fullList;
        return NULL;
    }
}

// Given a sGraph(there is no nfp), and begin_node(root_node),
// 0) begin_node is selected after Topsort by a calling function
// 1) find nodes within depth from begin_node, and insert it into a Set {S}
// 2) Generate power-set of {S}, and
//    select all possible sub-set of {S} except a) signe node which are not
//    root_node, b) sub-sets whos elements are not connected.
//    The selected sub-set {p}, have to have the root_node as a begin_node
//    of the sub-set. ==> #of selected sub-set <= 2^|S|.
// 3) insert each {p}, which are actually sub-graph of sGraph, into a Queue
//    and return the set of sub-graph. The called routine will generate BDD
//    based on this set P = {{p}|p is sub-set of sGraph}.
LinkedList<Graph*> *GetSetOfFuncGraphs(Graph *sGraph, Node *bNode, int depth)
{
    LinkedList<Graph*> *fgList = new LinkedList<Graph*>;
    LinkedList<Node*> *sPiList, *gateList;
    Queue<LinkedList<Node*>*> *allQ = new Queue<LinkedList<Node*>*>;
    LinkedList<Node*> *fullList, *subList = new LinkedList<Node*>;
    Queue<Node*> *Q;

    LStack<Node*> *S;
    char *bName;
    int bLevel;
    LinkedList<Node*> *bNodeFoList;
    NodeFunc bNodeFunc;

    //1) Starting from bNode, generate node lists which are going to be a Graph
    sPiList = sGraph->GetPiList();
    gateList = sGraph->GetGateList();
    bNodeFoList = bNode->GetFanoutList();
    bNodeFunc = bNode->GetNodeFunc();
    bLevel = bNode->GetLevel();
    bName = bNode->GetName();

    LinkedList<Node*> *subList_root = new LinkedList<Node*>;
    if(bNodeFunc == NODE_INV && bNodeFoList->ListSize() >= 2){
        subList_root->InsertRear(bNode);
        allQ->Qinsert(subList_root);
    } else {
        subList_root->InsertRear(bNode);
        allQ->Qinsert(subList_root);
        (void)subGetSubListRecursively(bName, bLevel, sPiList, gateList,
            sGraph, allQ, subList, depth);
        fullList = subGetFullList(bNode, sGraph, depth);
        if(fullList != NULL){
            allQ->Qinsert(fullList);
        }
    }

    /* generating graphs using node lists for the function to generate bdd*/
    LinkedList<Node*> *curList;
    Graph *newGraph;
    char *graph_name;
    int graph_num = -1;

    //2) Build subGraph for an individual lists
    while(!allQ->QEmpty()){
        graph_num++;
        curList = allQ->DeQueue();
        newGraph = new Graph;
        graph_name = newGraph->GetName();
        graph_name = (char *)calloc(strlen(bName)+10, sizeof(char));
        (void)sprintf(graph_name, "%s[%d]", bName, graph_num);
        (*newGraph).SetName(graph_name);
        free(graph_name);

        List2Graph(&newGraph, curList, bNode);
#ifdef DEBUG_FUNC_SET
        cout<<"\n*** "<< newGraph->GetName() <<endl;
        newGraph->PrintPiNodes();
        cout<<endl;
        newGraph->PrintPoNodes();
        cout<<endl;
        newGraph->PrintAllNodes();
        cout<<endl;
        sGraph->PrintNodesOfList(curList);
#endif
        fgList->InsertRear(newGraph);
        delete curList;
    }
    delete allQ;
    return fgList;
}

// Check if all the fanins are visited of a node (bNode)
int isAllFaninVisited(Graph *funcGraph, Node *bNode)
{
    LinkedList<Node*> *faninList;
    Node *aNode = NULL;
    char *aNode_name, *bNode_name;
    int count;

    if(funcGraph->FindNode(funcGraph->GetGateList(), bNode_name) == -1){
        cout << "Node : "<< bNode->GetName() << "cannot find in the \n";
        cout << "funcGraph (" << funcGraph->GetName() << ".\n";
        return(0);
    }
    faninList = bNode->GetFaninList();
    count = 0;
    for(faninList->Reset(); !faninList->EndOfList(); faninList->Next()){
        aNode = faninList->Data();
        aNode_name = aNode->GetName();
        if(funcGraph->FindNode(funcGraph->GetPiList(), aNode_name) != -1 && \
            aNode->GetVisit() > 0) {
            count++;
        }
    }
    if(count == aNode->GetNumFanin()){
        return(1);
    } else {
        return(0);
    }
}

// Reset (-1) visited flag for all nodes in a sub-graph except the nodes in
// PiList of the graph. The visited flag of the nodes in PiList is set to 1.
void ResetVisitFlag(Graph *fGraph)
{
    LinkedList<Node*> *piList, *allNodeList;
    Node *aNode;
    char *aNode_name;

    piList = fGraph->GetPiList();
    allNodeList = fGraph->GetGateList();
    for(allNodeList->Reset(); !allNodeList->EndOfList(); allNodeList->Next()){
        aNode = allNodeList->Data();
        aNode_name = aNode->GetName();
        if(fGraph->FindNode(piList, aNode_name) != -1){
            aNode->MarkVisit(1);
        } else {
            aNode->MarkVisit(-1);
        }
    }
}

// For a given function graph (Graph format), returns 1) a List of char which
// represents a function in Reverse Polish Notation (RPN) using dfs, 2) #of
// literals by passing address of a integer variable(integer pointer).
// -> can determine #of literals (equals to just (list size) - #of operators)
// Input : Graph *fGraph, Node *bNode, int &num_literal
// Returns : LinkedList<char*> *rpn_function.
// The next step of this subroutine is to identify support-set to generate
// BDD. BDD is generated using non-inverted and inverted signals, but
// to generate it in a different variable order (applied for library cells,
// especially PTL-cells) only non-inverted signals can be used.
LinkedList<char*> *GetFunctionForBdd(Graph *fGraph, Node *bNode, int &nLit,
    int bar_bar_delete = 1)
{
    LinkedList<char*> *rpnList = new LinkedList<char*>;
    LStack<Node*> *S;

    LinkedList<Node*> *finList;
    Node *curNode, *aNode;
    char *aNode_name, *tmp_var;
    int visited, num_visit, cur_pos;
    NodeFunc cur_func, finFunc;

    (void)ResetVisitFlag(fGraph);
    nLit = 0;

    if((visited = bNode->GetVisit()) < 0){
        S.Push(bNode);
    } else {
        cout <<" In GetFunctionForBdd(), starting node ("
            <<bNode->GetName()<<")\n has already been"
            <<" visited.\n";
        fGraph->PrintPiNodes();
        fGraph->PrintPoNodes();
        fGraph->PrintAllNodes();
        exit(1);
    }
    while(!S.Empty()){
        curNode = S.Pop();
        finList = curNode->GetFaninList();
        num_visit = 0;
        for(finList->Reset(); !finList->EndOfList(); finList->Next()){
            aNode = finList->Data();
            if((visited = aNode->GetVisit()) > 0)
                num_visit++;
        }
        if(num_visit == curNode->GetNumFanin()){
            for(finList->Reset(); !finList->EndOfList(); finList->Next()){
                aNode = finList->Data();
                aNode_name = aNode->GetName();
                // 02-20-01
                if(fGraph->FindNode(fGraph->GetPiList(), aNode_name) != -1){
                    rpnList->InsertRear(aNode_name);
                    nLit++;
                }
            }
            switch((cur_func = curNode->GetNodeFunc())){
                case NODE_INV:
                    if(bar_bar_delete){ // bar_bar_delete = 1(default)
                        /* 2 consecutive '\'' will be removed */
                        cur_pos = rpnList->CurrentPosition();
                        if((cur_pos) >= 1){

```

```

        rpnList->Reset(cur_pos);
        tmp_var = rpnList->Data();
        if(strcmp(tmp_var, "\\") == 0){
            delete rpnList->DeleteCurrentElem();
        } else {
            rpnList->InsertRear("\\");
        }
    } else {
        rpnList->InsertRear("\\");
    }
} else { // if bar_bar_delete = 0
    rpnList->InsertRear("\\");
}
break;
case NODE_AND: rpnList->InsertRear("&");
break;
case NODE_OR: rpnList->InsertRear("|");
break;
case NODE_NOR: rpnList->InsertRear("~");
break;
case NODE_NAND: rpnList->InsertRear("&~");
break;
default :
    cout<<"Error.....\n";
    cout<<"In GetFunctionForBdd(), node function of "
    <<" Decomposed Graph is not correct "
    <<"\n Expected node functions are NODE_INV, "
    <<" NODE_AND, NODE_NAND, NODE_OR, and NODE_NOR.\n";
    exit(0);
    break;
}
// set visit flag of curNode;
curNode->MarkVisit(1);
} else {
    S.Push(curNode);
    for(finList->Reset(); !finList->EndOfList(); finList->Next()){
        aNode = finList->Data();
        finFunc = aNode->GetNodeFunc();
        if((visited = aNode->GetVisit()) < 0){
            S.Push(aNode);
        }
    }
}
}
// Returns a function at node "bNode" based on fGraph@bNode
return rpnList;
}

int IsOperator(char *str)
{
    if(str == NULL) return (0);
    if(strcmp(str, "\\") == 0 || strcmp(str, "&") == 0 || strcmp(str, "|") == 0){
        return(1);
    } else {
        return(0);
    }
}

int IsConstant(char *str)
{
    if(str == NULL) return (0);
    if(strcmp(str, "VDD") == 0 || strcmp(str, "GND") == 0){
        return(1);
    } else {
        return(0);
    }
}

void PrintCharList(LinkedList<char *> *List)
{
    if(List->ListSize() != 0){
        for(List->Reset(); !List->EndOfList(); List->Next()){
            cout << List->Data() << " ";
        }
    } else {
        cout << "In PrintCharList(), ListSize is zero or negative !!!\n";
    }
    cout << endl;
}

void PrintCharList1(LinkedList<char *> *List)
{
    if(List->ListSize() != 0){
        for(List->Reset(); !List->EndOfList(); List->Next()){
            cout << List->Data() << " ";
        }
    } else {
        cout << "In PrintCharList(), ListSize is zero or negative !!!\n";
    }
}

int subgFindNode(LinkedList<Node *> *gList, char *curVar, int &tmp_pos)
{
    int position = -1;
    Node *curNode;
    char *curName;

    tmp_pos = -1;
    for(gList->Reset(); !gList->EndOfList(); gList->Next()){
        tmp_pos++;
        curNode = gList->Data();
        curName = curNode->GetName();
        cout<<tmp_pos<<" : "<<curName<<"\n";
        if(strcmp(curName, curVar) == 0){
            position = tmp_pos;
            break;
        }
    }
    return position;
}

// Get a support variable of a RPN function

// For a given a function in RPN, returns support variables in a List
LinkedList<char *> *GetSupportVarSet(Graph *fGraph, LinkedList<char *> *rpnFunc,
int &num_suf)
{
    LinkedList<char *> *sufList = new LinkedList<char *>;
    LinkedList<Node *> *gateList; // of original network
    Node *curNode;
    char *curNode_name;

    char *curVar;
    char *tmp_var, *sup_var, ch;
    int found, position = -1;
    int tmp_pos;

    num_suf = 0;
    gateList = fGraph->GetGateList();

    for(rpnFunc->Reset(); !rpnFunc->EndOfList(); rpnFunc->Next()){
        tmp_var = rpnFunc->Data();
        if(IsOperator(tmp_var) || IsConstant(tmp_var)) {
            continue;
        }
        ch = tmp_var[strlen(tmp_var)-1];
        if(strlen(tmp_var) <= 64){
            curVar = (char *)calloc(strlen(tmp_var)+1, sizeof(char));
        } else {
            cout<<"Variable name("<<tmp_var<<" is too long (max 64 chars).\n";
            cout<<"in "<<__FILE__<<" at line "<<__LINE__<<endl;
        }
        if(ch == '\\'){
            strcpy(curVar, tmp_var);
            curVar[strlen(tmp_var)-1] = '\\0';
        } else {
            strcpy(curVar, tmp_var);
        }

        // find non-inverted node and retrieve the node_name
        // from the original net.
        if((position = fGraph->FindNode(gateList, curVar)) != -1){
            gateList->Reset(position);
            curNode = gateList->Data();
            curNode_name = curNode->GetName();
        } else {
            fGraph->PrintPiNodes();
            fGraph->PrintPoNodes();
            fGraph->PrintAllNodes();
            cout<<"\n"; PrintCharList(rpnFunc);
            position = subgFindNode(gateList, curVar, tmp_pos);
            cout<<"In GetSupportVarSet(), "<< curNode_name << " cannot find\n";
            cout<<"tmp_var = "<< tmp_var << "\n";
            cout<<"In GetSupportVarSet(), "<< curVar << " cannot find\n";
            cout<<"Position = "<<position<<" : tmp_pos = "<<tmp_pos<<endl;
            cout<<"in "<<__FILE__<<" at line "<<__LINE__<<"\n";
            exit(1);
        }
        free(curVar);

        // search suplist to if curVar is already in the List(sufList)
        found = 0;
        if(!sufList->IsEmpty()){
            for(sufList->Reset(); !sufList->EndOfList(); sufList->Next()){
                tmp_var = sufList->Data();
                if(strcmp(curNode_name, tmp_var) == 0) found = 1;
            }
        }

        int pos;
        if(!found){
            if(sufList->ListSize() == 0){
                sufList->InsertRear(curNode_name);
            } else if(sufList->ListSize() == 1){
                pos = sufList->ListSize()-1;
                while(pos >= 0){
                    sufList->Reset(pos);
                    sup_var = sufList->Data();
                    if(strcmp(sup_var, curNode_name) > 0){
                        if(pos == 0){
                            sufList->InsertFront(curNode_name);
                            break;
                        } else {
                            pos--;
                        }
                    } else {
                        sufList->InsertRight(curNode_name);
                        break;
                    }
                }
            }
        }
        num_suf++;
    }
    return sufList;
}

//----- Bdd handling routines -----
//DdNode *GenerateBddVar(DdManager *bddm, LinkedList<char *> *sufSet,
//char *curVar )
//DdNode *GenerateBddVar(DdManager *bddm, LinkedList<char *> *sufSet,
//char *curVar)
{
    DdNode *temp = NULL;
    char *tmp_var, ch;
    //char *nonInvVar = new char[20];
    char *nonInvVar;
    int var_id, num_matches=0;

    ch = curVar[strlen(curVar)-1];
    nonInvVar = (char *)calloc(strlen(curVar)+1, sizeof(char));
    if(ch == '\\'){
        strcpy(nonInvVar, curVar, strlen(curVar)-1);
    }
}

```

```

        nonInvVar[strlen(curVar)-1] = '\0';
    } else {
        strcpy(nonInvVar, curVar);
    }

    int comp;
    for(sufSet->Reset(); !sufSet->EndOfList(); sufSet->Next()){
        tmp_var = sufSet->Data();
        if(strcmp(tmp_var, nonInvVar)==0){
            num_matches++;
            if(num_matches > 1) {
                cout << "Variable (" << curVar << ") "
                    << "matched more than one times in GenerateBddVar()\n";
                exit(1);
            }
            var_id = sufSet->CurrentPosition();
            if(ch != '\0'){
                temp = Cudd_bddIthVar(bddm, var_id);
                comp = Cudd_IsComplement(temp);
                Cudd_Ref(temp);
            } else {
                temp = Cudd_Not(Cudd_bddIthVar(bddm, var_id));
                Cudd_Ref(temp);
            }
            break;
        }
    }
    free(nonInvVar);
    return temp;
}

//DdNode *GetBdd(Graph *Ckt, DdManager *bddm, LinkedList<char *> *rpnFunc)
DdNode *GetBdd(DdManager *bddm, LinkedList<char *> *rpnFunc,
    LinkedList<char *> *sufSet)
{
    LinkedList<char *> *supportSet;
    DdNode *f, *f1, *tmp1, *tmp2, *zero, *one;
    LStack<DdNode *> S;
    char *curData;

    one = DD_ONE(bddm);
    zero = Cudd_Not(DD_ONE(bddm));
    supportSet = sufSet;

    for(rpnFunc->Reset(); !rpnFunc->EndOfList(); rpnFunc->Next()){
        curData = rpnFunc->Data();
        if(!IsOperator(curData) && !IsConstant(curData)){
            tmp1 = GenerateBddVar(bddm, supportSet, curData);
            S.Push(tmp1);
        } else if(IsConstant(curData)==1){
            if(strcmp(curData, "VDD")==0){
                S.Push(one);
            } else {
                S.Push(zero);
            }
        } else {
            if(strcmp(curData, "\'')==0){
                tmp1 = S.Pop();
                f = Cudd_Not(tmp1);
                Cudd_Ref(f);
                S.Push(f);
                Cudd_RecursiveDeref(bddm, tmp1);
            } else if(strcmp(curData, "*")==0){
                tmp1 = S.Pop(); //==Cudd_Ref(tmp2);
                tmp2 = S.Pop(); //==Cudd_Ref(tmp2);
                f = Cudd_bddIte(bddm, tmp1, tmp2, zero);
                Cudd_Ref(f);
                S.Push(f);
                Cudd_RecursiveDeref(bddm, tmp1);
                Cudd_RecursiveDeref(bddm, tmp2);
            } else if(strcmp(curData, "+")==0){
                tmp1 = S.Pop(); //==Cudd_Ref(tmp2);
                tmp2 = S.Pop(); //==Cudd_Ref(tmp2);
                f = Cudd_bddIte(bddm, tmp1, one, tmp2);
                Cudd_Ref(f);
                S.Push(f);
                Cudd_RecursiveDeref(bddm, tmp1);
                Cudd_RecursiveDeref(bddm, tmp2);
            }
        }
    }
    f1 = S.Pop();
    return f1;
}

// Bdd equivalence checking routine, returns (1) if f = g, or returns (0)
// This routine check if the both bdds have same children recursively.
//int BddEqCheck(DdManager *bddm, DdNode *f, DdManager *bddmg, DdNode *g)
int BddEqCheck(DdNode *f, DdNode *g)
{
    DdNode *fT, *fE, *gT, *gE;
    int fVal, gVal;

    if(Cudd_IsConstant(f) && Cudd_IsConstant(g)){
        if(Cudd_IsComplement(f) && Cudd_IsComplement(g)){
            fVal = (int)(Cudd_Regular(f)->type.value);
            gVal = (int)(Cudd_Regular(g)->type.value);
            if(fVal == gVal) {
                return (1);
            }
        } else {
            return (0);
        }
    } else if(!Cudd_IsComplement(f) && !Cudd_IsComplement(g)){
        fVal = (int)(f->type.value);
        gVal = (int)(g->type.value);
        if(fVal == gVal) {
            return (1);
        }
    } else {
        return (0);
    }
}

return (1);
} else {
    return (0);
}
} else {
    return (0);
}
}

else if(!Cudd_IsConstant(f) && Cudd_IsConstant(g) ||
    (Cudd_Regular(f)->index != Cudd_Regular(g)->index) ||
    Cudd_IsConstant(f) && !Cudd_IsConstant(g) ) {
    return (0);
} else {
    if(Cudd_IsComplement(f)){
        fT = cuddE(Cudd_Not(f));
        fE = cuddT(Cudd_Not(f));
        if(Cudd_IsComplement(g)){
            gT = cuddE(Cudd_Not(g));
            gE = cuddT(Cudd_Not(g));
            return(BddEqCheck(fT, gT) && BddEqCheck(fE, gE));
        }
        else {
            return (0);
        }
    }
    else {
        fT = cuddT(f); fE = cuddE(f);
        if(Cudd_IsComplement(g)){
            return (0);
        }
        else {
            gT = cuddT(g); gE = cuddE(g);
            return(BddEqCheck(fT, gT) && BddEqCheck(fE, gE));
        }
    }
}
}

// Symmetry check for variables x[i], x[j], here x[i] and x[j] are just
// index of variables used in a bdd F.
// Before call this routine, the variable index x or y should checked in order
// to protect fetching wrong bdd pointer from the dd_manager.
int IsSymmetry(DdManager *bddm, DdNode *f, int x, int y)
{
    DdNode *fx, *fx_, *fxy, *fxy_, *fx_y, *fx_y_;
    DdNode *ddx, *ddy;

    ddx = Cudd_bddIthVar(bddm, x);
    ddy = Cudd_bddIthVar(bddm, y);

    fx = Cudd_Cofactor(bddm, f, ddx);
    fx_ = Cudd_Cofactor(bddm, f, Cudd_Not(ddx));

    fxy = Cudd_Cofactor(bddm, fx, ddy);
    fxy_ = Cudd_Cofactor(bddm, fx, Cudd_Not(ddy));

    fx_y = Cudd_Cofactor(bddm, fx_, ddy);
    fx_y_ = Cudd_Cofactor(bddm, fx_, Cudd_Not(ddy));

    if(BddEqCheck(fxy_, fx_y)){
        Cudd_RecursiveDeref(bddm, ddx); Cudd_RecursiveDeref(bddm, ddy);
        Cudd_RecursiveDeref(bddm, fx); Cudd_RecursiveDeref(bddm, fx_);
        Cudd_RecursiveDeref(bddm, fxy); Cudd_RecursiveDeref(bddm, fxy_);
        Cudd_RecursiveDeref(bddm, fx_y); Cudd_RecursiveDeref(bddm, fx_y_);
        return(1); // positive symmetry
    }
    else if(BddEqCheck(fxy, fx_y_)){
        Cudd_RecursiveDeref(bddm, ddx); Cudd_RecursiveDeref(bddm, ddy);
        Cudd_RecursiveDeref(bddm, fx); Cudd_RecursiveDeref(bddm, fx_);
        Cudd_RecursiveDeref(bddm, fxy); Cudd_RecursiveDeref(bddm, fxy_);
        Cudd_RecursiveDeref(bddm, fx_y); Cudd_RecursiveDeref(bddm, fx_y_);
        return(2); // negative symmetry
    }
    else {
        Cudd_RecursiveDeref(bddm, ddx); Cudd_RecursiveDeref(bddm, ddy);
        Cudd_RecursiveDeref(bddm, fx); Cudd_RecursiveDeref(bddm, fx_);
        Cudd_RecursiveDeref(bddm, fxy); Cudd_RecursiveDeref(bddm, fxy_);
        Cudd_RecursiveDeref(bddm, fx_y); Cudd_RecursiveDeref(bddm, fx_y_);
        return(0); // not a symmetry variable
    }
}

}

//-- Symmetry sets
// returns #of variables in SymmSets, *nset is #of symm set.
int GetSymmSet(DdManager *bddm, DdNode *f, LinkedList<char *> *suppSet, \
    LinkedList<char *> *newArItem, int *nset)
{
    LinkedList<char *> arrayItem[10];
    char *var1, *var2;
    int var1Id, var2Id, symm;
    Queue<char *> Q;

    int index = -1;
    // check if the List is support set of a function.
    for(suppSet->Reset(); !suppSet->EndOfList(); suppSet->Next()){
        var1 = suppSet->Data();
        Q.Insert(var1);
    }
    // compare support var[i] which is in Queue and support var[j], j!=i.
    while(!Q.Empty()){
        var1 = Q.DeQueue();
        if(IsOperator(var1)){
            cout << "The argument List is not supportSet of a function\n";
            cout << "because it contains "<< var1 << "\n";
            exit(1);
        }
    }
}

```

```

    }
    index++; // increase symmetry set list index
    arrayItem[index].InsertRear(var1);
    var1Id = index;
    for(suppSet->Reset(); !suppSet->EndOfList(); suppSet->Next()){
        var2 = suppSet->Data();
        if(strcmp(var1, var2) != 0){
            var2Id = suppSet->CurrentPosition();
            symm = IsSymmetry(bddm, f, var1Id, var2Id);
            if(symm){ // positive and negative symmetry
                arrayItem[index].InsertRear(var2);
            }
        }
    }
}

#ifdef DEBUG_Symm
for(arrayItem[index].Reset(); !arrayItem[index].EndOfList();
    arrayItem[index].Next()){
    cout << arrayItem[index].Data() << " ";
}
cout << endl;
#endif

// delete list that duplicates another
//int n, listSize, listSize1, char_index;
int n, listSize, listSize1;
char *chr1, *chr2;
int dup_list[10], dup_count; // Here, I assumed that max{|suf|} <= 10.
Queue<char > Q2;

if(index > 10){
    cout << "Support variables are too many in GetSymmetrySet()\n";
    exit(1);
}
for(n=0; n<10; n++) dup_list[n] = 0;
for(n = 0; n <= index-1; n++){
    listSize = arrayItem[n].ListSize();
    for(int k = 1; k <= index; k++){
        listSize1 = arrayItem[k].ListSize();
        dup_count = 0;
        if(listSize == listSize1 && !(n==k) && dup_list[n] == 0){
            for(arrayItem[n].Reset(); !arrayItem[n].EndOfList();
                arrayItem[n].Next()){
                chr1 = arrayItem[n].Data();
                Q2.QInsert(chr1);
            }
            while(!Q2.QEmpty()){
                chr1 = Q2.DeQueue();
                for(arrayItem[k].Reset(); !arrayItem[k].EndOfList();
                    arrayItem[k].Next()){
                    chr2 = arrayItem[k].Data();
                    if(strcmp(chr1, chr2) == 0){
                        dup_count++;
                    }
                }
            }
            if((dup_count == listSize) && (dup_count == listSize1)) {
                dup_list[k] = 1;
            }
        }
    } // inner for
}

// copy arrayItem to newArItem and delete arrayItem.
int new_index = -1; // new ArItem's index
int num_var_of_symm_set = 0;
for(n=0; n<=index; n++){
    if(dup_list[n] != 1){
        new_index++;
        for(arrayItem[n].Reset(); !arrayItem[n].EndOfList();
            arrayItem[n].Next()){
            newArItem[new_index].InsertRear(arrayItem[n].Data());
        }
        num_var_of_symm_set += newArItem[new_index].ListSize();
    }
}

#ifdef DEBUG_Symm
cout << "Index = "<< index << " New_index = "<< new_index << endl;
for(n = 0; n <= new_index; n++){
    PrintCharList(&newArItem[n]);
}
#endif

*nsset = new_index+1;
return num_var_of_symm_set;
}

/*****
* Permute.cc
* Subroutines for BDD variable re-ordering
*****/
#include <fstream.h>
#include <stdio.h>
#include "LinkedList.h"

// UpperLimit is maximum number of elements to permute
const int UpperLimit = 8;

// copy n element array y to array x
void copy(int x[], int y[], int n)
{
    for(int i=0; i<n; i++){
        x[i] = y[i];
    }
}

```

```

// Permlist is an n elements array of integers. generate the permutations
// of the elements whose indices are in the range start <= i <= n-1. when
// a permutation is complete, print the entire array. to permute all n
// elements, begin with start=0.
void getPermute(int permlist[], int start, int n, LinkedList<int > *pList)
{
    int *tmparr = new int [UpperLimit];
    int temp, i;

    // stopping condition: land at last array elements
    if(start == n-1){
        int *permItem = new int[UpperLimit];

#ifdef PERM_TEST
        // print the permutation
        for(i=0; i < n; i++){
            cout << permlist[i] << " ";
        }
        cout << "\n";
#endif

        copy(permItem, permlist, n);
        pList->InsertRear(permItem);
    } else {
        // recursive step: exchange permlist[start] and permlist[i],
        // make a copy of the array in tmparr, and permute elements of
        // tmparr from start+1 through end of the array
        for(i=start; i < n; i++){
            // exchange permlist[i] with permlist[start]
            temp = permlist[i];
            permlist[i] = permlist[start];
            permlist[start] = temp;

            // create a new list and call permute
            copy(tmparr, permlist, n);
            getPermute(tmparr, start+1, n, pList);
        }
    }
    delete [] tmparr;
}

#ifdef TEST_main
void main(void)
{
    // permlist contains the n number we permute
    int *permlist = new int [UpperLimit];
    int n, i;

    LinkedList<int > *pList = new LinkedList<int >;

    cout << "Enter a number 'n' between a and "
        << UpperLimit << " ";
    cin >> n;

    // initialize permlist to {1, 2, ... ,n}
    for(i=0; i < n; i++){
        permlist[i] = i+1;
    }

    // print permutations fo elements in array
    // permlist from indices 0 to n-1
    getPermute(permlist, 0, n, pList);

    cout << "Data from pList\n";
    int index = 1;
    for(pList->Reset(); !pList->EndOfList(); pList->Next()){
        permlist = pList->Data();
        printf("%3d : ", index++);
        for(i = 0; i < n; i++){
            cout << permlist[i] << " ";
        }
        cout << "\n";
    }
}
#endif

/*****
* Match_depth.cc
* 1. Check match between cluster and cell
* 2. Builds match list at every nodes
*****/
#include <fstream.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <new.h>

// For a given subject-graph, search all nodes in the graph and insert the
// node into a Queue if it's not PI or PO.
// Returns the Queue to the caller to get possible functions for each node.
LStack<Node > *matchGetGateListOfSubGraph(Graph * sGraph)
{
    LStack<Node > * nodeS = new LStack<Node >;
    Queue<Node > * tmpQ;

    LinkedList<Node > * nodeList, * finList, * poList, * piList;
    Node * curNode, * tNode;
    char * curNode_name;

    nodeList = sGraph->GetGateList();
    poList = sGraph->GetPoList();
    piList = sGraph->GetPiList();

    for(nodeList->Reset(); !nodeList->EndOfList(); nodeList->Next()){
        curNode = nodeList->Data();
        curNode_name = curNode->GetName();
        if(sGraph->FindNode(poList, curNode_name) != -1){
            tmpQ.QInsert(curNode);
        }
    }
}

```

```

    }
    if(tmpQ.QLength() > 1){
        cout << "In matchGetGateListOfSubGraph(), there are more than";
        cout << " one outputs. Graph name is [" << sGraph->GetName();
        cout <<"]\n";
    }
}
curNode = tmpQ.QFront();
if(curNode->GetNodeFunc() != NODE_PD) {
    // since NODE_PD has same func as its fanin.
    nodeS->Push(curNode);
}
while(!tmpQ.QEmpty()){
    curNode = tmpQ.DeQueue();
    finList = curNode->GetFaninList();
    if(finList->ListSize() != 0){
        for(finList->Reset(); !finList->EndOfList(); finList->Next()){
            tNode = finList->Data();
            curNode_name = tNode->GetName();
            // if it is not in Pilist and in GateList
            if(sGraph->FindNode(pilist, curNode_name) == -1){
                tmpQ.QInsert(tNode);
                nodeS->Push(tNode);
            }
        }
    }
}
return nodeS;
}
// Print symmetry sets
void PrintSymmSet(LinkedList<char*> *symmSet, int num_symm)
{
    cout << "\n SymmSet :: "<< num_symm << "\n";
    for(int n = 0; n < num_symm; n++){
        PrintCharList(&symmSet[n]);
    }
}
void matchDeleteCharLinkedList(LinkedList<char*> *charList, int ld = 0)
{
    int listSize, n;
    listSize = charList->ListSize();
    if(listSize < 1) return;
    for(n=0; n<listSize; n++){
        charList->Reset();
        delete charList->DeleteCurrentElem();
    }
    if(ld == 1){
        delete charList;
    }
}
void matchDeleteDCharLinkedList(LinkedList<char*> *charList, int ld = 0)
{
    int listSize, n;
    listSize = charList->ListSize();
    if(listSize < 1) return;
    for(n=0; n<listSize; n++){
        charList->Reset();
        free(charList->DeleteCurrent());
    }
    if(ld == 1){
        delete charList;
    }
}
void MemFreeForSymmSet(LinkedList<char*> *symmSet, int num_symm)
{
    for(int n = 0; n<num_symm; n++){
        (void)matchDeleteCharLinkedList(&symmSet[n]);
    }
}
void MemFree_Bdd_SymmSet(DdManager *fbddm, DdManager *gbddm,
    DdNode *f, DdNode *g,
    LinkedList<char*> *symmSetF, int num_f_symm,
    LinkedList<char*> *symmSetG, int num_g_symm)
{
    Cudd_RecursiveDeref(fbddm, f);
    Cudd_RecursiveDeref(gbddm, g);
    (void) MemFreeForSymmSet(symmSetF, num_f_symm);
    (void) MemFreeForSymmSet(symmSetG, num_g_symm);
    Cudd_Quit(fbddm);
    Cudd_Quit(gbddm);
}
// Copy rpn
LinkedList<char*> * matchCopyRpn(LinkedList<char*> *old_rpn)
{
    LinkedList<char*> *new_rpn = new LinkedList<char*>;
    char *testVar;
    for(old_rpn->Reset(); !old_rpn->EndOfList(); old_rpn->Next()){
        testVar = old_rpn->Data();
        new_rpn->InsertRear(testVar);
    }
    return new_rpn;
}
// This routine modifies the phase of the function in rpn notation
// by looking at phaseFunction(integer) which is considered as binary value.
LinkedList<char*> *matchModifyVarPhase(LinkedList<char*> *rpnG,
    LinkedList<char*> *supSetG, int phaseFunc)
{
    LinkedList<char*> *new_rpnG = new LinkedList<char*>;
    int polarity = 0;
    char *targetVar, *testVar;
    // copy rpnG to new_rpnG
    for(rpnG->Reset(); !rpnG->EndOfList(); rpnG->Next()){
        testVar = rpnG->Data();
        new_rpnG->InsertRear(testVar);
    }
    if(phaseFunc > 0){ // if phaseFunc = 0, no variables are inverted.
        for(int n = 0; n < supSetG->ListSize(); n++){
            polarity = phaseFunc & 1<<n; // test bit position "n"
            if(polarity){
                (void)supSetG->Reset(n);
                targetVar = supSetG->Data();
                for(new_rpnG->Reset(); !new_rpnG->EndOfList(); new_rpnG->Next()){
                    testVar = new_rpnG->Data();
                    if(strcmp(targetVar, testVar) == 0){
                        new_rpnG->InsertRight("\'");
                    }
                }
            }
        }
    }
    return new_rpnG;
}
// Returns a List that contains arrays as its elements.
// Each element in the array is the index of the elements of symmetry set.
// Array size is equal to "num_set_elem"(symmetry set)
LinkedList<int*> *matchGetPermutationList(int num_g_symm)
{
    LinkedList<int*> *pList = new LinkedList<int*>;
    int *permArray = new int [8]; // max #of elements of symmSetG = 8.
    // initialize permutation array
    for(int i = 0; i < num_g_symm; i++){
        permArray[i] = i;
    }
    (void)getPermute(permArray, 0, num_g_symm, pList); // result is in pList
    delete [] permArray;
    return pList;
}
// Returns ordered supportset based on permutation array (ordArray).
LinkedList<char*> *matchGetNewOrderOfSupSetG(LinkedList<char*> *symmSetG,
    int *ordArray, int num_g_symm)
{
    LinkedList<char*> *orderList = new LinkedList<char*>;
    LinkedList<char*> *symmList;
    int index;
    // Get one permutation --> Actual variable ordering
    for(int i = 0; i < num_g_symm; i++){
        index = ordArray[i];
        symmList = &symmSetG[index];
        for(symmList->Reset(); !symmList->EndOfList(); symmList->Next()){
            orderList->InsertRear(symmList->Data());
        }
    }
    return orderList;
}
// Check Unateness, returns (0, positive unate), (1, neg. unate), (2, binate)
// in a function.
int matchCheckUnateness(LinkedList<char*> *rpnFunc, char *checkingVar)
{
    int unateness = 0, position;
    char *tmp_char;
    LinkedList<char*> *rpnCopy;
    // copy linkedlist to array;
    for(rpnFunc->Reset(); !rpnFunc->EndOfList(); rpnFunc->Next()){
        tmp_char = rpnFunc->Data();
        rpnCopy->InsertRear(tmp_char);
    }
    for(rpnFunc->Reset(); !rpnFunc->EndOfList(); rpnFunc->Next()){
        tmp_char = rpnFunc->Data();
        if(strcmp(tmp_char, checkingVar) == 0){
            position = rpnFunc->CurrentPosition();
        }
    }
    return unateness;
}
// general linked list;
int matchFindNameInGenList(LinkedList<char*> *genList, char *f_name)
{
    int position = -1;
    if((genList->ListSize() != 0)){
        for(genList->Reset(); !genList->EndOfList(); genList->Next()){
            if(strcmp(f_name, genList->Data()) == 0){
                position = genList->CurrentPosition();
                return position;
            }
        }
    }
    return position;
}
// Only for functions in Rpn format
int matchFindNameInRpnF(LinkedList<char*> *rpnF, char *f_name)

```

```

{
    int position = -1;
    char *prime;

    char *rpn_var, *tmp_var, *tmp_var1;
    LinkedList<char *> *rpnFunc = new LinkedList<char *>;

    if((rpnF->ListSize()!=0) && (rpnF->ListSize() > 2)){
        // copy rpnF to tmp_rpnFunc; added on May 12, 2000
        // to make correct rpn_func = d ' e * , etc...
        for(rpnF->Reset();!rpnF->EndOfList();rpnF->Next()){
            rpn_var = rpnF->Data();
            tmp_var = (char *)calloc(strlen(rpn_var)+1, sizeof(char));
            strcpy(tmp_var, rpn_var);

            if(strcmp(rpn_var, "\'") != 0){
                rpnFunc->InsertRear(tmp_var);
            } else if(strcmp(rpn_var, "\'") == 0){
                //pos = tmp_rpnG->CurrentPosition();
                //tmp_rpnG->Reset(pos-1);
                tmp_var1 = rpnFunc->Data();
                strcat(tmp_var1, rpn_var);
            }
        }

        for(rpnFunc->Reset();!rpnFunc->EndOfList();rpnFunc->Next()){
            if(strcmp(f_name, rpnFunc->Data()) == 0){
                position = rpnFunc->CurrentPosition();
                return position;
            }
        }
    } else if(rpnF->ListSize() == 2){ // Inverter function
        prime = &f_name[strlen(f_name)-1];
        if(!strcmp(prime, "\'")){
            return (1);
        }
    }

    (void)matchDeleteCharLinkedList(rpnFunc);
    delete rpnFunc;
    return position;
}

int matchIsCellOutputPhaseNegative(LibCell *curCell)
{
    char *cell_name;
    int negative = 0;

    cell_name = curCell->GetName();
    if(!strcmp(cell_name, "nor2") || !strcmp(cell_name, "nand2") ||
        !strcmp(cell_name, "nor3") || !strcmp(cell_name, "nand3") ||
        !strcmp(cell_name, "aoi2i") || !strcmp(cell_name, "inv1") ||
        // 11-01-01
        !strcmp(cell_name, "aoi22") ||
        !strcmp(cell_name, "oi2i") || !strcmp(cell_name, "inv2")){
        negative = 1;
    }
    return negative;
}

// Check if the var is binate, in library func (rpnG) binate
int matchIsVarBinate(LinkedList<char *> *rpnG, char *g_name)
{
    int count = 0;

    for(rpnG->Reset();!rpnG->EndOfList();rpnG->Next()){
        if(strcmp(rpnG->Data(), g_name) == 0){
            count++;
        }
    }
    if(count%2 == 0){
        return (1);
    } else {
        return (0);
    }
}

// Check rpn function if the list elemrnt is VDD or GND at listPos-1 (current
// List positio -1) or listPos+1.
// ex) E B ' * ' ' D B ' * ' ' + ' ' A ' * ' ' C A ' * ' ' + => 3-unate, 2-binate
// GND b * b ' d * * + a * a ' GND e * e ' g * * * + => 4-unate, 1-binate
// GND b * b ' d ' * * + a * a ' VDD e * e ' g * * * + => 4-unate, 1-binate
int matchCheckConstVarInRpn(LinkedList<char *> *rpnL, int listPos)
{
    char *prev_var=NULL, *next_var=NULL, *tmp_var;
    int prev_new = 0, next_new = 0, result = 0;
    int cur_pos, listSize;

    // Backward search
    if(listPos-1 >= 0){
        rpnL->Reset(listPos-1);
        prev_var = rpnL->Data();
    } else {
        prev_var = (char *)calloc(2, sizeof(char));
        prev_new = 1;
    }

    // Forward search
    listSize = rpnL->ListSize();
    cur_pos = listPos+1;
    if(cur_pos <= listSize){
        while(cur_pos <= listSize-1){
            rpnL->Reset(cur_pos);
            next_var = rpnL->Data();
            if(strcmp(next_var, "\'") == 0){
                cur_pos++;
                continue;
            }
        }
    }

    } else {
        break;
    }
} else {
    next_var = (char *)calloc(2, sizeof(char));
    next_new = 1;
}

if(IsConstant(prev_var)){
    result = 1;
    /* to check 'a ' ' GND e ' when current listPos points "a"
    * ex) GND b * b ' d ' * * + a * a ' ' GND e ' * e ' ' g * * * +
    */
} else if(IsConstant(next_var)){
    if((cur_pos+1) <= listSize-1){
        rpnL->Reset(cur_pos+1);
        tmp_var = rpnL->Data();
        if(!IsOperator(tmp_var)){
            result = 0;
        } else {
            result = 1;
        }
    } else {
        result = 1;
    }
} else {
    result = 0;
}

rpnL->Reset(listPos);
if(prev_new) free(prev_var);
if(next_new) free(next_var);

return result;
}

// The support set elements must be extracted from rpnL
//int matchIsVarBinatex(LinkedList<char *> *rpnG, char *g_name)
int matchIsVarBinatexS(LinkedList<char *> *rpnL, char *g_name)
{
    int pos_unate[10], neg_unate[10], name_len, listPos, pos_count;
    int var_match_count, n, bar_cnt;
    int posUnate, negUnate;
    //char *sup_var, *tmp_var, *rpn_var;
    char *tmp_var, *rpn_var;
    int result = 0;

    for(n=0;n<10;n++){
        pos_unate[n] = 0; neg_unate[n] = 0;
    }
    var_match_count = 0;
    posUnate = negUnate = 0;
    for(rpnL->Reset();!rpnL->EndOfList();rpnL->Next()){
        tmp_var = rpnL->Data();
        name_len = strlen(tmp_var);
        if(!IsOperator(tmp_var) && !IsConstant(tmp_var)){
            rpn_var = (char *)calloc(name_len+1, sizeof(char));
            if(name_len>1 && tmp_var[name_len-1] == '\') {
                strcpy(rpn_var, tmp_var, name_len-1);
                rpn_var[name_len] = '\0';
            } else {
                strcpy(rpn_var, tmp_var);
            }
        } else {
            continue;
        }

        if(strcmp(g_name, rpn_var) == 0){
            listPos = rpnL->CurrentPosition();
            if(matchCheckConstVarInRpn(rpnL, listPos)) continue;
            pos_count = listPos;
            var_match_count++;
            if(tmp_var[name_len-1] == '\') {
                // check next item in the rpn_List, if it is ' or * or +
                while(pos_count < (rpnL->ListSize()-1)){
                    rpnL->Next(); pos_count++;
                    //bar_cnt = 0;
                    if(strcmp(tmp_var = rpnL->Data(), "\'") == 0){
                        bar_cnt++;
                    }
                    if(bar_cnt%2 == 0){
                        neg_unate[var_match_count-1] = 1;
                        pos_unate[var_match_count-1] = 0;
                    } else {
                        neg_unate[var_match_count-1] = 0;
                        pos_unate[var_match_count-1] = 1;
                    }
                    if(!strcmp(tmp_var, "\'") == 0) break;
                }
                rpnL->Reset(listPos);
            } else {
                bar_cnt = 0;
                while(pos_count < (rpnL->ListSize()-1)){
                    rpnL->Next(); pos_count++;
                    if(strcmp(tmp_var = rpnL->Data(), "\'") == 0){
                        bar_cnt++;
                    }
                    if(bar_cnt%2 == 1){
                        neg_unate[var_match_count-1] = 1;
                        pos_unate[var_match_count-1] = 0;
                    } else {
                        neg_unate[var_match_count-1] = 0;
                        pos_unate[var_match_count-1] = 1;
                    }
                    if(!strcmp(tmp_var, "\'") == 0) break;
                }
                rpnL->Reset(listPos);
            }
        }
    }
}

```

```

    }
    free(rpn_var);
} // End of for_loop
for(n=0; n < var_match_count; n++){
    if(pos_unate[n]==1){
        posUnate = 1;
    }else if(neg_unate[n]==1){
        negUnate = 1;
    }
}
if(posUnate && negUnate){
    return result = 1;
}else if(!posUnate && negUnate || posUnate && !negUnate){
    return result = 0;
}
return result;
}
// check if a variable phase in function of G (library) is modified.
int matchIsVarPhaseInverted(LibCell *curCell, LinkedList<char *> *rpnG,
                             char *g_name)
{
    int t1_flag = 0, t2_flag = 0;
    char *rpn_var, *tmp_var, *tmp_var1;
    char *tVar1, *tVar2;
    LinkedList<char *> tmp_rpnG;
    // copy rpnG to tmp_rpnG; (debugging--ok)
    for(rpnG->Reset(); !rpnG->EndOfList(); rpnG->Next()){
        rpn_var = rpnG->Data();
        //cout<<"tmp_var = "<<tmp_var<<"\n";
        tmp_var = (char *)calloc(strlen(rpn_var)+2, sizeof(char));
        strcpy(tmp_var, rpn_var);
        if(strcmp(rpn_var, "\'')!=0){
            tmp_rpnG.InsertRear(tmp_var);
        } else if(strcmp(rpn_var, "\'')==0){
            tmp_var1 = tmp_rpnG.Data();
            strcat(tmp_var1, rpn_var);
            free(tmp_var);
        }
    }
    // check if a variable is inverted, i.e., variable phase is modified
    // during matching.
    tVar1 = (char *)calloc(strlen(g_name)+2, sizeof(char));
    tVar2 = (char *)calloc(strlen(g_name)+3, sizeof(char));
    strcpy(tVar1, g_name); strcat(tVar1, "\'');
    strcpy(tVar2, g_name); strcat(tVar2, "\'');
    for(tmp_rpnG.Reset(); !tmp_rpnG.EndOfList(); tmp_rpnG.Next()){
        rpn_var = tmp_rpnG.Data();
        if(matchIsVarBinate(rpnG, g_name)){
            if(!strcmp(rpn_var, tVar1) && (t1_flag==0)){
                t1_flag = 1;
            } else if(!strcmp(rpn_var, tVar2) && (t2_flag==0)){
                t2_flag = 1;
            }
        } else {
            if(!strcmp(rpn_var, tVar1) && (t1_flag==0)){
                t1_flag = 1;
            }
        }
    }
    (void)matchDeleteDCharLinkedList(&tmp_rpnG);
    free(tVar1);
    free(tVar2);
    if(matchIsVarBinate(rpnG, g_name)){
        return (t2_flag==1 && t1_flag==1);
    } else {
        return (t1_flag==1);
    }
}
// check if a variable phase in function of G (library) is modified.
// ONLY for non-control variables (ex: c, d, f, g of pyl_y3,
int matchIsVarPhaseInverted1(LibCell *curCell, LinkedList<char *> *rpnG,
                              char *g_name)
{
    char *rpn_var, *tmp_var, *tmp_var1;
    char *tVar1, *tVar2;
    LinkedList<char *> tmp_rpnG;
    int flag = 0, n;
    int pos = -1, count=-1, posArr[10];
    for(rpnG->Reset(); !rpnG->EndOfList(); rpnG->Next()){
        pos++;
        rpn_var = rpnG->Data();
        if(!strcmp(rpn_var, g_name)){
            count++;
            posArr[count] = pos;
        }
    }
    for(n = 0; n <= count; n++){
        pos = posArr[n];
        rpn_var = rpnG->Data(pos+1);
        if(!strcmp(rpn_var, "\'')){
            flag = 1;
        }
    }
    return flag;
}
// Change Pin name to reflect binate variable phase inversion.

