1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services Branch

Direction des acquisitions et
des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Reproduit avec l'autorisation

Reproduit avec l'autorisation

NOTICE

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

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

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

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

AVIS

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

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

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

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

Canada

**TRANSISTOR PLACEMENT ALGORITHM FOR AUTOMATIC LAYOUT
SYNTHESIS OF CMOS/BiCMOS LOGIC AND INTERFACE CIRCUITS**

Hongxia Xia, B.Eng

CARLETON UNIVERSITY

Ottawa, Canada

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

© Copyright October 1993, Hongxia Xia

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file *Votre référence*

Our file *Notre référence*

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

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

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

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

ISBN 0-315-89906-9

Canada

Name

Hongxia XIA

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Linguistics

SUBJECT TERM

0544

SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
General 0289
Linguistics 0290
Modern 0291
Literature 0401
General 0294
Classical 0295
Comparative 0297
Medieval 0298
Modern 0316
African 0591
American 0305
Asian 0352
Canadian (English) 0355
Canadian (French) 0593
English 0311
Germanic 0312
Latin American 0315
Middle Eastern 0313
Romance 0314
Slavic and East European 0579

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology 0323

SOCIAL SCIENCES

American Studies 0324
Anthropology 0326
Archaeology 0327
Cultural 0310
Physical 0272
Business Administration 0770
General 0454
Accounting 0338
Banking 0385
Management 0501
Marketing 0503
Canadian Studies 0505
Economics 0508
General 0509
Agricultural 0510
Commerce-Business 0511
Finance 0358
History 0366
Labor 0351
Theory 0578
Folklore 0366
Geography 0351
Gerontology 0351
History 0578
General 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science 0615
General 0616
International Law and 0617
Relations 0617
Public Administration 0814
Recreation 0452
Social Work 0626
Sociology 0627
General 0938
Criminology and Penology 0631
Demography 0628
Ethnic and Racial Studies 0629
Individual and Family 0630
Studies 0700
Industrial and Labor 0344
Relations 0709
Public and Social Welfare 0999
Social Structure and 0453
Development 0344
Theory and Methods 0709
Transportation 0999
Urban and Regional Planning 0453
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy 0475
Animal Culture and 0476
Nutrition 0359
Animal Pathology 0478
Food Science and 0479
Technology 0480
Forestry and Wildlife 0817
Plant Culture 0777
Plant Pathology 0746
Plant Physiology 0306
Range Management 0287
Wood Technology 0308
Biology 0309
General 0379
Anatomy 0329
Biostatistics 0353
Botany 0369
Cell 0793
Ecology 0410
Entomology 0307
Genetics 0317
Limnology 0416
Microbiology 0433
Molecular 0821
Neuroscience 0778
Oceanography 0472
Physiology 0786
Radiation 0760
Veterinary Science 0425
Zoology 0996
Biophysics 0425
General 0996
Medical 0996

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
General 0300
Audiology 0992
Chemotherapy 0567
Dentistry 0350
Education 0769
Hospital Management 0758
Human Development 0982
Immunology 0564
Medicine and Surgery 0347
Mental Health 0569
Nursing 0570
Nutrition 0380
Obstetrics and Gynecology 0354
Occupational Health and 0381
Therapy 0571
Ophthalmology 0419
Pathology 0572
Pharmacology 0382
Pharmacy 0573
Physical Therapy 0574
Public Health 0575
Radiology 0575
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences 0485
Chemistry 0749
General 0486
Agricultural 0487
Analytical 0488
Biochemistry 0738
Inorganic 0490
Nuclear 0491
Organic 0494
Pharmaceutical 0495
Physical 0754
Polymer 0405
Radiation 0605
Mathematics 0986
Physics 0606
General 0608
Acoustics 0748
Astronomy and 0607
Astrophysics 0798
Atmospheric Science 0759
Atomic 0609
Electronics and Electricity 0610
Elementary Particles and 0752
High Energy 0756
Fluid and Plasma 0611
Molecular 0463
Nuclear 0346
Optics 0984
Radiation 0346
Solid State 0984
Statistics 0984