void matchChangePinMap(LinkedList<pinMap *> *pinMapList, char *n1, char *n2)
{
    pinMap *pinInfo;
    if(pinMapList->ListSize()==0) return; // 0731
    for(pinMapList->Reset(); !pinMapList->EndOfList(); pinMapList->Next()){
        pinInfo = pinMapList->Data();
        if(!strcmp(pinInfo->cell_pin, n1)){
            strcpy(pinInfo->cell_pin, n2);
        }
    }
}

int matchIsNodeNameInverted(char *target_name)
{
    int n, name_len, count=0;
    char ch;
    name_len = strlen(target_name);
    for(n = 0; n<name_len; n++){
        ch = target_name[n];
        if(ch=='\') {
            count++;
        }
        if(count%2 == 0){
            return FALSE;
        } else {
            return TRUE;
        }
    }
}
// Check if all pins are same or not for both lists.
int matchMappedPinsAreEqual(LinkedList<pinMap *> *oldPinList,
                             LinkedList<pinMap *> *newPinList)
{
    pinMap *oldPin, *newPin;
    char *old_name, *new_name;
    int count, listSize;
    listSize = oldPinList->ListSize();
    count = 0;
    for(oldPinList->Reset(); !oldPinList->EndOfList(); oldPinList->Next()){
        oldPin = oldPinList->Data();
        old_name = oldPin->net_name;
        for(newPinList->Reset(); !newPinList->EndOfList(); newPinList->Next()){
            newPin = newPinList->Data();
            new_name = newPin->net_name;
            if(strcmp(old_name, new_name)==0){
                count++;
            }
        }
    }
    if(count == listSize){
        return (1);
    } else {
        return (0);
    }
}
// Check if current "match" is already in targetNode's matchList,
// if so, return (1), otherwise return (0).
int matchCurrentMatchIsDuplicated(LinkedList<Match *> *matchList, Match *match)
{
    Match *oldMatch;
    LinkedList<pinMap *> *newPinList, *oldPinList;
    char *new_match_name, *old_match_name;
    int duplicated = 0;
    new_match_name = match->cell_name;
    newPinList = match->pinmap;
    if(matchList->ListSize()==0) return (0);
    for(matchList->Reset(); !matchList->EndOfList(); matchList->Next()){
        oldMatch = matchList->Data();
        old_match_name = oldMatch->cell_name;
        if(strcmp(old_match_name, new_match_name)==0){
            oldPinList = oldMatch->pinmap;
            if(matchMappedPinsAreEqual(oldPinList, newPinList)){
                duplicated = 1;
                break;
            }
        }
    }
    return duplicated;
}
// Find binate variables whose polarity are inverted from orig func in libCell,
// and insert it to invlist.
void matchFindInvertedBinateVariable(LinkedList<char *> *supSetG,
                                      LinkedList<char *> *rpnG, LibCell *curCell,
                                      LinkedList<char *> *invList)
{
    char *g_name;
    for(supSetG->Reset(); !supSetG->EndOfList(); supSetG->Next()){
        g_name = supSetG->Data();
        if(matchIsVarBinate(rpnG, g_name)){ //appears two times
            if(matchIsVarPhaseInverted(curCell, rpnG, g_name)){
                invList->InsertRear(g_name);
            }
        }
    }
}
void matchInsertStrapInfo2PinList(LinkedList<char *> *strapT,
                                   LinkedList<pinMap *> *pinMapList)
{
    Queue<char *> Q;
    char *tmpchar;
    pinMap *strapInfo;
    if(strapT->ListSize()!=0){
        for(strapT->Reset(); !strapT->EndOfList(); strapT->Next()){

```



```

    delete sq;
}
}

// 11-25-00
// This routine changes support set-G's ordering based on symm set soze of F
// Ex) a match happens when : symmSetF={n4}, {i1 i3}, and supF={n4 i1 i3},
//      SymmsetG={b d}, {a}, and supG = {b d a}
//      after a match happens, but the correct order of support sets are
//      supF = {n4 i1 i3} and supG = {a b d}.
//      circuit example : c17
// 11-19-01
// if the order of supSetF has been changed, return (1)
int matchSupportSetFreorderBasedOnDepth(LinkedList<char *> *supSetF,
    LinkedList<char *> *symmSetF, int num_f_symm, Graph *origNet)
{
    LinkedList<char *> *symmList;
    Queue<char *> *Q = new Queue<char *>;
    char *symm_var, *f_name, *nf_name;
    int n, net_pos=-1, gLevel, ogLevel=0, pos;

    LinkedList<Node *> *gateList;
    Node *curNode;
    LinkedList<char *> *tmpList;

    LinkedList<char *> oldSetF; //11-19-01

    for_each(supSetF){
        oldSetF.InsertRear(supSetF->Data());
    }

    gateList = origNet->GetGateList();
    for(n=0; n<num_f_symm; n++){
        symmList = &symmSetF[n];
        if(symmList->ListSize() > 1){
            tmpList = new LinkedList<char *>;
            for(symmList->Reset(); !symmList->EndOfList(); symmList->Next()){
                symm_var = symmList->Data();
                Q->QInsert(symm_var);
            }
            while(!Q->QEmpty()){ // sort
                f_name = Q->DeQueue();
                if((net_pos=origNet->FindNode(gateList, f_name))!=-1){
                    curNode = gateList->Data(net_pos);
                    gLevel = curNode->GetGLevel();
                    if(tmpList->ListSize()==0){
                        tmpList->InsertRear(f_name);
                    }
                    pos = tmpList->ListSize()-1;
                    nf_name = tmpList->Data(pos);
                    while(pos>0){
                        if((net_pos=origNet->FindNode(gateList,
                            nf_name))!=-1){
                            curNode = gateList->Data(net_pos);
                            ogLevel = curNode->GetGLevel();
                        }
                        else {
                            cout<<"Error : "<<"at "<<__LINE__
                                <<"in "<<__FILE__<<"\n";
                        }
                        if(gLevel > ogLevel){
                            if(pos == 0){
                                tmpList->InsertFront(f_name);
                                break;
                            }
                            else {
                                pos--;
                            }
                        }
                        else {
                            tmpList->InsertRight(f_name);
                            break;
                        }
                    }
                }
            }
        }
        // end_of_sort
        (void)matchDeleteCharLinkedList(&symmSetF[n]);
        for(tmpList->Reset(); !tmpList->EndOfList(); tmpList->Next()){
            f_name = tmpList->Data();
            //symmList->InsertRear(f_name);
            (&symmSetF[n])->InsertRear(f_name);
        }
        (void)matchDeleteCharLinkedList(tmpList, 1);
    }
    // end_if

    // replace supSetF
    (void)matchDeleteCharLinkedList(supSetF);
    for(n=0; n<num_f_symm; n++){
        symmList = &symmSetF[n];
        for(symmList->Reset(); !symmList->EndOfList(); symmList->Next()){
            symm_var = symmList->Data();
            supSetF->InsertRear(symm_var);
        }
    }
    delete Q;

    // check if org order and new order are different (11-19-01)
    string new_str, old_str;
    int ret_val = 0;
    for(supSetF->Reset(), oldSetF.Reset(); !supSetF->EndOfList();
        supSetF->Next(), oldSetF.Next()){
        new_str.erase(); old_str.erase();
        new_str += supSetF->Data();
        old_str += oldSetF.Data();
        if(new_str.compare(old_str)){
            ret_val = 1;
            break;
        }
    }

    }
    return ret_val;
}

void matchCopyPinName(char *dest_ptr, char *src_ptr)
{
    int src_len;
    src_len = strlen(src_ptr);
    dest_ptr = (char *)realloc(dest_ptr, src_len+3);
    strcpy(dest_ptr, src_ptr);
}

// Get sub-network graph (cluster func) polarity by checking
// rpnF's last two elements. Because the last elem in rpnF should
// be either '\n' or (+/*).
// ex) if .... (+/*) ==> positive, ... (+/*) ' --> negative
// 11-08-01
void matchGetRpnFuncPolarity(LinkedList<char *> *rpnF, string& rpnF_pol)
{
    char *last_char, *last_1;
    int size;

    size = rpnF->ListSize();
    last_char = rpnF->Data(size-1);
    last_1 = rpnF->Data(size-2);

    rpnF_pol.erase();
    if(!string(last_char).compare("\n")){
        if(size > 2){
            if(!string(last_1).compare("*") || !string(last_1).compare("+")){
                rpnF_pol += "negative";
            }
            else {
                cout<<"Error :: RPN format is not correct ....\n";
                cout<<"In "<<__FILE__<<" at line "<<__LINE__<<"\n";
                exit(0);
            }
        }
        else {
            rpnF_pol += "negative";
        }
    }
    else{
        rpnF_pol += "positive";
    }
}

//11-08-01
int matchGetSTMatchPolarity(Match *match, string& match_pol)
{
    char *cell;
    int ret_value = 0;
    cell = match->cell_name;
    match_pol.erase();

    if(!strcmp(cell, "nand", 4) || !strcmp(cell, "nor", 3) ||
        !strcmp(cell, "aoi", 3) || !strcmp(cell, "oi", 3) ||
        !strcmp(cell, "inv", 3)){
        ret_value = 1;
        match_pol += "negative";
    }
    else if(!strcmp(cell, "and", 3) || !strcmp(cell, "or", 2) ||
        !strcmp(cell, "ao", 2) || !strcmp(cell, "oa", 2) ){
        ret_value = 0;
        match_pol += "positive";
    }
    else {
        cout<<"n*****\n";
        cout<<"Error :: Unsupported cell in Mixed PTL mapping: "<<cell<<"\n";
        cout<<"in "<<__FILE__<<" at line "<<__LINE__<<"\n";
        cout<<"n*****\n";
        exit(0);
    }
    return ret_value;
}

//11-08-01
// if f = s17GAT5 s27GAT8' * ' s227GAT99' + ', ??
int matchVariableInRpnFInvertedX(LinkedList<char *> *rpnF, char *f_name)
// Only works for 2-input gates (nand2, nor2,...)
{
    int ret_value = 0, found = 0;
    int size, len, n;
    string var, tmp_var, f_str, fn_str;

    f_str.erase(); f_str += f_name;
    fn_str.erase(); fn_str = f_str + "\n";

    if(!f_str.compare("s330GAT151")){
        cout<<"\n";
    }

    size = rpnF->ListSize();
    for(n=0; n < size; n++){
        tmp_var.erase();
        tmp_var += rpnF->Data(n);

        if(!tmp_var.compare("\n") || !tmp_var.compare("+") ||
            !tmp_var.compare("*")){
            continue;
        }

        len = tmp_var.size();
        var.erase();
        if((n+1 < size) && !string(rpnF->Data(n+1)).compare("\n")){
            if(tmp_var.substr(len-1,1) == "\n"){
                var = tmp_var.substr(0, len-1);
            }
            else {
                var = tmp_var + "\n";
            }
        }
        else {
            var = tmp_var;
        }
    }
}

```

```

        if(!f_str.compare(var) || !fn_str.compare(var)){
            found = 1;
            break;
        }
    }
    if(found){
        if(!fn_str.compare(var)){
            ret_value = 1;
        }
    } else {
        cout << "\n*****\n";
        cout << "Error :: A variable in support set F is not in rpnF: "
             << f_name << "\n";
        cout << "in "<< _FILE_ << " at line "<< _LINE_ << "\n";
        cout << "*****\n";
        exit(0);
    }
}
return ret_value;

int matchCheckNumInversionsX(LinkedList<char*> *rpnF, int& n, int& size)
{
    int kk, numinv=0;
    string var_n, var_n1;
    for(kk=n+1; kk < size-2; kk++){
        var_n.erase();
        var_n1.erase();
        var_n += rpnF->Data(kk);
        var_n1 += rpnF->Data(kk+1);
        if((!var_n.compare("+") || !var_n1.compare("*")) && !var_n1.compare("\n")){
            numinv++;
        }
    }
    return numinv;
}

int matchCheckNumInversions(Graph *fnGraph, Node *tgtNode, string& cur_str)
{
    int pos, numinv=0;
    LinkedList<Node*> *sGateList;
    LinkedList<Node*> *curFoList;
    Queue<Node*> Q;
    Node *curNode, *foNode;
    NodeFunc *fFunc, *foFunc, *tgtFunc;
    char *cur_name, *fo_name;
    string q_str, fo_str, tgt_str;
    int num_fanout;
    cur_name = (char *)cur_str.c_str();
    tgt_str.erase();
    tgt_str = tgtNode->GetName();
    tgtFunc = tgtNode->GetNodeFunc();
    sGateList = fnGraph->GetGateList();
    if((pos = fnGraph->FindNode(sGateList, cur_name)) != -1){
        curNode = sGateList->Data(pos);
        Q.Insert(curNode);
    } else {
        cout << "\n*****\n";
        cout << "Error :: Current node("<<cur_str
             <<") cannot be found in funcGraph.\n";
        cout << "in "<< _FILE_ << " at line "<< _LINE_ << "\n";
        cout << "*****\n";
        exit(0);
    }
    while(!Q.Empty()){
        curNode = Q.DeQueue();
        fFunc = curNode->GetNodeFunc();
        curFoList = curNode->GetFanoutList();
        num_fanout = curFoList->ListSize();
        q_str.erase();
        q_str += curNode->GetName();
        if(!q_str.compare(tgt_str)) break;
        if(num_fanout >= 1){
            for each(curFoList){
                foNode = curFoList->Data(i);
                fo_name = foNode->GetName();
                if((pos=fnGraph->FindNode(sGateList, fo_name)) != -1){
                    if(strcmp(fo_name,q_str.c_str())==0){
                        cout << "Error : fanin and fanout name are the same !!!\n";
                        cout << "cur_name="<<q_str.c_str()<< " fo_name="<<fo_name<< "\n";
                        cout << "in "<< _FILE_ << " at line "<< _LINE_ << "\n";
                        exit(0);
                    }
                    break;
                }
            }
        } else {
            cout << " num_fanout = "<<num_fanout<< "\n";
            cout << "Error :: Less than one fanout at Node ("<<cur_str<< "\n";
            cout << "in "<< _FILE_ << " at line "<< _LINE_ << "\n";
            exit(0);
        }
        fo_str.erase();
        fo_str += foNode->GetName();
        /* Ccheck root_node(target_node) if
        * 1) If fFunc (- (NAND, NOR) and tgtFunc = INV, because this combination
        * leads to a non-inverting function, ex) (a' b)' --> a' b *
        */
        /* during matching stage the graph contains only:
        * NAND2 NOR2 AND2 NOR2 INV
        */
        foFunc = foNode->GetNodeFunc();
        if(fo_str.compare(tgt_str) && {foFunc==NODE_NAND||foFunc==NODE_NOR}){
            numinv++;
        } else if(foFunc==NODE_INV){
            if(fo_str.compare(tgt_str)){
                numinv++;
            } else if(foFunc==NODE_NAND||foFunc==NODE_NOR||foFunc==NODE_INV){
                numinv++;
            }
        }
    }
    return numinv;
}

// if f = s17GAT5 s27GAT9' + ' s227GAT99' + ', ??
int matchVariableInRpnFinverted(LinkedList<char*> *rpnF, char *f_name,
                                Graph *fnGraph, Node *tgtNode)
{
    int ret_value = 0, found = 0;
    int size, len, n;
    int num_inv, var_len;
    string var, tmp_var, f_str, fn_str;
    f_str.erase(); f_str += f_name;
    fn_str.erase(); fn_str = f_str + "\n";
    if(!f_str.compare("s330GAT130")){
        cout << "\n";
    }
    size = rpnF->ListSize();
    for(n=0; n < size; n++){
        tmp_var.erase();
        tmp_var += rpnF->Data(n);
        if(!tmp_var.compare("(") || tmp_var.compare("+") ||
            !tmp_var.compare("*")){
            continue;
        }
        len = tmp_var.size();
        var.erase();
        if((n+1 < size) && !string(rpnF->Data(n+1)).compare("\n")){
            if(tmp_var.substr(len-1,1) == "\n"){
                var = tmp_var.substr(0,len-1);
            } else {
                var = tmp_var + "\n";
            }
        } else {
            var = tmp_var;
        }
        if(!f_str.compare(var) || !fn_str.compare(var)){
            // Check the number of inversions for "var"
            num_inv = matchCheckNumInversions(fnGraph, tgtNode, var);
            #ifdef MATCH_DEBUG
            cout << "var = "<<var<< " Num inversons = "<< num_inv << "\n";
            #endif
            if(num_inv%2==1){
                var_len = var.size();
                if(var.substr(var_len-1,1)=="\n"){
                    var = var.substr(0,var_len-1);
                } else {
                    var += "\n";
                }
            }
            found = 1;
            break;
        }
    }
    if(found){
        if(!fn_str.compare(var)){
            ret_value = 1;
        }
    } else {
        cout << "\n*****\n";
        cout << "Error :: A variable in support set F is not in rpnF: "
             << f_name << "\n";
        cout << "in "<< _FILE_ << " at line "<< _LINE_ << "\n";
        cout << "*****\n";
        exit(0);
    }
    return ret_value;
}

// Static cell pin assignment
void matchStPinMapping(LinkedList<char*> *supSetG, LinkedList<char*> *supSetF,
                      Match *match, LibCell *curCell, Node *tgtNode,
                      LinkedList<char*> *rpnG, LinkedList<char*> *rpnF,
                      Graph *fnGraph)
{
    LinkedList<pinMap*> *pinMapList;
    char *f_name, *g_name;
    char *fn_name, *target_name;
    pinMap *mapPin;
    int num_sup;
    string rpnF_pol, match_pol;
    target_name = tgtNode->GetName();
    pinMapList = match->pinmap;
    (void)matchGetRpnFuncPolarity(rpnF, rpnF_pol);
    (void)matchGetStMatchPolarity(match, match_pol);
    num_sup = supSetG->ListSize();
    for(supSetG->Reset(), supSetF->Reset(); !supSetG->EndOfList();
        supSetG->Next(), supSetF->Next()){
        mapPin = new pinMap;
        strcpy(mapPin->cell_pin, (g_name = supSetG->Data()));
        f_name = supSetF->Data();
    }
}

```

```

// determine polarity of mapPin->net_node
fn_name = (char *)calloc(strlen(f_name)+2, sizeof(char));
strcpy(fn_name, f_name);
strcat(fn_name, "\\0");

if(strcmp(curCell->GetName(), "inv", 3) == 0){
    if(matchIsNodeNameInverted(target_name)){
        strcpy(mapPin->net_node, f_name); //
    }else{
        strcpy(mapPin->net_node, f_name);
    }
} else {
    if(!rpnF_pol.compare(match_pol)){
        if(matchVariableInRpnFInverted(rpnF, f_name, fnGraph, tgtNode)){
            strcpy(mapPin->net_node, f_name);
        }else{
            strcpy(mapPin->net_node, f_name);
        }
    }else{
        if(matchVariableInRpnFInverted(rpnF, f_name, fnGraph, tgtNode)){
            strcpy(mapPin->net_node, f_name);
        }else{
            strcpy(mapPin->net_node, f_name);
        }
    }
}
pinMapList->InsertRear(mapPin);
free(fn_name);
}
}

// PTL Type1 (y1) pin mapping
void matchY1PinMap(LinkedList<char *> *supSetG, LinkedList<char *> *supSetF,
    Match *match, LibCell *curCell, Node *tgtNode,
    LinkedList<char *> *rpnG, LinkedList<char *> *invList)
{
    LinkedList<pinMap *> *pinMapList;
    char *f_name, *g_name;
    char *fn_name, *target_name;
    pinMap *mapPin;

    target_name = tgtNode->GetName();
    pinMapList = match->pinmap;

    for(supSetG->Reset(), supSetF->Reset(); !supSetG->EndOfList();
        supSetG->Next(), supSetF->Next()){
        mapPin = new pinMap;
        strcpy(mapPin->cell_pin, (g_name = supSetG->Data()));
        f_name = supSetF->Data();

        // determine polarity of mapPin->net_node
        fn_name = (char *)calloc(strlen(f_name)+2, sizeof(char));
        strcpy(fn_name, f_name); strcat(fn_name, "\\");

        if(!strcmp(g_name, "a")){
            strcpy(mapPin->net_node, f_name);
            free(fn_name);
        } else {
            if(matchIsVarPhaseInverted(curCell, rpnG, g_name)){
                strcpy(mapPin->net_node, f_name);
            } else {
                strcpy(mapPin->net_node, f_name);
                free(fn_name);
            }
        }
        if(matchFindNameInGenList(invList, "a") != -1){
            if(!strcmp(g_name, "c")){
                strcpy(mapPin->cell_pin, "b");
                (void)matchChangePinMap(pinMapList, "b", /*to*/ "c");
            } else if(!strcmp(g_name, "b")){
                strcpy(mapPin->cell_pin, "c");
                (void)matchChangePinMap(pinMapList, "c", /*to*/ "b");
            }
        }
    }
    pinMapList->InsertRear(mapPin);
}

// PTL Typeix (yix) pin mapping
void matchYixPinMap(LinkedList<char *> *supSetG, LinkedList<char *> *supSetF,
    Match *match, LibCell *curCell, Node *tgtNode,
    LinkedList<char *> *rpnG, LinkedList<char *> *invList)
{
    LinkedList<pinMap *> *pinMapList;
    char *f_name, *g_name;
    char *fn_name, *target_name;
    pinMap *mapPin, *xorPin;

    target_name = tgtNode->GetName();
    pinMapList = match->pinmap;

    for(supSetG->Reset(), supSetF->Reset(); !supSetG->EndOfList();
        supSetG->Next(), supSetF->Next()){
        mapPin = new pinMap;
        strcpy(mapPin->cell_pin, (g_name = supSetG->Data()));
        f_name = supSetF->Data();

        fn_name = (char *)calloc(strlen(f_name)+2, sizeof(char));
        strcpy(fn_name, f_name); strcat(fn_name, "\\");

        if(!strcmp(g_name, "a")){
            strcpy(mapPin->net_node, f_name);
            free(fn_name);
        } else if(!strcmp(g_name, "b")){
            if(matchFindNameInGenList(invList, "a") != -1){
                strcpy(mapPin->net_node, f_name);
            } else {
                strcpy(mapPin->net_node, f_name);
                free(fn_name);
            }
        }
    }
}

xorPin = new pinMap;
strcpy(xorPin->cell_pin, "c");
if(matchFindNameInGenList(invList, "a") != -1){
    strcpy(xorPin->net_node, f_name);
} else {
    strcpy(xorPin->net_node, f_name);
}
pinMapList->InsertRear(xorPin);
pinMapList->InsertRear(mapPin);
}

// PTL Type2 (y2) pin mapping
void matchY2PinMap(LinkedList<char *> *supSetG, LinkedList<char *> *supSetF,
    Match *match, LibCell *curCell, Node *tgtNode,
    LinkedList<char *> *rpnG, LinkedList<char *> *invList)
{
    LinkedList<pinMap *> *pinMapList;
    char *f_name, *g_name;
    char *fn_name, *target_name;
    pinMap *mapPin;

    target_name = tgtNode->GetName();
    pinMapList = match->pinmap;

    for(supSetG->Reset(), supSetF->Reset(); !supSetG->EndOfList();
        supSetG->Next(), supSetF->Next()){
        mapPin = new pinMap;
        strcpy(mapPin->cell_pin, (g_name = supSetG->Data()));
        f_name = supSetF->Data();

        // determine polarity of mapPin->net_node
        fn_name = (char *)calloc(strlen(f_name)+2, sizeof(char));
        strcpy(fn_name, f_name); strcat(fn_name, "\\");

        if(matchIsVarBinate(rpnG, g_name)){
            strcpy(mapPin->net_node, f_name);
            free(fn_name);
        } else {
            if(matchIsVarPhaseInverted(curCell, rpnG, g_name)){
                strcpy(mapPin->net_node, f_name);
            } else {
                strcpy(mapPin->net_node, f_name);
                free(fn_name);
            }
        }
        if(matchFindNameInGenList(invList, "b") != -1){
            if(!strcmp(g_name, "c")){
                strcpy(mapPin->cell_pin, "d");
                (void)matchChangePinMap(pinMapList, "d", /*to*/ "c");
            } else if(!strcmp(g_name, "d")){
                strcpy(mapPin->cell_pin, "c");
                (void)matchChangePinMap(pinMapList, "c", /*to*/ "d");
            }
        }
    }
    pinMapList->InsertRear(mapPin);
}

// PTL Type2x (y2x) pin mapping
void matchY2xPinMap(LinkedList<char *> *supSetG, LinkedList<char *> *supSetF,
    Match *match, LibCell *curCell, Node *tgtNode,
    LinkedList<char *> *rpnG, LinkedList<char *> *invList)
{
    LinkedList<pinMap *> *pinMapList;
    char *f_name, *g_name;
    char *fn_name, *target_name;
    pinMap *mapPin, *xorPin;

    target_name = tgtNode->GetName();
    pinMapList = match->pinmap;

    for(supSetG->Reset(), supSetF->Reset(); !supSetG->EndOfList();
        supSetG->Next(), supSetF->Next()){
        mapPin = new pinMap;
        strcpy(mapPin->cell_pin, (g_name = supSetG->Data()));
        f_name = supSetF->Data();

        // determine polarity of mapPin->net_node
        fn_name = (char *)calloc(strlen(f_name)+2, sizeof(char));
        strcpy(fn_name, f_name); strcat(fn_name, "\\");

        if(!strcmp(g_name, "a") || !strcmp(g_name, "b")){
            strcpy(mapPin->net_node, f_name);
            free(fn_name);
        } else if(!strcmp(g_name, "e")){
            if(matchIsVarPhaseInverted(curCell, rpnG, g_name)){
                strcpy(mapPin->net_node, f_name);
            } else {
                strcpy(mapPin->net_node, f_name);
                free(fn_name);
            }
        }
        if(!strcmp(g_name, "c")){
            if((matchFindNameInGenList(invList, "b") != -1 &&
                matchIsVarPhaseInverted(curCell, rpnG, g_name)) ||
                (matchFindNameInGenList(invList, "b") != -1 &&
                matchIsVarPhaseInverted(curCell, rpnG, g_name))){
                strcpy(mapPin->net_node, f_name);
            }
        }
        xorPin = new pinMap;
        strcpy(xorPin->cell_pin, "d");
        strcpy(xorPin->net_node, f_name);
        pinMapList->InsertRear(xorPin);
    } else if((matchFindNameInGenList(invList, "b") != -1 &&
        !matchIsVarPhaseInverted(curCell, rpnG, g_name)) ||
        (matchFindNameInGenList(invList, "b") != -1 &&
        !matchIsVarPhaseInverted(curCell, rpnG, g_name))){
    }
}

```

```

        (matchFindNameInGenList(invList,"b")==-1 &&
        matchIsVarPhaseInverted(curCell, rpnG, g_name))) {
    strcpy(mapPin->net_node, f_name);

    xorPin = new pinMap;
    strcpy(xorPin->cell_pin, "d");
    strcpy(xorPin->net_node, fn_name);
    pinMapList->InsertRear(xorPin);
}
}
pinMapList->InsertRear(mapPin);
}
}

void matchStrapChange(LinkedList<pinMap *> *pinMapList,
                    char *net_name, char *g_name)
{
    pinMap *mapPin;
    char *netNode, *cellPin;

    for(pinMapList->Reset(); !pinMapList->EndOfList(); pinMapList->Next()) {
        mapPin = pinMapList->Data();
        netNode = mapPin->net_node;
        cellPin = mapPin->cell_pin;
        if(!strcmp(netNode, g_name)) {
            strcpy(netNode, net_name);
        }
    }

    // PTL Type3 (y3) pin mapping
    void matchY3PinMap(LinkedList<char *> *supSetG, LinkedList<char *> *supSetF,
                    Match *match, LibCell *curCell, Node *tgtNode,
                    LinkedList<char *> *rpnG, LinkedList<char *> *invList)
    {
        LinkedList<pinMap *> *pinMapList;
        char *f_name, *g_name;
        char *fn_name, *target_name;
        pinMap *mapPin;
        int nsupG;

        target_name = tgtNode->GetName();
        pinMapList = match->pinmap;
        nsupG = supSetG->ListSize();

        for(supSetG->Reset(), supSetF->Reset(); !supSetG->EndOfList();
            supSetG->Next(), supSetF->Next()) {
            mapPin = new pinMap;
            strcpy(mapPin->cell_pin, (g_name = supSetG->Data()));
            f_name = supSetF->Data();

            // determine polarity of mapPin->net_node
            fn_name = (char *)calloc(strlen(f_name)+2, sizeof(char));
            strcpy(fn_name, f_name); strcat(fn_name, "\n");

            if(nsupG == 3) {
                if(!strcmp(g_name, "a")) {
                    strcpy(mapPin->net_node, f_name);
                    pinMapList->InsertRear(mapPin);
                    free(fn_name);
                } else if(!strcmp(g_name, "b")) {
                    strcpy(mapPin->net_node, f_name);
                    (void)matchStrapChange(pinMapList, f_name, g_name);
                    free(fn_name);
                } else {
                    if(matchIsVarPhaseInverted(curCell, rpnG, g_name)) {
                        strcpy(mapPin->net_node, fn_name);
                        (void)matchStrapChange(pinMapList, fn_name, g_name);
                    } else {
                        strcpy(mapPin->net_node, f_name);
                        (void)matchStrapChange(pinMapList, f_name, g_name);
                        free(fn_name);
                    }
                }
                if(matchFindNameInGenList(invList, "b") != -1) {
                    if(!strcmp(g_name, "c")) {
                        strcpy(mapPin->cell_pin, "d");
                        (void)matchChangePinMap(pinMapList, "d", /*to*/ "c");
                        // for "f" and "g"
                        (void)matchChangePinMap(pinMapList, "f", /*to*/ "x");
                        (void)matchChangePinMap(pinMapList, "g", /*to*/ "f");
                        (void)matchChangePinMap(pinMapList, "x", /*to*/ "g");
                    }
                }
            }
            pinMapList->InsertRear(mapPin);
        } else if(nsupG > 3) {
            if(matchIsVarBinate(rpnG, g_name)) {
                strcpy(mapPin->net_node, f_name);
                free(fn_name);
            } else {
                if(matchIsVarPhaseInverted(curCell, rpnG, g_name)) {
                    strcpy(mapPin->net_node, fn_name);
                } else {
                    strcpy(mapPin->net_node, f_name);
                    free(fn_name);
                }
            }
            if(matchFindNameInGenList(invList, "b") != -1) {
                if(!strcmp(g_name, "d")) {
                    strcpy(mapPin->cell_pin, "c");
                    (void)matchChangePinMap(pinMapList, "c", /*to*/ "d");
                } else if(!strcmp(g_name, "c")) {
                    strcpy(mapPin->cell_pin, "d");
                    (void)matchChangePinMap(pinMapList, "d", /*to*/ "c");
                }
            }
        } else if(matchFindNameInGenList(invList, "e") != -1) {
            if(!strcmp(g_name, "f")) {
                strcpy(mapPin->cell_pin, "g");
            }
        }
    }
}

(void)matchChangePinMap(pinMapList, "g", /*to*/ "f");
} else if(!strcmp(g_name, "g")) {
    strcpy(mapPin->cell_pin, "f");
    (void)matchChangePinMap(pinMapList, "f", /*to*/ "g");
}
}
}
pinMapList->InsertRear(mapPin);
} /* end of if */
} /* End of For */

// PTL Type3x (y3x) pin mapping
void matchY3xPinMap(LinkedList<char *> *supSetG, LinkedList<char *> *supSetF,
                    Match *match, LibCell *curCell, Node *tgtNode,
                    LinkedList<char *> *rpnG, LinkedList<char *> *invList)
{
    LinkedList<pinMap *> *pinMapList;
    char *f_name, *g_name;
    char *fn_name, *target_name;
    pinMap *mapPin, *xorPin;
    int nsupG;

    target_name = tgtNode->GetName();
    pinMapList = match->pinmap;
    nsupG = supSetG->ListSize();

    for(supSetG->Reset(), supSetF->Reset(); !supSetG->EndOfList();
        supSetG->Next(), supSetF->Next()) {
        mapPin = new pinMap;
        strcpy(mapPin->cell_pin, (g_name = supSetG->Data()));
        f_name = supSetF->Data();

        // determine polarity of mapPin->net_node
        fn_name = (char *)calloc(strlen(f_name)+2, sizeof(char));
        strcpy(fn_name, f_name); strcat(fn_name, "\n");

        if(nsupG == 3) {
            if(!strcmp(g_name, "a")) {
                strcpy(mapPin->net_node, f_name);
                pinMapList->InsertRear(mapPin);
                free(fn_name);
            } else if(!strcmp(g_name, "b")) {
                strcpy(mapPin->net_node, f_name);
                pinMapList->InsertRear(mapPin);

                xorPin = new pinMap; // for original "e"
                strcpy(xorPin->cell_pin, "e");
                strcpy(xorPin->net_node, f_name);
                pinMapList->InsertRear(xorPin);
                free(fn_name);
            } else if(!strcmp(g_name, "c")) {
                if((matchFindNameInGenList(invList, "b") != -1 &&
                    matchIsVarPhaseInverted(curCell, rpnG, g_name)) ||
                    (matchFindNameInGenList(invList, "b") != -1 &&
                    !matchIsVarPhaseInverted(curCell, rpnG, g_name))) {
                    strcpy(mapPin->net_node, fn_name);
                    pinMapList->InsertRear(mapPin);

                    xorPin = new pinMap; // for original "f"
                    strcpy(xorPin->cell_pin, "f");
                    strcpy(xorPin->net_node, f_name);
                    pinMapList->InsertRear(xorPin);

                    xorPin = new pinMap;
                    strcpy(xorPin->cell_pin, "d");
                    strcpy(xorPin->net_node, f_name);
                    pinMapList->InsertRear(xorPin);

                    xorPin = new pinMap; // for original "g"
                    strcpy(xorPin->cell_pin, "g");
                    strcpy(xorPin->net_node, fn_name);
                    pinMapList->InsertRear(xorPin);
                } else if((matchFindNameInGenList(invList, "b") != -1 &&
                    !matchIsVarPhaseInverted(curCell, rpnG, g_name)) ||
                    (matchFindNameInGenList(invList, "b") != -1 &&
                    matchIsVarPhaseInverted(curCell, rpnG, g_name))) {
                    strcpy(mapPin->net_node, f_name);
                    pinMapList->InsertRear(mapPin);

                    xorPin = new pinMap; // for original "f"
                    strcpy(xorPin->cell_pin, "f");
                    strcpy(xorPin->net_node, fn_name);
                    pinMapList->InsertRear(xorPin);

                    xorPin = new pinMap;
                    strcpy(xorPin->cell_pin, "d");
                    strcpy(xorPin->net_node, fn_name);
                    pinMapList->InsertRear(xorPin);

                    xorPin = new pinMap; // for original "g"
                    strcpy(xorPin->cell_pin, "g");
                    strcpy(xorPin->net_node, f_name);
                    pinMapList->InsertRear(xorPin);
                }
            }
        } else if(nsupG > 3) {
            if(!strcmp(g_name, "a") || !strcmp(g_name, "b")
                || !strcmp(g_name, "e")) {
                strcpy(mapPin->net_node, f_name);
                pinMapList->InsertRear(mapPin);
                free(fn_name);
            }
        }
    }
}

```

```

    } else if(!strcmp(g_name,"c")){
        if((matchFindNameInGenList(invList,"b")!=-1 &&
            matchIsVarPhaseInverted(curCell, rpnG, g_name))||
            (matchFindNameInGenList(invList,"b")!=-1 &&
            matchIsVarPhaseInverted(curCell, rpnG, g_name))){
            strcpy(mapPin->net_node, fn_name);

            xorPin = new pinMap;
            strcpy(xorPin->cell_pin, "d");
            strcpy(xorPin->net_node, f_name);

            pinMapList->InsertRear(xorPin);
        } else if((matchFindNameInGenList(invList,"b")!=-1 &&
            !matchIsVarPhaseInverted(curCell, rpnG, g_name))||
            (matchFindNameInGenList(invList,"b")!=-1 &&
            matchIsVarPhaseInverted(curCell, rpnG, g_name))){
            strcpy(mapPin->net_node, f_name);

            xorPin = new pinMap;
            strcpy(xorPin->cell_pin, "d");
            strcpy(xorPin->net_node, fn_name);

            pinMapList->InsertRear(xorPin);
        }
        pinMapList->InsertRear(mapPin);
    } else if(!strcmp(g_name,"f")){
        if((matchFindNameInGenList(invList,"e")!=-1 &&
            matchIsVarPhaseInverted(curCell, rpnG, g_name))||
            (matchFindNameInGenList(invList,"e")!=-1 &&
            matchIsVarPhaseInverted(curCell, rpnG, g_name))){
            strcpy(mapPin->net_node, fn_name);

            xorPin = new pinMap;
            strcpy(xorPin->cell_pin, "g");
            strcpy(xorPin->net_node, f_name);

            pinMapList->InsertRear(xorPin);
        } else if((matchFindNameInGenList(invList,"e")!=-1 &&
            !matchIsVarPhaseInverted(curCell, rpnG, g_name))||
            (matchFindNameInGenList(invList,"e")!=-1 &&
            matchIsVarPhaseInverted(curCell, rpnG, g_name))){
            strcpy(mapPin->net_node, f_name);

            xorPin = new pinMap;
            strcpy(xorPin->cell_pin, "g");
            strcpy(xorPin->net_node, fn_name);

            pinMapList->InsertRear(xorPin);
        }
        pinMapList->InsertRear(mapPin);
    }
}
//ptl_y3x
}
}
// PPrint pin-map of a match
void matchPrintPinMap(Node *targetNode, Match *matchS)
{
    LinkedList<pinMap *> *pinList;
    pinMap *curPin;

    pinList = matchS->pinmap;
    cout<<" <<<< PIN Mapping >>>>\n";
    cout<<targetNode->GetName()<<" : " <<"
    <<" matched cell --> "<<matchS->cell_name<<"\n";
    for(pinList->Reset(); !pinList->EndOfList(); pinList->Next()){
        curPin = pinList->Data();
        cout<<curPin->cell_pin<<"("<<curPin->net_node<<" " <<"
    }
    cout<<endl;
}

// Assign pins for matched function at targetNode, and record matched cell name
void matchAssignPins(LinkedList<char *> *rpnF, LinkedList<char *> *supSetF,
    LinkedList<char *> *rpnG, LinkedList<char *> *supSetG,
    LibCell *curCell, Node *tarNode,
    LinkedList<char *> *symmSetF, int num_f_symm,
    LinkedList<char *> *symmSetG, int num_g_symm,
    Graph *origNet, Graph *funcGraph=NULL)
{
    Match *match = new Match;
    pinMap *mapPin, *xorPin;

    LinkedList<pinMap *> *pinMapList;
    LinkedList<Match *> *matchList;
    LinkedList<char *> *invList = new LinkedList<char *>;

    char *f_name, *g_name;
    char *fn_name, *target_name, *curCellName;
    int nsupG;

    match->cell_name=(char *)realloc(match->cell_name,
        strlen(curCell->GetName())+2);
    strcpy(match->cell_name, curCell->GetName());
    pinMapList = match->pinmap;
    curCellName = curCell->GetName();

    if(curCell->GetCellType()==PTL){
        (void)matchFindInvertedBinateVariable(supSetG, rpnG, curCell, invList);

        // Inert Strap info to pinMapList
        (void)matchInsertStrapInfo2PinList(strapT, pinMapList);
        if(!strcmp(curCellName, "ptl_y2", 6) || !strcmp(curCellName, "ptl_y3", 6)){
            (void)matchDetermineY23CellName(invList, curCell, match);
        }
    }

    target_name = tarNode->GetName();

    nsupG = supSetG->ListSize();
    if(curCell->GetCellType()==STATIC){
        if(!strcmp(match->cell_name, "and2") &&
            !strcmp(target_name, "and3")){
            cout<<"and3...\n";
        }
        (void)matchSetPinMapping(supSetG, supSetF, match, curCell, tarNode,
            rpnG, rpnF, funcGraph);
    } else if(curCell->GetCellType()==PTL){
        if(strcmp(match->cell_name, "ptl_y1")==0){
            matchY1PinMap(supSetG, supSetF, match, curCell, tarNode, rpnG, invList);
        } else if(strcmp(match->cell_name, "ptl_y1x")==0){
            matchY1xPinMap(supSetG, supSetF, match, curCell, tarNode, rpnG, invList);
        } else if(strcmp(curCell->GetName(), "ptl_y2")==0){
            matchY2PinMap(supSetG, supSetF, match, curCell, tarNode, rpnG, invList);
        } else if(strcmp(curCell->GetName(), "ptl_y2x")==0){
            matchY2xPinMap(supSetG, supSetF, match, curCell, tarNode, rpnG, invList);
        } else if(strcmp(curCell->GetName(), "ptl_y3")==0){
            matchY3PinMap(supSetG, supSetF, match, curCell, tarNode, rpnG, invList);
        } else if(strcmp(curCell->GetName(), "ptl_y3x")==0){
            matchY3xPinMap(supSetG, supSetF, match, curCell, tarNode, rpnG, invList);
        }
    }

    if(curCell->GetCellType()==STATIC){
        matchList = tarNode->GetSMatchList();
    } else {
        matchList = tarNode->GetPMatchList();
    }

    // insert match to node's map list
    if(!matchCurrentMatchIsDuplicated(matchList, match)){
        #ifdef MATCH_PIN_DEBUG
            cout<<"\n INSERTING NEW MATCH ..... \n";
        #endif
        matchList->InsertRear(match);
        #ifdef MATCH_PIN_DEBUG
            Match *matchS;
            matchS = matchList->Data();
            (void)matchPrintPinMap(tarNode, matchS);
        #endif
    } else {
        delete match;
    }

    delete invList;
} /* End of pin assignment */

// The support set elements must be extracted from rpnL
void matchGetNumUBinateVars(LinkedList<char *> *rpnL, LinkedList<char *> *supSet,
    int *ufNum, int *bfNum)
{
    int pos_unate[10], neg_unate[10], name_len, listPos, pos_count;
    int var_match_count, s, bar_cnt;
    int posUnate, negUnate;
    char *sup_var, *tmp_var, *rpn_var;

    (*bfNum) = (*ufNum) = 0;
    for(supSet->Reset(); !supSet->EndOfList(); supSet->Next()){
        sup_var = supSet->Data();
        for(n=0; n<10; n++){
            pos_unate[n] = 0; neg_unate[n] = 0;
        }
        var_match_count = 0;
        posUnate = negUnate = 0;
        for(rpnL->Reset(); !rpnL->EndOfList(); rpnL->Next()){
            tmp_var = rpnL->Data();
            name_len = strlen(tmp_var);
            if(!IsOperator(tmp_var) && !IsConstant(tmp_var)){
                rpn_var = (char *)calloc(name_len+1, sizeof(char));
                if(name_len>1 && tmp_var[name_len-1]!='\'' ){
                    strncpy(rpn_var, tmp_var, name_len-1);
                } else {
                    strcpy(rpn_var, tmp_var);
                }
            } else {
                continue;
            }

            if(strcmp(sup_var, rpn_var)==0){
                listPos = rpnL->CurrentPosition();
                if(matchCheckConstVarInRpn(rpnL, listPos)) continue;
                pos_count = listPos;
                var_match_count++;
                if(tmp_var[name_len-1]!='\'' ){
                    bar_cnt = 0;
                    while(pos_count < (rpnL->ListSize()-1)){
                        rpnL->Next(); pos_count++;
                    }
                    //bar_cnt = 0;
                    if(strcmp(tmp_var, rpnL->Data(), "\'")==0){
                        bar_cnt++;
                    }
                    if(bar_cnt%2 == 0){
                        neg_unate[pos_count-1] = 1;
                        pos_unate[pos_count-1] = 0;
                    } else {
                        neg_unate[pos_count-1] = 0;
                        pos_unate[pos_count-1] = 1;
                    }
                }
                if(!strcmp(tmp_var, "\'")==0) break;
            }
            rpnL->Reset(listPos);
        } else {

```

```

        bar_cnt = 0;
        while(pos_count < (rpnL->ListSize()-1)){
            rpnL->Next(); pos_count++;
            if(strcmp(tmp_var = rpnL->Data(),"\'")==0){
                bar_cnt++;
            }
            if(bar_cnt%2 == 1){
                neg_unate[var_match_count-1] = 1;
                pos_unate[var_match_count-1] = 0;
            } else {
                neg_unate[var_match_count-1] = 0;
                pos_unate[var_match_count-1] = 1;
            }
            if(!(strcmp(tmp_var,"\'")==0)) break;
        }
        rpnL->Reset(listPos);
    }
    /* delete [] rpn_var; */
    free(rpn_var);
} // End of inner for_loop

for(n=0; n < var_match_count; n++){
    if(pos_unate[n]==1){
        posUnate = 1;
    } else if(neg_unate[n]==1){
        negUnate = 1;
    }
}

if(posUnate && negUnate){
    (*bfNum)++;
} else if(!posUnate && negUnate || posUnate && !negUnate){
    (*ufNum)++;
} else {
    // exit(0);
}
}

int matchCheckSupsetPinlistEquivalence(LinkedList<pinMap> *pinList,
                                       LinkedList<char> *supSetF)
{
    pinMap *oldPin;
    char *pinName, *sup_var, *wob_name;
    int num_pins, num_constant = 0, count = 0, name_len;

    num_pins = pinList->ListSize();
    for(pinList->Reset(); !pinList->EndOfList(); pinList->Next()){
        oldPin = pinList->Data();
        pinName = oldPin->net_node;
        if(strcmp(pinName,"VDD")==0 || strcmp(pinName,"GND")==0){
            num_constant++;
            continue;
        }
        name_len = strlen(pinName);
        if(pinName[name_len-1] == '\''){
            wob_name = (char *)calloc(name_len, sizeof(char));
            strncpy(wob_name, pinName, (name_len-1));
        } else {
            wob_name = (char *)calloc((name_len+1), sizeof(char));
            strcpy(wob_name, pinName);
        }
        for(supSetF->Reset(); !supSetF->EndOfList(); supSetF->Next()){
            sup_var = supSetF->Data();
            if(strcmp(wob_name, sup_var)==0){
                count++;
            }
        }
        /*delete [] wob_name; */
        free(wob_name);
    }
    if(count == (num_pins-num_constant)){
        return (1);
    } else {
        return (0);
    }
}

// This routine checks if curCell is already mapped for the targetNode,
// if so, do not generate BDD for matching
int matchCheckIfCurCellMapped(Node *curNode, LinkedList<char> *supSetF,
                               LibCell *curCell)
{
    LinkedList<Match> *matchList;
    LinkedList<pinMap> *prevPinList;
    char *cellName, *matchName;
    CellType cellType;
    Match *prevMatch;
    int numFVar, mapped = 0, equal_cell;

    cellType = curCell->GetCellType();
    cellName = curCell->GetName();
    numFVar = supSetF->ListSize();
    if(cellType == STATIC){
        matchList = curNode->GetSMatchList();
    } else {
        matchList = curNode->GetPMatchList();
    }
    if(matchList->ListSize()==0) return (0);
    for(matchList->Reset(); !matchList->EndOfList(); matchList->Next()){
        prevMatch = matchList->Data();
        matchName = prevMatch->cell_name;
        prevPinList = prevMatch->pinmap;
        equal_cell = 1;
        if(cellType==STATIC){
            equal_cell = strcmp(matchName, cellName);
        } else {
            equal_cell = strcmp(matchName, cellName, 6);
        }
        if(!equal_cell){
            if(matchCheckSupsetPinlistEquivalence(prevPinList, supSetF)){
                mapped = 1;
                break;
            }
        }
    }
    return mapped;
}

void matchDeletePermuList(LinkedList<int> *pList)
{
    int size, n;
    size = pList->ListSize();
    pList->Reset();
    if(pList != NULL){
        for(n=0; n<size; n++){
            //pList->Reset(n);
            delete [] pList->DeleteCurrent();
        }
        delete pList;
        pList = NULL;
    }
}

// Perform match with library
int matchDoStaticMatch(LinkedList<char> *rpnFunc,
                      LinkedList<char> *supSetF,
                      int num_sup,
                      LinkedList<char> *rpnG,
                      LinkedList<char> *supSetG,
                      int num_supG,
                      LibCell *curCell,
                      Node *targetNode,
                      DdNode *f, DdNode *g,
                      Graph *origNet, Graph *funcGraph = NULL)
{
    //LinkedList<char> *rpnFunc, *supSetF, *rpnG, *supSetG;
    int num_f_symm, num_g_symm;
    int match = 0;

    DdManager *fbddm, *gbddm;

    LinkedList<char> *symmSetF = new LinkedList<char> [10];
    LinkedList<char> *symmSetFA;
    symmSetFA = &symmSetF;
    int num_symsetF = 0, num_symsetG = 0;

    int phaseFunc;
    LinkedList<char> *new_rpnG, *new_ord_supSetG;
    LinkedList<int> *pList; // permutation list (list of integer array)
    int *ordArray, ufNum, bfNum, ugNum, bgNum;
    int mapped, ord_chn=0, type;

    fbddm = Cudd_Init(0,0, 1, 1, 1);
    f = GetBdd(fbddm, rpnFunc, supSetF);
    num_symsetF = GetSymmSet(fbddm, f, supSetF, symmSetF, &num_f_symm);

#ifdef MATCH_BASED_ON_DEPTH
    ord_chn=matchSupportSetFreorderBasedOnDepth(supSetF, symmSetF,
                                                num_f_symm, origNet);
    if(ord_chn){ //if supF order has been changed(11-19-01)
        Cudd_RecursiveDeref(fbddm,f);
        Cudd_Quit(fbddm);
        fbddm = Cudd_Init(0,0, 1, 1, 1);
        f = GetBdd(fbddm, rpnFunc, supSetF);
    }
#endif

#ifdef MATCH_FUNC_DEBUG
    PrintCharList(rpnFunc);
    PrintCharList(supSetF);
    cout<<"num_unate = "<<ufNum<<" num_binate = "<<bfNum<<"\n";
    PrintSymmSet(symmSetF, num_f_symm);
    Cudd_PrintDebug(fbddm, f, 1, 5);
#endif

    int maxNum = (int)pow((double)2, (double)num_supG);
    for(int n = 0; n < maxNum; n++){
        num_g_symm = 0;

        /* 12-26-00 */
        mapped = 0;
        mapped = matchCheckIfCurCellMapped(targetNode, supSetF, curCell);
        if(mapped) break;
        /* 12-26-00 */

        LinkedList<char> *symmSetG = new LinkedList<char> [5];
        LinkedList<char> *symmSetGA;
        symmSetGA = &symmSetG;

        phaseFunc = n & 0x0ff; // cannot be processed #elements > 8 in supSetG
        new_rpnG = matchModifyVarPhase(rpnG, supSetG, phaseFunc);
        (void)matchGetNumUBinateVars(new_rpnG, supSetG, &ugNum, &bgNum);

        if(ugNum==ufNum && bgNum==bfNum){
            gbddm = Cudd_Init(0,0, 1, 1, 1); //-->
            g = GetBdd(gbddm, new_rpnG, supSetG);

            num_symsetG = GetSymmSet(gbddm, g, supSetG, symmSetG, &num_g_symm);

#ifdef MATCH_FUNC_DEBUG
            PrintCharList(new_rpnG);
            PrintCharList(supSetG);
            cout<<"num_unate = "<<ugNum<<" num_binate = "<<bgNum<<"\n";
            PrintSymmSet(symmSetG, num_g_symm);
            Cudd_PrintDebug(gbddm, g, 1, 5);
#endif

```

```

#endif

Cudd_RecursiveDeref(gbddm,g);
Cudd_Quit(gbddm);

if(num_f_symm == num_g_symm){
    // Permute Symmetry set (Re-Ordering) and re-generate Bdd
    pList = matchGetPermutationList(num_g_symm);
    // for each permutation of rpnG
    for(pList->Reset(); !pList->EndOfList(); pList->Next()){
        /* 12-26-00 */
        mapped = 0;
        mapped = matchCheckIfCurCellMapped(targetNode, supSetF,
            curCell);
        if(mapped) continue;
        /* 12-26-00 */
        ordArray = pList->Data();
        new_ord_supSetG = matchGetNewOrderOfSupSetG(symmSetG,
            ordArray, num_g_symm);

        gbddm = Cudd_Init(0,0, 1, 1, 1);
        g = GetBdd(gbddm, new_rpnG, new_ord_supSetG);

        if((Cudd_DagSize(f)==Cudd_DagSize(g)) &&
            (Cudd_IsComplement(f) && Cudd_IsComplement(g)) ||
            !Cudd_IsComplement(f) && !Cudd_IsComplement(g)){
            match = BddEqCheck(f,g);
        }

        if(match){ //Static
            (void)matchAssignPins(rpnFunc, supSetF, new_rpnG,
                new_ord_supSetG, NULL,
                curCell, targetNode,
                NULL, 0, NULL, 0, origNet, funcGraph);
        }

#ifdef MATCH_DEBUG
        cout<<"n.... " << targetNode->GetName() <<" .....n";
        type = targetNode->GetNodeFunc();
        cout<<"n..NodeFunc= ";
        (void)origNet->Print_Node_Func(type);

        cout<<"n\n";
        PrintCharList(rpnFunc);
        PrintCharList(supSetF);
        PrintSymmSet(symmSetF, num_f_symm);
        Cudd_PrintDebug(fbddm, f, 1, 5);

        PrintCharList(new_rpnG);
        PrintCharList(new_ord_supSetG);
        PrintSymmSet(symmSetG, num_g_symm);
        cout<<curCell->GetName() << " ";
        Cudd_PrintDebug(gbddm, g, 1, 5);
#endif

        (void) MemFreeForSymmSet(symmSetG, num_g_symm);
        delete [] symmSetGA;
        delete new_rpnG;
        (void) MemFreeForSymmSet(symmSetF, num_f_symm);
        delete [] symmSetFA;

        Cudd_RecursiveDeref(fbddm,f);
        Cudd_Quit(fbddm);

        Cudd_RecursiveDeref(gbddm,g);
        Cudd_Quit(gbddm);

        (void)matchDeleteCharLinkedList(new_ord_supSetG, 1);
        delete pList;
        return (1);
    }
    delete [] ordArray;
    (void)matchDeleteCharLinkedList(new_ord_supSetG, 1);
    Cudd_RecursiveDeref(gbddm,g);
    Cudd_Quit(gbddm); //---
}
delete pList;
}
(void) MemFreeForSymmSet(symmSetG, num_g_symm);
delete [] symmSetGA;
(void)matchDeleteCharLinkedList(new_rpnG, 1);
}
else {
    (void) MemFreeForSymmSet(symmSetG, num_g_symm);
    delete [] symmSetGA;
    (void)matchDeleteCharLinkedList(new_rpnG, 1);
}
}

(void) MemFreeForSymmSet(symmSetF, num_f_symm);
delete [] symmSetFA;
Cudd_RecursiveDeref(fbddm,f);
Cudd_Quit(fbddm);
}
return match;

void matchChangeSupportSetOrder(LinkedList<char *> *supSetF,
    LinkedList<char *> *symmSetF, int num_f_symm)
{
    LinkedList<char *> *symmList;
    char *symm_var;
    int i, listSize;

    listSize = supSetF->ListSize();
    supSetF->Reset();
    for(i=0; i<listSize; i++){
        (void)supSetF->DeleteCurrent();
        }
    for(i=num_f_symm-1; i>=0; i--){
        symmList = &symmSetF[i];
        for(symmList->Reset(); !symmList->EndOfList(); symmList->Next()){
            symm_var = symmList->Data();
            supSetF->InsertRear(symm_var);
        }
    }
}

// PTL match
// Perform match with library
int matchDoPtlMatch(LinkedList<char *> *rpnFunc,
    LinkedList<char *> *supSetF,
    int num_sup,
    LinkedList<char *> *rpnG,
    LinkedList<char *> *supSetG,
    int num_supG,
    LinkedList<char *> *strapt,
    LibCell *curCell,
    Node *targetNode,
    DdNode *f, DdNode *g,
    Graph *origNet, Graph *funcGraph = NULL)
{
    int num_f_symm, num_g_symm;
    int match = 0;

    DdManager *fbddm, *gbddm;

    LinkedList<char *> *symmSetF = new LinkedList<char *> [10];
    LinkedList<char *> *symmSetFA;
    symmSetFA = &symmSetF;
    int num_symsetF = 0, num_symsetG = 0;

    int phaseFunc;
    LinkedList<char *> *new_rpnG, *new_ord_supSetG;
    LinkedList<int *> *pList; // permutation list (list of integer array)
    char *curCellName;
    int *ordArray, ufNum, bfNum, ugNum, bgNum;
    int mapped, ord_chn=0, tmp_int;

    fbddm = Cudd_Init(0,0, 1, 1, 1);
    f = GetBdd(fbddm, rpnFunc, supSetF);
    num_symsetF = GetSymmSet(fbddm, f, supSetF, symmSetF, &num_f_symm);

#ifdef MATCH_BASED_ON_DEPTH
    ord_chn = matchSupportSetFreorderBasedOnDepth(supSetF, symmSetF,
        num_f_symm, origNet);

    if(ord_chn){
        Cudd_RecursiveDeref(fbddm,f);
        Cudd_Quit(fbddm);
        fbddm = Cudd_Init(0,0, 1, 1, 1);
        f = GetBdd(fbddm, rpnFunc, supSetF);
    }
#endif

    (void)matchGetNumUBinateVars(rpnFunc, supSetF, &ufNum, &bfNum);

#ifdef MATCH_FUNC_DEBUG
    PrintCharList(rpnFunc);
    PrintCharList(supSetF);
    cout<<"num_ubinate = "<<ufNum<<" num_binate = "<<bfNum<<"n";
    PrintSymmSet(symmSetF, num_f_symm);
    Cudd_PrintDebug(fbddm, f, 1, 5);
#endif

    curCellName = curCell->GetName();
    int maxNum = (int)pow((double)2, (double)num_supG);

    if(num_f_symm <= 7 && !(bfNum==1 && !strcmp(curCellName, "ptl_y3",6))){
        for(int n = 0; n < maxNum; n++){
            num_g_symm = 0;

            mapped = 0;
            mapped = matchCheckIfCurCellMapped(targetNode, supSetF, curCell);
            if(mapped) break;

            LinkedList<char *> *symmSetG = new LinkedList<char *> [10];
            LinkedList<char *> *symmSetGA;
            symmSetGA = &symmSetG;

            phaseFunc = n & 0xff; // cannot be processed #elements > 8 in supSetG
            new_rpnG = matchModifyVarPhase(rpnG, supSetG, phaseFunc);
            (void)matchGetNumUBinateVars(new_rpnG, supSetG, &ugNum, &bgNum);

            if(ugNum==ufNum && bgNum==bfNum){
                gbddm = Cudd_Init(0,0, 1, 1, 1); //---
                g = GetBdd(gbddm, new_rpnG, supSetG);

                num_symsetG = GetSymmSet(gbddm, g, supSetG, symmSetG, &num_g_symm);
                Cudd_RecursiveDeref(gbddm,g);
            }

#ifdef MATCH_FUNC_DEBUG
            PrintCharList(new_rpnG);
            PrintCharList(supSetG);
            cout<<"num_ubinate = "<<ugNum<<" num_binate = "<<bgNum<<"n";
            PrintSymmSet(symmSetG, num_g_symm);
            Cudd_PrintDebug(gbddm, g, 1, 5);
            cout<<"Regular.....n";
            Cudd_PrintDebug(gbddm, Cudd_Regular(g), 1, 5);
#endif

            Cudd_Quit(gbddm);
            if((num_f_symm == num_g_symm)){
                // Permute Symmetry set (Re-Ordering) and re-generate Bdd
                pList = matchGetPermutationList(num_g_symm);

                // for each permutation of rpnG
                for(pList->Reset(); !pList->EndOfList(); pList->Next()){
                    /* 12-26-00 */

```

```

mapped = 0;
mapped = matchCheckIfCurCellMapped(targetNode, supSetF,
                                     curCell);
if(mapped) break;
/* 12-26-0 */

match = 0;
ordArray = pList->Data();
new_ord_supSetG = matchGetNewOrderOfSupSetG(symmSetG,
                                             ordArray, num_g_symm);

gbddm = Cudd_Init(0,0, 1, 1, 1); /*-*/
g = GetBdd(gbddm, new_rpnG, new_ord_supSetG);

if(Cudd_DagSize(f)==Cudd_DagSize(g) &&
    (Cudd_IsComplement(f) && Cudd_IsComplement(g)||
     !Cudd_IsComplement(f) && !Cudd_IsComplement(g))){
    match = BddEqCheck(f,g);
}

if(match){ // Ptl
    (void)matchAssignPins(rpnFunc, supSetF, new_rpnG,
                        new_ord_supSetG, strapT,
                        curCell, targetNode,
                        symmSetF, num_f_symm,
                        symmSetG, num_g_symm, origNet);
}

#ifdef MATCH_DEBUG
    cout<<"n.... "<< targetNode->GetName()<<" .....n";
    cout<<"\nn";
    PrintCharList(rpnFunc);
    PrintCharList(supSetF);
    cout<<"num_unate = "<< ifNum<<" num_binate = "
        << bNum<<"\nn";
    PrintSymmSet(symmSetF, num_f_symm);
    cout << "#symm_setF = "<< num_symsetF<<"\nn";
    Cudd_PrintDebug(fbddm, f, 1, 5);
    cout<<"Cell = "<< curCellName<<"\nn";
    PrintCharList(new_rpnG);
    PrintCharList(new_ord_supSetG);
    cout<<"num_unate = "<< ugNum<<" num_binate = "
        << bgNum<<"\nn";
    PrintSymmSet(symmSetG, num_g_symm);
    cout << "#symm_setG = "<< num_symsetG<<"\nn";
    cout<<curCell->GetName() << "\nn";
    Cudd_PrintDebug(gbddm, g, 1, 5);
#endif

    (void) MemFreeForSymmSet(symmSetG, num_g_symm);
    delete [] symmSetGA;
    delete new_rpnG;
    (void) MemFreeForSymmSet(symmSetF, num_f_symm);
    delete [] symmSetFA;
    Cudd_RecursiveDeref(fbddm,f);
    Cudd_Quit(fbddm);

    Cudd_RecursiveDeref(gbddm,g);
    Cudd_Quit(gbddm);

    (void)matchDeleteCharLinkedList(new_ord_supSetG, 1);
    delete pList;

    return (1);
}
delete [] ordArray;
(void)matchDeleteCharLinkedList(new_ord_supSetG, 1);
Cudd_RecursiveDeref(gbddm,g);
Cudd_Quit(gbddm); /*-*/
}
tmp_int = pList->ListSize();
for(int k=0; k<tmp_int; k++){
    pList->Reset();
    delete pList->DeleteCurrentElem();
}
delete pList;
}
(void) MemFreeForSymmSet(symmSetG, num_g_symm);
delete [] symmSetGA;
(void)matchDeleteCharLinkedList(new_rpnG, 1);
} //if(ugNum==ufNum && bgNum==bfNum)
else {
    (void) MemFreeForSymmSet(symmSetG, num_g_symm);
    delete [] symmSetGA;
    (void)matchDeleteCharLinkedList(new_rpnG, 1);
}
}
}
}
}

// In Cell library, unate variable appears only once in function(rpnGc)
LinkedList<char *> *matchGetUnateVars(LinkedList<char *> *rpnGc,
                                     LinkedList<char *> *supSetG,
                                     int num_unateG)
{
    LinkedList<char *> *unateVar = new LinkedList<char *>;
    char *supVar, *rpnVar;
    int match;

    num_unateG = 0;
    for(supSetG->Reset(); !supSetG->EndOfList(); supSetG->Next()){
        match = 0;

        supVar = supSetG->Data();
        for(rpnGc->Reset(); !rpnGc->EndOfList(); rpnGc->Next()){
            rpnVar = rpnGc->Data();
            if(strncmp(supVar, rpnVar)==0){
                match++;
            }
        }
        if(match==1){ // Unate variable
            unateVar->InsertRear(supVar);
            num_unateG++;
        }
    }
    return unateVar;
}

LinkedList<char *> *matchStrapRpn(LibCell *curCell, char *p1, char *v1,
                                  char *p2, char *v2,
                                  char *p3=NULL, char *v3=NULL,
                                  char *p4=NULL, char *v4=NULL,
                                  char *p5=NULL, char *v5=NULL)
{
    LinkedList<char *> *tmpRpn;
    LinkedList<char *> *rpnGc = new LinkedList<char *>;
    char *tmp_char;

    tmpRpn = curCell->GetFunc();
    // copy cell rpn to rpnGc, if a var is strapped insert strap info.
    for(tmpRpn->Reset(); !tmpRpn->EndOfList(); tmpRpn->Next()){
        tmp_char = tmpRpn->Data();
        if(strncmp(tmp_char, p1)==0){
            rpnGc->InsertRear(v1);
        }
        else if (strncmp(tmp_char, p2)==0){
            rpnGc->InsertRear(v2);
        }
        else if (p3!=NULL && strncmp(tmp_char, p3)==0){
            rpnGc->InsertRear(v3);
        }
        else if (p4!=NULL && strncmp(tmp_char, p4)==0){
            rpnGc->InsertRear(v4);
        }
        else if (p5!=NULL && strncmp(tmp_char, p5)==0){
            rpnGc->InsertRear(v5);
        }
        else {
            rpnGc->InsertRear(tmp_char);
        }
    }
    return rpnGc;
}

// strapT = matchGetStrapInfo("b", "GND", "", "");
LinkedList<char *> *matchGetStrapInfo(char *p1, char *v1, char *p2, char *v2,
                                       char *p3=NULL, char *v3=NULL,
                                       char *p4=NULL, char *v4=NULL,
                                       char *p5=NULL, char *v5=NULL)
{
    LinkedList<char *> *strapT = new LinkedList<char *>;

    if(strncmp(p1,"")!=0 && strcmp(v1,"")!=0){
        strapT->InsertRear(p1);
        strapT->InsertRear(v1);
    }
    if(strncmp(p2,"")!=0 && strcmp(v2,"")!=0){
        strapT->InsertRear(p2);
        strapT->InsertRear(v2);
    }
    if(p3!=NULL){
        if(v3!=NULL){
            strapT->InsertRear(p3);
            strapT->InsertRear(v3);
        }
        else {
            cout<<"In matchGetStrapInfo() of "<<__FILE__<<"\nn";
            cout<<"p3 is not NULL, but v3 is NULL. Program dies.\nn";
            exit(0);
        }
    }
    if(p4!=NULL){
        if(v4!=NULL){
            strapT->InsertRear(p4);
            strapT->InsertRear(v4);
        }
        else {
            cout<<"In matchGetStrapInfo() of "<<__FILE__<<"\nn";
            cout<<"p4 is not NULL, but v4 is NULL. Program dies.\nn";
            exit(0);
        }
    }
    if(p5!=NULL){
        if(v5!=NULL){
            strapT->InsertRear(p5);
            strapT->InsertRear(v5);
        }
        else {
            cout<<"In matchGetStrapInfo() of "<<__FILE__<<"\nn";
            cout<<"p4 is not NULL, but v4 is NULL. Program dies.\nn";
            exit(0);
        }
    }
    return strapT;
}

// Returns #of strapped pins, modify rpnG an insert it to pointer to array
int matchGetPtlStrapFunc(LibCell *curCell,
                        LinkedList<LinkedList<char *> *> *strapRpnG,
                        LinkedList<LinkedList<char *> *> *strapTable)
{
    Graph *cellGraph;
    LinkedList<char *> *rpnGc, *supSetG, *unateVar, *strapT;
    char *cell_name;

```

```

int num_supG, num_unateG;
int index = 0, num_strapFunc = 0;

cellGraph = curCell->GetCellGraph();
cell_name = curCell->GetName();
rpnGc = matchCopyRpn(curCell->GetFunc());

strapRpnG->InsertRear(rpnGc); // unmodified function
strapT = matchGetStrapInfo("0", "0", "0", "0");//dummy to align list index
strapTable->InsertRear(strapT);
index++;

supSetG = GetSupportVarSet(cellGraph, rpnGc, num_supG);
unateVar = matchGetUnateVars(rpnGc, supSetG, num_unateG);

// Strap is done only for unate variables.
if(unateVar->ListSize()==0 && !(strcmp(cell_name, "ptl_y3x")==0)){
    (void)matchDeleteCharLinkedList(unateVar, 1);
    (void)matchDeleteCharLinkedList(supSetG, 1);
    return num_strapFunc;
} else {
    if(!strcmp(cell_name, "ptl_y1")){
        rpnGc = matchStrapRpn(curCell, "b", "GND", "c", "c");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("b", "GND", "", "");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "b", "VDD", "c", "c");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("b", "VDD", "", "");
        strapTable->InsertRear(strapT); num_strapFunc++;

        (void)matchDeleteCharLinkedList(unateVar, 1);
        (void)matchDeleteCharLinkedList(supSetG, 1);
        return num_strapFunc;
    }
    else if(!strcmp(cell_name, "ptl_y2")){
        rpnGc = matchStrapRpn(curCell, "c", "GND", "d", "d");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("c", "GND", "", "");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "c", "VDD", "d", "d");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("c", "VDD", "", "");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "e", "GND", "c", "c");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("e", "GND", "", "");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "e", "VDD", "c", "c");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("e", "VDD", "", "");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "c", "GND", "e", "GND");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("c", "GND", "e", "GND");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "c", "GND", "e", "VDD");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("c", "GND", "e", "VDD");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "c", "VDD", "e", "GND");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("c", "VDD", "e", "GND");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "c", "VDD", "e", "VDD");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("c", "VDD", "e", "VDD");
        strapTable->InsertRear(strapT); num_strapFunc++;

        (void)matchDeleteCharLinkedList(unateVar, 1);
        (void)matchDeleteCharLinkedList(supSetG, 1);
        return num_strapFunc;
    }
    else if(!strcmp(cell_name, "ptl_y2x")){
        rpnGc = matchStrapRpn(curCell, "e", "GND", "c", "c");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("e", "GND", "", "");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "e", "VDD", "c", "c");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("e", "VDD", "", "");
        strapTable->InsertRear(strapT); num_strapFunc++;

        (void)matchDeleteCharLinkedList(unateVar, 1);
        (void)matchDeleteCharLinkedList(supSetG, 1);
        return num_strapFunc;
    }
    else if(!strcmp(cell_name, "ptl_y3x")){
        rpnGc = matchStrapRpn(curCell, "e", "b", "f", "c");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("e", "b", "f", "c");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "e", "b", "f", "c");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("e", "b", "f", "c");
        strapTable->InsertRear(strapT); num_strapFunc++;

        (void)matchDeleteCharLinkedList(unateVar, 1);
        (void)matchDeleteCharLinkedList(supSetG, 1);
        return num_strapFunc;
    }
    else if(!strcmp(cell_name, "ptl_y3")){
        // y3 match test with 3 variables
        rpnGc = matchStrapRpn(curCell, "d", "GND", "e", "b", "f", "VDD", "g", "c");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("d", "GND", "e", "b", "f", "VDD", "g", "c");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "d", "GND", "e", "e");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("d", "GND", "", "");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "d", "VDD", "e", "e");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("d", "VDD", "", "");
        strapTable->InsertRear(strapT); num_strapFunc++;

        // c,f
        rpnGc = matchStrapRpn(curCell, "c", "GND", "f", "GND");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("c", "GND", "f", "GND");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "c", "GND", "f", "VDD");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("c", "GND", "f", "VDD");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "c", "VDD", "f", "GND");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("c", "VDD", "f", "GND");
        strapTable->InsertRear(strapT); num_strapFunc++;

        rpnGc = matchStrapRpn(curCell, "c", "VDD", "f", "VDD");
        strapRpnG->InsertRear(rpnGc);
        strapT = matchGetStrapInfo("c", "VDD", "f", "VDD");
        strapTable->InsertRear(strapT); num_strapFunc++;

        (void)matchDeleteCharLinkedList(unateVar, 1);
        (void)matchDeleteCharLinkedList(supSetG, 1);
        return num_strapFunc;
    }
    }
}

// Do Boolean match at targetNode with library cells
void matchDoBooleanMatch(Graph *orgNet, Graph *sGraph, Node *targetNode,
    LinkedList<LibCell> *cellLib,
    DdNode *f, DdNode *g, int fDepth)
{
    int nliterals, num_sup, num_supG;
    LinkedList<Graph> *fgList; //pure
    LinkedList<char> *rpnFunc, *supSetF, *rpnG, *supSetG = NULL;
    LinkedList<char> *strapT = NULL;
    Graph *funcGraph, *cellGraph;
    LibCell *curCell;
    char *cur_cell_name;

    int num_str;
    int matchS = 0, matchP = 0;
    int mapped, dummy = -1;
    int n, listSize;

    // get possible functions at targetNode (all possible functions in Graph)
    fgList = GetSetOfFuncGraphs(sGraph, targetNode, fDepth);
    // for each function 1) get rpn 2) Graph Generate BDD
    for(fgList->Reset(); fgList->EndOfList(); fgList->Next++){
        funcGraph = fgList->Data();
        dummy++;

        rpnFunc = GetFunctionForBdd(funcGraph, targetNode, nliterals);
        #ifdef MATCH_FUNC_DEBUG
        cout<<"\n FUNCTION GRAPH "<<dummy<<"*****\n";
        funcGraph->PrintPNodes();
        funcGraph->PrintPoNodes();
        funcGraph->PrintAllNodes();
        cout<<"rpn_func("<<dummy<<")";
        #endif
        PrintCharList(rpnFunc);

        // orgNet is original network before generation of subject Graph;
        supSetF = GetSupportVarSet(orgNet, rpnFunc, num_sup);
        // For each library gate, check if #of support vars, check unateness
        // if same get symmetry sets of both functions.
        for(cellLib->Reset(); cellLib->EndOfList(); cellLib->Next()){
            matchS = 0; matchP = 0;
            curCell = cellLib->Data();
            cur_cell_name = curCell->GetName();
            cellGraph = curCell->GetCellGraph();

            mapped = 0;
            mapped = matchCheckIfCurCellMapped(targetNode, supSetF, curCell);
            if(mapped) continue;

            if(curCell->GetCellType()==STATIC){
                /* 01-03-01 */
            }
        }
    }
}

```

```

mapped = 0;
mapped = matchCheckIfCurCellMapped(targetNode, supSetF, curCell);
if(mapped) continue;
/* 01-03-01 */

rpnG = curCell->GetFunc();
supSetG = GetSupportVarSet(cellGraph, rpnG, num_supG);
if(num_sup == num_supG && num_sup <= 4 && num_supG<=4){//aoi22

#ifdef MATCH_FUNC_DEBUG
    cout<<"\n.....Function ..... \n";
    PrintCharList(rpnFunc);
    PrintCharList(supSetF);
    cout<<"...Cell Function .... "<<curCell->GetName()<<"\n";
    PrintCharList(rpnG);
    PrintCharList(supSetG);
#endif

    matchS = matchDoStaticMatch(rpnFunc, supSetF, num_sup,
                                rpnG, supSetG, num_supG,
                                curCell, targetNode,
                                f, g, orgNet, funcGraph);
    (void)matchDeleteCharLinkedList(supSetG, 1);
#ifdef MATCH_DEBUG_SHORT
    if(matchS){
        cout << "STATIC MATCHED !!!!!!!!!\n";
        //exit(1);
    }
#endif
} else {
    (void)matchDeleteCharLinkedList(supSetG, 1);
}

} else if(curCell->GetCellType()==PTL){
    /* 01-03-01 */
    mapped = 0;
    mapped = matchCheckIfCurCellMapped(targetNode, supSetF, curCell);
    if(mapped) continue;
    /* 01-03-01 */

    num_strG = 0;
    LinkedList<LinkedList<char *> *> *strapRpnG =
        new LinkedList<LinkedList<char *> *>();
    LinkedList<LinkedList<char *> *> *strapTable =
        new LinkedList<LinkedList<char *> *>();

    // Get strapped function for ptl_y1, ptl_y2, ptl_y3;
    num_strG = matchGetPtlStrapFunc(curCell, strapRpnG, strapTable);
    for(strapRpnG->Reset(); !strapRpnG->EndOfList();
        strapRpnG->Next()){
        // 01-03-01
        mapped = 0;
        mapped = matchCheckIfCurCellMapped(targetNode, supSetF,
                                            curCell);
        if(mapped) continue;
        // 01-03-01

        rpnG = strapRpnG->Data();
        supSetG = GetSupportVarSet(cellGraph, rpnG, num_supG);
        if(strapTable->ListSize()!=0){
            strapTable->Reset(strapRpnG->CurrentPosition());
            strapT = strapTable->Data();
        }
        if((num_sup==num_supG && (num_sup>=2) && (num_supG>=2) &&
            (num_sup<=7) && (num_supG<=7))){
#ifdef MATCH_FUNC_DEBUG
            cout<<"\n.....Function ..... \n";
            PrintCharList(rpnFunc);
            PrintCharList(supSetF);
            cout<<"...Cell Function .. "<<curCell->GetName()<<"\n";
            PrintCharList(rpnG);
            PrintCharList(supSetG);
            cout<<"PTL num_spuSet matched\n";
#endif

            matchP = matchDoPtlMatch(rpnFunc, supSetF, num_sup,
                                    rpnG, supSetG, num_supG,
                                    strapT, targetNode,
                                    f, g, orgNet);

            (void)matchDeleteCharLinkedList(supSetG, 1);
            if(matchP){
                break; //exit(1);
            }
#ifdef MATCH_DEBUG_SHORT
            cout<<"\n.....Function ..... \n";
            PrintCharList(rpnFunc);
            PrintCharList(supSetF);
            cout<<"...Cell Function .. "<<curCell->GetName()<<"\n";
            PrintCharList(rpnG);
            PrintCharList(supSetG);
            cout<<"PTL num_spuSet matched\n";
            cout << "PTL MATCHED !!!!!!!!!\n";
#endif
        } else {
            (void)matchDeleteCharLinkedList(supSetG, 1);
        }
    }

    listSize = strapRpnG->ListSize();
    for(n = 0; n < listSize; n++){
        strapRpnG->Reset();
        rpnG = strapRpnG->Data(); //
        (void)matchDeleteCharLinkedList(rpnG, 1);
        delete strapRpnG->DeleteCurrentElem();
    }
}

delete strapRpnG;
listSize = strapTable->ListSize();
for(n = 0; n < listSize; n++){
    strapTable->Reset();
    rpnG = strapTable->Data(); //
    (void)matchDeleteCharLinkedList(rpnG, 1);
    delete strapTable->DeleteCurrentElem();
}
delete strapTable;
(void)matchDeleteCharLinkedList(supSetG, 1);
}
if(matchP) break;
(void)matchDeleteCharLinkedList(supSetF, 1);
(void)matchDeleteCharLinkedList(rpnFunc, 1);
delete funcGraph;
}

listSize = fgList->ListSize();
for(n=0;n<listSize;n++){
    fgList->Reset();
    delete fgList->DeleteCurrentElem();
}
delete fgList;
}

/*****
 * MapGraph.h
 * Class definition of mapped circuit and its member functions
 *****/
#ifdef MAPGRAPH_CLASS
#define MAPGRAPH_CLASS

#include <iostream.h>
#include <fstream.h>

#include "MapNode.h"
#include "lElem.h"
#include "LinkedList.h"

// Definition of Graph class
class MapGraph
{
private:
    char * network_name;

    LinkedList<MapNode *> *pilist; // Lists of all primary inputs
    LinkedList<MapNode *> *polist; // Lists of all primary outputs
    LinkedList<MapNode *> *allnodes; // Lists of all pi/po/gates

    //int numgate, numpi, numpo, numedge;
    int numgate, numpi, numpo;
    int size, unique_id;

    int visited;

    double area_given; // Required area
    double delay_given; // Required delay
    double power_given; // Required power

    double area;
    double delay;
    double power;

    int used_in_gchom;

public:
    MapGraph();
    ~MapGraph(); // Destructor

    void SetName(char *name);
    char *GetName();
    int GetSize();

    void SetArea(double narea);
    double GetArea(void);

    void SetDelay(double ndelay);
    double GetDelay(void);

    void SetVisited(int flag);
    int GetVisited(void);

    void SetUsed(int flag);
    int GetUsed(void);

    // Node modification method
    void InsertPiNode(MapNode *node_name);
    void InsertPoNode(MapNode *node_name);
    void InsertGate(MapNode *gate_name);

    LinkedList<MapNode *> * GetPiList(void);
    LinkedList<MapNode *> * GetPoList(void);
    LinkedList<MapNode *> * GetGateList(void);
    MapNode * GetGate(char *gate_name, NodeType ntype, NodeFunc nfunc);
    int DeleteGate(char *gate_name);

    int FindNode(LinkedList<MapNode *> *L, char *item_name);
    MapNode * FindOrCreateNode(char *node_name, int &position,
                               NodeType ntype=UNASSIGNED,
                               NodeFunc nfunc=NODE_UNDEFINED);

    //int numgate, numpi, numpo, numedge;
    int GetNumPi(void);
    void SetNumPi(int npi);

```

```

    int GetNumPo(void);
    void SetNumPo(int npo);
    int GetNumGate(void);
    void SetNumGate(int ngate);

    void PrintPiNodes(void);
    void PrintPoNodes(void);
    void PrintAllNodes(void);
    void Print_Node_Type(int type);
    void Print_Node_Func(int type);

    void PrintNodesOfList(LinkedList<MapNode *> *nodeList);
};

// Constructor
MapGraph::MapGraph()
{
    network_name = NULL;
    allnodes = new LinkedList<MapNode *>;
    pilist = new LinkedList<MapNode *>;
    polist = new LinkedList<MapNode *>;

    area_given = area = 0.0;
    delay_given = delay = 0.0;
    power_given = power = 0.0;
    size = numgate = numpi = numpo = 0;
    unique_id = 1;
    visited = 0;
    used_in_gchrom = 0;
};

// Destructor
MapGraph::~MapGraph()
{
    polist->ClearList();
    delete polist;
    pilist->ClearList();
    delete pilist;
    allnodes->ClearList();
    delete allnodes;
    delete network_name;
};

void MapGraph::SetName(char *name)
{
    if(!name){
        free(network_name);
        network_name = strdup(name);
    }
}

char * MapGraph::GetName() { return network_name; }
int MapGraph::GetSize() { return size; }

void MapGraph::SetArea(double narea){ area = narea;}
double MapGraph::GetArea(void){return area;}

void MapGraph::SetDelay(double ndelay){delay = ndelay;}
double MapGraph::GetDelay(void){return delay;}

void MapGraph::SetVisited(int flag){visited = flag;}
int MapGraph::GetVisited(void){return visited;}

void MapGraph::SetUsed(int new_flag){used_in_gchrom = new_flag;}
int MapGraph::GetUsed(void){return used_in_gchrom;}

LinkedList<MapNode *> * MapGraph::GetGateList(void) {return allnodes;}
LinkedList<MapNode *> * MapGraph::GetPiList(void) {return pilist;}
LinkedList<MapNode *> * MapGraph::GetPoList(void) {return polist;}

void MapGraph::InsertPiNode(MapNode *item)
{
    item->SetNodeId(size);
    pilist->InsertRear(item);
    allnodes->InsertRear(item);
    numpi++; numgate++;
    size++;
}

void MapGraph::InsertPoNode(MapNode *item)
{
    item->SetNodeId(size);
    polist->InsertRear(item);
    allnodes->InsertRear(item);
    numpo++; numgate++;
    size++;
}

void MapGraph::InsertGate(MapNode *gate_name)
{
    gate_name->SetNodeId(size);
    allnodes->InsertRear(gate_name);
    numgate++;
    size++;
}

// Retrive MapNode class by searching the node_name
MapNode *
MapGraph::GetGate(char *gate_name, NodeType &ntype, NodeFunc &nfunc)
{
    int position;
    MapNode *tempNode;

    if ((position = FindNode(allnodes, gate_name)) != -1) {
        allnodes->Reset(position);
        tempNode = allnodes->Data();
        ntype = tempNode->GetNodeType();
        nfunc = tempNode->GetNodeFunc();
        return tempNode;
    }
    else {
        return tempNode = NULL;
    }
}

// Delete a MapNode (pi, po, gates) searching the node name
// Returns 0 if it exist and deleted, -1 otherwise
int MapGraph::DeleteGate(char *gate_name)
{
    int position;
    char *temp_name;
    MapNode *tempNode;
    NodeType type;

    if ((position = FindNode(allnodes, gate_name)) != -1) {
        allnodes->Reset(position);
        tempNode = allnodes->Data();
        temp_name = tempNode->GetName();
        type = tempNode->GetNodeType();

        // Delete all edges that are connected to this MapNode
        // 1. find all nodes (fanins and fanouts)
        // 2. find all edges (returns position in the edgelist)
        // 3. delete those edges from the list

        tempNode = allnodes->DeleteCurrent();
        numgate--;
        delete tempNode->GetFaninList();
        delete tempNode->GetFanoutList();
        delete tempNode;

        if (type == PRIMARY_INPUT) {
            position = FindNode(pilist, temp_name);
            if (position == -1) {
                cout << " MapGraph::DeleteGate("<< gate_name << ")" << endl;
                cout << " MapNode is in GateLists but not in PiList !!!";
                cout << endl;
                exit(1);
            }
            pilist->Reset(position);
            tempNode = pilist->DeleteCurrent();
            numpi--;
            delete tempNode->GetFaninList();
            delete tempNode->GetFanoutList();
            delete tempNode;
            return 0;
        }
        else if (type == PRIMARY_OUTPUT) {
            position = FindNode(polist, temp_name);
            if (position == -1) {
                cout << " MapGraph::DeleteNode("<< gate_name << ")" << endl;
                cout << " MapNode is in GateLists but not in PoList !!!";
                cout << endl;
                exit(1);
            }
            polist->Reset(position);
            tempNode = polist->DeleteCurrent();
            numpo--;
            delete tempNode;
            return 0;
        }
        return 0;
    }
    return -1;
}

// Find a MapNode by searching pilist, polist, gatelist.
// Returns item position , if not found return -1.
int MapGraph::FindNode(LinkedList<MapNode *> *L, char *item_name)
{
    int item_position = -1;

    if (L->front->Next != L->rear) {
        isElem<MapNode *> *temp_node = L->front->Next;
        while(temp_node->Next != NULL) {
            item_position++;
            if(strcmp(temp_node->data->name, item_name)==0) {
                return item_position;
            }
            temp_node = temp_node->Next;
        }
    }
    return -1;
}

// Find a node, if not exist create a node and return it's pointer and
// position in allnides. If it is ceated, position=-1 is returned.
MapNode * MapGraph::FindOrCreateNode(char *node_name, int &position,
    NodeType ntype, NodeFunc nfunc)
{
    position = -1;
    if ((position = FindNode(allnodes, node_name)) != -1) {
        MapNode *tempNode;
        allnodes->Reset(position);
        tempNode = allnodes->Data();
        return tempNode;
    }
    else {
        MapNode *tempNode = new MapNode;
        tempNode->SetName(node_name);

        if (ntype == PRIMARY_INPUT) {
            tempNode->SetNodeType(ntype);
            tempNode->SetNodeFunc(nfunc);
            InsertPiNode(tempNode);
        }
        else if (ntype == PRIMARY_OUTPUT) {

```

```

        tempNode->SetNodeType(nctype);
        tempNode->SetNodeFunc(nfunc);
        InsertPoNode(tempNode);
    }
    else {
        tempNode->SetNodeType(nctype);
        tempNode->SetNodeFunc(nfunc);
        InsertGate(tempNode);
    }
    return tempNode;
}
}

int MapGraph::GetNumPi(void){ return numpi;}
void MapGraph::SetNumPi(int npi){numpi = npi;}

int MapGraph::GetNumPo(void){ return numpo;}
void MapGraph::SetNumPo(int npo){ numpo = npo;}

int MapGraph::GetNumGate(void){return numgate;}
void MapGraph::SetNumGate(int ngate){numgate = ngate;}

// Print all Primary Input MapNodes and it's fanin/fanout nodes
void MapGraph::PrintPiNodes(void)
{
    int position = -1;
    int fanin_pos = -1;
    int fanout_pos = -1;
    char *node_name;
    NodeType node_type;
    NodeFunc node_func;

    cout << "Printing Input Lists of MapGraph !!!!"<<endl;
    cout << "Number of inputs = "<< numpi << endl;

    if (!polist->isEmpty()) {
        lsElem<MapNode *> *tempPI = polist->front->Next;
        while (tempPI->Next != NULL) {
            position++;
            polist->Reset(position);
            node_name = polist->Data()->name;
            node_type = polist->Data()->GetNodeType();
            node_func = polist->Data()->GetNodeFunc();
            printf("%s %d %d ", node_name, node_type, node_func);
            if (!polist->Data()->fanin->isEmpty()) {
                lsElem<MapNode *> *tempFI = polist->Data()->fanin->front->Next;
                fanin_pos = -1;
                cout << "fanin : ";
                while(tempFI->Next != NULL) {
                    fanin_pos++;
                    polist->Data()->fanin->Reset(fanin_pos);
                    cout<<polist->Data()->fanin->Data()->name <<" ";
                    tempFI = tempFI->Next;
                }
            }
            if (!polist->Data()->fanout->isEmpty()) { //is_polist_empty
                lsElem<MapNode *> *tempFO = polist->Data()->fanout->front->Next;
                fanout_pos = -1;
                cout << "fanout : ";
                while(tempFO->Next != NULL) {
                    fanout_pos++;
                    polist->Data()->fanout->Reset(fanout_pos);
                    cout<<polist->Data()->fanout->Data()->name <<" ";
                    tempFO = tempFO->Next;
                }
            }
            tempPI = tempPI->Next;
            cout<<endl;
        }
    }

    // Print all Primary Output MapNodes and it's fanin/fanout nodes
    void MapGraph::PrintPoNodes(void)
    {
        int position = -1;
        int fanin_pos = -1;
        int fanout_pos = -1;
        char *node_name;
        NodeType node_type;
        NodeFunc node_func;

        cout << "Printing Output Lists of MapGraph !!!!"<<endl;
        cout << "Number of outputs = "<< numpo << endl;

        if (!polist->isEmpty()) {
            lsElem<MapNode *> *tempPO = polist->front->Next;
            while (tempPO->Next != NULL) {
                position++;
                polist->Reset(position);
                node_name = polist->Data()->name;
                node_type = polist->Data()->GetNodeType();
                node_func = polist->Data()->GetNodeFunc();
                printf("%s %d %d ", node_name, node_type, node_func);

                if (!polist->Data()->fanin->isEmpty()) {
                    lsElem<MapNode *> *tempFI = polist->Data()->fanin->front->Next;
                    fanin_pos = -1;
                    cout << "fanin : ";
                    while(tempFI->Next != NULL) {
                        fanin_pos++;
                        polist->Data()->fanin->Reset(fanin_pos);
                        cout<<polist->Data()->fanin->Data()->name <<" ";
                        tempFI = tempFI->Next;
                    }
                }
                if (!polist->Data()->fanout->isEmpty()) { //is_polist_empty
                    lsElem<MapNode *> *tempFO = polist->Data()->fanout->front->Next;
                    fanout_pos = -1;
                    cout << "fanout : ";
                    while(tempFO->Next != NULL) {
                        fanout_pos++;
                        polist->Data()->fanout->Reset(fanout_pos);
                        cout<<polist->Data()->fanout->Data()->name <<" ";
                        tempFO = tempFO->Next;
                    }
                }
                tempPO = tempPO->Next;
                cout<<endl;
            }
        }

        // Print all MapNodes and it's fanin/fanout nodes
        void MapGraph::PrintAllNodes(void)
        {
            MapNode *aNode, *inNode, *outNode;
            LinkedList<MapNode *> *inList, *outList;

            cout << "Printing All Lists of MapGraph !!!!"<<endl;
            cout << "Number of allnodes = "<< numgate << endl;

            if (!allnodes->isEmpty()) {
                for(allnodes->Reset();allnodes->EndOfList();allnodes->Next()){
                    aNode = allnodes->Data();
                    printf("%3d[%2d] (%2d) : %6s ", allnodes->position,aNode->GetLevel(),\
                        aNode->GetVisit(),aNode->GetName());
                    Print_Node_Type(allnodes->Data()->type);
                    Print_Node_Func(allnodes->Data()->func);
                    cout << allnodes->Data()->id << " ";

                    inList = aNode->GetFaninList();
                    if (!inList->isEmpty()) {
                        cout << "fanin : ";
                        for(inList->Reset();inList->EndOfList();inList->Next()){
                            inNode = inList->Data();
                            cout<< inNode->GetName() <<" ";
                        }
                        cout << "\t";
                    }

                    outList = aNode->GetFanoutList();
                    if (!outList->isEmpty()) { //is_polist_empty
                        cout << "fanout : ";
                        for(outList->Reset();outList->EndOfList();outList->Next()){
                            outNode = outList->Data();
                            cout<< outNode->GetName()<<" ";
                        }
                    }
                    cout<<endl;
                } // for loop
            }

            void MapGraph::Print_Node_Type(int type)
            {
                switch (type) {
                    case 0: cout << "UNASSIGNED(0) ";
                           break;
                    //case 1: cout << "PRIMARY_INPUT(1) ";
                    case 1: cout << "RY_INPUT(1) ";
                           break;
                    //case 2: cout << "PRIMARY_OUTPUT(2) ";
                    case 2: cout << "Y_OUTPUT(2) ";
                           break;
                    case 3: cout << "INTERNAL(3) ";
                           break;
                    default: cout << "Invalid MapNode Type ("<<type<<") ";
                }
            }

            void MapGraph::Print_Node_Func(int type)
            {
                switch (type) {
                    case 0: cout << "NODE_UNDEFINED ";
                           break;
                    case 1: cout << "NODE_PI ";
                           break;
                    case 2: cout << "NODE_PO ";
                           break;
                    case 3: cout << "NODE_O ";
                           break;
                    case 4: cout << "NODE_1 ";
                           break;
                    case 5: cout << "NODE_BUF ";
                           break;
                    case 6: cout << "NODE_INV ";
                           break;
                    case 7: cout << "NODE_AND ";
                           break;
                    case 8: cout << "NODE_OR ";
                           break;
                    case 9: cout << "NODE_NAND ";
                           break;
                    case 10: cout << "NODE_NOR ";
                           break;
                    case 11: cout << "NODE_CPLX ";
                           break;
                    case 12: cout << "NODE_PTL1 ";
                           break;
                    case 13: cout << "NODE_PTL2 ";
                           break;
                    case 14: cout << "NODE_PTL3 ";
                           break;
                    default: cout << "Invalid MapNode Function("<<type<<")";
                }
            }

            // Print all MapNodes and it's fanin/fanout nodes
            void MapGraph::PrintNodesOfList(LinkedList<MapNode *> *nodeList)
            {
                int position = -1;
                int fanin_pos = -1;
                int fanout_pos = -1;

```

```

cout << "Printing All MapNodes In a Lists !!!"<<endl;
cout << "Size of List = "<< nodeList->ListSize() << endl;

if (!nodeList->isEmpty()) {
    lsElem<MapNode *> *tempAll = nodeList->front->Next;
    while (tempAll->Next != NULL) {
        position++;
        nodeList->Reset(position);
        printf("%3d : %6s ", position, nodeList->Data()->name);
        Print_Node_Type(nodeList->Data()->type);
        Print_Node_Func(nodeList->Data()->func);

        cout << nodeList->Data()->id << " ";

        if (!nodeList->Data()->fanin->isEmpty()) {
            lsElem<MapNode *> *tempFI = nodeList->Data()->fanin->front->Next;
            fanin_pos = -1;
            cout << "fanin : ";
            while(tempFI->Next != NULL) {
                fanin_pos++;
                nodeList->Data()->fanin->Reset(fanin_pos);
                cout<<nodeList->Data()->fanin->Data()->name <<" ";
                tempFI = tempFI->Next;
            }
            cout << "t";
        }
        if (!nodeList->Data()->fanout->isEmpty()) { //is_polist_empty
            lsElem<MapNode *> *tempFO = nodeList->Data()->fanout->front->Next;
            fanout_pos = -1;
            cout << "fanout : ";
            while(tempFO->Next != NULL) {
                fanout_pos++;
                nodeList->Data()->fanout->Reset(fanout_pos);
                cout<<nodeList->Data()->fanout->Data()->name <<" ";
                tempFO = tempFO->Next;
            }
        }
        tempAll = tempAll->Next;
        cout<<endl;
    }
}
#endif