Applied Sciences

Applied Mechanics 0346
Computer Science 0984

Engineering 0537
General 0538
Aerospace 0539
Agricultural 0540
Automotive 0541
Biomedical 0542
Chemical 0543
Civil 0544
Electronics and Electrical 0348
Heat and Thermodynamics 0545
Hydraulic 0546
Industrial 0547
Marine 0794
Materials Science 0548
Mechanical 0743
Metallurgy 0551
Mining 0552
Nuclear 0549
Packaging 0765
Petroleum 0554
Sanitary and Municipal 0790
System Science 0428
Geotechnology 0796
Operations Research 0795
Plastics Technology 0994
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

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

**“Transistor Placement Algorithm for Automatic Layout Synthesis of
CMOS / BiCMOS Logic and Interface Circuits”**

**submitted by Hongxia Xia (B. Eng.) in partial fulfillment of the
requirements for the degree of Master of Engineering**

Thesis Supervisor

Chairman, Department of Electronics

Carleton University

October 20, 1993

Abstract

The emergence of BiCMOS technology as a VLSI technology presents new challenges for CAD support for VLSI design. In particular, tools for layout synthesis that can handle standard cells comprising a mixture of MOS and bipolar devices are demanded. This thesis makes an important contribution towards enhanced CAD support for BiCMOS leaf cell layout. It contains an analysis of the principal characteristics of typical circuits contained in BiCMOS standard cell libraries that are important from a transistor placement point of view. It proposes a detailed formulation for the transistor placement problem in which a transistor placement problem is reduced to a module floorplanning problem. It proposes an efficient algorithm that can find the optimal module placements for small circuits (containing no more than 6 modules) and sub-optimal module placements for bigger circuits. By employing slicing structured floorplans and shape functions, the algorithm allows overlapping in the placement under certain conditions, and is able to enforce a pre-placement constraints by place the module inside the area specified. Finally, the thesis illustrates the performance of the algorithm on three types of circuits including BiCMOS logic, ECL logic, and I/O buffer circuits.

Acknowledgments

I would like to express my sincere thanks to my supervisor, Prof. Martin Lefebvre for his timely guidance and constant encouragement. It is his expert advice and continuous interest that made my thesis research work fruitful. I also wish to thank the fellow students in VLSI CAD group for all the help from them. In particular, I would like to thank David F. Skoll and Kim-minh Nguyen for helping me to incorporate my program with theirs.

Thanks are also due to Prof. Garry Tarr, graduate advisor of Dept. of Electronics, for his precious time spent on helping me, as a foreign student here, in all sorts of ways. I thank our system administrator Arthur Castonguay, who, without any complaint, solved the problems caused by my occasional abuse of the system. I must thank all the department staff. The academic, friendly environment made it a great pleasure studying here.

This research would not be possible without the financial support of Department of Electronics of Carleton University, the Natural Sciences and Engineering Research Council of Canada, and Micronet, a Network of Centres of Excellence focussing on microelectronics research funded by the Government of Canada. In addition, I would like to acknowledge the Canadian Microelectronics Corporation for providing the workstations and software licenses used for the prototype implementation.

Table of Contents

Abstract	iii
Acknowledgments	iv
Table of Contents	vi
List of Figures	vii
List of Tables	ix
 Chapter 1 Introduction.....	 1
1.1 Perspective	1
1.2 Objectives.....	3
1.3 Outline	3
 Chapter 2 Introduction to BiCMOS Logic and Interface Circuits	 5
2.1 BiCMOS Logic Cells	5
2.2 ECL Logic Cells.....	10
2.3 I/O Buffers.....	12
2.4 Summary.....	15
 Chapter 3 Algorithm Background	 16
3.1 Floorplanning -- Placement for Flexible Modules	16
3.1.1 Slicing structure	17
3.1.2 Shape Functions	18
3.2 Existing approaches to Block Placement	24
3.3 Branch and Bound Technique.....	25
 Chapter 4 Problem Definition	 28
4.1 Reduction of Transistor Placement Problem to Module Floorplanning Problem	28
4.2 Definition for the Floorplanning Problem	31
4.2.1 Objective Cost Function	31