/*****
 * MapGraph.h
 * Class definition of mapped circuit and its member funtions
 *****/
#ifdef MAPNODE_CLASS
#define MAPNODE_CLASS
#include "LinkedList.h"
#include "Node.h"

class MapNode
{
private:
    char *name; // node name
    Match *mappedCell; // match for a solution of the node
    NodeType type; // node type
    NodeFunc func; // node function
    int id; // node id
    int numfanin; // Number of Inputs
    LinkedList<MapNode *> *fanin; // List of fanin nodes
    int numfanout; // Number of fanout nodes
    LinkedList<MapNode *> *fanout; // List of fanin nodes

    double area; // Area
    double power; // Power for a cell (reserved)
    double tDR; // Rising delay
    double tDF; // Falling delay
    double tTR; // Rising slope
    double tTF; // Falling slope

    int level;
    int visited;
    int glevel;

public:
    MapNode(void); // Default constructor
    MapNode(char *&name, NodeType type, NodeFunc func, int id);
    ~MapNode(void); // Destructor
    void SetName(char *nname);
    char * GetName(void);
    void SetNodeType(NodeType gtype);
    NodeType GetNodeType(void);
    void SetNodeFunc(NodeFunc gfunc);
    NodeFunc GetNodeFunc(void);
    void SetNodeId(int gid);
    int GetNodeId(void);

    void SetLevel(int level);
    void SetGLevel(int level);
    int GetLevel(void);
    int GetGLevel(void);

    void MarkVisit(int mark=1);
    int GetVisit(void);

    // properties
    void SetArea(double narea);
    void SetPower(double npower);

    void SettDR(double ntDR);
    void SettDF(double ntDF);
    void SettTR(double ntTR);
    void SettTF(double ntTF);

    double GetArea(void);

    double GettDR(void);
    double GettDF(void);
    double GettTR(void);
    double GettTF(void);
    double GetPower(void);

    Match * GetMatch(void);
    void SetMatch(Match *&nmatch);

    int GetNumFanin(void);
    int GetNumFanout(void);
    void SetNumFanin(int nfanin);
    void SetNumFanout(int nfanout);

    void InsertFanin(MapNode *&N);
    void DeleteFanin(MapNode *&N);
    void InsertFanout(MapNode *&N);

    LinkedList<MapNode *> * GetFaninList(void);
    LinkedList<MapNode *> * GetFanoutList(void);

    friend class MapGraph;
};

// Default Constructor
MapNode::MapNode()
{
    name = new char[63];
    fanin = new LinkedList<MapNode *>;
    fanout = new LinkedList<MapNode *>;

    mappedCell = NULL;
    numfanin = 0;
    numfanout = 0;

    area = power = 0.0;
    tDR = tDF = tTR = tTF = 0.0;

    visited = -1;
    level = -1;
    glevel = -1;
}

// Constructor
MapNode::MapNode(char *&gname, NodeType gtype, NodeFunc gfunc, int gid)
{
    name = new char[63];
    name = gname;
    type = gtype;
    func = gfunc;
    id = gid;

    fanin = new LinkedList<MapNode *>;
    fanout = new LinkedList<MapNode *>;

    numfanin = 0;
    numfanout = 0;

    area = power = 0.0;
    tDR = tDF = tTR = tTF = 0.0;

    visited = -1;
    level = -1;
}

// Default destructor
MapNode::~MapNode(void) { };

// Set node name
void MapNode::SetName(char *nname) { strcpy(name, nname); }

// Get node name
char * MapNode::GetName(void) { return name; }

// Set node type
void MapNode::SetNodeType(NodeType gtype) { type = gtype; }

// Get node type
NodeType MapNode::GetNodeType(void) { return type; }

// Set node func
void MapNode::SetNodeFunc(NodeFunc gfunc) { func = gfunc; }

// Get node func
NodeFunc MapNode::GetNodeFunc(void) { return func; }

// set node id (number)
void MapNode::SetNodeId(int gid) { id = gid; }

// get node id
int MapNode::GetNodeId(void) { return id; }

// set node id (number)
void MapNode::SetLevel(int nlevel) { level = nlevel; }

// get node id
int MapNode::GetLevel(void) { return level; }

// set global level (same as the glevel in origNet)
void MapNode::SetGLevel(int nlevel) { glevel = nlevel; }

// return global level
int MapNode::GetGLevel(void) { return glevel; }

void MapNode::MarkVisit(int mark) {visited = mark;}
int MapNode::GetVisit(void) {return visited;}

void MapNode::SetArea(double narea) { if (narea >= 0){ area = narea; } }
void MapNode::SetPower(double npower) { if (npower >= 0){ power = npower; } }

void MapNode::SettDR(double ntDR)
{
    if(ntDR >= 0){tDR = ntDR;}
}
void MapNode::SettDF(double ntDF)
{
    if(ntDF >= 0){tDF = ntDF;}
}
void MapNode::SettTR(double ntTR) { if(ntTR >= 0){tTR = ntTR;}}

```

```

void MapNode::SettTF(double ntTF) { if(ntTF >= 0){tTF = ntTF;}}
double MapNode::GetArea(void) { return area; }
double MapNode::GetPower(void) { return power; }
double MapNode::GettDF(void) { return tDF; }
double MapNode::GettDR(void) { return tDR; }
double MapNode::GettTF(void) { return tTF; }
double MapNode::GettTR(void) { return tTR; }

Match * MapNode::GetMatch(void) { return mappedCell; }
void MapNode::SetMatch(Match * & nmatch){ mappedCell = nmatch; }

int MapNode::GetNumFanin(void) { return numfanin; }
int MapNode::GetNumFanout(void) { return numfanout; }
void MapNode::SetNumFanin(int nfanin) { numfanin = nfanin; }
void MapNode::SetNumFanout(int nfanout) { numfanout = nfanout; }

void MapNode::InsertFanin(MapNode * & gate)
{
    fanin->InsertRear(gate);
    numfanin++;
}

void MapNode::DeleteFanin(MapNode * & gate)
{
    char *gate_name, *in_name;
    MapNode *inNode;

    gate_name = gate->GetName();
    for(fanin->Reset(); !fanin->EndOfList(); fanin->Next()){
        inNode = fanin->Data();
        in_name = inNode->GetName();
        if(strcmp(in_name, gate_name)==0) break;
    }
    fanin->DeleteCurrent();
    numfanin--;
}

void MapNode::InsertFanout(MapNode * & gate)
{
    fanout->InsertRear(gate);
    numfanout++;
}

// Get fanin list of a node
LinkedList<MapNode * > * MapNode::GetFaninList(void)
{
    return fanin;
}

// Get fanout list of a node
LinkedList<MapNode * > * MapNode::GetFanoutList(void)
{
    LinkedList<MapNode * > *templist;

    templist = fanout;
    return templist;
}
#endif // MAPNODE_CLASS

/*****
 * Delay_cal_new.cc
 * Delay and Cost (Area) calculation
 *****/
#include <fstream.h>
#include <stdlib.h>
#include <ctype.h> // used for function 'isdigit'
#include <string.h>
#include <stdio.h> // used for function 'isdigit'
#include <math.h>
#include <new.h>
#include <time.h> // for seconds()

/* STL */
#include <string>
#include "util.h" // used for sug-graph utility
#include "Utils.h"

//----- Global Chrom struct -----
struct GlChrom {
public:
    LinkedList<MapGraph * > *gChromosome;
    double cp_delay;
    double area;

    GlChrom();
    ~GlChrom(void);
};

// Default glCHROM constructor
GlChrom::GlChrom()
{
    gChromosome = new LinkedList<MapGraph * >;
    cp_delay = 0;
    area = 0;
};

GlChrom::~GlChrom()
{
    gChromosome->ClearList();
    delete gChromosome;
};

//----- Print Routines -----
void gaPrintPinMap(Match *aMatch)
{
    LinkedList<pinMap * > *pMap;
    pinMap *curPin;

    cout << " "<<aMatch->cell_name<< " ";
    pMap = aMatch->pinmap;

    for(pMap->Reset(); !pMap->EndOfList(); pMap->Next()){
        curPin = pMap->Data();
        cout<<curPin->cell_pin<<"("<<curPin->net_node<<" ";
    }
    cout << "\n";

    // Print pin map of a node with library cells
    void gaPrintPinMapWithCells(MapGraph *subChromosome)
    {
        LinkedList<MapNode * > *subGList, *foList;
        MapNode *curNode, *foNode; Match *curMatch;
        NodeFunc curNode_func;

        subGList = subChromosome->GetGateList();
        for(subGList->Reset(); !subGList->EndOfList(); subGList->Next()){
            curNode = subGList->Data();
            curNode_func = curNode->GetNodeFunc();
            if((curNode_func!=NODE_PO && curNode_func!=NODE_PI){
                foList = curNode->GetFanoutList();
                if(foList->ListSize() > 0){
                    for_each(foList){
                        foNode = foList->Data();
                        //cout<<" "<<foNode->GetName();
                        cout<<" ("<<foNode->GetName()<<" ";
                    }
                } else {
                    cout<<" ("<<" ";
                }
                cout<<" "<<curNode->GetName();
                curMatch = curNode->GetMatch();
                (void)gaPrintPinMap(curMatch);
            }
        }
    }

    void gasPrintGlobalChromosome(LinkedList<MapGraph * > *gChromosome, int wNode=0)
    {
        MapGraph *subChrom;
        int index = -1;

        for(gChromosome->Reset(); !gChromosome->EndOfList(); gChromosome->Next()){
            index++;
            subChrom = gChromosome->Data();

            cout<<"\n -- SubChromosome # ["<<index
            <<" (u = "<<subChrom->GetUsed()<<"\n";
            if(wNode==1){
                subChrom->PrintPiNodes();
                subChrom->PrintAllNodes();
                cout<<"\n";
            }
            (void)gaPrintPinMapWithCells(subChrom);
            cout<<"\n";
        }
    }

    //----- Evaluation -----
    // Bottom to Top Traversal
    // Get a set of nodes whose all fanins's visit flags are 1.
    int gaB2TGetStartingNodes(MapGraph *aChromosome, Queue<MapNode * > *nodeQ)
    {
        LinkedList<MapNode * > *nodeList, *finList;
        MapNode *curNode, *aNode;
        int pos_node, num_node = 0, visit = -1;

        nodeList = aChromosome->GetGateList();
        for(nodeList->Reset(); !nodeList->EndOfList(); nodeList->Next()){
            curNode = nodeList->Data();
            if((curNode->GetVisit() < 0){
                finList = curNode->GetFaninList();
                //---- Debug
                if((finList->ListSize()==0){
                    cout<<" currNode name = "<<curNode->GetName()<<" fanin = 0\n";
                    aChromosome->PrintPiNodes();
                    aChromosome->PrintPoNodes();
                    aChromosome->PrintAllNodes();
                }
                //---- Debug
                pos_node = 0;
                for(finList->Reset(); !finList->EndOfList(); finList->Next()){
                    aNode = finList->Data();
                    if((visit = aNode->GetVisit()) > 0){
                        pos_node++;
                    }
                }
                if((finList->ListSize()==pos_node){
                    nodeQ->QInsert(curNode);
                    num_node++;
                }
            }
        }
        return num_node;
    }

    // read a cell and its area for a gene (mapped node), and update gene->area
    void gaRetriveNodeArea(MapNode *aGene, NodeFunc nfunc,
        LinkedList<LibCell * > *cellLibrary)
    {
        LibCell *aCell;
        Match *aMatch;
        char *matchName, *aCellName;

        aMatch = aGene->GetMatch();
        matchName = aMatch->cell_name;
        for(cellLibrary->Reset(); !cellLibrary->EndOfList(); cellLibrary->Next()){
            aCell = cellLibrary->Data();
            aCellName = aCell->GetName();
            if((nfunc==NODE_PTL1 || nfunc==NODE_PTL2 || nfunc==NODE_PTL3){
                if(strcmp(matchName, aCellName, 6)==0){
                    aGene->SetArea(aCell->GetArea());
                    return;
                }
            }
        }
    }
}

```

```

    } else {
        if(strcmp(matchName, aCellName)==0){
            aGene->SetArea(aCell->GetArea());
            return;
        }
    }
}

// returns Minimum cost Chromosome index and its cost
double gaEvaluateGetMinAreaCost(LinkedList<MapGraph *> *gaPop, int *index,
                                int *numMinChrom)
{
    MapGraph *aChromosome;
    double curCost, min_cost = RAND_MAX;
    int count = -1;

    for(gaPop->Reset(); !gaPop->EndOfList(); gaPop->Next()){
        count++;
        aChromosome = gaPop->Data();
        curCost = aChromosome->GetArea();
        if(curCost < min_cost){
            *index = count;
            min_cost = curCost;
        }
    }

    // Finds the number of chromosomes that have the min_cost value
    count = 0;
    for(gaPop->Reset(); !gaPop->EndOfList(); gaPop->Next()){
        aChromosome = gaPop->Data();
        curCost = aChromosome->GetArea();
        if((min_cost-curCost)==0){
            count++;
        }
    }
    *numMinChrom = count;
    return min_cost;
}

// returns Maximum cost Chromosome index and its cost
double gaEvaluateGetMaxAreaCost(LinkedList<MapGraph *> *gaPop, int *index,
                                int non_visited = 0)
{
    MapGraph *aChromosome;
    double curCost, max_cost = 0.0;
    int count = -1, visit_flag=0;

    for(gaPop->Reset(); !gaPop->EndOfList(); gaPop->Next()){
        count++;
        aChromosome = gaPop->Data();
        curCost = aChromosome->GetArea();
        visit_flag = aChromosome->GetVisited();
        if(non_visited==1){
            if(curCost > max_cost && !visit_flag){
                *index = count;
                max_cost = curCost;
            }
        } else {
            if(curCost > max_cost){
                *index = count;
                max_cost = curCost;
            }
        }
    }
    return max_cost;
}

int gaIsVddGnd(char *name)
{
    if(strcmp(name, "GND")==0 || strcmp(name, "VDD")==0){
        return (1);
    } else {
        return (0);
    }
}

// Returns cell name of current match
char *gaGetMatchedCellName(Match *curMatch)
{
    char *match_cell;
    match_cell = (char *)calloc(64, sizeof(char));
    strcpy(match_cell, curMatch->cell_name);

    if(!strcmp(match_cell, "ptl_y21") || !strcmp(match_cell, "ptl_y22")){
        strcpy(match_cell, "ptl_y2");
    } else if(!strcmp(match_cell, "ptl_y2x1") || !strcmp(match_cell, "ptl_y2x2")){
        strcpy(match_cell, "ptl_y2x");
    } else if(!strcmp(match_cell, "ptl_y31") || !strcmp(match_cell, "ptl_y32")){
        strcpy(match_cell, "ptl_y3");
    } else if(!strcmp(match_cell, "ptl_y3x1") || !strcmp(match_cell, "ptl_y3x2")){
        strcpy(match_cell, "ptl_y3x");
    } else {
        strcpy(match_cell, curMatch->cell_name);
    }

    return match_cell;
}

// Returns library cell pointer by retrieving current matched cell from Lib
LibCell *gaGetMatchedLibCell(Match *curMatch, LinkedList<LibCell *> *cellLib)
{
    char *match_cell, *curcell_name;
    LibCell *curCell;

    match_cell = gaGetMatchedCellName(curMatch);
    for(cellLib->Reset(); !cellLib->EndOfList(); cellLib->Next()){
        curCell = cellLib->Data();
        curcell_name = curCell->GetName();
        if(strcmp(match_cell, curcell_name)==0){
            free(match_cell);
            return curCell;
        }
    }
}

cout << "In gaGetMatchedCell(), no cell named "<<curMatch->cell_name
<<" found in Cell Library\n";
cout << "Returns NULL pointer for pinList for a cell\n";
free(match_cell);
return (NULL);
}

// Returns a MapNode by looking at a Graph based on given name
MapNode *gaRetrieveMapNodeFromName(MapGraph *aChrom, MapNode *curNode,
                                    char *insig_name)
{
    LinkedList<MapNode *> *gateList;
    MapNode *inNode;
    int pos=-1;

    gateList = aChrom->GetGateList();
    if((pos=aChrom->FindNode(gateList, insig_name))!=-1){
        gateList->Reset(pos);
        inNode = gateList->Data();
        return inNode;
    } else {
        cout << "In gaRetrieveMapNodeFromName(), "<<insig_name
        <<" does not exist in a MapGraph.\n"
        <<" In "<<__FILE__<<" at line no. "<<__LINE__<<endl
        <<"NULL pointer will be returned\n";
        cout << "----- "<<curNode->GetName()<<" \n";
        aChrom->PrintPiNodes();
        aChrom->PrintPoNodes();
        aChrom->PrintAllNodes();
        return (NULL);
    }
}

// Sets all visit flag to -1 except PI node (to 1) of the tree
void gaInitNodesVisitFlag(MapGraph *aChromosome)
{
    Queue<MapNode *> Q;
    LinkedList<MapNode *> *piList, *finList;
    LinkedList<MapNode *> *poList, *gateList;

    MapNode *bNode, *inNode;
    char *bnode_name, *inode_name;
    int count = 0;

    // find output node
    gateList = aChromosome->GetGateList();
    piList = aChromosome->GetPiList();
    poList = aChromosome->GetPoList();

    for(gateList->Reset(); !gateList->EndOfList(); gateList->Next()){
        bNode = gateList->Data();
        bnode_name = bNode->GetName();
        if(aChromosome->FindNode(poList, bnode_name)!=-1 &&
            (bNode->GetNumFanout() > 1 || bNode->GetNodeType()==PRIMARY_OUTPUT)){
            Q.Insert(bNode);
            bNode->MarkVisit(-1);
            count++;
        }
    }

    if(count > 1){
        cout << "In gaInitNodesVisitFlag(), "
        <<"the # of output node (MFP or PO) is more than ONE!!\n";
        exit(1);
    }

    while(!Q.Empty()){
        bNode = Q.DeQueue();
        finList = bNode->GetFaninList();
        if(finList->ListSize()==0 && bNode->GetNodeFunc()!=NODE_PI){
            cout << "In gaInitNodesVisitFlag(), "
            <<"#of fanin is 0 at node "<<bNode->GetName()<<"\n";
            printf("In %s at line %d\n", __FILE__, __LINE__);
            exit(1);
        }
        for(finList->Reset(); !finList->EndOfList(); finList->Next()){
            inNode = finList->Data();
            inode_name = inNode->GetName();
            if(aChromosome->FindNode(piList, inode_name)!=-1){
                inNode->MarkVisit(1);
            } else {
                inNode->MarkVisit(-1);
                Q.Insert(inNode);
            }
        } // for_loop
    } // while_loop

// Returns the rise arrival time at the inputs of a node(gate)
double gaGetRiseArrivalTimeAtNode(MapNode *inNode)
{
    double rise_delay;
    return(rise_delay = inNode->Gettdr());
}

// Returns the fall arrival time at the inputs of a node(gate)
double gaGetFallArrivalTimeAtNode(MapNode *inNode)
{
    double fall_delay;
    return(fall_delay = inNode->Gettdf());
}

// Returns the rise transition time at the inputs of a node(gate)
double gaGetRiseTransitionTimeAtNode(MapNode *inNode)
{
    double rise_tran_time;
    return(rise_tran_time = inNode->Gettdt());
}

// Returns the fall transition time at the inputs of a node(gate)
double gaGetFallTransitionTimeAtNode(MapNode *inNode)

```

```

{
    double fall_tran_time;
    return(fall_tran_time = inNode->GetTF());
}

// Returns load capacitance of a node
double gaGetLoadCapOfaNode(MapNode *curNode, LinkedList<LibCell *> *cellLibrary)
{
    LinkedList<MapNode *> *foutList;
    MapNode *foNode;
    Match *foMatch;
    LinkedList<pinMap *> *matchPinList;
    pinMap *foPin;
    LinkedList<Pin *> *cPinList;
    Pin *cellPin;
    LibCell *curCell;

    char *cur_node_name, *out_sig_name, *cell_pin_name;
    double loadCap = 0, pinCap = 0;
    int found_a_fo_cap;

    foutList = curNode->GetFanoutList();
    cur_node_name = curNode->GetName();

    // 1) for each fanout node of the current node
    for(foutList->Reset(); !foutList->EndOfList(); foutList->Next()){
        found_a_fo_cap = FALSE;
        foNode = foutList->Data();
        if(foNode->GetNodeType() != PRIMARY_OUTPUT){
            // 2) Find current node_name in the pinList of the curr fanout match
            foMatch = foNode->GetMatch();
            matchPinList = foMatch->pinmap;
            //for_each(matchPinList){
            for(matchPinList->Reset(); !matchPinList->EndOfList(); matchPinList->Next()){
                foPin = matchPinList->Data();
                out_sig_name = foPin->net_name;
                if(!gaIsVddGnd(out_sig_name) &&
                    strcmp(cur_node_name, out_sig_name) == 0){
                    // 3) Find cell pin name which is mapped to current
                    // node_name
                    curCell = gaGetMatchedLibCell(foMatch, cellLibrary);
                    cPinList = curCell->GetPinList();
                    //for_each(cPinList){
                    for(cPinList->Reset(); !cPinList->EndOfList(); cPinList->Next()){
                        cellPin = cPinList->Data();
                        cell_pin_name = cellPin->GetPinName();
                        // 4) If it is matched to foPin->cell_pin, get Cap
                        if(strcmp(foPin->cell_pin, cell_pin_name) == 0){
                            pinCap = cellPin->GetInputCap();
                            found_a_fo_cap = TRUE;
                            break;
                        }
                    }
                }
            }
            if(found_a_fo_cap == TRUE) break;
        } //for_each pin of fanout match(cur_node_name, sig_name in foMatch)
    } else if (foNode->GetNodeType() == PRIMARY_OUTPUT){
        pinCap = 3.0;
    }
    loadCap = loadCap + pinCap;
}

return loadCap;
}

// Returns max delay value at current node with the current match
// t = max(Arr(i) + a(i) + b(i)*Slope(i) + c(i)*C_load), i is fanin of the cell
double gaCalculateGeneDelay(MapGraph *aChrom, MapNode *curNode, Match *curMatch, LinkedList<LibCell *> *cellLibrary)
{
    LinkedList<pinMap *> *matPinList; // pin list of the current match
    pinMap *curPin;
    LinkedList<Pin *> *cellPinList; // pin list of the matched cell
    Pin *cellPin = NULL;
    LibCell *curCell;

    char *sig_name;
    MapNode *inNode;
    double cLoad, arrTime, tDr, tDr_slop, tD, max_delay_r = 0.0;

    // 1) Get fanout load cap
    if(curNode->GetNodeType() == PRIMARY_OUTPUT){
        //cLoad = OUTPUT_LOAD_CAP;
        cLoad = 3.0;
    } else {
        cLoad = gaGetLoadCapOfaNode(curNode, cellLibrary);
    }

    matPinList = curMatch->pinmap;
    curCell = gaGetMatchedLibCell(curMatch, cellLibrary);
    cellPinList = curCell->GetPinList();

    // 2) for each pins of the current match
    max_delay_r = 0.0;
    for_each(matPinList){ // for each pins of the current match
        curPin = matPinList->Data();
        sig_name = curPin->net_name;
        if(gaIsVddGnd(sig_name)) continue;
        if((inNode = gaRetrieveMapNodeFromName(aChrom, curNode, sig_name)) == NULL){
            //return (-1);
            return (0);
        }
        arrTime = gaGetRiseArrivalTimeAtNode(inNode);

        //3) for each matched cell pin, check if its name is the same as sig_name
        for_each(cellPinList){
            cellPin = cellPinList->Data();
            if(strcmp(curPin->cell_pin, cellPin->GetPinName()) == 0){
                break;
            }
        }
        // 3-1) get matched pin delay (load independent and dependent delay)
        tDr = cellPin->GetTdr(); tDr_slop = cellPin->GetTdr_slop();
        tD = arrTime + tDr + tDr_slop*cLoad;
        max_delay_r = Max(max_delay_r, tD);
    }

    // set delay(max_delay) of current node's delay valde in node Class
    curNode->SetTDR(max_delay_r);
    return max_delay_r;
}

// returns max delay of a chromosome and sets input_node_name
// 0) reset visit flag of all nodes : 1 for inputs, -1 for others
// 1) find all inputs in aChromosome(tree), and insert them into a Q
// 2) Calculate delay from input to output by dequeuing Q, and enqueue
// current node's fanout
double gaEvaluateChromosomeDelay(MapGraph *aChromosome,
    LinkedList<LibCell *> *cellLibrary)
{
    Queue<MapNode *> *Q = new Queue<MapNode *>;
    int nnode;

    MapNode *curNode, *foNode;
    LinkedList<MapNode *> *foutList;
    Match *curMatch;
    double delay, total_delay = 0.0;

    (void)gaInitNodesVisitFlag(aChromosome);
    nnode = gaB2TGetStartingNodes(aChromosome, Q);

    while(!Q->IsEmpty()){
        curNode = Q->Dequeue();

        // find mapped cell for this node and find max_delay at output
        curMatch = curNode->GetMatch();
        delay = gaCalculateGeneDelay(aChromosome, curNode, curMatch, cellLibrary);
        total_delay = total_delay + delay;

        foutList = curNode->GetFanoutList();
        if(foutList->ListSize() != 0){
            for_each(foutList){
                foNode = foutList->Data();
                if(foNode->GetNodeFunc() != NODE_PO){
                    Q->Qinsert(foNode);
                }
            }
        }
        aChromosome->SetDelay(total_delay);
        delete Q;
        return total_delay;
    }

    // Calculate cost(area, delay) value for a Chromosome (Tree)
    void gaEvaluateChromosome(MapGraph *aChromosome,
        LinkedList<LibCell *> *cellLibrary,
        double *areaCost, double *delayCost)
    {
        LinkedList<MapNode *> *mGateList;
        MapNode *aGene;

        *areaCost = 0.0;
        *delayCost = -1.0;
        mGateList = aChromosome->GetGateList();
        /* for debug
        cout << "\nin Evaluate Chromosome\n";
        aChromosome->PrintPoNodes();
        aChromosome->PrintPiNodes();
        aChromosome->PrintAllNodes();
        */

        for(mGateList->Reset(); !mGateList->EndOfList(); mGateList->Next()){
            aGene = mGateList->Data();
            //cout << "aGene name = "<< aGene->GetName() << endl;
            if((aGene->GetNodeType() == PRIMARY_OUTPUT &&
                (aGene->GetNodeFunc() != NODE_PO) ||
                (aGene->GetNodeFunc() != NODE_PI))){
                (void)gaRetrieveNodeArea(aGene, aGene->GetNodeFunc(), cellLibrary);
                *areaCost = *areaCost + aGene->GetArea();
            }
            aChromosome->SetArea(*areaCost);
            *delayCost = gaEvaluateChromosomeDelay(aChromosome, cellLibrary);
        }

        double gaEvaluatePopulation(LinkedList<MapGraph *> *gaPop,
            LinkedList<LibCell *> *cellLibrary)
        {
            MapGraph *aChromosome; // a legal solution (tree)
            double areaCost, delayCost, totalAreaCost = 0.0;
            int index = -1;

            if(gaPop->ListSize() == 0){
                cout << "Population doesn't have any node !!!!!\n";
                return (0.0);
            }
            for(gaPop->Reset(); !gaPop->EndOfList(); gaPop->Next()){
                aChromosome = gaPop->Data();
                index ++;

                (void)gaEvaluateChromosome(aChromosome, cellLibrary, &areaCost, &delayCost);
                #ifdef GA_EVALUATE_DEBUG
                cout << "Sol # [" << index << "] :: ";
                cout << "Cost (area) = " << aChromosome->GetArea() << " : ";
                #endif
            }
        }
    }
}

```

```

        cout<< "Cost (delay) = "<< aChromosome->GetDelay() <<"\n";
    #endif
    }
    totalAreaCost = totalAreaCost + areaCost;
    return totalAreaCost/(index+1); // acerage area
}

// ----- Global chromosome delay calculation
// Set node's visit flag if the ndodes are (Pis) in PiList of origNet, else
// reset node's visit flag.
void gasInitNodesVisitFaig(Graph *origNet, LinkedList<MapGraph *> *gChromosome)
{
    LinkedList<Node *> *origPiList;
    MapGraph *subChrom; // mapped sub-graph
    LinkedList<MapNode *> *sub_chromGList; MapNode *subNode;
    char *subNodeName; int pos;

    origPiList = origNet->GetPiList();
    for(gChromosome->Reset(); !gChromosome->EndOfList(); gChromosome->Next()){
        subChrom = gChromosome->Data();
        sub_chromGList = subChrom->GetGateList();
        for_each(sub_chromGList){
            subNode = sub_chromGList->Data();
            subNodeName = subNode->GetName();
            pos = origNet->FindNode(origPiList, subNodeName);
            if(pos!=-1){
                subNode->MarkVisit(1);
            } else {
                subNode->MarkVisit(-1);
            }
        } // inner-for-loop
    }

    //nstart_node = gasGetStartingNodes(gChromosome, Q);
    // get bottom to top traversing starting nodes whose fanins are all visited
    int gasB2TGetStartingNodes(LinkedList<MapGraph *> *gChrom, Queue<MapNode *> *Q)
    {
        int nnode=0, num_posi_node;
        MapGraph *subChrom; MapNode *curNode, *finNode;
        LinkedList<MapNode *> *subGateList, *finList;

        for(gChrom->Reset(); !gChrom->EndOfList(); gChrom->Next()){
            subChrom = gChrom->Data();
            subGateList = subChrom->GetGateList();
            for(subGateList->Reset(); !subGateList->EndOfList(); subGateList->Next()){
                curNode = subGateList->Data();
                finList = curNode->GetFaninList();
                if((curNode->GetVisit() < 1) && (finList->ListSize()>0)){
                    //finList = curNode->GetFaninList();
                    num_posi_node = 0;
                    for(finList->Reset(); !finList->EndOfList(); finList->Next()){
                        finNode = finList->Data();
                        if(finNode->GetVisit()==1){
                            num_posi_node++;
                        }
                    }
                    if(num_posi_node == finList->ListSize()){
                        nnode++;
                        Q->Insert(curNode);
                    }
                }
            }
        }
        return nnode;
    }

    // Collect all nodes in a subChromosome whose all inputs are visited
    int gasGetSubB2TStartingNodes(MapGraph *subChrom, Queue<MapNode *> *Q)
    {
        int nnode=0, num_posi_node;
        LinkedList<MapNode *> *subGateList, *finList;
        MapNode *curNode, *finNode;

        subGateList = subChrom->GetGateList();
        for(subGateList->Reset(); !subGateList->EndOfList(); subGateList->Next()){
            curNode = subGateList->Data();
            finList = curNode->GetFaninList();
            if((curNode->GetVisit() < 1) && (finList->ListSize()>0)){
                num_posi_node = 0;
                for(finList->Reset(); !finList->EndOfList(); finList->Next()){
                    finNode = finList->Data();
                    if(finNode->GetVisit()==1){
                        num_posi_node++;
                    }
                }
                if(num_posi_node == finList->ListSize()){
                    nnode++;
                    Q->Insert(curNode);
                }
            }
        }
        return nnode;
    }

    //pinCap = gasFindNodeAtOutnodeAndGetCap(cur_node_name, foNode);
    double gasFindNodeAtOutnodeAndGetCap(char *cur_node_name, MapNode *foNode,
        LinkedList<LibCell *> *cellLib)
    {
        int found_fo_cap; double pinCap=0;
        char *fo_in_sig_name, *cell_pin_name;

        Match *foMatch;
        LinkedList<PinMap *> *fo_matchPinList;
        pinMap *fo_in_Pin;

        LibCell *fo_matchCell;
        LinkedList<Pin *> *cPinList;
        Pin *cellPin;

        found_fo_cap = FALSE;

        if(foNode->GetNodeFunc()!=NODE_PO){
            /* 2) find current node name in the fanout match pinList */
            foMatch = foNode->GetMatch();
            fo_matchPinList = foMatch->pinmap;

            for_each(fo_matchPinList){
                fo_in_Pin = fo_matchPinList->Data();
                fo_in_sig_name = fo_in_Pin->net_name;
                if(!fo_in_sig_name) break;
                strcmp(cur_node_name, fo_in_sig_name)==0){
                    // 3) find cell pin name which is mapped to curr node
                    fo_matchCell = gasGetMatchedLibCell(foMatch, cellLib);
                    cPinList = fo_matchCell->GetPinList();
                    for_each(cPinList){
                        cellPin = cPinList->Data();
                        cell_pin_name = cellPin->GetPinName();

                        // 4) If cell_pin_name and fo_in_Pin name are same
                        // retrieve cell input cap
                        if(strcmp(fo_in_Pin->cell_pin, cell_pin_name)==0){
                            pinCap = cellPin->GetInputCap();
                            found_fo_cap = TRUE;
                            break;
                        }
                    } // for_each cPinList
                }
            } // for_each fo_matchPinList
            if(found_fo_cap == TRUE) break;
        } else if(foNode->GetNodeFunc()==NODE_PO){
            pinCap = OUTPUT_LOAD_CAP;
        }
        return pinCap;
    }

    //foQueue = gasGetFanoutListAcrossChromosome(curNode, gChromosome);
    // Returns a list of fanout nodes of the curNode across the sub-graph boundary
    void gasGetFanoutListAcrossChromosomeX(MapNode *curNode,
        LinkedList<MapGraph *> *gChromosome,
        Queue<MapNode *> *foQueue)
    {
        MapGraph *subChrom;
        MapNode *subNode, *finNode;
        LinkedList<MapNode *> *subGateList, *finList;
        char *cur_node_name, *fo_fin_name;

        cur_node_name = curNode->GetName();
        for(gChromosome->Reset(); !gChromosome->EndOfList(); gChromosome->Next()){
            subChrom = gChromosome->Data();
            subGateList = subChrom->GetGateList();
            for(subGateList->Reset(); !subGateList->EndOfList(); subGateList->Next()){
                subNode = subGateList->Data();
                if(subNode->GetNodeType()==PRIMARY_INPUT){
                    finList = subNode->GetFaninList();
                    for(finList->Reset(); !finList->EndOfList(); finList->Next()){
                        finNode = finList->Data();
                        fo_fin_name = finNode->GetName();
                        if(strcmp(cur_node_name, fo_fin_name)==0){
                            foQueue->Insert(subNode);
                        }
                    }
                }
            } // inner for_loop
        } // for_loop
        //return foQueue;
    }

    //foQueue = gasGetFanoutListAcrossChromosome(curNode, gChromosome);
    // Returns a list of fanout nodes of the curNode across the sub-graph boundary
    void gasGetFanoutListAcrossChromosome(MapNode *curNode,
        LinkedList<MapGraph *> *gChromosome,
        Queue<MapNode *> *foQueue)
    {
        MapGraph *subChrom;
        MapNode *subNode, *finNode;
        LinkedList<MapNode *> *subGateList, *finList;
        char *cur_node_name, *fo_fin_name;
        int used_sub_chrom;

        cur_node_name = curNode->GetName();
        for(gChromosome->Reset(); !gChromosome->EndOfList(); gChromosome->Next()){
            subChrom = gChromosome->Data();
            used_sub_chrom = subChrom->GetUsed();
            if(!used_sub_chrom) continue;

            subGateList = subChrom->GetGateList();
            for(subGateList->Reset(); !subGateList->EndOfList(); subGateList->Next()){
                subNode = subGateList->Data();
                if(subNode->GetNodeType()==PRIMARY_INPUT){
                    finList = subNode->GetFaninList();
                    for(finList->Reset(); !finList->EndOfList(); finList->Next()){
                        finNode = finList->Data();
                        fo_fin_name = finNode->GetName();
                        if(strcmp(cur_node_name, fo_fin_name)==0){
                            foQueue->Insert(subNode);
                        }
                    }
                }
            } // inner for_loop
        } // for_loop

        // Returns current node's load capacitance for global chromosome
        double gasGetCurrentNodeLoadCap(LinkedList<MapGraph *> *gChromosome,
            MapNode *curNode, LinkedList<LibCell *> *cellLib)
        {
            char *curName;
            LinkedList<MapNode *> *foutList; MapNode *foNode;
            int num_fanout; //10-05-00

            double loadCap = 0.0, pinCap;

```

```

//Queue<MapNode> *foQueue;
Queue<MapNode> *foQueue = new Queue<MapNode>;

curName = curNode->GetName();
num_fanout = curNode->GetNumFanout(); //10-05
// curNode is not mfp or po
if(curNode->GetNodeType() != PRIMARY_OUTPUT && num_fanout != 0){
    foutList = curNode->GetFanoutList();

    // 1) for_each fanout node of the current ndoe
    for(foutList->Reset(); !foutList->EndOfList(); foutList->Next()){
        foNode = foutList->Data();
        pinCap = gasFindNodeAtOutNodeAndGetCap(curName, foNode, cellLib);
        loadCap = loadCap + pinCap;
    } // for_each fanout node of current node

} else if(curNode->GetNodeType() == PRIMARY_OUTPUT &&
          curNode->GetNodeFunc() != NODE_PO){ // curNode = mfp
    (void)gasGetFanoutListAcrossChromosome(curNode, gChromosome, foQueue);
    while(!foQueue->IsEmpty()){
        foNode = foQueue->DeQueue();
        pinCap = gasFindNodeAtOutNodeAndGetCap(curName, foNode, cellLib);
        loadCap = loadCap + pinCap;
    }
}
delete foQueue;
return loadCap;
}

//inNode = gasRetrieveMappedNodeFromName(gChromosome, curNode, sig_name);
// Finds a curNode's fanin node (MapNode) based on sig_name and return it
MapNode *gasRetrieveMappedNodeFromName(LinkedList<MapGraph> *gChromosome,
                                       MapNode *curNode, char *in_sig_name)
{
    MapGraph *subChrom;
    LinkedList<MapNode> *subGateList, *foutList;
    MapNode *inNode = NULL;
    int pos, fout_pos;
    char *cur_name;

    cur_name = curNode->GetName();
    for(gChromosome->Reset(); !gChromosome->EndOfList(); gChromosome->Next()){
        subChrom = gChromosome->Data();
        subGateList = subChrom->GetGateList();
        pos = -1; fout_pos = -1;
        if((pos = subChrom->FindNode(subGateList, in_sig_name)) != -1){
            subGateList->Reset(pos);
            inNode = subGateList->Data();
            foutList = inNode->GetFanoutList();
            if((fout_pos = subChrom->FindNode(foutList, cur_name)) != -1 &&
                foutList->ListSize() != 0){
                break;
            }
        }
    }
    if(inNode == NULL){
        cout << "In gasRetrieveMappedNodeFromName(), "<< in_sig_name
              << " does not exist in a MapGraph.\n"
              << "NULL pointer will be returned\n";
    }
    return inNode;
}

void gasSetDelayOfCrossChromosomeNodes(LinkedList<MapGraph> *gChrom,
                                       MapNode *curNode, double tDr, double tDf,
                                       double tTr, double tTf,
                                       int gch_pos = -1)
{
    MapGraph *subChrom;
    LinkedList<MapNode> *subGateList, *subNodeInList;
    MapNode *subNode, *subInNode;
    char *cur_node_name, *subnode_name, *subinName;
    NodeFunc curFunc;
    NodeType curType;

    if(tDr < 0 && tDf < 0) return;
    if(tDr > 0) curNode->SettDR(tDr);
    if(tDf > 0) curNode->SettDF(tDf);
    if(tTr > 0) curNode->SettTR(tTr);
    if(tTf > 0) curNode->SettTF(tTf);

    curFunc = curNode->GetNodeFunc();
    curType = curNode->GetNodeType();
    curNode->MarkVisit(1);

    cur_node_name = curNode->GetName();
    for(gChrom->Reset(); !gChrom->EndOfList(); gChrom->Next()){
        subChrom = gChrom->Data();
        subGateList = subChrom->GetGateList();
        for_each(subGateList){
            subNode = subGateList->Data();
            subnode_name = subNode->GetName();
            if(curFunc == NODE_PO || curType == PRIMARY_OUTPUT){
                if(subNode->GetNodeType() == PRIMARY_INPUT){
                    //subnode_name = subNode->GetName();
                    if(strcmp(cur_node_name, subnode_name) == 0){
                        if(tDr >= 0) subNode->SettDR(tDr);
                        if(tDf >= 0) subNode->SettDF(tDf);
                        if(tTr >= 0) subNode->SettTR(tTr);
                        if(tTf >= 0) subNode->SettTF(tTf);
                        subNode->MarkVisit(1);
                    }
                } else {
                    // for_each(subGateList)
                }
            }
        }
    }

    // find mapped cell for this node and find max_delay at output
    // Set current node's delay at MapNode calss (a gene is a mapped Node)
    // t = max( Arr(1) + intrinsic(1) + b(i)*in_Slope(1) + c(i)*C_load )
}

void gasCalculateGlobalGeneDelay(LinkedList<MapGraph> *gChromosome,
                                MapNode *curNode,
                                LinkedList<LibCell> *cellLibrary, int gch_pos)
{
    Match *curMatch;
    LinkedList<pinMap> *matchPinList; // pin list of current node's match
    pinMap *curPin; // current pin's cell pin map to net node

    LibCell *curCell;
    LinkedList<Pin> *cellPinList; // pin list of the mapped cell
    Pin *cellPin=NULL;

    double cLoad, arrRise, arrFall, tranRise, tranFall;
    double tDr_pin, tDr_slop_pin, tDr_itran_pin, tDr_rise;
    double tDf_pin, tDf_slop_pin, tDf_itran_pin, tDf_fall;
    double max_dly_rise, max_dly_fall;

    double tTr_pin, tTr_slop_pin, tTr_itran_pin, tTr_rise;
    double tTf_pin, tTf_slop_pin, tTf_itran_pin, tTf_fall;
    double max_tran_rise, max_tran_fall;

    MapNode *inNode = NULL;
    char *sig_name;

    // 1) Get current node's fanout load capacitance
    cLoad = gasGetCurrentNodeLoadCap(gChromosome, curNode, cellLibrary);

    curMatch = curNode->GetMatch();
    matchPinList = curMatch->pinMap;
    curCell = gasGetMatchedLibCell(curMatch, cellLibrary);
    cellPinList = curCell->GetPinList();

    // 2) for each pins of the current match
    max_dly_rise = 0.0;
    max_dly_fall = 0.0;
    max_tran_rise = 0.0;
    max_tran_fall = 0.0;
    for(matchPinList->Reset(); !matchPinList->EndOfList(); matchPinList->Next()){
        curPin = matchPinList->Data();
        sig_name = curPin->net_name;
        if(gasVddGnd(sig_name)) continue;

        // 2-1) inNode arrival time
        inNode = gasRetrieveMappedNodeFromName(gChromosome, curNode, sig_name);
        if(inNode == NULL){
            cout << "In gasCalculateGlobalGeneDelay(), node "<< curNode->GetName()
                  << "'s fanin("<< sig_name <<") does not exist.....\n";
            arrRise = arrFall = tranRise = tranFall = 0.0;
        } else {
            arrRise = gasGetRiseArrivalTimeAtNode(inNode);
            arrFall = gasGetFallArrivalTimeAtNode(inNode);
            tranFall = gasGetRiseTransitionTimeAtNode(inNode);
            tranRise = gasGetFallTransitionTimeAtNode(inNode);
        }

        // 3) for each matched cell pin,
        for(cellPinList->Reset(); !cellPinList->EndOfList(); cellPinList->Next()){
            cellPin = cellPinList->Data();
            if(strcmp(curPin->cell_pin, cellPin->GetPinName()) == 0){
                break;
            }
        }

        // 3-1) get matched cell's pin delay (intrinsic delay and load
        // dependent slop and input_tran_time
        // Rise delay(1)
        tDr_pin = cellPin->GetTDR();
        tDr_slop_pin = cellPin->GetTDR_slop();
        tDr_itran_pin = cellPin->GetTDR_itran();
        tDr_rise = arrRise + tDr_pin*1e-12 + tDr_slop_pin*cLoad*1e-15
                  + tDr_itran_pin*tranRise;
        max_dly_rise = Max(max_dly_rise, tDr_rise);

        // Fall delay(2)
        tDf_pin = cellPin->GetTDF();
        tDf_slop_pin = cellPin->GetTDF_slop();
        tDf_itran_pin = cellPin->GetTDF_itran();
        tDf_fall = arrFall + tDf_pin*1e-12 + tDf_slop_pin*cLoad*1e-15
                  + tDf_itran_pin*tranFall;
        max_dly_fall = Max(max_dly_fall, tDf_fall);

        // Rising Transition time(3)
        tTr_pin = cellPin->GetTTR();
        tTr_slop_pin = cellPin->GetTTR_slop();
        tTr_itran_pin = cellPin->GetTTR_itran();
        tTr_rise = tTr_pin*1e-12 + tTr_slop_pin*cLoad*1e-15
                  + tTr_itran_pin*tranRise;
        max_tran_rise = Max(max_tran_rise, tTr_rise);

        // Falling Transition time(4)
        tTf_pin = cellPin->GetTTF();
        tTf_slop_pin = cellPin->GetTTF_slop();
        tTf_itran_pin = cellPin->GetTTF_itran();
        tTf_fall = tTf_pin*1e-12 + tTf_slop_pin*cLoad*1e-15
                  + tTf_itran_pin*tranFall;
        max_tran_fall = Max(max_tran_fall, tTf_fall);
    } // for_each_pin(current node's match)

    // 4) set delay (max_dly_rise) of curNode
    (void)gasSetDelayOfCrossChromosomeNodes(gChromosome, curNode, max_dly_rise,
                                             max_dly_fall, max_tran_rise, max_tran_fall, gch_pos);
}

// Check foNode's fanin, and insert foNode into Q if Q != NULL
int gasCheckFaninVisitAndEnQueue(MapNode *foNode, Queue<MapNode> *Q=NULL)
{
    int count, all_visited = 0;
    MapNode *inNode;
    LinkedList<MapNode> *finList;
}

```

```

finList = foNode->GetFaninList();
count = 0;
for_each(finList){
    finNode = finList->Data();
    if(finNode->GetVisit()==1) count++;
}

if(count == finList->ListSize()){
    all_visited = 1;
    if(Q != NULL){
        Q->QInsert(foNode);
    }
}
return all_visited;
}

// Returns Primary Output node which has max delay among POs
MapNode *gasEvaluateFindMaxDelay(Graph *origNet,
    LinkedList<MapGraph *> *gChromosome, double *max_dly)
{
    LinkedList<Node *> *oPolist;
    MapGraph *subChrom;
    LinkedList<MapNode *> *subChrGList;
    MapNode *curNode, *cpNode=NULL;
    char *cur_node_name;

    double fDelay, rDelay, arrTime;
    int pos;

    oPolist = origNet->GetPoList();
    *max_dly = 0.0;

    for(gChromosome->Reset();!gChromosome->EndOfList();gChromosome->Next()){
        subChrom = gChromosome->Data();
        subChrGList = subChrom->GetGatelist();
        for_each(subChrGList){
            curNode = subChrGList->Data();
            cur_node_name = curNode->GetName();
            if((pos=origNet->FindNode(oPolist, cur_node_name))!=-1){
                rDelay = curNode->GetDR();
                fDelay = curNode->GetDF();
                arrTime = Max(rDelay, fDelay);
                if(arrTime > *max_dly){
                    *max_dly = arrTime;
                    cpNode = curNode;
                }
            }
        }
    }
    return cpNode;
}