4.2.2	Constraints	33
Chapter 5	Algorithm Description	35
5.1	Algorithm Overview	35
5.2	Cost Approximation for Partial Placements	39
5.3	Details of Search Tree Node Expansion	44
5.4	Topology to Placement Conversion	47
5.5	The Effects of Pre-placement Constraints	51
5.5.1	Constraints on Placement Boundaries Imposed by Pre-placement	51
5.5.2	Pre-placement Constraints in Shape Function Construction	54
5.5.3	Pre-placement Constraints in Conversion of Topology to Placement	54
5.5.4	Pre-placement Constraints in Slicing Tree Validation	55
Chapter 6	Experimental Results	56
6.1	4-input BiCMOS nor gate	58
6.1.1	Circuit Description	58
6.1.2	Impact of control parameters	58
6.2	I/O Buffer (Narrow Pad Configuration)	71
6.3	2-input ECL nor/or Circuits	76
Chapter 7	Conclusions and Recommendations for Further Research	84
7.1	Summary	84
7.2	Contributions	85
7.3	Recommendations for Future Research	85
	Glossary of Terms and Symbols	87
	References	89

List of Figures

Figure 2.1	Schematics of N-type BiCMOS gates	7
Figure 2.2	Schematics of R-type BiCMOS Gates.	8
Figure 2.3	Layout of a Standard Cell of BiCMOS Logic Circuits	9
Figure 2.4	Schematic of the Basic nor/or ECL gate	10
Figure 2.5	Layout of a Standard Cell of ECL Circuits	11
Figure 2.6	Schematic of a Typical Bi-directional CMOS I/O Buffer.. . . .	12
Figure 2.7	Typical layout templates for I/O buffer	14
Figure 3.1	slicing and non-slicing structure	17
Figure 3.2	Slicing trees corresponding to Figure 3.1 (a)	21
Figure 3.3	Shape Function Definition.	21
Figure 3.4	Shape functions prediction	22
Figure 3.5	Constraint propagation through shape functions.	23
Figure 3.6	Two types of shape functions	23
Figure 3.7	Decision Tree.	27
Figure 3.8	Search Trees	27
Figure 4.1	Transistor to Module Placement Problem Reduction	30
Figure 5.1	Branch-and-Bound Decision Tree	36
Figure 5.2	Branch and Bound Placement Algorithm	37
Figure 5.3	Leaf Node Shape Function Definition.	41
Figure 5.4	Wire Length Estimation.	43
Figure 5.5	Slicing Tree Generation.	48
Figure 5.6	Leaf Node Expansion	48
Figure 5.7	Impact of Pre-placement Constraints on Placement Boundary.	53
Figure 5.8	Impact of Pre-Placement Constraints' on Shape Functions	54
Figure 6.1	Problem formulation for 4-input BiCMOS nor gate	59

Figure 6.2	Impact of <i>*cellHeight*</i> : Placements Achieved for 4-input BiCMOS Nor Gate60
Figure 6.3	Impact of <i>*bus_penalty*</i> and <i>*wire_length_to_area*</i>64
Figure 6.4	Impact of <i>*horizontal_penalty*</i>65
Figure 6.5	Impact of <i>*exhaustive_threshold*</i>66
Figure 6.6	Impact of <i>*blk_number_difference_bound*</i> (1).....67
Figure 6.7	Impact of <i>*blk_number_difference_bound*</i> (2)68
Figure 6.8	Impact of <i>*blk_number_difference_bound*</i>69
Figure 6.9	A Manual Layout of 4-input BiCMOS nor gate70
Figure 6.10	Connectivity Graph $G(V, E)$ for the I/O Buffer circuit.71
Figure 6.11	Shape functions for the modules in the I/O buffer72
Figure 6.12	Placement Achieved for I/O buffer (narrow configuration)74
Figure 6.13	A layout for I/O Buffer (narrow configuration).75
Figure 6.14	Schematic of 2-input ECL nor/or circuit76
Figure 6.15	Problem Formulation for 2-input ECL nor/or Gate.78
Figure 6.16	ECL standard cell specification in BATMOS80
Figure 6.17	Shape functions for modules in ECL 2-input nor/or Gate.81
Figure 6.18	Placement for 2-input ECL nor/or Gate82
Figure 6.19	Layout for 2-input ECL nor/or gate.83

List of Tables

Table 4.1	Summary of three kinds of modules	29
Table 4.2	Weights on wire lengths.	33
Table 6.1	Parameters of Group I: user requirements.	57
Table 6.2	Parameters of Group II: software behavior control.	57
Table 6.3	Typical settings for some of the parameters	57
Table 6.4	Locations and dimensions for fixed modules in I/O buffer	72
Table 6.5	Pre-placement restrictions	72
Table 6.6	Dimensions of the devices.	79
Table 6.7	Dimensions and locations for fixed modules.	79

Chapter 1

Introduction

1.1 Perspective

In VLSI physical design, the designer transforms a circuit description into a geometric description, which is known as a layout. A layout consists of a set of planar geometric shapes in several layers. A general overview of circuit layout is given in [1].

VLSI design styles can be broadly classified as either full-custom or semi-custom. In a full-custom layout, different blocks of a circuit can be placed at any location on a silicon wafer as long as all the blocks are non-overlapping. On the other hand, in semi-custom layout, some parts of a circuit are pre-designed and placed on some specific place on the silicon wafer. [2]

Standard cell design style is an important category of semi-custom design. Standard cell architecture considers the layout to consist of rectangular cells of the same height. Cells are placed in rows and the space between two rows is called a channel, and is used to perform interconnections between cells. A special cell called feedthrough may be inserted between cells in a row for the interconnection of cells that belong to two non-adjacent rows.

While standard cell layout designs are quicker to develop, a substantial initial investment is needed in the development of the cell library, which may consist of several hundred cells, or even more. Each type of cell must be created with several transistor sizes to meet different circuit requirements.

With the power of design automation, a lot of effort can be saved in standard cell library development. Similar to chip-level layout design automation, standard cell layout

design automation can be divided into subproblems of placement, routing and compaction.

The cell level placement problem involves placing circuit elements inside a cell, in such a way as to maximally preserve the connectivity among the elements, thereby maximizing the routability of that unit and minimizing the area required. Like general placement problems, it has been shown that the problem is NP-hard [3]. The computation time required to obtain the optimum solution increases exponentially as the number of modules increases [4].

For CMOS standard cells, the placement problem consists of finding a relative order as well as orientation for the devices. Many methods have been reported to solve this problem. An overview is provided in [5]. For BiCMOS cells, on the other hand, layout synthesis tools are facing the challenges from cells comprising a mixture of MOS and bipolar devices. Bipolar devices tend to have more flexibility in their shapes and locations. For example, even with the same size a bipolar transistor can have a number of layout configurations by adjusting the aspect ratio of its emitter and other parts accordingly. Unlike MOS devices, the layout of a bipolar transistor is usually stand-alone, hence presents more freedom in choosing location for the individual transistor. The placement for BiCMOS cells must be able to take advantage of such flexibility to produce better placements.

This thesis addresses the problem of transistor level placement for logic and interface cells in the context where both mixed MOS and bipolar transistors are present in the circuit. The circuits in such placement problem have the following characteristics in common: (i) They have small numbers of objects to place; (ii) The objects to be placed may have flexible shape configurations to choose from. (iii) Many problems allow overlapping of certain objects, or demand restricted regions for the locations of certain objects.

1.2 Objectives

Two objectives are set for this thesis. The first objective is to develop an efficient algorithm to solve the transistor placement problem in the context of BiCMOS leaf cell generation. The second objective is to validate the algorithm by implementation in Common Lisp and by comparing the placements produced by the software with hand-crafted layouts.

1.3 Outline

This thesis is organized into 7 Chapters. Chapter 2 introduces three types of circuits targeted by the algorithm, emphasizing the characteristics affecting placement.

Chapter 3 provides the background necessary for the understanding of the algorithm presented in this thesis. The features of slicing floorplans and shape functions are discussed. Existing placement algorithms and the fundamentals of branch and bound techniques are also explained.

Chapter 4 formulates the placement problem as an optimization problem. The strategy is to recast the transistor placement problem into a layout floorplan design problem with a mixture of rigid and flexible modules. The MOS devices are merged to form locally optimum chains of transistors which are then treated as rigid modules. Other devices such as diodes, bipolar transistors, resistors etc., are treated as flexible modules which can be characterized by a shape function. In order to determine the locations and shapes of these modules, an optimization problem is formulated.