// Check if all nodes in gChromosome are visited, if yes return TRUE, else (0)
int gasAreAllNodesInGlobalChromosomeVisited(LinkedList<MapGraph *> *gChromosome)
{
    MapGraph *subChrom;
    LinkedList<MapNode *> *gatelist, *piList;
    MapNode *curNode, *tmpNode;
    int all_visited = 1;

    for(gChromosome->Reset();!gChromosome->EndOfList();gChromosome->Next()){
        subChrom = gChromosome->Data();
        if(!subChrom->GetUsed()) continue;
        gatelist = subChrom->GetGatelist();
        for(gatelist->Reset();!gatelist->EndOfList();gatelist->Next()){
            curNode = gatelist->Data();
            //if(curNode->GetNodeFunc() != NODE_PO){
            if(curNode->GetVisit() != 1){
                all_visited = 0;
                //cout<<"Node "<<curNode->GetName()<<" is not visited.\n";
                piList = curNode->GetFaninList();
                for_each(piList){
                    tmpNode = piList->Data();
                    cout<<"piNode "<<tmpNode->GetName()<<" \n";
                }
            }
            //if(curNode->GetNodeFunc() != NODE_PO){
            (void)gasPrintGlobalChromosome(gChromosome, 1);
            //break;
            exit(0);
        }
    }
    if(all_visited==0){
        break;
    }
    return all_visited;
}

// Clear used sub-chromosome flagged 04-23-01
void gasClearSubChromUsedFlag(LinkedList<MapGraph *> *gChrom)
{
    MapGraph *subChrom;

    for(gChrom->Reset();!gChrom->EndOfList();gChrom->Next()){
        subChrom = gChrom->Data();
        subChrom->SetUsed(0);
    }
}

// Set used sub-chromosome flagged 04-23-01
int gasSetSubChromUsedFlagInGChrom(LinkedList<MapGraph *> *gChrom)
{
    MapGraph *rsubChrom, *csubChrom;
    LinkedList<MapNode *> *rsubPoList, *csubPiList;
    MapNode *rsubPoNode, *csubPiNode;
    char *rsubPoName, *csubPiName;
    int index = -1, c_index = -1, used_sub_chrom = 0;
    int pos;

    (void)gasClearSubChromUsedFlag(gChrom); // Clear used flag
    for(gChrom->Reset();!gChrom->EndOfList();gChrom->Next()){
        index++; c_index = -1;
        rsubChrom = gChrom->Data();
        rsubPoList = rsubChrom->GetPoList();
        rsubPoNode = rsubPoList->Data(0);
        rsubPoName = rsubPoNode->GetName();
        if(rsubPoNode->GetNodeFunc() != NODE_PO){
            for(gChrom->Reset();!gChrom->EndOfList();gChrom->Next()){
                c_index++;
                csubChrom = gChrom->Data();
                if(index != c_index && csubChrom->GetUsed()==0){
                    csubPiList = csubChrom->GetPiList();
                    if((pos=csubChrom->FindNode(csubPiList, rsubPoName))!=-1){
                        rsubChrom->SetUsed(1);
                        used_sub_chrom++;
                        break;
                    }
                }
            }
        }
        else {
            rsubChrom->SetUsed(1);
            used_sub_chrom++;
        }
        gChrom->Reset(index); // get back to the orig pos
    }
    return used_sub_chrom;
}

// Calculate delay at every node (mod : 04-19-01)
double gasEvaluateGlobalChromosomeDelay(Graph *origNet, GChrom *globalChrom,
    LinkedList<LibCell *> *cellLib)
{
    Queue<MapNode *> *Q = new Queue<MapNode *>;
    double tDr, tDf, tTr, tTf, delay;

    MapNode *curNode, *finNode, *cfoNode;
    MapNode *cpNode;
    LinkedList<MapNode *> *finList, *curFoList;
    LinkedList<MapGraph *> *gChrom;
    MapGraph *subChrom;
    int gflag=0, itr_cnt = -1;

    int gch_pos = -1, cfoVisit, all_visited;
    NodeType curType;

    gChrom = globalChrom->gChromosome;
    (void)gasInitNodesVisitFlag(origNet, gChrom);
    (void) gasSetSubChromUsedFlagInGChrom(gChrom);

    for(gChrom->Reset();!gChrom->EndOfList();gChrom->Next()){
        gch_pos++;
        subChrom = gChrom->Data();
        if(!subChrom->GetUsed()) continue;
        //visited PI nodes
        nstart_node = gasGetSubB2TStartingNodes(subChrom, Q);
        while(!Q->QEmpty()){
            curNode = Q->DeQueue();
            curType = curNode->GetNodeType();
            if(curNode->GetNodeFunc() == NODE_PO){
                finList = curNode->GetFaninList();
                if(finList->ListSize()==1){
                    finNode = finList->Data(0);
                    if(finNode->GetVisit()==1){
                        tDr = finNode->GetDR();
                        tDf = finNode->GetDF();
                        tTr = finNode->GetTR();
                        tTf = finNode->GetTF();
                        (void)gasSetDelayOfCrossChromosomeNodes(gChrom, curNode,
                            tDr, tDf, tTr, tTf, gch_pos);
                    }
                    else {
                        cout<<"In gasEvaluateGlobalChromosomeDelay().\n";
                        cout<<"PO node ("<<curNode->GetName()<<" fanin ("
                            <<finNode->GetName()<<" is not visited ..... \n";
                        exit(1);
                    }
                }
            }
            curNode->MarkVisit();
        }
        else {
            (void)gasCalculateGlobalGeneDelay(gChrom, curNode, cellLib,
                gch_pos);
        }
        curFoList = curNode->GetFanoutList();
        if(curFoList->ListSize() >= 1 && curType == INTERNAL){
            for_each(curFoList){
                cfoNode = curFoList->Data();
                cfoVisit = cfoNode->GetVisit();
                if(gasCheckFaninVisitAndEnQueue(cfoNode)&& (cfoVisit<1)){
                    Q->QInsert(cfoNode);
                }
            }
        }
        gChrom->Reset(gch_pos); //recover gChrom pos in for_loop
    }
    gflag = gasAreAllNodesInGlobalChromosomeVisited(gChrom);
    // find max_delay of a gChromosome and set chromosome delay
    cpNode = gasEvaluateFindMaxDelay(origNet, gChrom, &delay);
    globalChrom->cp_delay = delay;

    delete Q;
    if(gflag==0){
        return (-1);
    }
    else {
        return delay;
    }
}

double gasEvaluateGlobalChromosomeArea(GChrom *globalChrom)
{
}

```

```

LinkedList<MapGraph *> *gChromosome;
MapGraph *subChrom;
double sub_chrom_area, total_area = 0;

gChromosome = globalChrom->gChromosome;
(void)gasSetSubChromUsedFlagInGChrom(gChromosome);

for(gChromosome->Reset();!gChromosome->EndOfList();gChromosome->Next()){
    subChrom = gChromosome->Data();
    if(subChrom->GetUsed()){
        sub_chrom_area = subChrom->GetArea();
        total_area += sub_chrom_area;
    }
}
globalChrom->area = total_area;
return total_area;
}

// Returns critical path delay
double gasEvaluateGlobalChromosome(Graph *origNet, GChrom *globalChrom,
    LinkedList<LibCell *> *cellLibrary,
    double *gChromArea=NULL)
{
    double cp_dly=0.0, global_chrom_area;

    //12-24-00
    cp_dly = gasEvaluateGlobalChromosomeDelay(origNet,globalChrom,cellLibrary);
    global_chrom_area = gasEvaluateGlobalChromosomeArea(globalChrom);
    if(gChromArea!=NULL){
        *gChromArea = global_chrom_area;
    }
    return cp_dly;
}

//----- Global Chromosome Levelizing -----
// 10-28-01
void dlyResetVisitFlag(LinkedList<MapGraph *> *gChromosome)
{
    MapGraph *subChrom;
    MapNode *subNode;
    LinkedList<MapNode *> *subGateList;

    for(gChromosome->Reset();!gChromosome->EndOfList();gChromosome->Next()){
        subChrom = gChromosome->Data();
        subGateList = subChrom->GetGateList();
        for_each(subGateList){
            subNode = subGateList->Data();
            subNode->MarkVisit(-1);
            //subNode->SetLevel(-1);
            subNode->SetGLevel(-1);
            if(subNode->GetNodeFunc()==NODE_PO){
                subNode->SetLevel(0);
            } else {
                subNode->SetLevel(-1);
            }
        }
    }
}

// 10-28-01
int dlyIsCurrentMapNodeInQueue(Queue<MapNode *> *Q,MapNode *foNode)
{
    LinkedList<MapNode *> *qList;
    MapNode *qNode;
    char *q_name, *fo_name;
    int return_val = 0;

    if(Q->QEmpty()) return (0);
    fo_name = foNode->GetName();
    qList = Q->QList();
    for(qList->Reset();!qList->EndOfList();qList->Next()){
        qNode = qList->Data();
        q_name = qNode->GetName();
        if(strcmp(q_name, fo_name)==0){
            return_val = 1;
            break;
        }
    }
    return return_val;
}

// 10-28-01
// Q contains all primary input's fanout node
// to traverse the global chromosome form bottom to top.
void dlyInitGlobalDepth(LinkedList<MapGraph *> *gChromosome, Graph *origNet,
    Queue<MapNode *> *Q)
{
    LinkedList<Node *> *orgPiList;
    MapGraph *subChrom;
    MapNode *subNode, *foNode;
    LinkedList<MapNode *> *subGateList, *foList;
    int used_sub_chrom,pos, num_fanout;

    char *subName;

    orgPiList = origNet->GetPiList();
    (void)dlyResetVisitFlag(gChromosome);

    for(gChromosome->Reset();!gChromosome->EndOfList();gChromosome->Next()){
        subChrom = gChromosome->Data();
        used_sub_chrom = subChrom->GetUsed();
        if(!used_sub_chrom) continue;
        subGateList = subChrom->GetGateList();
        for(subGateList->Reset();!subGateList->EndOfList();subGateList->Next()){
            subNode = subGateList->Data();
            subName = subNode->GetName();
            if((pos=origNet->FindNode(orgPiList, subName)) != -1){
                // more than one different subChromosome can have same
                // node(PI in origNet) as their input node.
                subNode->SetGLevel(0);
                subNode->MarkVisit(1);
                if((num_fanout=subNode->GetNumFanout())==0){
                    cout<<"fanout is zero\n";
                    // This "if" is not necessary because the node whose inputs
                    // are connected to PI (at least one) goes to Q.
                } else {
                    foList=subNode->GetFanoutList();
                    for_each(foList){
                        foNode = foList->Data();
                        if(!dlyIsCurrentMapNodeInQueue(Q,foNode)){
                            Q->QInsert(foNode);
                        }
                    }
                } else {
                    subNode->SetGLevel(-1);
                }
            }
        }
    }

    // 10-29-01
    // Check if all the fanins are visited and maximum fanin depth(gLevel)
    int dlyAreAllFaninsVisitedAndMaxInDepth(MapNode *qNode, int &max_depth)
    {
        LinkedList<MapNode *> *finList;
        MapNode *inNode;
        int count=0, visited, gLevel, num_fanin;
        int return_val = 0;

        max_depth = 0;
        finList = qNode->GetFaninList();
        num_fanin = finList->ListSize();
        for_each(finList){
            visited = -1;
            inNode = finList->Data();
            visited = inNode->GetVisit();
            gLevel = inNode->GetGLevel();
            if(visited == 1){
                count++;
                max_depth = Max(max_depth, gLevel);
            }
        }
        if(num_fanin == count){
            return_val = 1;
        }
        return return_val;
    }

    // 10-29-01
    void dlyGetGlobalFanoutNodes(LinkedList<MapGraph *> *gChromosome,
        MapNode *qNode, Queue<MapNode *> *foutQ)
    {
        LinkedList<MapNode *> *foList;
        MapNode *foNode;
        NodeType qType;

        qType = qNode->GetNodeType();
        if(qType == INTERNAL){
            foList = qNode->GetFanoutList();
            for_each(foList){
                foNode = foList->Data();
                foutQ->QInsert(foNode);
            }
        } else if(qType == PRIMARY_OUTPUT){
            // Returns a list of fanout nodes of the qNode
            // crossing the sub-graph boundary
            (void)gasGetFanoutListAcrossChromosome(qNode, gChromosome, foutQ);
        }
    }

    // 10-29-01
    // Set visit flag on curNode fanout's fanin (may cross-chromosome)
    void dlySetVisitFlagCurNodeOutsInNode(MapNode *curNode,MapNode *fanoutNode)
    {
        LinkedList<MapNode *> *foInList;
        MapNode *foInNode;
        string curName;
        char *foInName;
        int comp_value;

        curName = (string)curNode->GetName();

        foInList = fanoutNode->GetFaninList();
        for_each(foInList){
            foInNode = foInList->Data();
            //foInName = (string)foInNode->GetName();
            foInName = foInNode->GetName();
            if((comp_value=curName.compare(foInName))==0){
                foInNode->MarkVisit();
                foInNode->SetGLevel(curNode->GetGLevel());
            }
        }
    }

    // 10-28-01
    // Global Chromosome Levelizing
    // Calculate Global Depth for the Global Chromosome
    void dlySetGlobalDepthOfGlobalChromosome(LinkedList<MapGraph *> *gChromosome,
        Graph *origNet)
    {
        MapGraph *subChrom;
        MapNode *subNode, *finNode, *qNode, *qfoNode;
        LinkedList<MapNode *> *subGList, *finList;
        char *cur_node_name, *fo_fin_name;
        int used_sub_chrom, visited = 0;
        int all_visited, max_depth, glevel, count=0;

        string tmp_str;

        Queue<MapNode *> searchQ, foutQ;

        // Initialize Global depth, 0 for all PI Node(orgNet), -1 otherwise
        (void)dlyInitGlobalDepth(gChromosome, origNet, &searchQ);
    }
}

```

```

while(!searchQ.QEmpty()){
    qNode = searchQ.DeQueue();
    tmp_str = (string)qNode->GetName();
    count++;

    visited = qNode->GetVisit();
    all_visited=dlyAreAllFaninsVisitedAndMaxInDepth(qNode, max_depth);
    if(all_visited && visited == -1){
        if(qNode->GetNodeFunc() == NODE_PO){
            // Since a node whose function is NODE_PO doesnot have any
            // logic function, the glevel shold the same its fanin node.
            qNode->SetGLevel((glevel=max_depth));
        } else {
            qNode->SetGLevel((glevel=max_depth+1));
        }
        qNode->MarkVisit();

        (void)dlyGetGlobalFanoutNodes(gChromosome, qNode, &foutQ);
        while(!foutQ.QEmpty()){
            qfoNode = foutQ.DeQueue();

            // Set visit flag on qNode fanout's fanin (if cross-chromosome)
            (void)dlySetVisitFlagCurNodeOutsInNode(qNode, qfoNode);

            if(!dlyIsCurrentMapNodeInQueue(&searchQ, qfoNode)){
                searchQ.QInsert(qfoNode);
            }
        }
    } else {
        if(!dlyIsCurrentMapNodeInQueue(&searchQ, qNode) && visited == -1){
            searchQ.QInsert(qNode);
        }
    }

    //Debug
    if(count > 100000){
        (void)gasPrintGlobalChromosome(gChromosome);
        exit(1);
    }
}

// 10-31-01
void dlyGetGlobalChromosomePoList(LinkedList<MapGraph *> *gChrom,
    Graph *origNet, Queue<MapNode *> *poQ)
{
    LinkedList<Node *> *poList;
    MapGraph *subChrom;
    MapNode *subNode, *tmpNode;
    LinkedList<MapNode *> *subGateList;
    int used_sub_chrom, pos, num_fanout;
    int curGLevel, oldGLevel;

    char *subName;
    NodeFunc subFunc;

    // Clear poQ
    while(!poQ->QEmpty()){
        (void)poQ->DeQueue();
    }

    poList = origNet->GetPoList();
    for(gChrom->Reset(); !gChrom->EndOfList(); gChrom->Next()){
        subChrom = gChrom->Data();
        used_sub_chrom = subChrom->GetUsed();
        if(!used_sub_chrom) continue;

        subGateList = subChrom->GetGateList();
        for_each(subGateList){
            subNode = subGateList->Data();
            subName = subNode->GetName();
            subFunc = subNode->GetNodeFunc();
            curGLevel = subNode->GetGLevel();

            if((pos=origNet->FindNode(poList, subName)) != -1 && subFunc==NODE_PO){
                if(!dlyIsCurrentMapNodeInQueue(poQ, subNode)){
                    poQ->QInsert(subNode);
                }
            }
        }
    }

    // 10-31-01
    MapNode *dlyGetFaninNodesCrossChromosome(LinkedList<MapGraph *> *gChromosome,
        MapNode *curNode)
    {
        MapGraph *subChrom;
        //MapNode *subNode, *finNode;
        //LinkedList<MapNode *> *subGateList, *finList;
        //char *cur_node_name, *fo_fin_name;
        MapNode *subNode=NULL;
        LinkedList<MapNode *> *subPoList;
        char *cur_node_name;
        int used_sub_chrom, pos;

        cur_node_name = curNode->GetName();
        for(gChromosome->Reset(); !gChromosome->EndOfList(); gChromosome->Next()){
            subChrom = gChromosome->Data();
            used_sub_chrom = subChrom->GetUsed();
            if(!used_sub_chrom) continue;

            subPoList = subChrom->GetPoList();
            if((pos=subChrom->FindNode(subPoList, cur_node_name))!= -1){
                subNode = subPoList->Data(pos);
                subNode->SetLevel(curNode->GetLevel());
                break;
            } else {
                subNode = NULL;
            }
        }
    }

    }
}

return subNode;

/* Sort PO nodes in descending order.
 * Assume gLevel(PO[i]) <= gLevel(PO[i+1]).
 * In order to reflect the path length form PO[i+1]
 * to PO[i] when the traversal goes from PO[i].
 */
void dlySortByGLevel(Queue<MapNode *> *poQ)
{
    LinkedList<MapNode *> *queueList, tmpList;
    MapNode *curNode, *oldNode;
    int pos, cur_level, old_level;

    queueList = poQ->QList();
    while(!poQ->QEmpty()){
        curNode = poQ->DeQueue();
        cur_level = curNode->GetGLevel();
        if(tmpList.ListSize()==0){
            tmpList.InsertRear(curNode);
        } else {
            pos = tmpList.ListSize()-1;
            while(pos>0){
                oldNode = tmpList.Data(pos);
                old_level = oldNode->GetGLevel();
                if(cur_level > old_level){
                    if(pos == 0){
                        tmpList.InsertFront(curNode);
                        break;
                    } else {
                        tmpList.InsertRight(curNode);
                        break;
                    }
                }
                pos--;
            }
        }
    }

    for(tmpList.Reset(); !tmpList.EndOfList(); tmpList.Next()){
        curNode = tmpList.Data();
        poQ->QInsert(curNode);
    }
}

// 10-31-01
void dlyFanoutNodesDepthFromPO(LinkedList<MapGraph *> *gChrom,
    Graph *origNet,
    MapNode *mfpNode, Queue<MapNode *> *Q)
void dlyGlobalNodesDepthFromPO(LinkedList<MapGraph *> *gChrom, Graph *origNet)
{
    Queue<MapNode *> poQ, searchQ;
    MapNode *poNode, *sNode, *finNode;
    LinkedList<MapNode *> *finList;
    char *finName;
    int sLevel, pos;

    LinkedList<Node *> *orgPiList;

    orgPiList = origNet->GetPiList();

    (void)dlyGetGlobalChromosomePoList(gChrom, origNet, &poQ);
    (void)dlySortByGLevel(&poQ);

    while(!poQ.QEmpty()){
        poNode = poQ.DeQueue();
        searchQ.QInsert(poNode);

        while(!searchQ.QEmpty()){ // BFS
            sNode = searchQ.DeQueue();
            sLevel = sNode->GetLevel();
            finList = sNode->GetFaninList(); // (1)
            for_each(finList){
                finNode = finList->Data(); // (2)
                finName = finNode->GetName(); // (3)

                if((pos=origNet->FindNode(orgPiList, finName)) != -1){
                    finNode->SetLevel(sLevel);
                } else {
                    /* Since the "finNode" can be a mfp and its fanout's
                     * depth from PO can be different,
                     * depth(finNode) = max(level(FO(i), i=1..n), level(finNode)).
                     */
                    finNode->SetLevel(Max(sLevel+1, finNode->GetLevel()));

                    /* if fanin node is a NODE_PI of a sub-shromosome, faninList(1)
                     * size is "0", and get an error at (2) or (3).
                     * If finNode is NODE_PI, get real node which is in another
                     * sub-chromosome so that we can traverse down.
                     */
                    if(finNode->GetNodeFunc()==NODE_PI){
                        finNode = dlyGetFaninNodesCrossChromosome(gChrom, finNode);
                    }
                    if(!dlyIsCurrentMapNodeInQueue(&searchQ, finNode)){
                        searchQ.QInsert(finNode);
                    }
                }
            }
        }
    }
}

/*****
 * Gacover_sub_dPTL.cc
 *
 * This Pgm allows following PTL structure during covering stage.
 * 1) PTL2 + PTL1, or PTL3 + PTL1
 * 2) PTL1 + PTL2, or PTL1 + PTL3
 * This means max transistors in series through PTL are 3.
 */

```

```

*****/
#include <fstream>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <new.h>
#include <time.h>

#include "eqn_stack.h"
#include "eqn_opr.h"
#include "Node.h"
#include "LinkedList.h"
#include "Graph.h"

#include "MapNode.h"
#include "MapGraph.h"

#include "Queue.h"
#include "Stack.h"
#include "util.h"
#include "cuddInt.h"

//----- Initialize Genetic Population -----
// subOutList will be used to determine the inputs of a MapGraph.
// The reason of using this info is that mapped graph may introduce new PIs
// which were not PI in a subject graph. (04-26-01)
void gaEnumerateSubOutNodes(LinkedList<Graph*> *subGList,
                           LinkedList<Node*> *subOutList)
{
    Graph *sGraph;
    LinkedList<Node*> *sPoList;
    Node *curPoNode, *prevPoNode;
    int index, pos, cur_depth, prev_depth;

    index = -1;
    for(subGList->Reset(); !subGList->EndOfList(); subGList->Next()){
        index++;
        sGraph = subGList->Data();
        sPoList = sGraph->GetPoList();
        if(sPoList->ListSize() == 1){
            curPoNode = sPoList->Data(0);
            cur_depth = curPoNode->GetGLevel();
            if(index==0){
                subOutList->InsertRear(curPoNode);
            } else {
                pos = subOutList->ListSize()-1;
                while(pos>0){
                    prevPoNode = subOutList->Data(pos);
                    prev_depth = prevPoNode->GetGLevel();
                    if(prev_depth > cur_depth){
                        if(pos == 0){
                            subOutList->InsertFront(curPoNode);
                            break;
                        } else {
                            pos--;
                        }
                    } else {
                        subOutList->InsertRight(curPoNode);
                        break;
                    }
                }
            }
        } else {
            printf("SubGraph[%d] does not have ONLY one PO nodes\n", index);
            printf("In %s at line %d\n", __FILE__, __LINE__);
        }
    }
}

// Returns the starting node and the #of node(except PI & PO) in the subGraph
Node *gaGetStartingNode(Graph *sGraph, int *size, Node *fNode)
{
    LinkedList<Node*> *gateList, *poList, *piList, *finList;
    Node *curNode;
    char *curNodeName;
    *size = 0;

    piList = sGraph->GetPiList();
    poList = sGraph->GetPoList();
    gateList = sGraph->GetGateList();
    for(gateList->Reset(); !gateList->EndOfList(); gateList->Next()){
        curNode = gateList->Data();
        curNodeName = curNode->GetName();
        if(curNode->GetNodeType() == INTERNAL ||
           (curNode->GetNodeType() == PRIMARY_OUTPUT &&
            curNode->GetNodeFunc() != NODE_PO &&
            (sGraph->FindNode(piList, curNodeName) == -1))) ||
           (sGraph->FindNode(poList, curNodeName) != -1 &&
            curNode->GetNodeFunc() != NODE_PO &&
            sGraph->FindNode(piList, curNodeName) == -1){
            *size = *size + 1; // number of nodes in the Sgraph except Input
        }
    }

    // Returns starting node for further processing from top to bottom.
    if(poList->ListSize() != 1) {
        cout << "In " << __FILE__ << " at line " << __LINE__ << " : \n";
        cout << sGraph->GetName() << " does not have one OUTPUT node. \n";
        cout << "Nodeally a subGraph has only one output ndoe.... \n";
        cout << "Program is terminating .... \n";
        exit(1);
    } else {
        poList->Reset(0);
        curNode = poList->Data();
        fNode = NULL;
        //if(curNode->GetNodeType() == PRIMARY_OUTPUT &&
    }
}

if( curNode->GetNodeFunc() == NODE_PO){
    finList = curNode->GetFaninList();
    if(finList->ListSize() == 1){
        finList->Reset(0);
        fNode = curNode;
        curNode = finList->Data();
        //return finList->Data();
    } else {
        cout << "In " << __FILE__ << " at line " << __LINE__ << " : \n";
        cout << sGraph->GetName() << " 's primary output node has more "
            << " one fanin nodes. \n";
        cout << "curNode = " << curNode->GetName() << " \n";
        cout << "curFunc = " << (int)curNode->GetNodeFunc() << " \n";
        cout << "Program is terminating .... \n";
        exit(0);
    }
} // else {
return curNode;
}

// generate random # between 0 ~ max_val
double gaGenRandomNumber(double max_val)
{
    //srand(time(NULL));
    return (max_val*rand()/(RAND_MAX + 1.0));
}

// Coin flipping to get root node's cell type
CellType gaFlip(int index, double probability)
{
    double tmp_random;

    tmp_random = gaGenRandomNumber(1.0);
    if((tmp_random - probability)<0) {
        #ifdef GA_FLIP_DEBUG
            cout << tmp_random ;
            cout << " : STATIC " ;
        #endif
        return STATIC;
    } else {
        #ifdef GA_FLIP_DEBUG
            cout << tmp_random ;
            cout << " : PTL " ;
        #endif
        return PTL;
    }
}

// Decide current node type by searching PO, PI, and Gate List of sGraph
// Not using --> 04-26-01
NodeType gaGetCurrentNodeType(Graph *sGraph, Node *curNode)
{
    LinkedList<Node*> *piList, *poList, *gateList;
    char *cName; //current node name

    piList = sGraph->GetPiList();
    poList = sGraph->GetPoList();
    gateList = sGraph->GetGateList();
    cName = curNode->GetName();

    if(sGraph->FindNode(poList, cName) != -1 &&
       sGraph->FindNode(gateList, cName) != -1){
        return PRIMARY_OUTPUT;
    } else if(sGraph->FindNode(piList, cName) != -1 &&
              sGraph->FindNode(gateList, cName) != -1){
        return PRIMARY_INPUT;
    } else if(sGraph->FindNode(poList, cName) == -1 &&
              sGraph->FindNode(piList, cName) == -1 &&
              sGraph->FindNode(gateList, cName) != -1){
        return INTERNAL;
    } else {
        return UNASSIGNED;
    }
}

// Decide current node type by searching PO, PI, and Gate List of sGraph
// >> Important for deciding PI nodes<<, all nodes, Vn, Vn (- subOutList are PI
// --> 04-26-01 NodeType gaGetCurrentNodeType(Graph *sGraph, Node *curNode)
NodeType gaGetCurrentNodeType(Graph *sGraph, LinkedList<Node*> *subOutList,
                             Node *curNode)
{
    LinkedList<Node*> *piList, *poList, *gateList;
    char *cName; //current node name

    Node *piNode, *piNode;
    LinkedList<Node*> *piNodeFoList;
    int pos;

    piList = sGraph->GetPiList();
    poList = sGraph->GetPoList();
    gateList = sGraph->GetGateList();
    cName = curNode->GetName();

    if(sGraph->FindNode(poList, cName) != -1 &&
       sGraph->FindNode(gateList, cName) != -1){
        return PRIMARY_OUTPUT;
    } else if(sGraph->FindNode(subOutList, cName) != -1 &&
              sGraph->FindNode(poList, cName) == -1 &&
              sGraph->FindNode(gateList, cName) != -1){
        /* added for cc.slif a0 -t' situation 05-08-01*/
        pos = sGraph->FindNode(gateList, cName);
        if(pos != -1){
            piNode = gateList->Data(pos);
            piNodeFoList = piNode->GetFanoutList();
            piNode = piNodeFoList->Data(0);
            if(piNode->GetNodeFunc() == NODE_PO &&
               piNode->GetNodeFunc() == NODE_INV){
                return INTERNAL;
            } else {
                return PRIMARY_INPUT;
            }
        }
    }
}

```

```

    }
    } else {
        /* end of addition 05-08-01*/
        return PRIMARY_INPUT;
    }
} else if (sGraph->FindNode(poList, cName) == -1 &&
           sGraph->FindNode(piList, cName) == -1 &&
           sGraph->FindNode(subOutList, cName) == -1 &&
           sGraph->FindNode(gateList, cName) != -1) {
    return INTERNAL;
} else {
    return UNASSIGNED;
}
}

// Returns the function type of a matched cell (04-26-01)
NodeFunc gaGetMatchedCellFunc(Graph *sGraph, LinkedList<Node> *subOutList,
                              Node *curNode, Match *aMatch)
{
    NodeType curType;
    NodeFunc curFunc;
    char *matchName;

    matchName = aMatch->cell_name;
    curType = gaGetCurrentNodeType(sGraph, subOutList, curNode);
    if (curType == UNASSIGNED) {
        cout << "Node (" << curNode->GetName() << ")\'s node type is "
              << (int) curType << endl;
        cout << "In "<< _FILE_ << "at line "<< _LINE_ << endl;
        exit(0);
    }
    curFunc = curNode->GetNodeFunc();
    if (curType == INTERNAL || curType == PRIMARY_OUTPUT) {
        if (strcmp(matchName, "ptl_y1", 6) == 0) {
            return NODE_PTL1;
        } else if (strcmp(matchName, "ptl_y2", 6) == 0) {
            return NODE_PTL2;
        } else if (strcmp(matchName, "ptl_y3", 6) == 0) {
            return NODE_PTL3;
        } else if (strcmp(matchName, "and", 3) == 0) {
            return NODE_AND;
        } else if (strcmp(matchName, "or", 2) == 0) {
            return NODE_OR;
        } else if (strcmp(matchName, "nand", 4) == 0) {
            return NODE_NAND;
        } else if (strcmp(matchName, "nor", 3) == 0) {
            return NODE_NOR;
        } else if (strcmp(matchName, "aoi", 3) == 0) ||
            (strcmp(matchName, "oi", 3) == 0) || (strcmp(matchName, "oa2", 3) == 0) ||
            (strcmp(matchName, "ao2", 3) == 0) || (strcmp(matchName, "mux", 3) == 0) ||
            (strcmp(matchName, "xnor", 4) == 0) || (strcmp(matchName, "xor", 3) == 0) {
            return NODE_COMPLEX;
        } else if (strcmp(matchName, "inv", 3) == 0) {
            return NODE_INV;
        }
    } else if (curType == PRIMARY_INPUT) { // 04-26-01
        return NODE_PI;
    } else if (curType == PRIMARY_OUTPUT && curFunc == NODE_PO) {
        cout << "In gaGetMatchedCellFunc(), PO node is processing ... \n";
        cout << "It is strange.... Check it.... \n";
        cout << "node name is " << curNode->GetName() << "\n";
        return NODE_PO;
    }
    return NODE_UNDEFINED;
}

// Check if matched cell's output is inverted (except Inverter)
int gaIsMatchedCellOutputInverted(char *cName)
{
    if ((strcmp(cName, "nand", 4) == 0) || (strcmp(cName, "nor", 3) == 0) ||
        (strcmp(cName, "aoi", 3) == 0) || (strcmp(cName, "oi", 3) == 0)) {
        return (1);
    } else {
        return (0);
    }
}

// is Cell name inverted and fanins are not in oGraph?
int gaMatchedCellHaveMissingSignals(Graph *origNet, Match *aMatch)
{
    char *match_name, *sig_name;
    LinkedList<pinMap> *pinList;
    pinMap *pinSet;
    LinkedList<Node> *oGateList;
    int pos = -1;

    oGateList = origNet->GetGateList();
    match_name = aMatch->cell_name;
    pinList = aMatch->pinmap;
    for (pinList->Reset(); !pinList->EndOfList(); pinList->Next()) {
        pinSet = pinList->Data();
        sig_name = pinSet->net_name;
        if (IsConstant(sig_name)) continue;
        if ((pos = origNet->FindNode(oGateList, sig_name)) == -1) {
            return (1);
        }
    }
    return (0);
}

// Check if all net signals can be found in the original network.
int gaAreSignalsLegal(Graph *origNet, Graph *sGraph, CellType ctype,
                     Match *aMatch, Node *curNode = NULL)
{
    LinkedList<pinMap> *pMapList;
    LinkedList<Node> *oGateList, *sGateList;
    pinMap *curPin;
    char *cell_name, *net_name;
    int legal_signal = 1;

    cell_name = aMatch->cell_name;
    // is Cell name inverted and fanins are not in oGraph?
    if (gaMatchedCellHaveMissingSignals(origNet, aMatch)) {
        return (0);
    }

    pMapList = aMatch->pinmap;
    oGateList = origNet->GetGateList();
    sGateList = sGraph->GetGateList();
    for (pMapList->Reset(); !pMapList->EndOfList(); pMapList->Next()) {
        curPin = pMapList->Data();
        net_name = curPin->net_name;
        if (strcmp(net_name, "GND") == 0 || strcmp(net_name, "VDD") == 0) {
            continue;
        }
        if ((ctype == PTL) && (origNet->FindNode(oGateList, net_name) == -1)) {
            legal_signal = 0;
        } else if ((ctype == STATIC) && (origNet->FindNode(oGateList, net_name) == -1)) {
            legal_signal = 0;
        }
    }
    return legal_signal;
}

// Returns node type for an input pin (04-26-01)
NodeType gaGetInPinNodeType(Graph *origNet, Graph *sGraph,
                           LinkedList<Node> *subOutList, char *pinName,
                           CellType ctype)
{
    LinkedList<Node> *sGpiList, *sGpoList, *sGgateList;
    LinkedList<Node> *oGpiList, *oGgateList;

    oGpiList = origNet->GetPiList();
    sGpiList = sGraph->GetPiList();
    sGpoList = sGraph->GetPoList();
    sGgateList = sGraph->GetGateList();

    if (sGraph->FindNode(sGpoList, pinName) != -1 &&
        sGraph->FindNode(sGgateList, pinName) != -1) {
        return PRIMARY_OUTPUT;
    } else if (sGraph->FindNode(subOutList, pinName) != -1 &&
        sGraph->FindNode(oGpiList, pinName) != -1) {
        return PRIMARY_INPUT;
    } else if (sGraph->FindNode(sGpoList, pinName) == -1 &&
        sGraph->FindNode(sGpiList, pinName) == -1 &&
        sGraph->FindNode(subOutList, pinName) == -1 &&
        sGraph->FindNode(sGgateList, pinName) != -1) {
        return INTERNAL;
    } else {
        return UNASSIGNED;
    }
}

void gaCreateGaGraphPoNode(Graph *sGraph, Node *bNode, MapGraph *gaGraph)
{
    LinkedList<Node> *foutList, *poList;
    Node *outNode;
    char *bnode_name, *outnode_name;
    MapNode *mapNode, *finNode;
    int position = -1;

    poList = sGraph->GetPoList();
    bnode_name = bNode->GetName();
    if (sGraph->FindNode(poList, bnode_name) != -1) { // bNode is PO of sGraph
        mapNode = gaGraph->FindOrCreateNode(bnode_name, position,
                                           PRIMARY_OUTPUT);
    } else { // bNode is not a PO node of sGraph (of origNet)
        foutList = bNode->GetFanoutList();
        if (foutList->ListSize() != 1) {
            cout << "During Processing "<< sGraph->GetName() << ", \n";
            cout << "In gaCreateGaGraphPoNode() "
                  << bnode_name << "\'s fanout ndoe is more than ONE !! \n";
            exit(1);
        } else {
            foutList->Reset(0);
            outNode = foutList->Data();
            outnode_name = outNode->GetName();
            mapNode = gaGraph->FindOrCreateNode(outnode_name, position,
                                           PRIMARY_OUTPUT, NODE_PO);
            finNode = gaGraph->FindOrCreateNode(bnode_name, position);
            mapNode->InsertFanin(finNode);
            finNode->InsertFanout(mapNode);
        }
    }
}

CellType gaGetMappedCellType(MapNode *curNode)
{
    NodeFunc curNodeFunc;

    curNodeFunc = curNode->GetNodeFunc();
    if (curNodeFunc == NODE_PTL1 || curNodeFunc == NODE_PTL2 ||
        curNodeFunc == NODE_PTL3) {
        return PTL;
    } else if (curNodeFunc == NODE_OR || curNodeFunc == NODE_NOR ||
        curNodeFunc == NODE_AND || curNodeFunc == NODE_NAND ||
        curNodeFunc == NODE_COMPLEX || curNodeFunc == NODE_INV) {
        return STATIC;
    } else {
        return UNKNOWN;
    }
}

// 1) get curNode's fanout list, and get fanout node (only one node)
// 2) Find cur fanout node's mapped function type (NODE_PTL1, NODE_OR, ...)
// 3) if curNode's fanout is NODE_PO, return NULL

```



```

        count = 0;
    } else {
        ctype = PTL;
        curNode->SetMatchType(ctype);
        matchList = curNode->GetPMatchList();
        mListSize = matchList->ListSize();
        count = 0;
    }
}
return aMatch;
}

MapNode *gaFindFanoutNodeInGaGraph(MapGraph *gaGraph, char *curName)
{
    LinkedList<MapNode> *gateList, *foList;
    MapNode *curMapNode, *foMapNode;
    int cur_pos, foListSize;

    gateList = gaGraph->GetGateList();
    cur_pos = gaGraph->FindNode(gateList, curName);
    if (cur_pos != -1) {
        gateList->Reset(cur_pos);
        curMapNode = gateList->Data();
        foList = curMapNode->GetFanoutList();
        foListSize = foList->ListSize();
        if (foListSize == 1) {
            foList->Reset();
            foMapNode = foList->Data();
            if (foMapNode->GetNodeFunc() == NODE_PO) {
                foMapNode = NULL;
            }
        } else if (foListSize > 1) {
            foMapNode = NULL;
        } else {
            foMapNode = NULL;
        }
    } else {
        foMapNode = NULL;
    }
    return foMapNode;
}

// Decide curNode's fanin celltype that allows 3 Trs in series for PTLs.
// 1) PTL2 + PTL1, or PTL3 + PTL1
// 2) PTL1 + PTL2, or PTL1 + PTL3
// 3) PTL1 + PTL1 + PTL1
CellType gaDetermineFaninCellType(MapNode *cmapNode, Node *curNode, Node *inNode)
{
    LinkedList<Match> *tmpMList;
    NodeType curMapType;
    NodeFunc curMapFunc, cfo_mapFunc, cfo_fo_mapFunc;
    CellType ctype, faninType = UNKNOWN, cfo_Type = UNKNOWN, cfo_fo_Type;
    MapNode *cfo_Node, *cfo_fo_Node;

    // Current node Cell Type and curNode's fanout cellType & matchedCell
    // 1) if curNode's fanout is not NODE_PO (means fanout node has at least one
    // match), and curNode is not MFP node, get curNode's fanout type and cell.
    ctype = curNode->GetMatchType(); /* PTL or STATIC */
    curMapType = cmapNode->GetNodeFunc(); /* INTERNAL or PRIMARY_(IN/OUT)PUT */
    curMapFunc = cmapNode->GetNodeFunc(); /* NODE_... */
    cfo_Node = gaP21GetFanoutNodeMapFunc(cmapNode, &cfo_mapFunc);
    if (cfo_Node != NULL) {
        cout << "cmapNode = "<<cmapNode->GetName()<<"\n";
        cfo_Type = gaGetMappedCellType(cfo_Node);
    }
    cfo_fo_Node = gaP21GetFanoutNodeMapFunc(cfo_Node, &cfo_fo_mapFunc);
    if (cfo_fo_Node != NULL) {
        cfo_fo_Type = gaGetMappedCellType(cfo_fo_Node);
    }

    if (inNode->GetNodeFunc() == NODE_INV) {
        if (ctype == PTL) { // Current node type
            if (curMapFunc == NODE_PTL3 || curMapFunc == NODE_PTL2) {
                if (cfo_Type == PTL) {
                    faninType = STATIC;
                } else {
                    tmpMList = inNode->GetPMatchList(); // PtlList
                    if (tmpMList->ListSize() != 0) { //
                        faninType = PTL;
                    } else {
                        faninType = STATIC;
                    }
                }
            } else if (curMapFunc == NODE_PTL1) {
                if ((cfo_Type == PTL && cfo_mapFunc == NODE_PTL1) || cfo_Type == STATIC) {
                    tmpMList = inNode->GetPMatchList(); // PtlList
                    if (tmpMList->ListSize() != 0) { //
                        faninType = PTL;
                    } else {
                        faninType = STATIC;
                    }
                } else {
                    faninType = STATIC;
                }
            }
        } else {
            inNode->SetMatchType(faninType);
            return faninType;
        }
    } else {
        tmpMList = inNode->GetPMatchList(); // PtlList
        if (tmpMList->ListSize() != 0) { //
            faninType = PTL;
        } else {
            faninType = STATIC;
        }
        inNode->SetMatchType(faninType);
        return faninType;
    }
}

} else { // Inverter doesn't have Ptl match.
    if (ctype == PTL) {
        if (curMapFunc == NODE_PTL3 || curMapFunc == NODE_PTL2) {
            if (cfo_Type == PTL) {
                faninType = STATIC;
            } else {
                tmpMList = inNode->GetPMatchList(); // PtlList
                if (tmpMList->ListSize() != 0) { //
                    faninType = PTL;
                } else {
                    faninType = STATIC;
                }
            }
        } else if (curMapFunc == NODE_PTL1) {
            if ((cfo_Type == PTL && cfo_mapFunc == NODE_PTL1) || cfo_Type == STATIC) {
                tmpMList = inNode->GetPMatchList(); // PtlList
                if (tmpMList->ListSize() != 0) { //
                    faninType = PTL;
                } else {
                    faninType = STATIC;
                }
            } else {
                faninType = STATIC;
            }
        }
    } else {
        faninType = STATIC;
    }
    inNode->SetMatchType(faninType);
    return faninType;
}

}

// Sets curNode's fanin cellType based on curNode's cell type
void gaSetFaninCellType(Node *curNode, Node *inNode)
{
    LinkedList<Match> *tmpMList;
    CellType ctype;

    ctype = curNode->GetMatchType(); /* PTL or STATIC */
    if (inNode->GetNodeFunc() == NODE_INV) {
        if (ctype == PTL) { // Current node type
            inNode->SetMatchType(STATIC);
        } else {
            tmpMList = inNode->GetPMatchList(); // PtlList
            if (tmpMList->ListSize() != 0) { //
                inNode->SetMatchType(PTL);
            } else { //
                inNode->SetMatchType(STATIC); //
            }
        }
    } else { // Inverter doesn't have Ptl match.
        if (ctype == PTL) {
            inNode->SetMatchType(STATIC);
        } else {
            inNode->SetMatchType(PTL);
        }
    }
}

// legalSol = gaGetChromosome(sGraph, bNode, STATIC/PTL);
// int gaGetChromosome(Graph *origNet, Graph *sGraph, Node *bNode,
// Construct a sub_graph circuit by checking a legality of a mapped node's
// input signals.
void gaGetChromosome(Graph *origNet, Graph *sGraph,
    LinkedList<Node> *subOutList,
    Node *bNode, MapGraph *gaGraph)
{
    LinkedList<pinMap> *pMapList;
    pinMap *pin_map;
    int pos = -1;

    Match *aMatch;
    Queue<Node> *Q;
    MapNode *mapNode, *finNode, *foMapNode;

    Node *curNode, *sfinNode;
    char *curName, *netpinName;
    NodeType curNodeType;
    NodeFunc matchFuncType;
    LinkedList<Node> *sGateList;
    int pos_sgate = -1;

    NodeType curPinNodeType;
    CellType ctype, finType;

    Q.Insert(bNode); // Only Internal or Output node will be inserted in Q
    (void)gaCreateGaGraphPoNode(sGraph, bNode, gaGraph);

    while (!Q.Empty()) {
        curNode = Q.Dequeue();
        curName = curNode->GetName();
        ctype = curNode->GetMatchType();

        // PTL2 + PTL1
        foMapNode = gaFindFanoutNodeInGaGraph(gaGraph, curName);
        aMatch = gaGetALegalMatch(origNet, sGraph, subOutList,
            curNode, ctype, foMapNode);

        // Find or create mapped ndoe
        // internal or out
        curNodeType = gaGetCurrentNodeType(sGraph, subOutList, curNode);
        if (curNodeType == UNASSIGNED) {
            cout << "Node("<<curName<<")\'s node type is "
                <<(int)curNodeType<<endl;
            cout << "In "<<__FILE__<<" at line "<<__LINE__<<endl;

```

```

    }
    exit(0);
}

matchFuncType = gaGetMatchedCellFunc(sGraph, subOutList, curNode, aMatch);
mapNode = gaGraph->FindOrCreateNode(curName, pos,
                                     curNodeType, matchFuncType);
mapNode->SetMatch(aMatch);
if((pos!=1) && mapNode->GetNodeFunc()==NODE_UNDEFINED){
    mapNode->SetNodeType(curNodeType);
    mapNode->SetNodeFunc(matchFuncType);
}

pMapList = aMatch->pinmap;
for(pMapList->Reset(); !pMapList->EndOfList(); pMapList->Next()){
    pin_map = pMapList->Data();
    netpinName = pin_map->net_node;
    if((strcmp(netpinName, "GND")==0) || (strcmp(netpinName, "VDD")==0)){
        continue;
    }
    curPinNodeType = gaGetInPinNodeType(origNet, sGraph, subOutList,
                                       netpinName, ctype);
    if(curPinNodeType == UNASSIGNED){
        printf("Input of cur_node(%s), %s, has UNASSIGNED NodeType.\n",
              curNode->GetName(), netpinName);
        printf("In %s at line %d\n", __FILE__, __LINE__);
        sGraph->PrintPoNodes();
        sGraph->PrintPiNodes();
        sGraph->PrintAllNodes();
        exit(1);
    } else if(curPinNodeType == PRIMARY_INPUT){
        finNode = gaGraph->FindOrCreateNode(netpinName, pos,
                                             curPinNodeType, NODE_PI);
    } else {
        finNode = gaGraph->FindOrCreateNode(netpinName, pos,
                                             curPinNodeType);
    }

    // Insert finNode into current mapNode's fanin
    //if(pos==1){
    mapNode->InsertFanin(finNode);
    finNode->InsertFanout(mapNode);
    //}

    // Insert newly created node into Queue to reaverse network
    if(curPinNodeType==INTERNAL){
        sGateList = sGraph->GetGateList();
        pos_sgate=sGraph->FindNode(sGateList, netpinName);
        sGateList->Reset(pos_sgate);
        sfinNode = sGateList->Data();
        Q.Insert(sfinNode);

        //(void)gaSetFaninCellType(curNode, sfinNode);
        // Allow PTL2*PTL1, i.e., 3 trs in series
        finType=gaDetermineFaninCellType(mapNode, curNode, sfinNode);
    } // end for_loop
} //return (1);
}

MapGraph *gaCreateMapGraph(void)
{
    MapGraph *aLegalSol = new MapGraph;
    char *graph_name = new char[10];
    strcpy(graph_name, "");
    aLegalSol->SetName(graph_name);
    return aLegalSol;
}

// 1) Initialize Population for a sub_graph (returns legal mapping solutions)
// a) traverse network starting from a root node of a sub-tree
// b) generate a random number to select logic type (ptl or static)
// c) generate a random number to select a MATCH from the match
// list (at this point ptl or static is already decided by step b)
// d) traverse the tree by looking at the inputs of the selected MATCH.
// the logic type of input node is the opposite type of the parent
// node. (e.g., if root is static input node type is ptl, and so on)
// e) repeat step c) to select a match
LinkedList<MapGraph *> gaInitPopulation(Graph *origNetwork, Graph *sGraph,
                                       LinkedList<Node *> subOutList, int subPopSize)
{
    Node *bNode, *foNode=NULL;
    char *bnode_name;
    int n_node, popSize, i;
    CellType root_type;
    LinkedList<Match *> *ptlMatchList;