Chapter 5 describes a constructive branch-and-bound algorithm applied to generate a slicing floorplan. It concentrates on the process of child generation and the definition of the bounding function for partial placements of the branch-and-bound algorithm. The method proposed merges the processes of optimization and topology construction so that better topologies can be found. It has important features of being able to deal with

overlapping blocks as well as pre-placement constraints, features which are missing in most of the existing placement algorithms.

Chapter 6 presents detailed examples of the algorithm on a set of benchmark circuits. The proposed approach is validated with experimental results on these circuits, including BiCMOS logic circuits, ECL logic circuits, and an I/O buffer circuit. The controlling effects of user-specified parameters are illustrated. A qualitative comparison between generated placements generated and hand-crafted layouts is also made.

Chapter 7 draws conclusions and provides recommendations for further research.

Chapter 2

Introduction to BiCMOS Logic and Interface Circuits

This chapter describes the nature of the circuits targeted by the research as well as the physical constraints that must be supported. Three kinds of circuits, BiCMOS logic, ECL logic and I/O buffer are introduced. From the placement point of view, BiCMOS circuits are unique in that they are comprised of both MOS and bipolar devices. This leads to a placement problem with a mixture of rigid and flexible modules. ECL logic contains only bipolar devices. The ECL standard cells usually have three more buses running through for reference signals. The buses have both fixed width and location. Therefore, how to place the transistors in the presence of these buses becomes a more important issue here. I/O buffer circuits emphasize the importance of certain constraints on the feasible region that a device can be placed.

2.1 BiCMOS Logic Cells

BiCMOS technology is being actively applied to VLSI because of its high speed, density, and its low power dissipation characteristics. Among the available types of BiCMOS logic gates are N-type and R-type gates. The main difference between the two is the way they discharge the bases of the bipolar transistors. In the N-type, it is done by NMOS transistors. In the R-type, it is done by resistors. [41].

Figure 2.1 and Figure 2.2 show three basic logic gates implemented as N-type and R-type configurations respectively. The circuits consist of a number of MOS transistors, resistors, and bipolar devices.

The layout attributes of each element are affected by several aspects: First, electrical features of the circuit basically decide the size of the devices. Secondly, Bipolar devices

have significant differences from MOS devices in terms of layout. They are usually much bigger than MOS devices and they usually don't share diffusion area with other devices. They are usually flexible in layout dimensions. MOS devices are usually not flexible in layout dimensions. Thirdly, size could also affect the device layout attributes. Although MOS devices normally have fixed layout dimensions, significantly larger size makes MOS devices have flexible layout dimensions. In this case gates are bent to save area.

A common design for the layout of BiCMOS logic cells is shown in Figure 2.3, with V_{dd} and V_{ss} buses at the top and bottom of the cell respectively. V_{dd} and V_{ss} 's positions, bus width and the height of the cell are fixed as part of the standard cell design specifications. The width of the cell is flexible. Most of the devices are placed between the power buses without overlapping with each other or with the power buses. Depending on the library specifications, there may also be additional power rails and some restrictions on the positions of I/O pins.

(a) N-type BiCMOS
Inverter

(b) N-type BiCMOS
2-input nand gate

(c) N-type BiCMOS 2-
input nor gate

Figure 2.1 Schematics of N-type BiCMOS gates

(a) R-type BiCMOS Inverter

(b) R-type BiCMOS 2-input nand gate

(a) R-type BiCMOS 2-input nor gate

Figure 2.2 Schematics of R-type BiCMOS Gates

A layout of the 2-input BiCMOS nand gate

Figure 2.3 Layout of a Standard Cell of BiCMOS Logic Circuits

2.2 ECL Logic Cells

Emitter-coupled logic (ECL) is a popular logic family for integrated circuit design. It contains bipolar devices only and has the properties of high speed and enhanced drive capabilities. BiCMOS technology made ECL logic family even more important, because in this new technology, it is possible for ECL circuits to be fabricated on the same wafer as CMOS circuits. In this context the application of ECL greatly improves the quality of the circuit where fast speed or high drive capability are required.

Figure 2.4 shows the schematic of the basic nor/or ECL gate. There are 5 resistors and 6 bipolar transistors. Q1, Q2 and Q3 require symmetric electrical characteristics. Therefore, the same transistor layout is used for them. Also, the transistors are usually placed very close to each other whenever possible in order to improve matching. The same may apply to R1 and R2, R3 and R4, Q4 and Q5.

Figure 2.5 shows the standard cell layout for the circuit. Five power buses (V_{dd} , V_{ss} , V_{ref1} , V_{ref2} , V_{ref3}) run through the cell horizontally. Again, the positions and widths of power rails as well as total cell height are fixed as part of the standard cell design specifications. Most of the devices are not overlapped by the power rails since the same layer of metal is used. There may be restricted positions for I/O pins.

Figure 2.4 Schematic of the Basic nor/or ECL gate

A layout of the 2-input ECL nor/or gate

Figure 2.5 Layout of a Standard Cell of ECL Circuits

2.3 I/O Buffers.

The schematic of an I/O Pad circuit is given in Figure 2.6. The circuit is composed of a PMOS transistor, an NMOS transistor, an n-well Resistor, a p+/n diode, a pad, a lateral silicon controlled rectifier (LSCR), a snap back transistor (SBT), and logic blocks including a tri-state buffer, pre-drivers and an input buffer.

Figure 2.6 Schematic of a Typical Bi-directional CMOS I/O Buffer.

Protection from latch-up and electric static discharge (ESD) are two of the main issues in I/O buffer design. The reason is that I/O buffers are the interface between the outside and the chip, hence they deal with currents that are much bigger than the current dealt with by core circuits and are exposed to a more unpredictable environment.

For the above reasons, the PMOS, NMOS transistor, the resistor and the diode are made much larger than those in the core circuits. Their layouts are made flexible. The logic block, the Pad, LSCR and SBT have fixed dimensions.

There are two typical layout configurations for I/O pad buffers: narrow and wide. The narrow configuration is used in circuits with very large pin numbers (pad limited) and the wide configuration is found in circuits with relatively few I/O pins (core limited) [42]. The typical layouts for both configurations are shown in Figure 2.7.

The layout template of the narrow pad is a tall and rectangular cell with the pad opening located near the top. The PMOS and NMOS output drivers and ESD protection circuitry are typically near the center of the cell whereas the logic block is near the bottom. Separate power rails are usually used to supply the pad macro and the chip's core circuitry. Wide pad macros are usually squares with the pad opening typically placed near the middle of the layout. The PMOS and NMOS output drivers are frequently situated on opposite sides of the pad opening. The ESD protection circuitry is typically located near the top while the logic block is usually found near the bottom of the cell [42].

In I/O buffers, the pad opening and power rails are usually realized on the top metal layer. The power rails don't share any layer of metal with the devices except the pad. This contributes two special characters to I/O buffers' layouts: 1) The position of the pad is not fixed but restricted to a certain range. 2) Except for the pad, all the other devices are allowed to be placed under the power rails, i.e. overlapping is allowed.

Figure 2.7 Typical layout templates for I/O buffer

2.4 Summary

From the above discussion, it can be seen that the targeted circuits are generally small in scale, containing no more than 20 elements. The number of the objects handled by placement programs can be reduced further because of the possibilities of merging several elements to a composite module when these elements need to be closely matched.

It is also obvious that the circuits present some physical features which are usually not supported by chip-level placement algorithms. First, some elements have to be placed in certain ranges, such as the pad. These kinds of constraints are referred to as “pre-placement constraints”. Secondly, it is possible for some elements to be overlapped. For example, the power rails can be overlapped by some devices in I/O buffers. Thirdly, some objects need special treatment since they are already placed but they still affect the placement by their connections to the objects to be placed. Power rails of the circuits are examples of such objects.

Finally, many of the elements in the circuits are flexible in terms of their layout dimensions, such as bipolar transistors in ECL and BiCMOS circuits and MOS transistors in I/O buffers, while others have fixed layouts, such as MOS transistors in BiCMOS circuits.

Chapter 3

Algorithm Background

This chapter presents a summary of related research from the design automation and operations research literature, which is the background for the methods and algorithms proposed in this thesis. With respect to design automation, subjects related to floorplanning are discussed. With respect to operation research, branch and bound technique is reviewed.