    // this is a tree because there is no node whose out deg>1
    MapGraph *aLegalSol;
    LinkedList<MapGraph *> *popList = new LinkedList<MapGraph *>;
    bNode = gaGetStartingNode(sGraph, &n_node, foNode);
    bnode_name = bNode->GetName();
    if(bNode->GetNodeFunc()==NODE_INV && n_node==1){
        popSize = 1; // sGraph has only one node (INV)
    } else {
        popSize = subPopSize;
    }

    // Create LinkedList match_graph based on the match lists at each node
    ptlMatchList = bNode->GetPMatchList();
    for(i=0; i<popSize; i++){
        // coin flip to get root node's (bNode) gate type (0:static, 1:ptl)
        //root_type = gaFlip(0.5);

        /*
        if(bNode->GetNodeFunc()==NODE_INV ){
            root_type = STATIC;
        }else if(popSize!=1){
            root_type = gaFlip(1, 0.5);
            if(root_type==PTL && (ptlMatchList->ListSize()==0 ||
                                bNode->GetNumFanout() >=2 || foNode != NULL)){
                //bNode->GetNumFanout() > 2 }){
                root_type = STATIC;
            }
        } else {
            root_type = STATIC; // for sub-graph which has only one INV node
        }
        */

        root_type = STATIC; // for sub-graph which has only one INV node
        bNode->SetMatchType(root_type);
        aLegalSol = gaCreateMapGraph();
        (void)gaGetChromosome(origNetwork, sGraph, subOutList, bNode, aLegalSol);
        popList->InsertRear(aLegalSol);
#ifdef GA_INIT_POP_DEBUG
        aLegalSol->PrintPoNodes();
        aLegalSol->PrintPiNodes();
        aLegalSol->PrintAllNodes();
#endif
    }
    return popList;
}

//----- Cross Over -----
void gaPrint2Parents(MapGraph *parent1, int id1, MapGraph *parent2, int id2)
{
    cout<<"\n --- Parent 1 --- " <<"Index = "<<id1<<"\n";
    cout<<" Area Cost = "<<parent1->GetArea() <<"\n";
    parent1->PrintPoNodes();
    parent1->PrintPiNodes();
    parent1->PrintAllNodes();
    cout<<"\n --- Parent 2 --- " <<"Index = "<<id2<<"\n";
    cout<<" Area Cost = "<<parent2->GetArea() <<"\n";
    parent2->PrintPoNodes();
    parent2->PrintPiNodes();
    parent2->PrintAllNodes();
}

// returns fitness sum of all chromosome in the population
double gaGetFitnessSum(LinkedList<MapGraph *> *aPopulation)
{
    double fitness_sum = 0.0, aChroArea;
    MapGraph *aChromosome;
    if(aPopulation->ListSize()==0){
        return fitness_sum;
    }
    for(aPopulation->Reset(); !aPopulation->EndOfList(); aPopulation->Next()){
        aChromosome = aPopulation->Data();
        aChroArea = aChromosome->GetArea();
        fitness_sum = fitness_sum + (1/aChroArea);
    }
    return fitness_sum;
}

// Returns single individual's (MapGraph) via Roulette Wheel selection
//int gaSelect1Parent(LinkedList<MapGraph *> *gaPop)
MapGraph * gaSelect1Parent(LinkedList<MapGraph *> *gaPop, int *index)
{
    //int pop_size = 0;
    //int index = -1;
    MapGraph *aChromosome;
    double rand, fitnessSum, revArea, partSum = 0.0;

    *index = -1;
    fitnessSum = gaGetFitnessSum(gaPop);
    //pop_size = gaPop->ListSize();

    // Wheel point calculation [0,1]
    rand = fitnessSum*gaGenRandomNumber(1.0);

    // Find wheel slot
    if(gaPop->ListSize()==0){
        return NULL;
    }
    for(gaPop->Reset(); !gaPop->EndOfList(); gaPop->Next()){
        *index = *index + 1;
        aChromosome = gaPop->Data();
        revArea = (1.0/aChromosome->GetArea()); // the fitness of a individual
        partSum = partSum + revArea;
        if(partSum >= rand){
            break;
        }
    }
    //return index;
    gaPop->Reset(*index);
    return gaPop->Data();
}

MapNode * gaGetAParentNode(MapGraph *parent, int *node_pos)
{
    LinkedList<MapNode *> *parGateList;
    int parent_size;
    double par_node_posD;
    MapNode *parNode;
    NodeType parNodeType;
    NodeFunc parNodeFunc;

```

```

parGateList = parent->GetGateList();
parent_size = parGateList->ListSize();
while(1){
    par_node_posD = gaGenRandomNumber(parent_size);
    *node_pos = (int)par_node_posD;
    parGateList->Reset(*node_pos);
    parNode = parGateList->Data();

    parNodeType = parNode->GetNodeType();
    parNodeFunc = parNode->GetNodeFunc();
    if(parNodeType != PRIMARY_INPUT && parNodeFunc != NODE_PO){
        break;
    }
}
return parNode;
}

// Check cross-over target node's func type and its fanins.
// returns 1 if same type and fanins, and parent2 doesn't have target_node
// 1 if p1Node is PRIMARY_OUTPUT or its fanout is NODE_PO
// If 0 is returned cross-over can be applied.
int gaCheckTargetNodeTypeAndFanins(MapNode *p1Node, MapGraph *P1, MapGraph *P2)
{
    LinkedList<MapNode> *p2GateList, *p1nInList, *p2nInList;
    LinkedList<MapNode> *p1fanoutList;
    MapNode *p2Node, *p1fanoutNode;
    NodeFunc p1Func, p2Func, p1fanoutFunc;
    NodeType p1Type;
    int p2_pos;
    char *tgt_name;

    tgt_name = p1Node->GetName();
    p1Type = p1Node->GetNodeType();
    p1fanoutList = p1Node->GetFanoutList();
    if(p1fanoutList->ListSize()==1){
        p1fanoutList->Reset();
        p1fanoutNode = p1fanoutList->Data();
        p1fanoutFunc = p1fanoutNode->GetNodeFunc();
    } else {
        return (1);
    }
    if(p1Type==PRIMARY_OUTPUT || p1fanoutFunc==NODE_PO){
        return (1);
    }

    p2GateList = P2->GetGateList();
    if((p2_pos=P2->FindNode(p2GateList, tgt_name))!=-1){
        return (1);
    } else {
        p2GateList->Reset(p2_pos);
        p2Node = p2GateList->Data();
        if((p2Func = p2Node->GetNodeFunc())!=(p1Func = p1Node->GetNodeFunc())){
            return (0);
        } else {
            // nodes input check
            p1nInList = p1Node->GetFaninList();
            p2nInList = p2Node->GetFaninList();
            if(p1nInList->ListSize()!=p2nInList->ListSize()){
                return (0);
            } else {
                return (1); // may need to modify. Q<-p1nInList, comp with p2L
            }
        }
    }
}

// if crn != NULL, copy parent nodes from bNode to crn Node to the child,
// == NULL, copy parent nodes from bNode to PI to the child
void gaCopyPartialChromosome(MapNode *bNode, MapGraph *child, char *crn = NULL)
{
    MapNode *pCurNode, *chCurNode, *pcInNode, *chInNode;
    NodeType nodeType;
    NodeFunc nodeFunc;
    char *pCurName, *inNodeName;

    Match *mapped_cell;
    LinkedList<MapNode> *finList;
    int pos;

    Queue<MapNode> *Q;
    Q.Insert(bNode); // parent begin node

    while(!Q.Empty()){
        pCurNode = Q.DeQueue(); // parent current node
        pCurName = pCurNode->GetName();
        nodeType = pCurNode->GetNodeType();
        nodeFunc = pCurNode->GetNodeFunc();
        mapped_cell = pCurNode->GetMatch();

        //chCurNode = child->FindOrCreateNode(pCurName, pos, nodeType, nodeFunc);
        chCurNode = child->FindOrCreateNode(pCurName, pos, nodeType);
        chCurNode->SetNodeType(nodeType);
        chCurNode->SetNodeFunc(nodeFunc);

        // copy mapped cell pointer to child node
        chCurNode->SetMatch(mapped_cell);

        finList = pCurNode->GetFaninList();
        for(finList->Reset(); finList->EndOfList(); finList->Next()){
            pcInNode = finList->Data();
            inNodeName = pcInNode->GetName();
            nodeType = pcInNode->GetNodeType();
            nodeFunc = pcInNode->GetNodeFunc();
            chInNode = child->FindOrCreateNode(inNodeName, pos,
                                                nodeType, nodeFunc);
            chCurNode->InsertFanin(chInNode);
        }

        chInNode->InsertFanout(chCurNode);
        if(crn != NULL){
            if(strcmp(inNodeName, crn)!=0 && nodeFunc!=NODE_PI){
                Q.Insert(pcInNode);
            } else {
                if(nodeFunc!=NODE_PI){
                    Q.Insert(pcInNode);
                }
            }
        }
    } // while

    void gaCrossOver(MapGraph *parent1, MapGraph *parent2, MapNode *crossNode,
                    MapGraph *child1, MapGraph *child2)
    {
        LinkedList<MapNode> *p1PoList, *p1gateList, *p2PoList, *p2gateList;
        MapNode *p1RootNode, *p2RootNode, *p1CrossNode, *p2CrossNode;
        char *cross_node_name;
        int pos = -1;

        cross_node_name = crossNode->GetName();
        p1PoList = parent1->GetPoList();
        p2PoList = parent2->GetPoList();
        if(p1PoList->ListSize()>1 || p2PoList->ListSize()>1){
            cout<<"In function gaCrossOver(), Parent(s) has more than one output\n";
            cout<<"Check the Parent1 (a Chromosome)....\n";
            exit(1);
        } else {
            p1PoList->Reset(0);
            p1RootNode = p1PoList->Data();
            p2PoList->Reset(0);
            p2RootNode = p2PoList->Data();
        }

        (void)gaCopyPartialChromosome(p1RootNode, child1, cross_node_name);
        (void)gaCopyPartialChromosome(p2RootNode, child2, cross_node_name);

        p1gateList = parent1->GetGateList();
        pos = parent1->FindNode(p1gateList, cross_node_name);
        //cout << "Cross_node_name = "<<cross_node_name<<" Pos = "<<pos<<endl;
        p1gateList->Reset(pos);
        p1CrossNode = p1gateList->Data();

        p2gateList = parent2->GetGateList();
        pos = parent2->FindNode(p2gateList, cross_node_name);
        //cout << "Cross_node_name = "<<cross_node_name<<" Pos = "<<pos<<endl;
        p2gateList->Reset(pos);
        p2CrossNode = p2gateList->Data();

        (void)gaCopyPartialChromosome(p1CrossNode, child2);
        (void)gaCopyPartialChromosome(p2CrossNode, child1);
    }

    // Select crossNode's fanoutNode's MatchList to merge crossNode and foNode
    LinkedList<Match> * *gaCrossNodeFanoutMatchListX(Graph *sGraph,
        MapNode *crossNode, CellType crossType,
        MapNode *foNode, CellType foType, Node *sNode)
    {
        LinkedList<MapNode> * *fo_fanoutList;
        MapNode *fo_foNode;

        NodeFunc cross_Func, fo_Func;
        char *crossNode_name, *fo_name;

        LinkedList<Match> * *sMatchList = NULL;

        cross_Func = crossNode->GetNodeFunc();
        crossNode_name = crossNode->GetName();

        fo_Func = foNode->GetNodeFunc();
        fo_name = foNode->GetName();

        /* To merge fo_node & cross_node, both cell type should be same type
        * Modified on 08-01-00
        */
        if(fo_Func == NODE_INV){
            fo_fanoutList = foNode->GetFanoutList();
            if(fo_fanoutList->ListSize()==1){
                fo_fanoutList->Reset();
                fo_foNode = fo_fanoutList->Data();
                if(gaGetMappedCellType(fo_foNode)==PTL){
                    sMatchList = sNode->GetSMatchList();
                } else {
                    sMatchList = sNode->GetPMatchList();
                }
            } else if(foNode->GetNodeFunc()!=NODE_UNDEFINED){
                sMatchList = sNode->GetPMatchList();
            } else {
                cout<<_FILE_<<" line "<<_LINE_<<endl;
                cout<<"Error :: node("<<foNode->GetName()
                    <<")'s func is not defined\n";
            }
        } else {
            (foType==PTL) ? sMatchList = sNode->GetPMatchList() : \
                sMatchList = sNode->GetSMatchList();
        }
        return sMatchList;
    }

    // Find a match from "sMatchList" that covers the childNode rooted at foNode.
    Match * gaFindMergedMatch(LinkedList<Match> * *sMatchList, MapNode *foNode,
        MapNode *childNode)
    {
        char *child_name, *net_pin_name, *sm_pin_name;
        char *cur_pin_name, *new_pin_name;
        LinkedList<MapNode> * *foFaninList, *chFaninList;
        MapNode *curNode;
    }

```

```

LinkedList<char *> newPinList;
Queue<char *> Q;

Match *snodeMatch;
LinkedList<pinMap *> *smatchPins;
pinMap *pin_map;
int count;

child_name = childNode->GetName();
foFaninList = foNode->GetFaninList();

// Insert pins of fanout (of cross node) into temp nameList
for(foFaninList->Reset(); !foFaninList->EndOfList(); foFaninList->Next()){
    curNode = foFaninList->Data();
    net_pin_name = curNode->GetName();
    if(strcmp(net_pin_name, child_name) != 0 && !gaIsVddGnd(net_pin_name)){
        newPinList.InsertRear(net_pin_name);
    }
}

// Insert input pins of cross node into temp nameList
chFaninList = childNode->GetFaninList();
for(chFaninList->Reset(); !chFaninList->EndOfList(); chFaninList->Next()){
    curNode = chFaninList->Data();
    net_pin_name = curNode->GetName();
    newPinList.InsertRear(net_pin_name);
}

// finds new merged match
for(sMatchList->Reset(); !sMatchList->EndOfList(); sMatchList->Next()){
    snodeMatch = sMatchList->Data();
    smatchPins = snodeMatch->pinmap;
    for_each(smatchPins){
        pin_map = smatchPins->Data();
        sm_pin_name = pin_map->net_name;
        if(!gaIsVddGnd(sm_pin_name)){
            Q.Insert(sm_pin_name);
        }
    }
    count = 0;
    if(Q.Length() != newPinList.ListSize()){
        while(!Q.Empty()){ (void)Q.Dequeue(); }
        continue;
    }
    while(!Q.Empty()){
        cur_pin_name = Q.Dequeue();
        for(newPinList.Reset(); !newPinList.EndOfList(); newPinList.Next()){
            new_pin_name = newPinList.Data();
            if(strcmp(cur_pin_name, new_pin_name) == 0){
                count++;
            }
        }
        if(newPinList.ListSize() == count){
            return snodeMatch;
        }
    }
}
return (NULL);

// delete remove childNode from child1 MapGraph, and modify foNode fanins
void gaMergeSameTypeNodes(MapGraph *child, MapNode *foNode, MapNode *childNode,
    Match *mergedMatch, NodeFunc new_fo_func)
{
    LinkedList<MapNode *> *foutList, *finList;
    MapNode *curNode;
    char *childNode_name;
    int pos;

    childNode_name = childNode->GetName();

    // find fanins of the childNode, and delete its fanout nodes
    finList = childNode->GetFaninList();
    for(finList->Reset(); !finList->EndOfList(); finList->Next()){
        curNode = finList->Data();
        foutList = curNode->GetFanoutList();
        if((pos = child->FindNode(foutList, childNode_name)) != -1){
            foutList->Reset(pos);
            foutList->DeleteCurrent(); //==
        } else {
            cout<<"In gaMergeSameTypeNodes(), "
                <<childNode_name<<" can't be found in childNode's fanoutList\n";
        }
    }

    finList = foNode->GetFaninList();
    if((pos = child->FindNode(finList, childNode_name)) != -1){
        finList->Reset(pos);
        finList->DeleteCurrent();
    } else {
        cout<<"In gaMergeSameTypeNodes(), "
            <<childNode_name<<" can't be found in foNode's faninList\n";
    }

    // delete childNode
    child->DeleteGate(childNode_name);

    LinkedList<pinMap *> *pMapList;
    pinMap *curPin;
    char *net_name;
    MapNode *faninNode;

    foNode->SetNodeFunc(new_fo_func);
    foNode->SetMatch(mergedMatch); // 06-24

    pMapList = mergedMatch->pinmap;
    for(pMapList->Reset(); !pMapList->EndOfList(); pMapList->Next()){
        curPin = pMapList->Data();
        net_name = curPin->net_name;
        if(gaIsVddGnd(net_name) ||
            (pos = child->FindNode(finList, net_name)) != -1){
            continue;
        }
    }

    faninNode = child->FindOrCreateNode(net_name, pos);
    if(pos == -1){
        cout<<"In gaMergeSameTypeNodes(), "
            <<net_name<<" is created. Strange... Check it.\n";
    }
    foNode->InsertFanin(faninNode);
    faninNode->InsertFanout(foNode);
}

// returns 1 if the child Chromosome has valid structure, and if possible
// merge the node (crossovered node) and its fanout node
int gaCheckValidityOfChildAndMerge(MapNode *crossNode, MapGraph *child,
    Graph *sGraph, LinkedList<Node *> *subOutList,
    int num_merge)
{
    LinkedList<Node *> *sgraphGateList;
    LinkedList<MapNode *> *childGateList, *foList;
    char *crossNode_name, *fo_name;
    int child_pos, sg_pos;
    MapNode *childNode=NULL, *foNode=NULL, *fofoNode;
    Node *sNode;

    CellType fo_cell_type, cross_cell_type, fo_fo_Type;
    NodeFunc new_fo_func, fo_Func, fo_fo_Func;

    LinkedList<Match *> *sMatchList;
    Match *mergedMatch;

    crossNode_name = crossNode->GetName();
    childGateList = child->GetGateList();
    if((child_pos = child->FindNode(childGateList, crossNode_name)) != -1){
        childGateList->Reset(child_pos);
        childNode = childGateList->Data();

        foList = childNode->GetFanoutList();
        if(foList->ListSize() == 1){
            foList->Reset();
            foNode = foList->Data();
            fo_Func = foNode->GetNodeFunc();
            fo_name = foNode->GetName();
        } else {
            return (0);
        }
    }

    // returns PTL or STATIC
    cross_cell_type = gaGetMappedCellType(childNode);
    fo_cell_type = gaGetMappedCellType(foNode);

    if(cross_cell_type != fo_cell_type){
        num_merge = 0;
        return (1);
    }

    sgraphGateList = sGraph->GetGateList();
    if((sg_pos = sGraph->FindNode(sgraphGateList, fo_name)) != -1){
        sgraphGateList->Reset(sg_pos);
        sNode = sgraphGateList->Data();
    } else {
        cout<<" In gaCheckValidityOfChildAndMerge(), crossNode("
            <<crossNode_name<<")'s fanout("<<fo_name<<") cannot find.\n";
        exit(1);
    }

    sMatchList = gaCrossNodeFanoutMatchList(sGraph, childNode, cross_cell_type,
        foNode, fo_cell_type, sNode);

    mergedMatch = gaFindMergedMatch(sMatchList, foNode, childNode);

    if(mergedMatch != NULL){
        new_fo_func = gaGetMatchedCellFunc(sGraph, subOutList, sNode, mergedMatch);

        // check if the mergedMatch is PTL, (if PTL2 + PTL1 mode is enabled)
        // then check new_fo_func and fo_fo_Func are PTL2,3 + PTL1...
        // if not return (0), if OK do next routine
        fofoNode = gaP21GetFanoutNodeMapFunc(foNode, &fo_fo_Func);
        if(fofoNode != NULL){
            fo_fo_Type = gaGetMappedCellType(fofoNode);
        }
        if(new_fo_func == NODE_PTL2){
            // Since merging cell will be ptl_y2 or ptl_y3.
            if(fo_fo_Func == NODE_PTL2 || fo_fo_Func == NODE_PTL3){
                return (0);
            }
        } else if(new_fo_func == NODE_PTL1){
            return (0);
        }

        (void)gaMergeSameTypeNodes(child, foNode, childNode, mergedMatch,
            new_fo_func);
    } else {
        return (0);
    }
    return (1);
}

void gaDeleteAChromosome(MapGraph *child)
{
    delete child;
}

// Evaluate cost of the each child, and replace these if there is any
// chromosome that has more expensive cost than child(s).
void gaUpdatePopulation(LinkedList<MapGraph *> *gaPop,
    LinkedList<LibCell *> *cellLibrary,
    MapGraph *child1, int valid1, MapGraph *child2, int valid2)
{
    double areaCost, delayCost, maxArea;
    int pop_index; // index of gaPop which has max cost value
    if(valid1 == 1){

```

```

        maxArea = gaEvaluateGetMaxAreaCost(gaPop, &pop_index);
        (void)gaEvaluateChromosome(child1, cellLibrary, &areaCost, &delayCost);
        if (areaCost < maxArea) {
            gaPop->Reset(pop_index);
            delete gaPop->DeleteCurrentElem(); //==
            gaPop->InsertRear(child1);
        } else {
            delete child1;
        }
    }
    if (valid2==1) {
        maxArea = gaEvaluateGetMaxAreaCost(gaPop, &pop_index);
        (void)gaEvaluateChromosome(child2, cellLibrary, &areaCost, &delayCost);
        if (areaCost < maxArea) {
            //delete a chromosome which has max_area
            gaPop->Reset(pop_index);
            delete gaPop->DeleteCurrentElem(); //==
            //insert
            gaPop->InsertRear(child2);
        } else {
            delete child2;
        }
    }
}

void gaSelectCrossover(LinkedList<MapGraph*> *gaPop,
    LinkedList<LibCell*> *cellLibrary, Graph *sGraph,
    LinkedList<Node*> *subOutList)
{
    int indexP1, indexP2;
    MapGraph *parent1, *parent2;

    int par1_node_pos, type_fanin, itr = 0, itr_max = 10;
    MapNode *pariNode;
    char *target_name;
    int valid1 = 0, valid2 = 0, num_merge = 0;

    double area, delay;

    while(1) {
        // returns (-1) if Pop.size = 0
        itr++;
        parent1 = gaSelectiParent(gaPop, &indexP1);
        parent2 = gaSelectiParent(gaPop, &indexP2);
#ifdef GA_SELECT_DEBUG
        (void)gaPrint2Parents(parent1, indexP1, parent2, indexP2);
#endif

        // get the chromosomes by indexing indexP1 and indexP2
        // using parent_1, select a gene(internal node) and identify its fanout
        // node and its matched type. Then goto parent_2, find same node which
        // has same matched cell type. And perform crossover

        pariNode = gaGetAParentNode(parent1, &pari_node_pos);
        target_name = pariNode->GetName();

        type_fanin = gaCheckTargetNodeTypeAndFanins(pariNode, parent1, parent2);
#ifdef GA_SELECT_DEBUG
        cout << "Target Node Name = "<<target_name<<" St = "<<type_fanin<<"\n";
#endif
        if (!type_fanin || itr > itr_max) {
            break;
        }
    } // while

    if (type_fanin == 1) {
        return;
    }

    MapGraph *child1 = new MapGraph;
    MapGraph *child2 = new MapGraph;
    (void)gaCrossover(parent1, parent2, pariNode, child1, child2);
#ifdef GA_CROSS_DEBUG
    cout << "\n Crossovered Two Childs \n";
    (void)gaPrint2Parents(child1, 101, child2, 102);
#endif

    valid1 = gaCheckValidityOfChildAndMerge(pariNode, child1, sGraph,
        subOutList, num_merge);
    if (valid1) gaEvaluateChromosome(child1, cellLibrary, &area, &delay);
    valid2 = gaCheckValidityOfChildAndMerge(pariNode, child2, sGraph,
        subOutList, num_merge);
    if (valid2) gaEvaluateChromosome(child2, cellLibrary, &area, &delay);
#ifdef GA_CROSS_DEBUG
    cout << "\n Child 1 valid = "<<valid1<<" Child 2 valid = "<<valid2<<"\n";
    (void)gaPrint2Parents(child1, 201, child2, 202);
#endif

    // Evaluate Cost function and replace if the is less tha max cost
    (void)gaUpdatePopulation(gaPop, cellLibrary, child1, valid1, child2, valid2);
}

//----- Mutation -----
void gaResetMapGraphVisitedFlag(LinkedList<MapGraph*> *gaPop)
{
    MapGraph *aChromosome;

    for (gaPop->Reset(); !gaPop->EndOfList(); gaPop->Next()) {
        aChromosome = gaPop->Data();
        aChromosome->SetVisited(0);
    }
}

// select random parent from the population, and select random node
// do mutation and evaluate if its cost ???
void gaDoRandomMutation(LinkedList<MapGraph*> *gaPop, double mutation_rate)
{
    MapNode *gaMutateSelectANode(MapGraph *aChromosome, CellType *cType, int *flag)
    {
        int chroSize, genePos, itr_cnt=0;
        LinkedList<MapNode*> *chroGateList;
        MapNode *curNode;
        NodeFunc curFunc;
        NodeType curType;

        chroGateList = aChromosome->GetGateList();
        chroSize = chroGateList->ListSize();
        while(1) {
            itr_cnt++;
            genePos = (int)gaGenRandomNumber((double)(chroSize));
            chroGateList->Reset(genePos);
            curNode = chroGateList->Data();
            curFunc = curNode->GetNodeFunc();
            curType = curNode->GetNodeType();
            if (curFunc != NODE_PI && curFunc != NODE_PO || itr_cnt>30) { //0720
                *flag = 1;
                break;
            }
        }
        *cType = gaGetMappedCellType(curNode);
        if (itr_cnt>30) *flag=0;
        return curNode;
    }

    // for PerformMutation for only NAND_NOR_INV
    Node *gaFindsubGraphNode(Graph *sGraph, MapNode *curMapNode)
    {
        char *mG_node_name;
        int pos;

        LinkedList<Node*> *sGateList;
        Node *sNode;

        mG_node_name = curMapNode->GetName();
        sGateList = sGraph->GetGateList();
        if ((pos = sGraph->FindNode(sGateList, mG_node_name)) != -1) {
            sGateList->Reset(pos);
            sNode = sGateList->Data();
        } else {
            cout << "In gaGetUnMappedNode(), cannot find "<<mG_node_name<<"....\n";
            sNode = NULL;
        }
        return sNode;
    }

    LinkedList<Match*> *gaGetUnMappedGraphNode(MapNode *curNode, Graph *sGraph,
        Node *sNode)
    {
        char *mG_node_name;
        int pos;

        LinkedList<Node*> *sGateList;
        LinkedList<Match*> *mapList;
        CellType cType;
        NodeType nodeType; //0720

        mG_node_name = curNode->GetName();
        sGateList = sGraph->GetGateList();
        if ((pos = sGraph->FindNode(sGateList, mG_node_name)) != -1) {
            sGateList->Reset(pos);
            sNode = sGateList->Data();
        } else {
            cout << "In gaGetUnMappedNode(), cannot find "<<mG_node_name<<"....\n";
            sNode = NULL;
            exit(0);
        }

        cType = gaGetMappedCellType(curNode);
        nodeType = curNode->GetNodeType(); // 0720
        if (nodeType != PRIMARY_OUTPUT) { //0720
            if (cType==PTL) {
                mapList = sNode->GetPMatchList();
            } else {
                mapList = sNode->GetSMatchList();
            }
        } else { // 0720
            if (cType==PTL) {
                mapList = sNode->GetSMatchList();
            } else {
                mapList = sNode->GetPMatchList();
            }
        }
        return mapList;
    }

    void gaMutateDeleteAMapNode(MapGraph *aChrom, MapNode *curNode)
    {
        LinkedList<MapNode*> *piList, *poList, *gateList, *finList, *foList;
        MapNode *finNode, *foNode, *no_foNode;
        NodeType cType;
        int pos;
        char *fo_name, *cur_name, *no_fo_name;

        Queue<MapNode*> Q;

        gateList = aChrom->GetGateList();
        piList = aChrom->GetPiList();
        poList = aChrom->GetPoList();

        cType = curNode->GetNodeType();
        cur_name = curNode->GetName();

        // 1) delete current node in current fanin's fanout list
        finList = curNode->GetFaninList();
        if (finList->ListSize()>0) {
            for (finList->Reset(); finList->EndOfList(); finList->Next()) {
                finNode = finList->Data();
                foList = finNode->GetFanoutList();
                if (foList->ListSize()>0) {
                    for_each(foList) {
                        foNode = foList->Data();
                    }
                }
            }
        }
    }
}

```

```

        fo_name = foNode->GetName();
        if(strcmp(cur_name, fo_name)==0){
            //foList->DeleteCurrent(); // 11-13-00
            (void)foList->DeleteCurrentElem(); //
            break;
        }
    }
    // 2) if foList is 0, enqueue Q to delete after for_loop
    if(foList->ListSize()==0){
        Q.Insert(finNode);
    }
}
// 3) current node's fanin node, which doesn't have any fanout, delete
while(!Q.Empty()){
    no_foNode = Q.Dequeue();
    no_fo_name = no_foNode->GetName();
    if((pos=aChrom->FindNode(gateList, no_fo_name))!=-1){
        gateList->Reset(pos);
        gateList->DeleteCurrent(); //
        aChrom->SetNumGate(aChrom->GetNumGate()-1);
    }
    if((pos=aChrom->FindNode(piList, no_fo_name))!=-1){
        piList->Reset(pos);
        piList->DeleteCurrent(); //==
        aChrom->SetNumPi(aChrom->GetNumPi()-1);
    }
}
// 4) delete current node
if((pos=aChrom->FindNode(gateList, cur_name))!=-1){
    gateList->Reset(pos);
    gateList->DeleteCurrent(); //==
    aChrom->SetNumGate(aChrom->GetNumGate()-1);
} else { // Possible
#ifdef GA_MUTATE_DEBUG
    cout << "In gaMutateDeleteNodes(), "<<cur_name
        << " is not in GATE List...\n";
#endif
}
// Delete all node starting from bNode to the PI nodes in a Chromosome
void gaMutateDeleteNodes(MapGraph *aChromosome, MapNode *beginNode)
{
    Queue<MapNode> *Q;
    MapNode *curNode, *finNode;
    char *cur_name;
    LinkedList<MapNode> *finList;
    int pos;

    // 1) Delete beginNode and its fanins while traversing down the network
    Q.Insert(beginNode);
    while(!Q.Empty()){
        curNode = Q.Dequeue();
        cur_name = curNode->GetName();

        finList = curNode->GetFaninList();
        if(finList->ListSize()!=0){
            for_each(finList){
                finNode = finList->Data();
                if(finNode->GetNodeType()!=PRIMARY_INPUT){
                    Q.Insert(finNode); // To traverse down the network
                }
            }
        }
        (void)gaMutateDeleteAMapNode(aChromosome, curNode);
    }

    // 2) Delete begin node's fanins to use current node during mutated
    // illegal solution creation
    finList = beginNode->GetFaninList();
    if(finList->ListSize()>0){
        for(finList->Reset(); finList->EndOfList(); finList->Next()){
            Q.Insert(finList->Data());
        }
    }
    while(!Q.Empty()){
        curNode = Q.Dequeue();
        cur_name = curNode->GetName();
        if((pos = aChromosome->FindNode(finList, cur_name))!=-1){
            finList->Reset(pos);
            finList->DeleteCurrent(); //==
            beginNode->SetNumFanin(beginNode->GetNumFanin()-1);
        }
    }
}

void gaMutateBuildALegalSol(Graph *origNet, Graph *sGraph,
    LinkedList<Node> *subOutList,
    MapGraph *gaGraph, MapNode *bNode, Match *sMatch)
{
    LinkedList<pinMap> *pMapList;
    pinMap *pin_map;
    Match *aMatch;

    MapNode *mapNode, *finNode, *foMapNode;
    LinkedList<Node> *sGateList;
    Queue<Node> *Q;
    Node *curNode, *sfinNode, *bNode_in_sGraph;
    char *bnode_name, *curName, *netpinName;
    int pos, begin_i;
    CellType ctype, finType;

    NodeType curNodeType, curPinNodeType;
    NodeFunc matchFuncType, bNode_new_match_Func;

    //LinkedList<Node> *sGateList;
    int pos_sgate = -1;

    // 1) Set begin node's match which is different from old(deleted) match
    // and its Node Function(NODE_ADN or ...)
    bNode->SetMatch(sMatch);
    bnode_name = bNode->GetName();

    sGateList = sGraph->GetGateList();
    if((pos=sGraph->FindNode(sGateList, bnode_name))!=-1){
        sGateList->Reset(pos);
        bNode_in_sGraph = sGateList->Data();
        bNode_new_match_Func = gaGetMatchedCellFunc(sGraph, subOutList,
            bNode_in_sGraph, sMatch);
    } else {
        cout << "In gaMutateBuildALegalSol(), "<< bnode_name<< " cannot be "
            << " found in sGraph(un_mapped circuit)\n";
    }

    // 2) Update begin node's new mapped cell type
    if((pos=sGraph->FindNode(sGateList, bnode_name))!=-1){
        //sGateList->Reset(pos);
        sGateList->Reset(pos);
        curNode = sGateList->Data();
        curNode->SetMatchType(gaGetMappedCellType(bNode));
    }

    // 3) Build new legal solution
    Q.Insert(curNode);
    while(!Q.Empty()){
        curNode = Q.Dequeue();
        curName = curNode->GetName();
        ctype = curNode->GetMatchType();
        if(begin==1){
            aMatch = bNode->GetMatch();
            gaGraph->InsertGate(bNode);
            begin--;
        } else {
            foMapNode = gaFindFanoutNodeInGaGraph(gaGraph, curName);
            aMatch=gaGetALegalMatch(origNet, sGraph, subOutList,
                curNode, ctype, foMapNode);
        }

        curNodeType = gaGetCurrentNodeType(sGraph, subOutList, curNode);
        if(curNodeType==UNASSIGNED){
            cout<<"Node("<<curName<<")'s node type is "
                <<((int)curNodeType)<<endl;
            cout<<"In "<<__FILE__<<"at line "<<__LINE__<<endl;
            exit(0);
        }
        matchFuncType = gaGetMatchedCellFunc(sGraph, subOutList, curNode, aMatch);
        mapNode = gaGraph->FindOrCreateNode(curName, pos, curNodeType,
            matchFuncType);

        mapNode->SetMatch(aMatch);
        if(pos!=-1 && mapNode->GetNodeFunc()!=NODE_UNDEFINED){
            mapNode->SetNodeType(curNodeType);
            mapNode->SetNodeFunc(matchFuncType);
        }

        pMapList = aMatch->pinmap;
        for(pMapList->Reset(); !pMapList->EndOfList(); pMapList->Next()){
            pin_map = pMapList->Data();
            netpinName = pin_map->net_node;
            if((strcmp(netpinName, "GND")==0) || (strcmp(netpinName, "VDD")==0)){
                continue;
            }
            curPinNodeType = gaGetInPinNodeType(origNet, sGraph, subOutList,
                netpinName, ctype);
            if(curPinNodeType == PRIMARY_INPUT){
                finNode = gaGraph->FindOrCreateNode(netpinName, pos,
                    curPinNodeType, NODE_PI);
            } else {
                finNode = gaGraph->FindOrCreateNode(netpinName, pos,
                    curPinNodeType);
            }

            // Insert finNode into current mapNode's fanin
            mapNode->InsertFanin(finNode);
            finNode->InsertFanout(mapNode);

            // Insert newly created node into Queue to reaverse network
            if(curPinNodeType==INTERNAL){
                pos_sgate=sGraph->FindNode(sGateList, netpinName);
                sGateList->Reset(pos_sgate);
                sfinNode = sGateList->Data();
                Q.Insert(sfinNode);
            }

            // Allow PTL2+PTL1, i.e., 3 trs inseries
            finType=gaDetermineFaninCellType(mapNode, curNode, sfinNode);
        }
    } // end for_loop
} // while()

// perform mutation, updates a chromosome in the population
// Select a node in the MapGraph, and identify the current mapping. Then go
// to sGraph and find the same node selected in the MapGraph, and find
// a different match from the current mapping in the node. Map the selected
// match to the node and make the sub-tree rooted at selected node legal.
int gaPerformMutation(Graph *origNet, Graph *sGraph,
    LinkedList<Node> *subOutList, MapGraph *gaGraph)
{
    MapNode *curNode;
    CellType ctype;

```

```

Node *subNode, *dummyNode=NULL;
LinkedList<Match *> *matchList;
Match *curCellMatch, *sMatch;
char *cur_match_name, *smatch_name;

int flag=0, found_different_match=0;
int numMatches, match_pos, count;

MapNode *foMapNode;
NodeFunc fo_Func, matchFunc;

curNode = gaMutateSelectANode(gaGraph, &cType, &flag);
//For mapping only NAND_ONR_INV (09-30-00)
foMapNode = gaP21GetFanoutNodeMapFunc(curNode, &fo_Func);

if(flag==0){
    return flag;
}
curCellMatch = curNode->GetMatch();
cur_match_name = curCellMatch->cell_name;

matchList = gaGetUnMappedGraphNode(curNode, sGraph, dummyNode);
subNode = gaFindsubGraphNode(sGraph, curNode); //curNode=current mapped Node
if(matchList == NULL) return (0);
numMatches = matchList->ListSize();
if(numMatches==0) return (0);

count = 0;
while(1){
    count++;
    match_pos = (int)gaGenRandomNumber((double)numMatches);
    matchList->Reset(match_pos);
    sMatch = matchList->Data();

    //for NANDNOR
    matchFunc = gaGetMatchedCellFunc(sGraph, subOutList, subNode, sMatch);

    smatch_name = sMatch->cell_name;
    if(strcmp(cur_match_name, smatch_name)!=0){
        if(gaAreSignalsLegal(origNet, sGraph, cType, sMatch)){
            #ifndef ONLY_NOR_NAND_INV
                found_different_match = 1;
                break;
            #else
                if(subNode->GetMatchType()==STATIC){
                    if((matchFunc==NODE_NAND || matchFunc==NODE_INV ||
                        matchFunc==NODE_NOR)) {
                        found_different_match = 1;
                        break;
                    }
                } else if (foMapNode==NULL){
                    if(matchFunc==NODE_PTL1 || matchFunc==NODE_PTL2 ||
                        matchFunc==NODE_PTL3 || matchFunc==NODE_NAND ||
                        matchFunc==NODE_INV || matchFunc==NODE_NOR){
                        found_different_match = 1;
                        break;
                    }
                }
            }
        }
    }
    #endif
    if(count >= 10*numMatches) {
        return (0);
    }
}

if(found_different_match==0){
    flag = 0;
    return flag;
}

// If a different match type was not found, go to the fanout node of
// curNode and find
if(strcmp(cur_match_name, smatch_name)==0){
    (void)gaMutateDeleteNodes(gaGraph, curNode);
    (void)gaMutateBuildLegalSol(origNet, sGraph, subOutList,
                                gaGraph, curNode, sMatch);
    flag = 1;
} else {
    // 1) delete the cone from cur node's fanin and its transitive fanins
    (void)gaMutateDeleteNodes(gaGraph, curNode);
    #ifdef GA_MUTATE_DEBUG
        cout << "**** Current Node = "<<curNode->GetName()<<" ****\n";
        gaGraph->PrintPoNodes();
        gaGraph->PrintPiNodes();
        gaGraph->PrintAllNodes();
    #endif
    // 2) build new legal cone from the cur_node
    (void)gaMutateBuildLegalSol(origNet, sGraph, subOutList,
                                gaGraph, curNode, sMatch);
    #ifdef GA_MUTATE_DEBUG
        cout << "Recovered Chromosome ....\n";
        gaGraph->PrintPoNodes();
        gaGraph->PrintPiNodes();
        gaGraph->PrintAllNodes();
    #endif
    flag = 1;
}
return flag;
}

// Select chromosomes that does not have best cost. The number of chromosomes
// to be mutated is determined by the "mutation_rate", which represent the
// fraction of the population.
void gaDoRateMutation(Graph *origNet, Graph *sGraph,
    LinkedList<Node *> *subOutList,
    LinkedList<MapGraph *> *gaPop, double mr)
{
    double min_cost, max_cost;
    int min_first_index, max_first_index, pop_size, numMutate;

    MapGraph *aChromosome;
    double chroCost;
    int n, visit_flag, n_min_chrom;
    int flag;

    min_cost = gaEvaluateGetMinAreaCost(gaPop, &min_first_index, &n_min_chrom);
    pop_size = gaPop->ListSize();
    numMutate = (int)(pop_size*mr); // mr is mutation_rate
    if((pop_size - n_min_chrom) > numMutate){
        numMutate = pop_size - n_min_chrom;
    }
    for(n = 0; n < numMutate; n++){
        max_cost = gaEvaluateGetMaxAreaCost(gaPop, &max_first_index, 1);
        gaPop->Reset(max_first_index);
        aChromosome = gaPop->Data();
        chroCost = aChromosome->GetArea();
        visit_flag = aChromosome->GetVisited();
        if(!visit_flag && (chroCost > min_cost)){
            flag=gaPerformMutation(origNet, sGraph, subOutList, aChromosome);
            if(flag==1){
                aChromosome->SetVisited(1);
                break;
            }
        }
    }
    (void)gaResetMapGraphVisitedFlag(gaPop);
}

//----- Initialize Global Chromosome
// returns multiple fanout node list of a and2orNetwork excluding
// mfp node whose fanin is MFP node or PRIMARY_INPUT
void gaGetMfpNodeList(Graph *origNet, LinkedList<Node *> *mfpList)
{
    Node *mfpNode, *mfp_fiNode;
    LinkedList<Node *> *oGateList, *mfpn_foList, *mfpn_fiList;
    int num_fanout, num_mfp_fiNode_fanout;
    NodeFunc node_func, mfpn_fiNode_func;
    char *mfp_name;

    //mfp_poList = GetMultiFanoutAndPoNode(origNet); // in Subgraph_Bdd.cc
    oGateList = origNet->GetGateList();
    for(oGateList->Reset(); oGateList->EndOfList(); oGateList->Next()){
        mfpNode = oGateList->Data();
        mfpn_foList = mfpNode->GetFanoutList();
        num_fanout = mfpNode->GetNumFanout();
        if(num_fanout != mfpn_foList->ListSize() && num_fanout>1){
            mfp_name = mfpNode->GetName();
            cout<<"Error:Num fanout != foList size at Node "<<mfp_name<<"\n";
            cout<<"In "<<FILE__<<"a at "<<LINE__<<"\n";
            exit(0);
        }
        node_func = mfpNode->GetNodeFunc();
        if(mfpNode->GetNodeFunc()!=NODE_PO && num_fanout>1 &&
            node_func!=NODE_INV && node_func!=NODE_PI){
            mfpn_fiList = mfpNode->GetFaninList();
            if(mfpn_fiList->ListSize()>1){
                cout<<"Error : In "<<FILE__<<"at line "<<LINE__<<"\n";
                exit(0);
            }
            mfpn_fiList->Reset();
            mfp_fiNode = mfpn_fiList->Data();
            num_mfp_fiNode_fanout = mfp_fiNode->GetNumFanout();
            mfpn_fiNode_func = mfp_fiNode->GetNodeFunc();
            if(mfpn_fiNode_func!=NODE_PI && num_mfp_fiNode_fanout==1){
                mfpList->InsertRear(mfpNode);
            }
        }
    }
}

// popIndex[0] contains subGraph[0]'s index from subPopulation, and so on..
MapNode *gaGetMapNodeFromIndexForGchrom(
    LinkedList<MapGraph *> *subPList,
    int popIndex[], char *mfpName,
    CellType *mfpCellType, int get_po_node = 1)
{
    LinkedList<MapGraph *> *subPop;
    MapGraph *subChrom;
    LinkedList<MapNode *> *subGateList;
    MapNode *mapNode=NULL;
    NodeType map_node_type;
    int chrom_pos, n;

    chrom_pos = -1;
    for(subPList->Reset(); n=0; !subPList->EndOfList(); subPList->Next(), n++){
        subPop = subPList->Data();
        subPop->Reset(popIndex[n]);
        subChrom = subPop->Data();
        subGateList = subChrom->GetGateList();
        chrom_pos = subChrom->FindNode(subGateList, mfpName);
        if(chrom_pos != -1){
            subGateList->Reset(chrom_pos);
            mapNode = subGateList->Data();
            map_node_type = mapNode->GetNodeType();
            if(map_node_type==PRIMARY_OUTPUT && get_po_node ==1 ||
                map_node_type==PRIMARY_INPUT || get_po_node ==0){
                *mfpCellType = gaGetMappedCellType(mapNode);
                break;
            }
        } else {
            mapNode = NULL;
            *mfpCellType = UNKNOWN;
        }
    }
}

```

```

    } else {
        mapNode = NULL;
        *mapCellType = UNKNOWN;
    }
}
return mapNode;
}

int gasGetFanoutListFromGIndex(LinkedList<LinkedList<MapGraph *> *> *subPList,
int popIndex[], MapNode *mapNode, Queue<MapNode *> *Q)
{
    LinkedList<MapGraph *> *subPop;
    LinkedList<MapNode *> *subGateList, *fin_foList;
    MapGraph *subChrom;
    MapNode *subG_map_Node;

    char *map_node_name, *fin_name;
    int chrom_pos, k, num_fanout = 0;

    if(mapNode==NULL) return (0);
    map_node_name = mapNode->GetName();
    for(subPList->Reset(), k=0; !subPList->EndOfList(); subPList->Next(), k++){
        chrom_pos = -1;
        subPop = subPList->Data();
        subPop->Reset(popIndex[k]);

        subChrom = subPop->Data();
        subGateList = subChrom->GetGateList();
        for_each(subGateList){
            subG_map_Node = subGateList->Data();
            if(subG_map_Node->GetNodeType()==PRIMARY_INPUT){
                fin_name = subG_map_Node->GetName();
                if(strcmp(map_node_name, fin_name)==0){
                    Q->QInsert(subG_map_Node);
                    fin_foList = subG_map_Node->GetFanoutList();
                    num_fanout = num_fanout + fin_foList->ListSize();
                }
            }
        }
    }
    return num_fanout;
}

// Returns inverted signal name
char *gasGetInvertedSignalName(char *sig_name)
{
    int name_len;
    char *sig_name_b;

    name_len = strlen(sig_name);
    if(sig_name[name_len-1]!='\'){
        sig_name_b = (char *)calloc(name_len+1, sizeof(char));
        strncpy(sig_name_b, sig_name, name_len-1);
    } else {
        sig_name_b = (char *)calloc(name_len+2, sizeof(char));
        strcpy(sig_name_b, sig_name);
        strcat(sig_name_b, "\'");
    }
    return sig_name_b;
}

int gasCheckValidityFanoutCellType(CellType mnCellType,
Queue<MapNode *> *foQ,
LinkedList<LinkedList<MapGraph *> *> *subPList,
int popIndex[])
{
    CellType mfp_outCellType = UNKNOWN;
    LinkedList<MapGraph *> *subPop;
    MapGraph *subChrom;
    LinkedList<MapNode *> *subGateList;
    MapNode *mfoNode, *mapNode=NULL;
    int n, chrom_pos, valid=1;
    char *foName;

    while(!foQ->QEmpty()){
        mfoNode = foQ->DeQueue();
        foName = mfoNode->GetName();
        chrom_pos = -1;
        // gChrom is a global chromosome ( L<L MapNode *> > )
        for(subPList->Reset(), n=0; !subPList->EndOfList(); subPList->Next(), n++){
            subPop = subPList->Data();
            subPop->Reset(popIndex[n]);

            subChrom = subPop->Data();
            subGateList = subChrom->GetGateList();
            chrom_pos = subChrom->FindNode(subGateList, foName);
            if(chrom_pos != -1){
                subGateList->Reset(chrom_pos);
                mapNode = subGateList->Data();
                mfp_outCellType = gasGetMappedCellType(mapNode);
                break;
            }
        }

        if((mnCellType==mfp_outCellType) && mapNode->GetNodeFunc()!=NODE_INV){
            valid = 0;
            break;
        }
    }
    return valid;
}

int gasCheckValidityFanoutCellType(CellType mnCellType, Queue<MapNode *> *foQ,
LinkedList<LinkedList<MapGraph *> *> *subPList,
int popIndex[])
{
    CellType foCellType, mapCellType;
    LinkedList<MapNode *> *fList, *foList;

    MapNode *mfoNode, *mapNode, *inNode;
    int valid=1;

    while(!foQ->QEmpty()){
        mfoNode = foQ->DeQueue();
        if(mfoNode->GetNodeFunc()==NODE_PO){
            fList = mfoNode->GetFanInList();
            inNode = fList->Data(0);
            mapCellType = gasGetMappedCellType(inNode);
        } else {
            mapCellType = mnCellType;
        }
        if(mfoNode->GetNodeFunc() == NODE_INV) continue;
        foList = mfoNode->GetFanoutList();
        for(foList->Reset(); !foList->EndOfList(); foList->Next()){
            mapNode = foList->Data(0);
            if(mapNode->GetNodeFunc() == NODE_INV) continue;
            foCellType = gasGetMappedCellType(mapNode);
            if(foCellType==mapCellType){
                valid = 0;
                break;
            }
        }
    }
    return valid;
}

//C1355 has reconvergent fanout
int utilCheckIfReconvergence(Graph *origNet, Node *mfpNode)
{
    LinkedList<Node *> *mfoList, *foList1, *foList2;
    Node *foNode1, *foNode2;
    char *name1, *name2;
    int recon = 0;

    mfoList = mfpNode->GetFanoutList();
    if(mfoList->ListSize()==2){
        foNode1 = mfoList->Data(0);
        foNode2 = mfoList->Data(1);

        foList1 = foNode1->GetFanoutList();
        foNode1 = foList1->Data(0);
        name1 = foNode1->GetName();
        foList2 = foNode2->GetFanoutList();
        foNode2 = foList2->Data(0);
        name2 = foNode2->GetName();
        if(strcmp(name1, name2)==0){
            return (1);
        }
    } else {
        foList1 = foNode1->GetFanoutList();
        foNode1 = foList1->Data(0);
        name1 = foNode1->GetName();
        foList2 = foNode2->GetFanoutList();
        foNode2 = foList2->Data(0);
        name2 = foNode2->GetName();
        if(strcmp(name1, name2)==0) return (1);
    }
    return recon;
}

// returns TRUE if the gChromosome has valid circuit structure (P-S)
int gasCheckGlobalChromosomeValidity(Graph *origNet,
LinkedList<LinkedList<MapGraph *> *> *subPList,
LinkedList<Node *> *mfpList, int popIndex[])
{
    Node *mfpNode;
    char *mfpName, *mfpName_b=NULL;

    int num_fo_mfpNode, num_fo_mfpNode_b, num_fo_inSub;
    int num_fo_inMap, valid1 = 1, valid2 = 1, valid=1;
    int fanout_not_matched = 0, recon = 0;
    MapNode *mapNode, *mapNode_b;
    CellType mnCellType, mnbCellType;

    Queue<MapNode *> *foQ = new Queue<MapNode *>;
    Queue<MapNode *> *fobQ = new Queue<MapNode *>;

    #ifdef GCHROM_VALIDITY
    #endif
    if(mfpList->ListSize()>0){
        for_each(mfpList){
            num_fo_mfpNode = num_fo_mfpNode_b = 0;

            mfpNode = mfpList->Data(); // Nodes in sGraph (Graph)
            recon = utilCheckIfReconvergence(origNet, mfpNode);
            num_fo_inSub = mfpNode->GetNumFanout();
            mfpName = mfpNode->GetName();
            mfpName_b = gasGetInvertedSignalName(mfpName);

            // retrieve mfpNode from gChromosome and
            // get its mapped cell type(P or S)
            mapNode = gasGetMapNodeFromIndexForGChrom(subPList, popIndex,
mfpName, &mnbCellType);

            num_fo_mfpNode =
gasGetFanoutListFromGIndex(subPList, popIndex, mapNode, foQ);

            if(foQ->QLength()!=num_fo_inSub || foQ->QEmpty()){
                mapNode_b = gasGetMapNodeFromIndexForGChrom(subPList, popIndex,
mfpName_b, &mnbCellType, 0);

                num_fo_mfpNode_b =
gasGetFanoutListFromGIndex(subPList, popIndex, mapNode_b, fobQ);
            }

            num_fo_inMap = num_fo_mfpNode + num_fo_mfpNode_b;
            if(num_fo_mfpNode == 0 && num_fo_mfpNode_b!=0){
                if((num_fo_mfpNode_b-1) != num_fo_inSub && !recon){
                    fanout_not_matched = 1;
                }
            } else if(num_fo_mfpNode !=0 && num_fo_mfpNode_b==0){
                if(num_fo_mfpNode != num_fo_inSub && !recon){

```

```

        fanout_not_matched = 1;
    }else if(num_fo_mfpNode != 0 && num_fo_mfpNode_b != 0){
        if((num_fo_inMap - 1) != num_fo_inSub && !recon){
            fanout_not_matched = 1;
        }
    } else {
        fanout_not_matched = 1;
    }
    if(fanout_not_matched == 1){
#ifdef GCHROM_VALIDITY_DEBUG
        cout<<"Error: # of fanouts of mfp node "<<num_fo_mfpNode
            <<" are not match in Global Chromosome.\n";
        cout<<"num_fo_inSub = "<<num_fo_inSub<<"\n";
        cout<<"num_fo_mfpNode = "<<num_fo_mfpNode<<"\n";
        cout<<"num_fo_mfpNode_b = "<<num_fo_mfpNode_b<<"\n";
        cout<<"In "<<__FILE__<<" at line "<<__LINE__<<endl;
        cout<<"mapNode = "<<mapNode->GetName()<<"\n";
        cout<<"mapNode_b = "<<mapNode_b->GetName()<<"\n";
        cout<<" mapNode Fanouts.....\n";
        while(!foQ->IsEmpty()){
            tmpNode = foQ->Dequeue();
            cout<<tmpNode->GetName()<<" : ";
            tmpList = tmpNode->GetFanoutList();
            for_each(tmpList){
                tmpNode = tmpList->Data();
                cout<<tmpNode->GetName()<<" ";
            }
            cout<<endl;
        }
        cout<<" mapNode_B Fanouts.....\n";
        while(!fobQ->IsEmpty()){
            tmpNode = fobQ->Dequeue();
            cout<<tmpNode->GetName()<<" : ";
            tmpList = tmpNode->GetFanoutList();
            for_each(tmpList){
                tmpNode = tmpList->Data();
                cout<<tmpNode->GetName()<<" ";
            }
            cout<<endl;
        }
    }
#endif
    valid = 0;
    break;
}

// check if all the nodes in foutList has different
// mapped cell type
valid1=gasCheckValidityFanoutCellType(mnCellType, foQ,
    subPList, popIndex);
valid2=gasCheckValidityFanoutCellType(mnCellType, fobQ,
    subPList, popIndex);
}

delete [] mfpName_b;
while(!foQ->IsEmpty()){
    (void)foQ->Dequeue();
}
while(!fobQ->IsEmpty()){
    (void)fobQ->Dequeue();
}
delete foQ;
delete fobQ;
} // if(mfpList->Size()>0)
return valid;
}

// 11-03-2000, to calculate cost(area) accurately
// If any of the sub-chromosome's output in G1Chrom is not used in global
// chromosome delete the sub-chromosome from the List.
void gasRemoveUnusedSubChromosome(LinkedList<MapGraph*> *gChrom)
{
    MapGraph *tgtChrom, *subChrom;
    LinkedList<MapNode*> *poList, *piList, *doList;
    Queue<MapGraph*> grQ;
    Queue<MapNode*> nq;
    MapNode *curNode, *doNode;
    char *curName, *doName;
    int gchromLSize, found = 0, pos;

    gchromLSize = gChrom->ListSize();
    for(gChrom->Reset();!gChrom->EndOfList();gChrom->Next()){
        subChrom = gChrom->Data();
        grQ.Insert(subChrom);
    }

    while(!grQ.IsEmpty()){
        tgtChrom = grQ.Dequeue();
        poList = tgtChrom->GetPoList();
        found = 0;
        if(poList->ListSize()==1){
            curNode = poList->Data(0);
            if(curNode->GetNodeFunc() != NODE_PO){
                curName = curNode->GetName();

                // Check if "curName" is in any of the pi nodes. if yes,
                // found=1.
                for(gChrom->Reset();!gChrom->EndOfList();gChrom->Next()){
                    pos = -1;
                    subChrom = gChrom->Data();
                    piList = subChrom->GetPiList();
                    if((pos=subChrom->FindNode(piList, curName))!=-1){
                        found = 1;
                    }
                }
                // if found = 0;
                // for each graph find the graph whose output name is curNode,
                // and delete
            }
        }
    }
}

if(!found){
    for(gChrom->Reset();!gChrom->EndOfList();gChrom->Next()){
        subChrom = gChrom->Data();
        doList = subChrom->GetPoList();
        doNode = doList->Data(0);
        doName = doNode->GetName();
        if(strcmp(curName, doName)==0){
            (void)gChrom->DeleteCurrentElem();
            //(void)gChrom->DeleteCurrent();
            break;
        }
    }
}

} else {
    cout<<"Num po is one in "<<__FILE__<<" at line "<<__LINE__<<endl;
    exit(0);
}
}

// return and set the global depth of the PO node of a subChromosome
// 04-19-01
int gasGetGlobalDepthOfPoNode(MapGraph *subChrom, Graph *origNet, int index)
{
    LinkedList<MapNode*> *subPoList, *subPoInList = NULL;
    MapNode *subPoNode, *subPoInNode;
    char *subPoName, *subPoInName;

    LinkedList<Node*> *origGateList;
    Node *origNode;
    int max_glevel = 0, glevel, pos = -1;

    subPoList = subChrom->GetPoList();
    if(subPoList->ListSize() == 1){
        subPoNode = subPoList->Data(0);
        subPoName = subPoNode->GetName();
        subPoInList = subPoNode->GetFanInList();
    } else {
        printf("Error !!!\n");
        printf("sub-Chromosome[%d] has more than one PO nodes\n", index);
        printf("In %s at line %d\n", __FILE__, __LINE__);
        exit(0);
    }

    origGateList = origNet->GetGateList();
    for(subPoInList->Reset();!subPoInList->EndOfList();subPoInList->Next()){
        subPoInNode = subPoInList->Data(0);
        subPoInName = subPoInNode->GetName();
        pos = origNet->FindNode(origGateList, subPoInName);
        if(pos != -1){
            origNode = origGateList->Data(pos);
            glevel = origNode->GetGlevel();
        } else {
            printf("Error !!!\n");
            printf("PO node (%s) in sub-Chrom[%d] doesn't exist in origNet\n",
                subPoName, index);
            printf("In %s at line %d\n", __FILE__, __LINE__);
            exit(0);
        }
        max_glevel = Max(max_glevel, glevel);
    }
    subPoNode->SetGlevel(max_glevel+1);
    return (max_glevel+1);
}

#ifdef old0
int gasGetNewInsPos(LinkedList<MapGraph*> *gChrom, MapGraph *subChrom,
    int pos, int cur_depth)
{
    MapGraph *osubChrom;
    LinkedList<MapNode*> *osubPoList, *osubPiList;
    MapNode *osubPoNode, *piNode;
    char *piName;
    int osub_depth, new_pos;

    LinkedList<MapNode*> *csubPoList;
    MapNode *csubPoNode;
    char *csub_poName;
    int csub_po_name_found = 0;

    csubPoList = csubChrom->GetPoList();
    csubPoNode = csubPoList->Data(0);
    csub_poName = csubPoNode->GetName();

    // Backward search for finding cur_subPO name
    new_pos = pos;
    while(new_pos >= 0){
        osubChrom = gChrom->Data(new_pos);
        osubPoList = osubChrom->GetPoList();
        osubPoNode = osubPoList->Data(0);
        osub_depth = osubPoNode->GetGlevel();
        if(osub_depth == cur_depth){
            osubPiList = osubChrom->GetPiList();
            for_each(osubPiList){
                piNode = osubPiList->Data(0);
                piName = piNode->GetName();
                if(strcmp(csub_poName, piName)==0){
                    csub_po_name_found = 1;
                    break;
                }
            }
            if(csub_po_name_found == 1) break;
            new_pos--;
        } else {
            new_pos--;
        }
    }

    if(csub_po_name_found){
        if(new_pos >= 1){
            return new_pos-1;
        }
    }
}

```

```

    } else {
        return new_pos;
    }
} else {
    return pos;
}
}
#endif

int gasGetNewInsPos(LinkedList<MapGraph *> *gChrom, MapGraph *csubChrom,
    int pos, int cur_depth)
{
    MapGraph *osubChrom;
    LinkedList<MapNode *> *osubPoList, *osubPiList;
    MapNode *osubPoNode, *piNode;
    char *piName;
    int osub_depth, new_pos;

    LinkedList<MapNode *> *csubPoList;
    MapNode *csubPoNode;
    char *csub_poName;
    int csub_po_name_found = 0;
    int ins_pos, ins_pos_updated = 0;

    csubPoList = csubChrom->GetPoList();
    csubPoNode = csubPoList->Data(0);
    csub_poName = csubPoNode->GetName();

    // Backward search for finding cur_subPO name
    new_pos = pos;
    while(new_pos >= 0){
        osubChrom = gChrom->Data(new_pos);
        osubPoList = osubChrom->GetPoList();
        osubPoNode = osubPoList->Data(0);
        osub_depth = osubPoNode->GetGLevel();
        if(osub_depth == cur_depth){
            osubPiList = osubChrom->GetPiList();
            for_each(osubPiList){
                piNode = osubPiList->Data(0);
                piName = piNode->GetName();
                if(strcmp(csub_poName, piName)==0){
                    csub_po_name_found = 1;
                    break;
                }
            }
            if(csub_po_name_found == 1){
                ins_pos_updated++;
                ins_pos = new_pos;
                csub_po_name_found = 0;
            }
            new_pos--;
        } else if (osub_depth > cur_depth){
            new_pos--;
        } else {
            break;
        }
    }

    if(ins_pos_updated > 0){
        if(ins_pos >= 1){
            return ins_pos-1;
        } else {
            return ins_pos;
        }
    } else {
        return pos;
    }
}

// 04-19-01
void gasBuildSortedGChromosome(LinkedList<MapGraph *> *gChrom,
    MapGraph *subChrom, int cur_depth)
{
    MapGraph *oldSChrom;
    LinkedList<MapNode *> *oldSPoList;
    MapNode *oldSPoNode;
    int old_depth, pos = -1, new_pos=-1;

    if (gChrom->ListSize() == 0){
        gChrom->InsertRear(subChrom);
    } else if (gChrom->ListSize() > 1){
        pos = gChrom->ListSize()-1;
        /* Insert in ascending order */
        while(pos >= 0){
            oldSChrom = gChrom->Data(pos);
            oldSPoList = oldSChrom->GetPoList();
            oldSPoNode = oldSPoList->Data(0);
            old_depth = oldSPoNode->GetGLevel();
            if(old_depth > cur_depth){
                if(pos == 0){
                    gChrom->InsertFront(subChrom);
                    break;
                } else {
                    pos--;
                }
            } else if (old_depth < cur_depth){
                gChrom->InsertRight(subChrom);
                break;
            } else {
                new_pos = gasGetNewInsPos(gChrom, subChrom, pos, cur_depth);
                if(new_pos > -1){
                    gChrom->Reset(new_pos);
                    if(new_pos == 0){
                        gChrom->InsertFront(subChrom);
                    } else {
                        gChrom->InsertRight(subChrom);
                    }
                    break;
                } else {
                    printf("Error !!\nSubChromosome Insertion failed !!!\n");
                }
            }
        }
    }
}

printf("In %s at line %d\n", __FILE__, __LINE__);
printf("new_pos = %d, old_pos = %d\n", new_pos, pos);
exit(1);
}
}
}
}

// returns a global chromosome by randomly selecting a sub-chrom
// and if any of the sub-chromosome's output in G1Chrom is not used global
// chromosome delete the sub-chromosome from the List.
G1Chrom *gasGetRanGlobalChromosome(Graph *origNet,
    LinkedList<LinkedList<MapGraph *> *> *subPList,
    LinkedList<Node *> *mfplList, int wheel = 0)
{
    LinkedList<MapGraph *> *subPop;
    int sub_pop_size, sub_pos, n, k;
    int num_subg;
    int popIndex[num_subg+1];
    LinkedList<MapGraph *> *gChromosome;
    double rand, fitnessSum, revArea, partSum;
    MapGraph *subChromosome;
    int valid, gd, idx = -1;

    num_subg = subPList->ListSize();
    for(subPList->Reset(), n=0; !subPList->EndOfList(); subPList->Next(), n++){
        subPop = subPList->Data(0);
        sub_pop_size = subPop->ListSize();

        if(wheel!=0){ // roulette wheel
            fitnessSum = gasGetFitnessSum(subPop); // total area of the sub Pop
            rand = fitnessSum*gaGenRandomNumber(1.0); // Wheel point cal. [0,1]
            if(subPop->ListSize()==0){
                cout<<"In gasGetRanGlobalChromosome(), subPop("<<n<<")"
                <<"\n's size = 0, Check it ...!\n";
                exit(1);
            }
            partSum = 0.0;
            for(subPop->Reset(), k=0; !subPop->EndOfList(); subPop->Next(), k++){
                subChromosome = subPop->Data(0);
                revArea = (1.0/subChromosome->GetArea()); //fitness of indivi..
                partSum += revArea;
                if(partSum > rand){
                    popIndex[n] = k;
                    break;
                }
            }
        } else {
            sub_pos = (int)gaGenRandomNumber(sub_pop_size);
            popIndex[n] = sub_pos;
        }
    }

    valid = 1;
    #ifdef GCHROM_VALIDITY
    valid = gasCheckGlobalChromosomeValidity(origNet, subPList, mfplList, popIndex);
    #endif
    if(valid==1){
        G1Chrom *globalChrom = new G1Chrom;
        gChromosome = globalChrom->gChromosome;

        for(subPList->Reset(), n=0; !subPList->EndOfList(); subPList->Next(), n++){
            idx++;
            subPop = subPList->Data(0);
            subPop->Reset(popIndex[n]);
            subChromosome = subPop->Data(0);

            /* sort subChrom based on global depth of its out node in orgNet */
            gd = gasGetGlobalDepthOfPoNode(subChromosome, origNet, idx);
            (void)gasBuildSortedGChromosome(gChromosome, subChromosome, gd);
        }
        return globalChrom;
    } else {
        return NULL;
    }
}

// returns a global chromosome by selecting a minimum sub-chrom from each subpop
LinkedList<MapGraph *> *gasGetMinGlobalChromosome( \
    LinkedList<LinkedList<MapGraph *> *> *subPopList)
{
    LinkedList<MapGraph *> *subPop;
    int sub_pos, num_min_chrom;
    double min_cost;

    LinkedList<MapGraph *> *gChromosome = new LinkedList<MapGraph *>;
    MapGraph *subChromosome;

    for(subPopList->Reset(); !subPopList->EndOfList(); subPopList->Next()){
        subPop = subPopList->Data(0);
        min_cost = gaEvaluateGetMinAreaCost(subPop, &sub_pos, &num_min_chrom);
        subPop->Reset(sub_pos);
        subChromosome = subPop->Data(0);
        gChromosome->InsertRear(subChromosome);
    }
    return gChromosome;
}

void gasInitGlobalPopulation(Graph *origNet, LinkedList<G1Chrom *> *globalPop,
    LinkedList<LinkedList<MapGraph *> *> *subPopList,
    LinkedList<Node *> *mfplList,
    LinkedList<LibCell *> *cellLibrary,
    double gPopSize, double tReq)
{
    int index, valid, wheel = 1;
    G1Chrom *globalChrom;
    LinkedList<MapGraph *> *gChromosome;
    double dly;

```

```

index = 0;
cout << "\n ..... Initializing Global Population ..... \n";
while(1){
    valid = 0;
    globalChrom = gasGetRamGlobalChromosome(origNet, subPopList,
        mfpList, wheel);
    if(globalChrom != NULL){
        gChromosome = globalChrom->gChromosome;
        dly = gasEvaluateGlobalChromosome(origNet, globalChrom, cellLibrary);
        if(dly <= tReq && dly >= 0){
            #ifdef GLOBAL_CHROM_PRINT_DEBUG
                cout << "\n ===== Global Chromosome # [" << index << "] ===== \n";
                (void) gasPrintGlobalChromosome(gChromosome);
            #endif
            globalPop->InsertRear(globalChrom);
            index++;
        } else {
            globalChrom->area = 100000.0*(dly-tReq)*1e13;
            globalPop->InsertRear(globalChrom);
            index++;
            cout << "gPop # = " << index-1 << " is processed. \n";
        }
        if(index > (gPopSize-1)){
            break;
        }
    }
}

int gasGetGlobalMaxCostChrom(LinkedList<GChrom *> *globalPop, double *maxCost)
{
    int index = -1, max_pos = -1;
    double cost, max_cost = 0.0;
    GChrom *globalChrom;
    for(globalPop->Reset(); !globalPop->EndOfList(); globalPop->Next()){
        index++;
        globalChrom = globalPop->Data();
        cost = globalChrom->area;
        if(cost > max_cost){
            max_cost = cost;
            max_pos = index;
        }
    }
    *maxCost = max_cost;
    return max_pos;
}

void gasPrintGlobalPopPrintShort(LinkedList<GChrom *> *globalPop, double popS,
    char *with_pin_map=NULL)
{
    GChrom *globalChrom;
    int index = -1, min_pos = -1;
    double area, total_cost = 0.0, max_cost=0.0, min_cost=100000000;
    cout << "\n";
    for(globalPop->Reset(); !globalPop->EndOfList(); globalPop->Next()){
        index++;
        globalChrom = globalPop->Data();
        area = globalChrom->area;
        if(with_pin_map!=NULL && strcmp(with_pin_map, "yes")==0){
            (void)gasPrintGlobalChromosome(globalChrom->gChromosome);
        }
        max_cost = Max(max_cost, area);
        if(min_cost > area){
            min_cost = area;
            min_pos = index;
        }
    }
    total_cost += area;
    cout << "GChrom(" << index << ") :: Area = " << area
        << " Delay = " << globalChrom->cp_delay << "\n";
}
if(with_pin_map!=NULL && strcmp(with_pin_map, "yes")==0){
    cout << "\n*** Min cost global chromosome *** \n";
    globalChrom = globalPop->Data(min_pos);
    (void)gasPrintGlobalChromosome(globalChrom->gChromosome);
    area = globalChrom->area;
    cout << "GChrom(" << min_pos << ") :: Area = " << area
        << " Delay = " << globalChrom->cp_delay << "\n";
}
cout << "\n Max Cost(area) = " << max_cost << "\n";
cout << " Avg Cost(area) = " << (total_cost/popS) << "\n";
cout << " Min Cost(area) = " << min_cost << "\n";
}

/*****
 * Spice net.cc
 * This Pgm generates ELDO circuit netlist
 *****/

#include <fstream.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <new.h>
#include <time.h>

#include "eqn_stack.h"
#include "eqn_opr.h"
#include "Node.h"
#include "LinkedList.h"
#include "Graph.h"

#include "MapNode.h"
#include "MapGraph.h"

#include "Queue.h"
#include "Stack.h"

void spiceIncludeCells(int dual = 0)
{
    char *cell_dir;
    if(dual == 0){
        cell_dir = "-/MCNC/TSMC/STD_CELL_LIB/Sch/CellNet/";
    } else {
        cell_dir = "-/MCNC/TSMC/STD_CELL_LIB/DualRail/dCellNet/";
    }
    cout << "---- Including sub-circuits \n";
    if(dual==0){
        printf(".INCLUDE %s\n", cell_dir, "ao22.spi");
        printf(".INCLUDE %s\n", cell_dir, "aoi22.spi");
        printf(".INCLUDE %s\n", cell_dir, "buf1.spi");
        printf(".INCLUDE %s\n", cell_dir, "buf4.spi");
        printf(".INCLUDE %s\n", cell_dir, "mux2.spi");
        printf(".INCLUDE %s\n", cell_dir, "inv1.spi");
        printf(".INCLUDE %s\n", cell_dir, "nand2.spi");
        printf(".INCLUDE %s\n", cell_dir, "and2.spi");
        printf(".INCLUDE %s\n", cell_dir, "nand3.spi");
        printf(".INCLUDE %s\n", cell_dir, "and3.spi");
        printf(".INCLUDE %s\n", cell_dir, "nand4.spi");
        printf(".INCLUDE %s\n", cell_dir, "and4.spi");
        printf(".INCLUDE %s\n", cell_dir, "nor2.spi");
        printf(".INCLUDE %s\n", cell_dir, "or2.spi");
        printf(".INCLUDE %s\n", cell_dir, "nor3.spi");
        printf(".INCLUDE %s\n", cell_dir, "or3.spi");
        printf(".INCLUDE %s\n", cell_dir, "nor4.spi");
        printf(".INCLUDE %s\n", cell_dir, "or4.spi");
        printf(".INCLUDE %s\n", cell_dir, "xnor2.spi");
        printf(".INCLUDE %s\n", cell_dir, "xor2.spi");
        printf(".INCLUDE %s\n", cell_dir, "tptl_y1.spi");
        printf(".INCLUDE %s\n", cell_dir, "tptl_y21.spi");
        printf(".INCLUDE %s\n", cell_dir, "tptl_y22.spi");
        //
        printf(".INCLUDE %s\n", cell_dir, "tptl_y1_opt.spi");
        printf(".INCLUDE %s\n", cell_dir, "tptl_y21_opt.spi");
        printf(".INCLUDE %s\n", cell_dir, "tptl_y22_opt.spi");
        //
        printf(".INCLUDE %s\n", cell_dir, "tptl_y31.spi");
        printf(".INCLUDE %s\n", cell_dir, "tptl_y32.spi");
    } else {
        printf(".INCLUDE %s\n", cell_dir, "div1.spi");
        printf(".INCLUDE %s\n", cell_dir, "dnand2.spi");
        printf(".INCLUDE %s\n", cell_dir, "dnand3.spi");
        printf(".INCLUDE %s\n", cell_dir, "dnor2.spi");
        printf(".INCLUDE %s\n", cell_dir, "dnor3.spi");
        printf(".INCLUDE %s\n", cell_dir, "dptl_y1.spi");
        printf(".INCLUDE %s\n", cell_dir, "dptl_y21.spi");
        printf(".INCLUDE %s\n", cell_dir, "dptl_y22.spi");
        printf(".INCLUDE %s\n", cell_dir, "dptl_y31.spi");
        printf(".INCLUDE %s\n", cell_dir, "dptl_y32.spi");
    }
}

void spicePrintSignal(char *cur_net, int start, int end, int bar=0)
{
    int n;
    char *tab = " ";
    if(strcmp(cur_net, "VDD") != 0 && strcmp(cur_net, "GND") != 0){
        if(start > 0 && start == 1){
            printf("%c", '%');
        }
        for(n=start; n < end; n++){
            printf("%c", cur_net[n]);
        }
        if(bar > 0){
            printf("%c%s", 'b', tab);
        } else {
            printf("%s", tab);
        }
    } else if(strcmp(cur_net, "VDD") == 0){
        if(bar > 0){
            printf("%s%s", "GND", tab);
        } else {
            printf("%s%s", "VDD", tab);
        }
    } else if(strcmp(cur_net, "GND") == 0){
        if(bar > 0){
            printf("%s%s", "VDD", tab);
        } else {
            printf("%s%s", "GND", tab);
        }
    }
}

void spicePrintNodeName(char *cur_net, int dual = 0)
{
    int n, len_cur_net;
    len_cur_net = strlen(cur_net);
    if((cur_net[len_cur_net-1] != '\'') && (cur_net[0] != '$')){
        (void)spicePrintSignal(cur_net, 1, len_cur_net-1, 1);
        /* printing signal for dual-rail format */
        if(dual){
            (void)spicePrintSignal(cur_net, 1, len_cur_net-1, 0);
        }
    } else if((cur_net[len_cur_net-1] != '$') && (cur_net[0] != '$')){
        (void)spicePrintSignal(cur_net, 1, len_cur_net, 0);
        /* printing signal_bar for dual-rail format */
    }
}

```

```

        if(dual){
            (void)spicePrintSignal(cur_net, 1, len_cur_net, 1);
        }
    }else if((cur_net[len_cur_net - 1] == '\'') && (cur_net[0] != '$')){
        (void)spicePrintSignal(cur_net, 0, len_cur_net-1, 1);
        /* printing signal for dual-rail format */
        if(dual){
            (void)spicePrintSignal(cur_net, 0, len_cur_net-1, 0);
        }
    } else {
        (void)spicePrintSignal(cur_net, 0, len_cur_net, 0);
        /* printing signal_bar for dual-rail format */
        if(dual){
            (void)spicePrintSignal(cur_net, 0, len_cur_net, 1);
        }
    }
}

// Printing spice-format sub-ckt that begins with x...
void spicePrintEldoSubcircuit(MapNode *curNode, Match *curMatch, int dual=0)
{
    LinkedList<char*> *sortedPinL = new LinkedList<char*>;
    LinkedList<char*> *celPinList = new LinkedList<char*>;
    LinkedList<MapNode*> *foList;
    MapNode *foNode;
    LinkedList<pinMap*> *pMap;
    pinMap *curPin;
    char *cur_cell_pin, *prev_cell_pin = NULL, *cur_net;

    char *tmp_name;
    char *tab = " ";
    int pos = 0;

#ifdef SPICE_NET_DEBUG
    cout << " "<<curMatch->cell_name<<" ";
#endif

    // Sort PinList
    pMap = curMatch->pinmap;
    for(pMap->Reset(); !pMap->EndOfList(); pMap->Next()){
        curPin = pMap->Data();
        cur_cell_pin = curPin->cell_pin;
#ifdef SPICE_NET_DEBUG
        cout<<curPin->cell_pin<<(" "<<curPin->net_node<<" ");
#endif
        if(prev_cell_pin==NULL){
            celPinList->InsertRear(curPin->cell_pin);
            sortedPinL->InsertRear(curPin->net_node);
            prev_cell_pin = cur_cell_pin;
        } else {
            pos = celPinList->ListSize()-1;
            while(1){
                celPinList->Reset(pos);
                sortedPinL->Reset(pos);
                prev_cell_pin = celPinList->Data();
                if(strcmp(prev_cell_pin, cur_cell_pin) > 0){
                    if(pos==0){
                        celPinList->InsertFront(curPin->cell_pin);
                        sortedPinL->InsertFront(curPin->net_node);
                        break;
                    } else {
                        pos--;
                    }
                } else {
                    celPinList->InsertRight(curPin->cell_pin);
                    sortedPinL->InsertRight(curPin->net_node);
                    break;
                }
            }
        }
    }

#ifdef SPICE_NET_DEBUG
    cout << "\n";
#endif
    printf("x%d\t", ++sub_idx);
    for(sortedPinL->Reset(); !sortedPinL->EndOfList(); sortedPinL->Next()){
        cur_net = sortedPinL->Data();
        (void)spicePrintNodeName(cur_net, dual);

        // output node name of a cell
        foList = curNode->GetFanoutList();
        if(foList->ListSize()==1){
            foNode = foList->Data(0);
        } else {
            foNode = NULL;
        }
        if(foNode != NULL && foNode->GetNodeFunc()==NODE_PO){
            tmp_name = foNode->GetName();
        } else {
            tmp_name = curNode->GetName();
        }

        (void)spicePrintNodeName(tmp_name, dual); // output_name
        printf("VDD%s", tab);
        if(strcmp(curMatch->cell_name, "ptl_y1")==0){
            printf("%s", "ptl_y1");
        } else if(strcmp(curMatch->cell_name, "ptl_y2x1")==0){
            printf("%s", "ptl_y21");
        } else if(strcmp(curMatch->cell_name, "ptl_y2x2")==0){
            printf("%s", "ptl_y22");
        } else if(strcmp(curMatch->cell_name, "ptl_y3x1")==0){
            printf("%s", "ptl_y31");
        } else if(strcmp(curMatch->cell_name, "ptl_y3x2")==0){
            printf("%s", "ptl_y32");
        } else {
            printf("%s", curMatch->cell_name);
        }
    }

#ifdef SPICE_DELAY_DEBUG
    printf("\tr=%10.4g f=%10.4g",
        curNode->GetDR(), curNode->GetDF());
#endif
    printf("\n");
    delete sortedPinL;
    delete celPinList;
}

//031501
// This routine is applied only for dual-rail format to avoid errors like :
//Single rail format
//x63 %103 %105 GND %108b GND %106b VDD ptl_y21
//x64 %103b %103 VDD inv1
//x67 res_u0p0 OVFFLAGOp0 VDD DRIVE_ROM VDD %103b VDD ptl_y21
//
//Dual Rail
//x63 %103 %103b %105 %105b GND VDD %108b %108 GND VDD %106b %106 VDD ptl_y21
//x64 %103b %103 %103 %103b VDD inv1
//x67 res_u0p0 res_u0p0b OVFFLAGOp0 OVFFLAGOp0b VDD GND DRIVE_ROM
// + DRIVE_ROMb VDD GND %103b %103 VDD ptl_y21
//
// Mod results by CHANGING both (Match->pinMap->net_node) in foNode & cur_name.
//x63 %103x %103xb %105 %105b GND VDD %108b %108 GND VDD %106b %106 VDD ptl_y21
//x64 %103b %103 %103x %103xb VDD inv1
//
//mod : 120701
int spiceCheckModifyFoName_local(MapNode *curNode, char *sfile, int sline)
{
    LinkedList<MapNode*> *subGList, *foList, *fiList;
    MapNode *foNode;
    NodeFunc foFunc;
    char *cur_name, *fo_name, *net_name, *tmp_name;

    Match *foMatch;
    LinkedList<pinMap*> *pMap;
    pinMap *curPin;
    int name_len, found=0, ret_value = 0;

    string cur_str_name, tmp_str;

    cur_name = curNode->GetName();
    cur_str_name.erase();
    cur_str_name += cur_name;

    foList = curNode->GetFanoutList();
    if(foList->ListSize()!=1){
        printf("In %s at line %d\n", sfile, sline);
        printf("Inv Node (%s) doesn't have ", cur_name);
        printf("ONE fanout\n");
        printf("... Program is terminated ..\n");
        return(-1);
    }
    foNode = foList->Data(0);
    foFunc = foNode->GetNodeFunc();
    if(foFunc == NODE_PO){
        return(0);
    } else {
        foMatch = foNode->GetMatch();
        pMap = foMatch->pinmap;
        for(pMap->Reset(); !pMap->EndOfList(); pMap->Next()){
            curPin = pMap->Data();
            net_name = curPin->net_node;
            if(strcmp(cur_name, net_name)==0) {
                found = 1;
                break;
            }
        }
    }

    // Rename cur_node and net_name (120701)
    name_len = cur_str_name.size();
    if(found == 1 && (cur_str_name.substr(name_len-1,1) != "%") ||
        cur_str_name.substr(name_len-2,2) != "%;")){
        tmp_str.erase();
        cur_name = (char *)realloc(cur_name, name_len+2);
        net_name = (char *)realloc(net_name, name_len+2);

        if(cur_str_name.substr(name_len-1,1) == "'"){
            tmp_str = cur_str_name.substr(0, name_len-1) + "%;";
        }else{
            tmp_str = cur_str_name + "%;";
        }

        strcpy(cur_name, tmp_str.c_str());
        strcpy(net_name, tmp_str.c_str());

        ret_value = 1;
    }
    return ret_value;
}

// Check and Modify node name
void spiceCheckSubChrom(LinkedList<MapNode*> *subGList)
{
    LinkedList<MapNode*> *fiList;
    MapNode *curNode, *fiNode;
    NodeFunc curNode_func;
    int foerr;

    for(subGList->Reset(); !subGList->EndOfList(); subGList->Next()){
        curNode = subGList->Data();
        curNode_func = curNode->GetNodeFunc();

        if(curNode_func!=NODE_PO && curNode_func!=NODE_PI){
            if(curNode_func==NODE_INV){
                fiList = curNode->GetFaninList();
                fiNode = fiList->Data(0);
                if(fiNode->GetNodeFunc()!=NODE_PI){
                    foerr = spiceCheckModifyFoName_local(curNode,
                        __FILE__, __LINE__);
                }
            }
        }
    }
}

```

```

        if(foerr==1) exit(1);
    }
}
}

//01-19-01
int spiceGetFanoutNodesCrossChromosome(LinkedList<MapGraph *> *gChromosome,
MapNode *targetNode, Queue<MapNode *> *Q)
{
    MapGraph *subChrom;
    LinkedList<MapNode *> *gateList, *curInList;
    LinkedList<MapNode *> *subPolist;
    MapNode *curNode, *curInNode;
    NodeFunc curFunc;

    char *targetName, *curInName;
    int pos, used_sub_chrom, cur_level, max_depth = -1;

    targetName = targetNode->GetName();
    for(gChromosome->Reset();!gChromosome->EndOfList();gChromosome->Next()){
        subChrom = gChromosome->Data();
        if(!(used_sub_chrom = subChrom->GetUsed())) continue;
        subPolist = subChrom->GetPolist();
        pos = -1;
        /* if current sub-chromosomes output node is the same as target
        * name, the target node cannot be used as its(crrnt sub-chrom) inputs
        */
        if((pos=subChrom->FindNode(subPolist, targetName))!=-1) continue;
        gateList = subChrom->GetGateList();
        for(gateList->Reset();!gateList->EndOfList();gateList->Next()){
            curNode = gateList->Data();
            curFunc = curNode->GetNodeFunc();
            if(curFunc != NODE_PT){
                curInList = curNode->GetFaninList();
                for_each(curInList){
                    curInNode = curInList->Data();
                    curInName = curInNode->GetName();
                    if(strcmp(targetName, curInName)==0){
                        Q->Insert(curNode);
                        cur_level = curNode->GetLevel();
                        max_depth = Max(cur_level, max_depth);
                    }
                }
            }
        }
        return max_depth;
    }
}

// 110501
void spiceDualrailNodeNameChange(string& from_str_name, char *to_char_ptr)
{
    string tmp_str;
    int name_len;

    tmp_str.erase();
    name_len = from_str_name.size();
    if(from_str_name.substr(name_len-1,1)=="\"){
        tmp_str = from_str_name.substr(0,name_len-1)+"\X'";
    } else {
        tmp_str = from_str_name + "\X";
    }

    to_char_ptr = (char *)realloc(to_char_ptr, tmp_str.size()+1);
    strcpy(to_char_ptr,tmp_str.c_str());
}

// 110501
// Modify Node_name and pinMap (A' -> A'X'):
// 1) cur_node_name
// 2) foNode's pinMap
// 3) foNode's fanin_name (=cur_node_name)
// revised from 031501 which uses char * to modify node_name)
int spiceModifyCur_and_FoName_global(MapNode *curNode, Queue<MapNode *> *Q)
{
    MapNode *foNode, *foInNode;
    LinkedList<MapNode *> *foInList;
    NodeFunc foFunc;
    char *cur_name, *fo_in_name, *net_name, *tmp_name;

    Match *foMatch;
    LinkedList<pinMap *> *pMap;
    pinMap *curPin;
    int name_len,qlength, found=0, ret_value = 0;
    int po_cnt = 0;

    string cur_str_name, net_str_name;
    string fo_in_str_name;

    cur_name = curNode->GetName();
    cur_str_name += curNode->GetName();

    qlength = Q->QLength();
    while(!Q->IsEmpty()){
        foNode = Q->DeQueue();
        /* 1) foNode's fanin_name change */
        foInList = foNode->GetFaninList();
        for_each(foInList){
            foInNode = foInList->Data();
            fo_in_name = foInNode->GetName();

            fo_in_str_name.erase();
            fo_in_str_name += fo_in_name;
            if(fo_in_str_name.compare(cur_str_name)) continue;

            (void)spiceDualrailNodeNameChange(fo_in_str_name, fo_in_name);
        }
    }

    //Since NODE_PO doesn't have mapch(08-12-01).
    if(foNode->GetNodeFunc()==NODE_PO){
        po_cnt++;
        continue;
    }

    /* 2) find cur_name in foNode's pinMap */
    foMatch = foNode->GetMatch();
    pMap = foMatch->pinmap;
    for(pMap->Reset();!pMap->EndOfList();pMap->Next()){
        curPin = pMap->Data();
        net_name = curPin->net_node;

        net_str_name.erase();
        net_str_name += curPin->net_node;
        if(cur_str_name.compare(net_str_name)==0){
            found++;
            break;
        }
    }

    /* 3) foNode's pinMap change */
    name_len = net_str_name.size();
    if(found != 0 && net_str_name.substr(name_len-1)!="\X"){
        (void)spiceDualrailNodeNameChange(net_str_name, net_name);
        ret_value = 1;
    }
}

/* 4) curNode name change */
if((found + po_cnt) == qlength){
    (void)spiceDualrailNodeNameChange(cur_str_name, cur_name);
} else {
    cout<<"Error : curNode's fanout does not match with Q.\n";
    cout<<"cur_name = "<<cur_name<<"\n";
    cout<<"found = "<<found<<" po_cnt = "
    <<po_cnt<<" qlength = "<<qlength<<"\n";
    cout<<"in "<<__FILE__<<" at line "<<__LINE__<<".\n";
    exit(0);
}

return ret_value;
}

// 01-07-02
// Check if curNode's fanin is in orgPiList, if yes, return TRUE
int spiceCheckFaninIsInPiList(MapNode *curNode,LinkedList<Node *> *orgPiList)
{
    LinkedList<MapNode *> *finList;
    MapNode *finNode;
    Node *opiNode;
    int name_match = 0;
    string fin_name, org_pi_name;

    finList = curNode->GetFaninList();
    if(finList->ListSize() == 1){
        finNode = finList->Data();
        fin_name.erase();
        fin_name += finNode->GetName();
    } else {
        cout << "Error : at "<<__LINE__<<" in "<<__FILE__<<"\n";
        cout << "INV node has more than one finain nodes.\n";
        exit(0);
    }

    for(orgPiList->Reset();!orgPiList->EndOfList();orgPiList->Next()){
        opiNode = orgPiList->Data();
        org_pi_name.erase();
        org_pi_name += opiNode->GetName();
        if(fin_name.compare(org_pi_name)==0){
            name_match++;
        }
    }

    if(name_match > 1){
        cout << "Error : at "<<__LINE__<<" in "<<__FILE__<<"\n";
        cout << "Name match occurred more than once.\n";
        exit(0);
    }

    return name_match;
}

void spiceInvNameModifyInGlobalChrom(LinkedList<MapGraph *> *gChromosome,
LinkedList<Node *> *orgPiList)
{
    MapGraph *subChrom;
    LinkedList<MapNode *> *gateList, *curInList;
    LinkedList<MapNode *> *polist;
    MapNode *poNode;
    NodeFunc poFunc;
    char *poName;
    int index, gChromSize, qlength;
    int used_sub_chrom, in_pi_list;

    Queue<MapNode *> Q;

    gChromSize = gChromosome->ListSize();
    for(index = 0; index <gChromSize; index++){
        subChrom = gChromosome->Data(index);
        if(!(used_sub_chrom = subChrom->GetUsed())) continue;

        in_pi_list = 0;
        polist = subChrom->GetPolist();
        if(polist->ListSize() == 1){
            poNode = polist->Data(0);
            poName = poNode->GetName();
        } else {
            cout<<"Mapped Subgrap(final solution) ["<<index<<"] doesn't have"
            <<" only ONE primary output node.\n";
            cout<<"in "<<__FILE__<<" at line "<<__LINE__<<".\n";
            exit(0);
        }
    }
}

```

```

    poFunc = poNode->GetNodeFunc();
    if(poFunc == NODE_INV) {
        /* 01-07-2002
        * Check if fanin of the poNode is in primary input list of
        * unMapped graph which is original graph.
        * if yes, disable current subChrom by setting used-flag to "0"
        * use special sub-routine because the nodes(Node) in pilist have
        * different from poNode(MapNode)
        */
        in_pilist = spiceCheckFaninIsInPilist(poNode,orgPiList);
        if(in_pilist){
            // do not modify by "spiceModifyCur_and_FoName_global"
            continue;
        }

        qLength = 0;
        (void)spiceGetFanoutNodesCrossChromosome(gChromosome, poNode, &q);
        qLength = q.QLength();
        if(qLength >= 6){ //need to match {}
            (void)spiceModifyCur_and_FoName_global(poNode, &q);
        } else {
            while(!q.QEmpty()){
                (void)q.DeQueue();
            }
        }
    }
}

// Print pin map of a node with library cells
void spiceSubChromosome2EldoNet(MapGraph *subChromosome, int dual =0)
{
    LinkedList<MapNode*> *subGList, *foList, *fiList;
    MapNode *curNode, *foNode, *fiNode;
    Match *curMatch;
    NodeFunc curnode_func;
    char *cur_name, *fi_name;
    int foerr, cur_len, fi_len;

    subGList = subChromosome->GetGateList();
    if(dual){
        (void)spiceCheckSubChrom(subGList); //0315
    }

    for(subGList->Reset(); !subGList->EndOfList(); subGList->Next()){
        curNode = subGList->Data();
        curnode_func = curNode->GetNodeFunc();

        if(curnode_func!=NODE_PO && curnode_func!=NODE_PI){
            foList = curNode->GetFanoutList();
            if(foList->ListSize() > 0){
                for_each(foList){
                    foNode = foList->Data();
                    #ifdef SPICE_NET_DEBUG
                        cout<<"---: ("<<foNode->GetName()<<") ";
                    #endif
                }
            } else {
                #ifdef SPICE_NET_DEBUG
                    cout<<"---: ("<<") ";
                #endif
            }
            cur_name = curNode->GetName();
            cur_len = strlen(cur_name);
            #ifdef SPICE_NET_DEBUG
                cout<<" "<<curNode->GetName();
            #endif
            curMatch = curNode->GetMatch();

            /* Filtering inverters whose input is NODE_PI for dual-rail */
            if(curnode_func==NODE_INV){
                fiList = curNode->GetFaninList();
                fiNode = fiList->Data(0);
                fi_name = fiNode->GetName();
                fi_len = strlen(fi_name);
                if(fiNode->GetNodeFunc() == NODE_PI){
                    (void)spicePrintEldoSubcircuit(curNode, curMatch, dual);
                } else {
                    (void)spicePrintEldoSubcircuit(curNode, curMatch, dual);
                }
            } else {
                /* End of Filtering for dual-rail */
                (void)spicePrintEldoSubcircuit(curNode, curMatch, dual);
            }
        }
    }
}

int spiceFindMinCostGlobalChromosome(LinkedList<G1Chrom*> *globalPop,
double popS)
{
    G1Chrom *globalChrom;
    int index = -1, min_pos = -1;
    double area, total_cost = 0.0, max_cost=0.0, min_cost=100000000;

    for(globalPop->Reset(); !globalPop->EndOfList(); globalPop->Next()){
        index++;
        globalChrom = globalPop->Data();
        area = globalChrom->area;
        max_cost = Max(max_cost, area);

        if(min_cost > area){
            min_cost = area;
            min_pos = index;
        }

        total_cost += area;
    }
    cout<<"* Max Cost(area) = "<<max_cost<<"\n";
    cout<<"* Avg Cost(area) = "<<(total_cost/popS)<<"\n";
    cout<<"* Min Cost(area) = "<<min_cost<<"\n";

    return min_pos;
}

int spiceFindMinDelayGlobalChromosome(LinkedList<G1Chrom*> *globalPop,
double popS)
{
    G1Chrom *globalChrom;
    int index = -1, min_pos = -1;
    double delay, total_delay = 0.0, max_delay = 0.0, min_delay=100000000;
    double area;

    for(globalPop->Reset(); !globalPop->EndOfList(); globalPop->Next()){
        index++;
        globalChrom = globalPop->Data();
        delay = globalChrom->cp_delay;
        max_delay = Max(max_delay, delay);

        if(delay < min_delay){
            min_delay = delay;
            area = globalChrom->area;
            min_pos = index;
        }

        total_delay += delay;
    }
    cout<<"* Max Cost(delay) = "<<max_delay<<"\n";
    cout<<"* Avg Cost(delay) = "<<(total_delay/popS)<<"\n";
    cout<<"* Min Cost(delay) = "<<min_delay<<" Area = "<<area<<"\n";

    return min_pos;
}

//-- 01-19-01
void spiceFanoutOptMappedCellPinNameChange(MapNode *curNode, char *targetName,
char *l2_buf_name)
{
    Match *curMatch;
    LinkedList<pinMap*> *pinList;
    pinMap *aPin;
    char *net_name;

    string net_str_name, target_str, l2_buf_str;
    int new_len;

    curMatch = curNode->GetMatch();
    target_str.erase(); l2_buf_str.erase();
    target_str += targetName;
    l2_buf_str += l2_buf_name;
    new_len = l2_buf_str.size();

    pinList = curMatch->pinmap;
    for(pinList->Reset(); !pinList->EndOfList(); pinList->Next()){
        aPin = pinList->Data();
        net_name = aPin->net_name;

        net_str_name.erase();
        net_str_name += net_name;
        if(!net_str_name.compare(target_str)){
            net_name = (char *)realloc(net_name, new_len+1);
            strcpy(net_name, l2_buf_str.c_str());
            break;
        }
    }
}

void spiceHighFanoutRedistribution(LinkedList<MapGraph*> *gChrom,
MapNode *targetNode, Queue<MapNode*> *Q,
Queue<char*> *bufListQ, int nFANOUT, int max_depth=-1)
{
    MapGraph *subChrom;
    LinkedList<MapNode*> *gateList, *curInList;
    LinkedList<MapNode*> *subPolist;
    MapNode *curNode, *tfoNode, *outNode;
    NodeFunc curFunc, outFunc;
    double NUM_F0_PER_BUF;

    char *targetName, *curName, *tfoName;
    char *l1_buf_name, *l2_buf_name;
    double fnum_buf = 0.0;

    int pos, num_fanout, inum_buf=0, inum_buf_res=0, n;
    int dis_buf_count, array_index, array_value, index;
    int cur_depth;

    NUM_F0_PER_BUF = (double)nFANOUT;
    targetName = targetNode->GetName();
    num_fanout = Q->QLength();

    fnum_buf = num_fanout/NUM_F0_PER_BUF;
    inum_buf = (int)fnum_buf;
    inum_buf_res = (int)((fnum_buf - (double)inum_buf)*NUM_F0_PER_BUF + 0.1);

    cout<<"*inum_buf = "<<inum_buf<<" res = "<<inum_buf_res<<endl;
    dis_buf_count = inum_buf*(int)NUM_F0_PER_BUF+inum_buf_res;
    if(dis_buf_count != num_fanout){
        cout<<"Distributed buffer count does not match with num_fanout.\n";
        cout<<"Distributed buffer count = "<<dis_buf_count<<endl;
        cout<<"Num Fanout = "<<num_fanout<<endl;
    }

    int *numFanout = new int[inum_buf];
    for(n=0; n<inum_buf; n++){ // initialize num_buf
        numFanout[n] = (int)NUM_F0_PER_BUF;
    }

    // distribute remainder (if inum_buf_res < inum_buf) -- 102601
    for(n=0; n<inum_buf_res; n++){
        // get an index of numFanout using random number (max=inum_buf)
        pos = (int)gaGenRandomNumber((double)inum_buf);
        numFanout[pos] = numFanout[pos]+1;
    }

    /* this if-statement may not needed */
    if(num_fanout == 7 || num_fanout == 8 || num_fanout == 9){
        delete [] numFanout;
    }
}

```

```

int *numFanout = new int[2];
if(num_fanout == 8){
    numFanout[0] = 4;
    numFanout[1] = 4;
}else if(num_fanout == 7){
    numFanout[0] = 4;
    numFanout[1] = 3;
}else {
    numFanout[0] = 4;
    numFanout[1] = 5;
}
}

// create a master buffer
bufListQ->QInsert(targetName);
l1_buf_name = (char *)calloc(strlen(targetName)+2, sizeof(char));
if(targetName[strlen(targetName)-1]!='\0'){
    strcpy(l1_buf_name, targetName, strlen(targetName)-1);
    strcat(l1_buf_name, "b");
}else {
    strcpy(l1_buf_name, targetName);
}
strcat(l1_buf_name, "p");
bufListQ->QInsert(l1_buf_name);

array_index = 0;
array_value = numFanout[array_index];

// initialize l2_buf_name
l2_buf_name = (char *)calloc(strlen(l1_buf_name)+2, sizeof(char));
strcpy(l2_buf_name, l1_buf_name);
sprintf(l2_buf_name, "%s/d", l2_buf_name, array_index);
bufListQ->QInsert(l1_buf_name);
bufListQ->QInsert(l2_buf_name);

index = -1;
/* Here, Q contains the fanouts of targetNode */
while(!Q.Empty()){
    tfoNode = Q->Dequeue();
    if((cur_depth == tfoNode->GetLevel())==max_depth) continue;

    tfoName = tfoNode->GetName();
    cout<<"tfoName = "<<tfoName<<"\n";
    cout<<"array_value = "<<array_value<<" ";
    for(gChrom->Reset(); !gChrom->EndOfList(); gChrom->Next()){
        subChrom = gChrom->Data();
        subPolist = subChrom->GetPolist();
        pos = -1;

        /* if a subChrom's output name = tfoName(target's fanout), skip */
        if((pos=subChrom->FindNode(subPolist, tfoName))!=-1){
            outNode = subPolist->Data(pos);
            if((outFunc = outNode->GetNodeFunc())==NODE_PD) continue;
            gateList = subChrom->GetGateList();
            /* if tfoName is in subChrom's gateList, more specifically,
            * it's in subChrom's pList.
            */
            if((pos=subChrom->FindNode(gateList, tfoName))!=-1){
                curNode = gateList->Data(pos);
                curName = curNode->GetName();
                if((curFunc=curNode->GetNodeFunc())==NODE_PI) continue;
                // fanin name change ??