3.1 Floorplanning -- Placement for Flexible Modules

Placement is an important step in the physical design of VLSI circuits. In the placement phase, modules are assigned specific shapes and are positioned on a layout surface in such a fashion that no two blocks are overlapping and enough space is left on the layout surface for the routing phase. [2]

Based on whether the participating modules are flexible in shape, the placement can be divided into two classes, block placement and floorplanning. Block placement only deals with fixed-shape modules, while floorplanning can deal with both fixed-shape and flexible-shape modules. Flexible-shape modules can adjust their shapes provided certain rules are met. Shape functions are employed to mathematically summarize these rules.

A common approach to floorplan design is to first determine the topology of the floorplan primarily using the interconnection information among the modules [25]-[28]. After the topology of the floorplan is determined, optimizations are then performed on it to minimize some cost measure. The optimization determines the shape of each module such that the overall area is minimized [29]-[32]. The shortcoming of this approach is that the optimization is only performed on the determined topology, which may not be optimal when both area and interconnection information are taken into account.

Many techniques of general block placement have been adapted to floorplanning.

3.1.1 Slicing structure

An important sub-set of all floorplans is the class of slicing floorplans [27]-[28]. A slicing floorplan can be obtained by recursive dissection of a rectangle into two parts by either a vertical line or a horizontal line. The dissection process begins with a rectangle which represents the entire area. This rectangle is split with either a vertical or horizontal slice line to form two sub-rectangles. These rectangles are in turn split with slice lines to form smaller sub-rectangles, and the process continues until there is one rectangle for each module in the circuit. The final rectangles generated by the slicing process are referred to as leaf rectangles, and a rectangle which is split in two during the dissection process is referred to as a composite rectangle.

In Figure 3.1, (a) is slicing structure, (b) is a non-slicing structure.

Figure 3.1 slicing and non-slicing structure

A convenient way to represent a slicing floorplan is slicing tree. In a slicing tree, each node represents a rectangle in the floorplan. The leaf nodes represent leaf rectangles and the internal nodes represent composite rectangles. Internal nodes are associated with either a vertical or a horizontal cut, and also represent the set of circuit elements that fall within the sub-tree rooted at that node (cluster). The left and right children of an internal node represent the two sub-rectangles. The slicing tree is an oriented tree in that the relative position of sibling nodes is significant. The left child of

a node re_l represents the left or bottom sub-rectangle, and the right child represents the right or top sub-rectangle.

Different sequences of dissection result in different slicing trees, but not necessarily different floorplans. Therefore, a slicing floorplan may be represented by different slicing trees. As an example, Figure 3.2 shows two slicing trees that represent the same slicing floorplan in Figure 3.1 (a).

3.1.2 Shape Functions

A module's shape function $SF(x, y) = 0$ is a curve in the first quadrant of the xy plane. Suppose (x, y) is a point on the curve. Each point $(x + \Delta_1, y + \Delta_2)$ $\Delta_1 \geq 0, \Delta_2 \geq 0$, denotes an enveloping rectangle which is large enough to accommodate the module. Each point $(x - \Delta_1, y - \Delta_2)$ $\Delta_1 > 0, \Delta_2 > 0$, denotes an enveloping rectangle which is not large enough to accommodate the module. In Figure 3.3, the shape function $SF(x, y)$ indicates that only rectangles specified by points from the shaded area (edge inclusive) are able to accommodate the module. The shaded area is referred to as the feasible region.

Shape functions serve two purposes in floorplanning. The first is the prediction of the feasible dimensions of a layout block resulting from the combination of two or more modules in a sequence of horizontal and vertical merges specified in a slicing tree. An example is shown in Figure 3.4 which illustrates how the shape function of the resulting block can be computed from the shape function of the constituent modules.

Figure 3.4 (a) illustrates the bounding rectangle of a layout block which results from the merger of two modules, whereas Figure 3.4 (b) illustrates how the shape function of the resulting block can be derived.

Shape functions also serve a second purpose. Once a hierarchical floorplan exists, constraints imposed on layout block at the root level of the cut tree can be back

propagated into constraints on the individual constituents as shown in Figure 3.5. In the case shown, a height constraint is imposed on a block, which is a horizontal union of two modules. The width of the block can be obtained by calling the block's shape function predicted. Since it is a horizontal union, the widths allocated to the constituent modules are the same as the width of the block. Then, by calling the shape functions, the heights the spaces assigned to the modules are also obtained.