                // Global level of mapped node
                printf("curNode = %s (glevel = %d) (llevel = %d)\n",
                    curName, curNode->GetGlevel(), curNode->GetLevel());

                curInList = curNode->GetFaninList();
                if((pos=subChrom->FindNode(curInList, targetName))!=-1){
                    index++;
                    cout<<"Index = "<<index<<endl;
                    (void)spiceFanoutOptMappedCellPinNameChange(curNode,
                        targetName, l2_buf_name);
                }
                if((array_value-1)==0){
                    array_index++;
                    if(array_index >= inum_buf) break;
                    cout<<"array_INDEX = "<<array_index<<endl;
                    array_value = numFanout[array_index];
                    l2_buf_name = (char *)calloc(strlen(l1_buf_name)+2, sizeof(char));
                    strcpy(l2_buf_name, l1_buf_name);
                    sprintf(l2_buf_name, "%s/d", l2_buf_name, array_index);
                    bufListQ->QInsert(l1_buf_name);
                    bufListQ->QInsert(l2_buf_name);
                } else {
                    array_value--;
                }
            } // if tfoName id found
        }
    }
}

Queue<char *>
*spiceGlobalChromFanoutRedistribution(LinkedList<MapGraph *> *gChromosome,
    Graph *origNet, int nFANOUT)
{
    MapGraph *subChrom;
    LinkedList<MapNode *> *polist;
    Queue<MapNode *> Q;
    MapNode *sub_o_node;
    char *sub_out_name;
    NodeFunc outFunc;
    int index, gChromSize, qLength;
    int maxDepth, used_sub_chrom = 0;

    Queue<char *> *bufListQ = new Queue<char *>; // return value

    // 10-28-01
    (void)dlySetGlobalDepthOfGlobalChromosome(gChromosome, origNet);
    (void)dlyGlobalNodesDepthFromP0(gChromosome, origNet);

    gChromSize = gChromosome->ListSize();
    for(index = 0; index < gChromSize; index++){
        subChrom = gChromosome->Data(index);
        used_sub_chrom = subChrom->GetUsed();
        if(!used_sub_chrom){
            continue;
        }

        polist = subChrom->GetPolist();
        if(polist->ListSize() == 1){
            sub_o_node = polist->Data(0);
            sub_out_name = sub_o_node->GetName();
        } else {
            cout<<"Mapped Subgrap(final solution) ["<<index<<"] doesn't have"
                <<" only ONE primary output node.\n";
            cout<<"in "<<__FILE__<<" at line "<<__LINE__<<"\n";
            exit(0);
        }

        // Find the number of fanouts of sub_o_node (nodeType of
        // sub_o_node is either NODE_PD or NODE_PTL1,2,3/NODE_INV...)
        outFunc = sub_o_node->GetNodeFunc();
        qLength = 0;

        maxDepth=spiceGetFanoutNodesCrossChromosome(gChromosome, sub_o_node, &Q);
        qLength = Q.QLength();

        if(qLength == nFANOUT && outFunc != NODE_INV){
            cout<<"xcurNode = "<<sub_out_name<<" : qLength = "<<qLength<<endl;
            cout << " *MaxDepth = "<<maxDepth<<"\n";
            /* After this routine, network connectivity is broken because
            * the fanin name of the node in Q is changed, but after generating
            * Spice net the network connectivity is OK.
            */
            (void)spiceHighFanoutRedistribution(gChromosome, sub_o_node, &Q,
                bufListQ, nFANOUT, maxDepth);
            while(!Q.Empty()){ // Clear Queue
                (void)Q.Dequeue();
            }
        } else {
            while(!Q.Empty()){ // Clear Queue
                (void)Q.Dequeue();
            }
        }
    }
    return bufListQ;
}

void spiceBufferList2EldoNet(Queue<char *> *bufListQ, int dual = 0)
{
    int buf_index = -1, name_len;
    char *sig_in, *sig_out, *tmp_name;

    cout<<"\n* Printing Buffers for fanout re-distribution\n";
    while(!bufListQ->QEmpty()){
        buf_index++;
        sig_in = bufListQ->Dequeue();
        sig_out = bufListQ->Dequeue();
        name_len = strlen(sig_in);
        if(sig_in[name_len-1]!='\0'){
            tmp_name = (char *)calloc(name_len, sizeof(char));
            strcpy(tmp_name, sig_in);
            tmp_name[name_len-1] = '\0';
            printf("x%d\t%s", buf_index, tmp_name);
            printf("%s ", "b");
            if(dual){
                printf("%s ", tmp_name);
            }
            printf("%s ", sig_out);
            if(dual){
                printf("%s", sig_out);
                printf("%s ", "b");
            }
            printf("VDD invl\n");
            free(tmp_name);
        } else {
            printf("x%d\t%s ", buf_index, sig_in);
            if(dual){
                printf("%s", sig_in);
                printf("%s ", "b");
            }
            printf("%s ", sig_out);
            if(dual){
                printf("%s", sig_out);
                printf("%s ", "b");
            }
            printf("VDD invl\n");
        }
    }
}

void spiceGenerateSpiceNet(LinkedList<G1Chrom *> *globalPop, double popS,
    Graph *origNet, LinkedList<LibCell *> *cellLibrary,
    int nFout = 0, int dual = 0, int min_delay=0)
{
    G1Chrom *globalChrom;
    LinkedList<MapGraph *> *gChromosome;
    LinkedList<MapNode *> *subPolist, *subPIList;
    MapNode *subP0Node;
    NodeFunc subP0Func;
    MapGraph *subChrom;
    int g_pos, index = -1, in_pilist, qLength;

    char *network_name;
    LinkedList<Node *> *piList, *polist;
    Node *tmp_Node;
    char *tmp_name;

```

```

Queue<char *> *bufListQ;
Queue<MapNode *> Q;

network_name = origNet->GetName();
piList = origNet->GetPiList();
poList = origNet->GetPoList();

if(min_delay == 1){
    printf("Min delay Global Chromosome.\n");
}
printf("\n\n* Eldo net list for %s\n", network_name);
printf("\n*.LIB "geunc/MCNC/TSMC/Eldo_Par/log018.eldo53 TT\n");
printf("\n*.LIB "geunc/MCNC/TSMC/Eldo_Par/mix025_1.eldo53 TT\n");
(void)spiceIncludeCells(dual);

// Primary Inputs
//cout<<"\n* Primary Inputs : ";
cout<<"\n* i ";
for(piList->Reset();!piList->EndOfList();piList->Next()){
    printf("%s ", piList->Data()->GetName());
}
cout<<"\n";

// Primary Outputs
cout<<"* o ";
for(poList->Reset();!poList->EndOfList();poList->Next()){
    printf("%s ", poList->Data()->GetName());
}
cout<<"\n\n";

if(min_delay == 0){
    g_pos = spiceFindMinCostGlobalChromosome(globalPop, popS);
    globalChrom = globalPop->Data(g_pos);
    gChromosome = globalChrom->gChromosome;
} else {
    g_pos = spiceFindMinDelayGlobalChromosome(globalPop, popS);
    globalChrom = globalPop->Data(g_pos);
    gChromosome = globalChrom->gChromosome;
    (void)gasEvaluateGlobalChromosomeDelay(origNet, globalChrom, cellLibrary);
}

(void)gasSetSubChromUsedFlagInGChrom(gChromosome); //check used sub chrom
if(dual){
    (void)spiceInvNameModifyInGlobalChrom(gChromosome, piList); //010702
}

if(nFout > 4) {
    bufListQ=spiceGlobalChromFanoutReDistribution(gChromosome, origNet, nFout);
}

for(gChromosome->Reset();!gChromosome->EndOfList();gChromosome->Next()){
    index++;
    subChrom = gChromosome->Data();
    //cout<<"\n*-- SubChromosome # ["<index<<"\n";

    /* 01-07-02 */
    subPoList = subChrom->GetPoList();
    subPoNode = subPoList->Data(0);
    subPoFunc = subPoNode->GetNodeFunc();

    subPiList = subChrom->GetPiList();
    in_piList = 0;

    qLength = 0;
    if(subPoFunc == NODE_INV){
        in_piList = spiceCheckFaninInPiList(subPoNode, piList);
        // check faout in global-chromosome, if < 7 do not print.
        (void)spiceGetFanoutNodesCrossChromosome(gChromosome, subPoNode, &q);
        qLength = q.Qlength();
        //clear q
        while(!q.QEmpty()){ (void)q.DeQueue(); }
    }
    /* 01-07-08-02 */
    if((dual==1 && (subChrom->GetUsed()==0 || in_piList==1 ||
        ((qLength < 6) && (subPoFunc == NODE_INV)))) ||
        (dual==0 && (subChrom->GetUsed()==0))){ //01-08-02()
    } else {
        cout<<"\n*-- SubChromosome # ["<index<<"\n";
        (void)spiceSubChromosome2EldoNet(subChrom, dual);
    }
    gChromosome->Reset(index); //get back to the orig pos
}

if(nFout > 4) {
    (void)spiceBufferList2EldoNet(bufListQ, dual);
    delete bufListQ;
}

cout<<"\n*-- Output capacitances\n";
int out_idx = 0;
for(poList->Reset();!poList->EndOfList();poList->Next()){
    printf("c%d\t%s\tGND\t20f\n", out_idx++, poList->Data()->GetName());
    if(dual){
        printf("c%d\t%sb\tGND\t20f\n", out_idx++, poList->Data()->GetName());
    }
}
cout<<"\n";

cout<<"*-- Electrical source definition(s)\n";
cout<<"v0\tGND\t0\tDC\t0\n";
cout<<"vdd\tVDD\tGND\tDC\t2.5v\n";

cout<<"\n*-- Inputs \n";
index = 0;
for(piList->Reset();!piList->EndOfList();piList->Next()){
    tmp_Node = piList->Data();
    tmp_name = tmp_Node->GetName();

    printf("v%d\t%s\tGND\tpulse(", index++, tmp_name);
    printf("0 2.5 0 100p 100p 5n 13n\n");
}
cout<<"\n";

cout<<"*-- Simulation options and definitions\n";
printf("temp 25\n");
printf("tran 0.3n 13n\n");
printf("plot tran ");
int res = 0;
index = 12;
for(piList->Reset();!piList->EndOfList();piList->Next()){
    tmp_Node = piList->Data();
    tmp_name = tmp_Node->GetName();
    printf("v(%s)", tmp_name);
    index = index+4+strlen(tmp_name);
    res = index%30;
    if(res>73 && res<=79 || res==0){
        cout<<"\n* ";
    }
}
//cout<<"\n* ";
for(poList->Reset();!poList->EndOfList();poList->Next()){
    tmp_Node = poList->Data();
    tmp_name = tmp_Node->GetName();
    printf("v(%s)", tmp_name);
    index = index+4+strlen(tmp_name);
    res = index%30;
    if(res>73 && res<=79 || res==0){
        cout<<"\n* ";
    }
}
cout<<"\n";
cout<<"END\n\n";

/*****
* main.cc
*****/
#include <fstream.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <new.h>
#include <time.h>

#include "eqn_stack.h"
#include "eqn_opr.h"
#include "Node.h"
#include "LinkedList.h"
#include "Graph.h"

#include "MapNode.h"
#include "MapGraph.h"
#include "Utils.h"

#include "Queue.h"
#include "Stack.h"
#include "util.h"

#define ADD_INVERTER
#ifdef ADD_INVERTER
    #define ONLY_NOR_NAND_INV
    #define DEPTH_FOR_FUNC 7
#else
    #define DEPTH_FOR_FUNC 4
#endif

#define MAKE_BALANCED_GRAPH
#define MATCH_BASED_ON_DEPTH
#define OUTPUT_LOAD_CAP 3

#define NUM_GLOBAL_CHROMOSOME 10
#define OUTPUT_LOAD_CAP 3
#define NUM_GLOBAL_CHROMOSOME 10
#define CHR_RANDOM 0
#define READ_EQN_DEBUG
#define BAL_GRAPH_DEBUG

#define MATCH_PIN_DEBUG

int sub_idx = -1;

#include "Read_eqn.cc"
#include "Subgraph_Bdd_new.cc"
#include "Read_lib_new.cc"
#include "Permute.cc"
#include "Match_depth.cc"
#include "Delay_cal_new.cc"
#include "Gcover_sub_dPTL.cc"
#include "Spice_net.cc"

#define WITH_ARG

int main(int argc, char *argv[])
{
    char *lib_name = "/home/cho/Research/Work.dly/CellLib/csu_static_ptl.lib";

    int dual = 0, subPopSize = 5;
    int max_gen_count=10, gPopSize = 50, nFanout=5;
    double m_Fate = 0.35, delta = 0.25, REQ_TIME=30;
    int fDepth = DEPTH_FOR_FUNC;

#ifdef WITH_ARG
    char *file_name;
    if(argc < 9){
        cout<<"Error : the # of arguments must be 9 or 10.\n";
        cout<<"Use : pgm .slif fDepth pop_size"
            <<" num_gen mu_rate delta treq nFanout [dual].\n";
        exit(1);
    } else if(argc == 10 || argc == 11){

```

```

file_name = argv[1];
fDepth = atoi(argv[2]);
gPopSize = atoi(argv[3]);
max_gen_count = atoi(argv[4]);
m_rate = atof(argv[5]);
delta = atof(argv[6]);
REQ_TIME = atof(argv[7]);
nFanout = atoi(argv[8]);
dual = atoi(argv[9]);
}
if(argc == 11){
    subPopSize = atoi(argv[10]);
}
#endif

LStack<Node> *nodeStack;
Node *targetNode;
DdNode *f = new DdNode;
DdNode *g = new DdNode;

Graph *and2orNetwork = new Graph, *sGraph;
LinkedList<LibCell> *cellLibrary;
LinkedList<Graph> *subGList;

int index, gen_count=0;
time_t match_start_TIME, match_end_TIME;
double total_time, itr_start_time, itr_end_time, start_time, end_time;
time_t start_TIME, end_TIME, itr_start_TIME, itr_end_TIME;
double r_val = 0.0;

// Read logic network described in boolean equation, and convert to into
// a graph
and2orNetwork = Read_eqn(file_name);

// read cell library
cellLibrary = libReadLibrary(lib_name);

// Get subject-graph lists of a graph
subGList = GetSubjectGraphs(and2orNetwork);
match_start_TIME = time(NULL);

// Matching Stage
cout<<"\n----- Num sub-graph = "<< subGList->ListSize()<<"\n";
for(subGList->Reset(); !subGList->EndOfList(); subGList->Next()){
    sGraph = subGList->Data();
    (void) subGInitLevel(sGraph);
    cout<<"GRAPH :: "<< sGraph->GetName() <<"\n";

    //candidate node for match
    nodeStack = matchGetGateListOfSubGraph(sGraph);
    (void) fflush(stdout);
    while(!nodeStack->Empty()){
        targetNode = nodeStack->Pop();
        if(strcmp(targetNode->GetName(), "$51") == 0){
            cout << " $26 \n";
        }
        (void) matchDoBooleanMatch(and2orNetwork, sGraph, targetNode,
                                   cellLibrary, f, g, fDepth);
    }
    delete nodeStack;
}

match_end_TIME = time(NULL);
printf("MATCH_Elapsed_TIME(sec) = %f\n",
       difftime(time(NULL), match_start_TIME));

// for Genetic Covering
LinkedList<MapGraph> *subGPop;
LinkedList<LinkedList<MapGraph> *subGList = \
    new LinkedList<LinkedList<MapGraph> * >;

LinkedList<G1Chrom> *globalPop = new LinkedList<G1Chrom> *;

// Covering using Genetic Algorithm
LinkedList<Node> *mfplList = new LinkedList<Node> *;
(void) gasGetMfpNodeList(and2orNetwork, mfplList);
cout << "\n ..... Printing MFP nodes ..... \n";
(void) and2orNetwork->PrintNodesOfList(mfplList);

start_time = seconds();
start_TIME = time(NULL);
printf("Start time = %f\n", start_time);

// Generate sub-population
LinkedList<Node> *subOutList = new LinkedList<Node> *;
(void) gaEnumerateSubOutNodes(subGList, subOutList); //04-26-01
index = -1;
for(subGList->Reset(); !subGList->EndOfList(); subGList->Next()){
    index++;
    sGraph = subGList->Data();
    (void) subGInitLevel(sGraph);

    // Get initial population (genome is a tree [graph])
    cout << "\n ----- Sub-graph Population :: graph # = "
         << index << "\n";
    subGPop = galnitPopulation(and2orNetwork, sGraph, subOutList,
                               subPopSize);

    r_val = gaEvaluatePopulation(subGPop, cellLibrary); // just for area
    if(r_val == 0.0){
        printf("In %s at line %d\n", __FILE__, __LINE__);
        sGraph->PrintAllNodes();
        exit(0);
    }
    subPList->InsertRear(subGPop);
}

// Initialize Global Population
G1Chrom *globalChrom;
LinkedList<MapGraph> *gChromosomes;
int valid, i, num_subg, max_pos;

num_subg = subGList->ListSize();
double gDelay;
double old_cost[num_subg], avgPopCost[num_subg];
double costDiff;
double gChromArea, maxArea, tReq;

tReq = REQ_TIME;
//cout << "\n ..... Printing MFP nodes ..... \n";
(void) gasInitGlobalPopulation(and2orNetwork, globalPop, subPList, mfplList,
                               cellLibrary, gPopSize, tReq);

// GA
// Select, Crossover, and Mutation
// Evaluation Subpopulation
for(subPList->Reset(), i=0; !subPList->EndOfList(); subPList->Next(), i++){
    subGPop = subPList->Data();
    avgPopCost[i] = gaEvaluatePopulation(subGPop, cellLibrary);
    cout << "Average Pop Cost = "<< avgPopCost[i] << "\n";
    old_cost[i] = avgPopCost[i];
}

int gchrom_itr;
gen_count = 0;
cout << "\n ..... Optimizing Global Population ..... \n";
printf("Optimization start time = %f\n", seconds());
while(1){
    gen_count++;
    itr_start_time = seconds();
    itr_start_TIME = time(NULL);

    cout << "\n\n Itr # -- "<< gen_count << "\n\n";
    for(subPList->Reset(), i=0; !subPList->EndOfList(); subPList->Next(), i++){
        subGPop = subPList->Data();
        subGList->Reset(1); sGraph = subGList->Data();
        (void) gasSelectCrossover(subGPop, cellLibrary, sGraph, subOutList);
        avgPopCost[i] = gaEvaluatePopulation(subGPop, cellLibrary);

        //Percentage Difference
        costDiff = 100*((old_cost[i] - avgPopCost[i])/old_cost[i]);
        if(costDiff < delta){
            (void) gaDoRateMutation(and2orNetwork, sGraph, subOutList,
                                     subGPop, m_rate);
        }
        old_cost[i] = avgPopCost[i];
    }

    // Update Global Population
    gchrom_itr = -1;
    while(1){
        valid = 0;
        globalChrom = gasGetRanGlobalChromosome(and2orNetwork, subPList,
                                                  mfplList, CHR_RANDOM);

        if(globalChrom != NULL){
            gChromosome = globalChrom->gChromosome;
            gDelay = gasEvaluateGlobalChromosome(and2orNetwork, globalChrom,
                                                  cellLibrary, gChromArea);
            max_pos = gasGetGlobalMaxCostChrom(globalPop, &maxArea);
            #ifdef GLOBAL_ITERATION_DEBUG
            cout << "--G Chrom Area = "<< gChromArea
                 << "\n";
            cout << " Delay = "<< gDelay << "\n";
            cout << " G Pop MaxArea = "<< maxArea << "\n";
            #endif
            if(gDelay <= REQ_TIME && (maxArea > gChromArea)){
                globalPop->Reset(max_pos);
                delete globalPop->DeleteCurrentElem();
                globalPop->InsertRear(globalChrom);
                break;
            }
            else if((gDelay > REQ_TIME) && (maxArea > gChromArea)){
                globalPop->Reset(max_pos);
                delete globalPop->DeleteCurrentElem();
                globalChrom->area = 100000.0 + (gDelay - REQ_TIME) * 1e13;
                globalPop->InsertRear(globalChrom);
                break;
            }
            else {
                delete globalChrom;
                break;
            }
        }
    }
    itr_end_time = seconds();

    (void) gasPrintGlobalPopPrintShort(globalPop, gPopSize);
    printf("TIME_for_Iteration = %f\n",
           difftime(time(NULL), itr_start_TIME));
    printf("TIME_Cover_Acc = %f\n",
           difftime(time(NULL), start_TIME));
    printf("TIME_Match_Cover_Acc = %f\n",
           difftime(time(NULL), match_start_TIME));

    printf("Time_for_Iteration = %f\n", (itr_end_time - itr_start_time));
    printf("Itr_start_time = %f ", itr_start_time);
    printf("Itr_end_time = %f\n", itr_end_time);

    if(gen_count >= max_gen_count){
        break;
    }
}

end_time = seconds();
printf("Elapsed_TIME(sec) = %f\n", difftime(time(NULL), start_TIME));
printf("Elapsed_TIME_W_MATCH(sec) = %f\n",
       difftime(time(NULL), match_start_TIME));

```

```

sub_idx = -1;
dual = 0;
(void)spiceGenerateSpiceNet(globalPop,gPopSize,and2orNetwork,cellLibrary,
                             nFanout,0);
/*
sub_idx = -1;
dual = 1;
(void)spiceGenerateSpiceNet(globalPop,gPopSize,and2orNetwork,cellLibrary,
                             nFanout,dual);
*/

cout<<"\n*Circuit = "<< file_name <<"\n";
cout<<"*Library = "<< lib_name <<"\n";
cout<<"*max_gen_cout = "<<max_gen_count
<<" gPopSize = "<<gPopSize
<<" nFanout = "<<nFanout<<"\n";
cout<<"*m_rate = "<<m_rate
<<" delta = "<< delta
<<" depth = "<<DEPTH_FOR_FUNC <<endl;

cout<<"*Start Time = "<<start_time<<" : End Time = "<<end_time<<"\n";
printf("Elapsed Time(sec) = %f\n", (end_time - start_time));

printf("\n*Elapsed_TIME(sec) = %f\n", difftime(time(NULL),start_TIME));
printf("Elapsed_TIME_W_MATCH(sec) = %f\n",
        difftime(time(NULL),match_start_TIME));

delete mfpList;
delete and2orNetwork;
exit(0);
}

#####
# Make file for mixed PTL/Static Synthesis package
#####
##-- g++ version 2.96
CC = g++

CUDD = /home/cho/CUDD/cudd-2.1.3
WORK = /home/cho/Research/Work.dly

ICUDD = $(CUDD)/include IWORK = $(WORK)/Include

INCLUDE = -I$(IWORK) -I$(ICUDD)
#ICFLAGS = -O
#CCFLAGS = -g -O6 -Wall
CCFLAGS = -g

LIBS = -L$(CUDD)/lib -lcudd -lst -lutil -lmtr -lm
OBS1 = main.o
OBS2 =
OBS = $(OBS1) $(OBS2)
SRCS = $(OBS1.o=.cc)

TARGET = map
all : $(TARGET) $(TARGET) : $(OBS)
$(CC) -N -o $@ $(OBS) $(INCLUDE) $(LIBS)
main.o : main.cc
$(CC) -c $(SRCS) $(CCFLAGS) $(INCLUDE)
clean :
rm -f *.o core $(TARGET) map.pure
new :
$(MAKE) clean
$(MAKE)
##### End of Make File #####

```

Bibliography

- [1] F. Assaderaghi, D. Sinitsky, S. A. Parke, J. Boker, P. K. Ko, and C. Hu, "A Dynamic threshold voltage MOSFET (DTMOS) for ultra-low voltage operation," in *IEDM Tech. Dig.*, pp. 809–812, 1994.
- [2] F. Assaderaghi, D. Sinitsky, S. A. Parke, J. Boker, P. K. Ko, and C. Hu, "Dynamic Threshold-Voltage MOSFET (DTMOS) for Ultra-Low Voltage VLSI," *IEEE Trans. on Electron Devices*, vol. 44, pp. 414–422, Mar. 1997.
- [3] V. Tiwari, S. Malik, and A. Wolfe, "Compilation techniques for low energy: an overview," in *Proceedings of 1994 IEEE Symposium on Low Power Electronics*, pp. 39–39, 1994.
- [4] R. B. adn R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Trans. on Computer-Aided Design of Integrated Circuit and Systems*, vol. 6, pp. 1602–1082, Nov. 1987.
- [5] K. Roy and C. S. Prasad, "Circuit activity based logic synthesis for low power reliable operations," *IEEE Transactions on VLSI Systems*, vol. 1, pp. 503–513, Dec. 1993.
- [6] H. Savoj, R. Brayton, and H. Touati, "Extracting Local Don't Cares for Network Optimization," in *IEEE International Conference on Computer-Aided Design*, pp. 514–517, 1991.
- [7] A. Shen, S. Devadas, A. Ghosh, and K. Keutzer, "On Average Power Dissipation and Random Pattern Testability of Combinational Logic Circuits," in *IEEE International Conference on Computer-Aided Design*, pp. 402–407, 1992.
- [8] C. A. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, Mass.:Addison-Wesley, 1980.
- [9] D. Radhakrishnan and S. R. Whitaker, "Formal Design Procedures for Pass Transistor Switching Circuits," *IEEE Journal of Solid-State Circuits*, vol. SC-20, pp. 531–536, April 1985.
- [10] K. Yano, Y. Sasaki, K. Rikino, , and K. Seki, "Top-Down Pass-Transistor Logic Design," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 792–803, June 1996.

- [11] P. Buch, A. Narayan, A. R. Newton, and A. Sangiovanni-Vincentelli, "Logic synthesis for large pass transistor circuits," *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 1997.
- [12] S. Yamashita, K. Yano, and et. al, "Pass-Transistor/CMOS Collaborated Logic: The Best of Both Worlds," in *Symposium on VLSI Circuits Digest of Technical Papers*, pp. 31–32, 1997.
- [13] C. Yang and M. Ciesielski, "Synthesis For Mixed CMOS/PTL Logic : Preliminary Results," in *International Workshop on Logic Synthesis*, (Lake Tahoe, CA), 1999.
- [14] Y. Jiang, S. S. Sapatneker, and C. Bamji, "Technology Mapping for High Performance Static CMOS and Pass Transistor Logic Designs," in *Technical Report, Dept of ECE, Iowa State University*, 1999.
- [15] G. R. Cho and T. Chen, "On Mixed PTL/Static Logic for Low-Power and High-Speed Circuits," *VLSI Design : An International Journal of Custom-Chip Design, Simulation, and Testing*, vol. 12, no. 3, pp. 399–406, 2001.
- [16] Y. Jiang, S. S. Sapatneker, and C. Bamji, "Technology Mapping for High Performance Static CMOS and Pass Transistor Logic Designs," *IEEE Trans. on Very Large Scale Intergration (VLSI) Systems*, vol. 9, pp. 577–589, Oct. 2001.
- [17] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*. Computer Science Press, Inc., 1978.
- [18] K. Keutzer, "DAGON : Technology Binding and Local Optimization by DAG Matching," in *24th ACM/IEEE Design Automation Conference*, 1987.
- [19] R. L. Rudell, *Logic Synthesis for VLSI Design*. Ph.D. dissertation, University of California, Berkeley, CA 94720, 1989.
- [20] K. Chaudhary and M. Pedram, "A near optimal algorithm for technology mapping minimizing area under delay constraints," in *Design Automation Conference*, pp. 492–498, 1992.
- [21] K. Chaudhary and M. Pedram, "Computing the Area versus Delay Trade-Off Curves in Technology Mapping," *IEEE Trans. on Computer-Aided Design of Integrated Circuit and Systems*, vol. 14, pp. 1480–1489, Dec. 1995.
- [22] F. Mailhot and G. D. Micheli, "Algorithms for Technology Mapping Based on Binary Decision Diagrams and Boolean Operations," *IEEE Trans. on Computer-Aided Design of Integrated Circuit and Systems*, vol. 12, pp. 599–620, May 1993.

- [23] M. Munteanu, P. A. Ivey, L. Seed, M. Psilogorgopoulos, N. Powell, and I. Bogdan, "Single Ended Pass-Transistor Logic," in *VLSI: Systems on a Chip*(Kluwer Academic Publisher), pp. 206–217, 1999.
- [24] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi, and A. Shimuzu, "A 3.8ns CMOS 16x16-b Multiplier Using Complementary Pass-Transistor Logic," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 388–394, April 1990.
- [25] S. B. Akers, "Binary Decision Diagrams," *IEEE Trans. on Computers*, vol. C-27, pp. 509–516, June 1978.
- [26] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, vol. C-35, pp. 677–691, Aug. 1986.
- [27] R. Drechsler and B. Becker, *Binary Decision Diagrams: Theory and Implementation*. Kluwer Academic Publishers, 1998.
- [28] N. H. E. Weste, *Principles of CMOS VLSI Design : A system Perspective*. Addison Wesley, 2nd ed., 1993.
- [29] *Star-Hspice Manual, Release 1998.2*. Avanti, 1998.
- [30] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentilli, "Sequential circuit design using synthesis and optimization," in *IEEE Int. Conf. Comput. Design*, 1992.
- [31] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, *SIS: A System for Sequential Circuit Synthesis*. Memorandum No. UCB/ERL M92/41, University of California, Berkeley, CA 94720, 1992.
- [32] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publisher, 1996.
- [33] M. R. Garey and D. S. Johnson, *COMPUTERS AND INTRACTABILITY : A Guide to the Theory of NP-Completeness*. W. H. FREEMAN AND COMPANY, 1979.
- [34] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc., 1994.
- [35] T. Sakurai, "Alpha-Power Law MOSFET Models and its Applications to CMOS Inverter Delay and Other Formulas," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 584–594, April 1990.
- [36] Semiconductor Industry Association (SIA), *International Roadmap for Semiconductors 2001 Edition*. Austin, Texas: International SEMATECH, 2001.

- [37] A. Bellaouar and M. I. Elmasry, *Low-Power Digital VLSI Design: Circuits and Systems*. Kluwer Academic Publisher, 1996.
- [38] H. Kotaki, S. Kakimoto, M. Nakano, T. Matsuoka, K. Adachi, K. Sikimoto, T. Fukushima, and Y. Sato, "Novel Bulk Dynamic Threshold Voltage MOSFET (B-DTMOS) with Advanced Isolation (SITOS) and Gate to Shallow-Well Contact (SSS-C) Process for Ultra Low Power Dual Gate CMOS," in *Proc. of IEEE International Electron Device Meeting (IEDM)*, pp. 459–462, 1996.
- [39] K. K. Young, "Analysis of Conduction in Fully Depleted SOI MOSFET's," *IEEE Trans. on Electron Devices*, vol. 36, pp. 504–506, Mar. 1989.
- [40] P.-F. Lu, C.-T. Chuang, J. Ji, L. F. Wanger, and et. al., "Floating-Body Effects in Partially Depleted SOI CMOS Circuits," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 1241–1253, Aug. 1997.
- [41] K. Bernstein and N. J. Rohrer, *SOI: Circuit Design Concepts*. Kluwer Academic Publisher, 1st ed., 2000.
- [42] N. Lindert, T. Sugii, S. Tang, and C. Hu, "Dyanmic Threshold Pass-Transistor Logic for Improved Delay at Lower Power Supply Voltages," *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 85–89, Jan. 1999.
- [43] M. A. Casu, G. Masera, G. Piccinini, M. R. Roch, and M. Zamboni, "Comparative Analysis of PD-SOI Active Body-Biasing Circuits," in *2000 IEEE International SOI Conference*, pp. 94–95, 2000.
- [44] I.-Y. Chung, Y. J. Park, and H. S. Min, "A New SOI Inverter Using Dynamic Threshold for Low-Power Applications," *IEEE Electron Device Letters*, vol. 18, pp. 248–250, June 1997.
- [45] T. Fuse, Y. Oowaki, T. Yamada, and et al, "A 0.5V 200MHz 1-Stage 32b ALU using a Body Bias Controlled SOI Pass-Gate Logic," in *1997 IEEE International Solid-State Circuit Conference*, (San Francisco, CA), 1997.
- [46] B.-T. Wang and J. B. Kuo, "A Novel Low-Voltage Silicon-On-Insulator (SOI) CMOS Complementary Pass-Transistor Logic (CPL) Circuit using Asymetrical Dynamic Threshold Pass-Transistor (ADTPT) Technique," in *Proc. 43rd Midwest Symp. on Circuit and Systems*, (Lansing, MI), pp. 694–697, 2000.
- [47] R. Puri and C.-T. Chuang, "Hysterisis Effect in Pass-Transistor-Based, Partially Depleted SOI CMOS Circuits," *IEEE Journal of Solid-State Circuits*, vol. 35, pp. 625–631, April 2000.
- [48] F. Mailhot and G. D. Micheli, "Sequential circuit design using synthesis and optimization," in *The proceedings of the European Desing Automation Conference*, 1990.

- [49] L. Benni and G. D. Micheli, "A Survey of Boolean Matching Techniques for Library Binding," *ACM Transactions on Design Automation of Electronics Systems*, vol. 2, pp. 193–226, July 1997.
- [50] Y. Kukimoto, R. K. Brayton, and P. Sawkar, "Delay-Optimal Technology Mapping by DAG Covering," in *Design Automation Conference*, (San Francisco, CA), pp. 348–351, 1998.
- [51] M. Zhao and S. S. Sapatnekar, "A New Structural Pattern Matching Algorithm for Technology Mapping," in *38th ACM/IEEE Design Automation Conference*, (Las Vegas, NV), 2001.
- [52] S. Ercolani and G. D. Micheli, "Technology mapping for electrically programmable gate array," in *Design Automation Conference*, pp. 234–239, 1991.
- [53] B. M. E. Moret, "Decision Trees and diagrams," *Computer Surveys*, vol. 14, pp. 593–623, 1982.
- [54] F. Brown, *Boolean Reasoning: The Logic of Boolean Equations*. Kluwer Academic Publisher, 1990.
- [55] D. Moller, P. Molitor, and R. Drechsler, "Symmetry based variable ordering for ROBDDs," in *Proc. IFIP Workshop on Logic and Architecture Synthesis*, pp. 47–53, 1994.
- [56] C. Scholl, D. Moller, and P. Molitor, "BDD Minimization Using Symmetries," *IEEE Trans. on Computer-Aided Design of Integrated Circuit and Systems*, vol. 18, pp. 81–100, Feb. 1999.
- [57] L. Heinrich-Litan and P. Molitor, "Least Upper Bounds for the Size of OBDDs Using Symmetry Properties," *IEEE Trans. on Computers*, vol. 49, pp. 360–368, April 2000.
- [58] S. Panda, F. Somenzi, and B. F. Plessier, "Symmetry detection and dynamic variable ordering of decision diagrams," in *Proc. International Conference on Computer Aided Design*, pp. 628–631, 1994.
- [59] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design : OBDD-Foundations and Applications*. Springer-Verlag, 1998.
- [60] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
- [61] Z. Michalewicz, *Genetic Algorithms + Data Structure = Evolution Programs*. Springer-Verlag, 1996.

- [62] K. A. D. Jong, *Analysis of the behavior of genetic adaptive systems*. Ph.D. Dissertation, University of Michigan, Ann Arbor, MI, 1975.
- [63] J. J. Grefenstett, "Technology Mapping for High Performance Static CMOS and Pass Transistor Logic Designs," *IEEE Transaction on Systems, Man, and Cybernetics*, vol. SMC 16, p. 128, Jan./Feb. 1986.
- [64] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*. John Wiley & Sons, Inc., 1998.
- [65] C. L. Berman, J. L. Carter, and K. Day, "The Fanout Problem: From Theory to Practice," In C. L. Seitz editor, *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conferences*, pp. 66–99, 1989.
- [66] H. Touati, *Performance-Oriented Technology Mapping*. Ph.D. dissertation, ERL Memo M90/109, University of California, Berkeley, CA 94720, Nov. 1990.
- [67] T. Okamoto and J. Kong, "Buffered Steiner Tree Construction with Wire Sizing," in *Proceeding of the 1996 ICCAD*, (San Francisco, CA), pp. 44–49, 1996.
- [68] D. Kung, "A Fast Optimization Algorithm for Near-Continuous Buffer Libraries," in *Proceeding of the 1998 DAC*, (San Francisco, CA), pp. 352–355, Oct. 1998.
- [69] P. Rezvani, A. H. Ajami, M. Pedram, and H. Savoj, "LEOPARD: A Logical Effort-based fanout Optimizer for ARea and Delay," in *Proceeding of the 1999 ICCAD*, (San Jose, CA), pp. 516–519, 1999.
- [70] R. Murgai, "Efficient Fanout Optimization Algorithms," in *Proceedings of the conference on Asia South Pacific Design Automation Conference*, (Yokohama, Japan), pp. 571–577, 2001.
- [71] H. Touati, C. Moon, R. K. Brayton, and A. Wang, "Performance Oriented Technology Mapping," in *Proceeding of the 6th MIT VLSI Conference*, pp. 79–97, 1990.
- [72] P. Cocchini, M. Pedram, G. Piccinini, and M. Zamboni, "Fanout Optimization under a Submicron Transistor-Level Delay Model," in *Proceeding of the 1998 ICCAD*, (San Jose, CA), pp. 551–556, 1998.
- [73] *Powermill and Pathmill User Guide*. Synopsys, 2000.
- [74] S. Palnitkar, *Verilog HDL*. Prentice Hall, 1st ed., 1996.
- [75] S. Sjöholm and L. Lindh, *VHDL for Designers*. Prentice Hall, 1st ed., 1996.
- [76] *DesignWare Product Overview*. Synopsys, 1999.

- [77] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 1996.
- [78] H. Iwai, "CMOS Technology – Year 2010 and Beyond," *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 357–366, Mar. 1999.
- [79] D. Sylvester and H. Kaul, "Future Performance Challenge in Nanometer Design," in *Proceedings of the 38th Design Automation Conference*, (Las Vegas, Nevada), pp. 3–8, 2001.
- [80] G. R. Cho and T. Chen, "Mixed PTL/Static Logic Synthesis Using Genetic Algorithms for Low-Power Applications," in *International Symposium on Quality Electronic Design*, (San Jose, California), pp. 458–463, 2002.
- [81] G. Gristede and W. Hwang, "A Comparison Of Dual-Rail Pass Transistor Logic Family in 1.5V, 0.18 μ m CMOS Technology For Low Power Applications," in *GLSVLSI*, (Evanston, Illinois), 2000.
- [82] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, pp. 114–117, April 1965.
- [83] P. K. Bondyopadhyay, "Moore's Law Governs the Silicon Revolution," *Proceedings of the IEEE*, vol. 86, pp. 78–81, Jan. 1998.
- [84] C. Hu, "Future CMOS Scaling and Reliability," *Proceedings of the IEEE*, vol. 81, pp. 682–689, May 1993.
- [85] C. R. Barrett, "Microprocessor Evolution And Technology Impact," in *1993 Symposium on VLSI Technology, Digest of Technical Papers*, pp. 7–10, 1993.
- [86] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Simon, Y. Taur, and H.-S. P. Wong, "Device Scaling Limits of Si MOSFETs and Their Application Dependencies," *Proceedings of the IEEE*, vol. 89, pp. 259–288, Mar. 2001.
- [87] J. P. Uyemura, *CMOS Logic Circuit Design*. Kluwer Academic Publishers, 2nd ed., 1999.
- [88] T. Sakurai, "CMOS Inverter Delay and Other Formulas Using α -Power Law MOS Model," in *International Conference on Computer-Aided Design 1998*, pp. 74–277, 1988.
- [89] R. Rajsuman, *Iddq Testing for CMOS VLSI*. Artech House, 1995.
- [90] R. H. Krambeck, C. M. Lee, and H. F. Law, "High speed compact circuits with CMOS," *IEEE Journal of Solid-State Circuits*, vol. 17, pp. 614–619, June 1982.

- [91] B. Ramasubramanian, H. Schmit, and L. R. Carley, "Mixed-Swing Quadrail for Low Power Dual-Rail Domino Logic," in *Proc. of the International Symposium on Low Power Electronics and Design*, (San Diego, CA), 1999.
- [92] B. J. Sheu and et al., "BSIM: Berkeley short-channel IGFET model for MOS transistors," *IEEE Journal of Solid-State Circuits*, vol. 22, pp. 558 – 566, Aug. 1987.
- [93] Z. Chen, M. Johnson, L. Wel, and K. Roy, "Estimation of Standby Leakage Power in CMOS Circuits Considering Accurate Modeling of Transistor Stacks," in *Proc. of the ISLPED'98*, (Monterey, California, USA), 1998.
- [94] W. Jiang, V. Tiwari, E. de la Iglesia, and A. Sinha, "Topological Analysis for Leakage Prediction of Digital Circuits," in *Proc. of the 15th International Conference in VLSI Design*, 2002.
- [95] K. F. Sheugraf and C. Hu, "Hole Injection SiO₂ Breakdown Model for Very Low Voltage Lifetime Extrapolation," *IEEE Trans. on Electron Devices*, vol. 41, pp. 761–767, May 1994.
- [96] C.-H. Choi, K.-H. Oh, J.-S. Goo, Z. Yu, and R. W. Dutton, "Direct Tunneling Current Models for Circuit Simulaiton," in *Proc. of IEEE International Electron Device Meeting (IEDM)*, pp. 735–738, 1999.
- [97] F. Somenzi, *CUDD:CU Decision Diagram Package (Release 2.1.3)*. Dept. of ECE, University of Colorado at Boulder, Aug. 1997.
- [98] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide (Ver. 3.0)*. Triangle Park, NC:Microelectronics Center of North Carolina Research, 1991.
- [99] R. Burch, F. N. Najam, P. Yang, and T. N. Trick, "A Monte Carlo Approach for Power Estimation," *IEEE Trans. on Computer-Aided Design of Integrated Circuit and Systems*, vol. 1, pp. 63–71, Mar. 1993.
- [100] F. N. Najm, "A survey of Power Estimation Techniques in VLSI Circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuit and Systems*, vol. 2, pp. 446–455, Dec. 1994.
- [101] G. R. Cho and T. Chen, "On the Impact of Fanout Optimization and Redundant Buffer Removal for Mixed PTL Synthesis," in *IEEE/ACM 11th International Workshop on Logic & Synthesis*, (New Orleans, Louisiana), 2002.