The shape functions fall into two categories: discrete or continuous as illustrated in Figure 3.6. Discrete shape functions summarize the geometrical requirements for elements that have a discrete set of feasible shapes, while continuous shape functions summarize the geometrical requirements for elements whose shapes can be adjusted continuously. As a special case, fixed modules have "L" shaped discrete shape functions.

In order to define shape functions, size limitations are necessities. *size limitations* give the minimum widths and heights of the object. Size limitations of layout blocks are also determined by its constituent modules. In the example of Figure 3.4 (a) config1, the minimum width of the block is equal to the minimum width of module A or module B, whichever is bigger. The minimum height of the block is equal to the sum of the minimum heights of module A and module B.

The shape function of a layout block can be expressed as:

$$y(x) = \begin{cases} \text{Max}\{F(x), \min_height\} & \text{if } x \geq \min_width \\ \infty & \text{if } x < \min_width \end{cases} \quad (EQ 5.1)$$

or

$$x(y) = \begin{cases} \text{Max}\{F^{-1}(y), \min_width\} & \text{if } y \geq \min_height \\ \infty & \text{if } y < \min_height \end{cases} \quad (EQ 5.2)$$

Property of shape functions:

A close study indicates that a shape function is non-increasing. $SF_y(x)$ or $SF_x(y)$ satisfies: if $x_1 \leq x_2$, then $y_1 \geq y_2$, where $y_1 = SF_y(x_1)$, $y_2 = SF_y(x_2)$, (or alternatively, $x_1 = SF_x(y_1)$, $x_2 = SF_x(y_2)$)

Proof: Assume that $x_1 \leq x_2$, but $y_1 < y_2$. Two contradicting results can be drawn out of this assumption. first, Since the rectangle of width= x_1 , height = y_1 is large enough to accommodate the module, the rectangle width= $x_2 \geq x_1$, height= y_1 is certainly large enough to accommodate the module. On the other hand, because (x_2, y_2) is on the curve and $y_1 < y_2$, the point (x_2, y_1) belongs to the non-feasible region, the rectangle width= x_2 , height= y_1 , is not large enough to accommodate the module. These two results about (x_2, y_2) are obviously contradicting. The only possibility is that the assumption does not stand.

Figure 3.2 Slicing trees corresponding to Figure 3.1 (a)

Figure 3.3 Shape Function Definition

Config 1

moduleA and moduleB
form a block by
a "horizontal union"

Config 2

moduleA and moduleB
form a block by
a "vertical union"

(a) Different configurations in forming a block

SF_A: shape function of module A

SF_B: shape function of module B

(1) : shape function for the block in Config. 1

(2) : shape function for the block in Config. 2

(b) Block's shape function prediction

Figure 3.4 Shape functions prediction

Constraint propagation in Config. 1 of Figure 3.4 (a)

h_block : the height constraint of the block (given)

w_block : the width of the block (calculated from SF_block)

Figure 3.5 Constraint propagation through shape functions

Figure 3.6 Two types of shape functions

3.2 Existing approaches to Block Placement

Placement methods fall into two classes, constructive and iterative. Constructive methods include cluster growth [6]-[8], partitioning-based placement [9]-[12], and analytical methods [13]-[18].

The cluster growth algorithm selects an unplaced module based on an evaluation function which measures signal net connectivity to modules already or not yet placed, and then decides the best position for it relative to the modules already placed. This process is repeated until all modules are placed.

The top-down partitioning algorithm divides the global modules into subsets such that the number of weighted connections among the subsets is minimized and the area of the modules in each subset is more or less uniform.

The analytical method models the placement problem as a linear or nonlinear optimization problem where all modules are moved simultaneously along a dimensional gradient in the placement search space.

Iterative methods include pairwise interchange [36], force-directed interchange and force-directed relaxation [36], simulated annealing [19] - [21], evolution [22], and the genetic method [23].

In the pairwise interchange method, two modules in a layout are interchanged and the cost function of the new layout is computed. If there is an improvement, the new layout replaces the old one; otherwise it does not.

Force-directed interchange and force-directed relaxation methods use an analogy to forces in springs to determine the position of the selected modules. These iteration methods accept the next configuration only if the objective function does not increase. This property may cause the results to fall into a local minimum.

Simulated annealing uses a probabilistic hill-climbing method to avoid local minimum. In simulated annealing, one cell is moved or two cells are exchanged in one iteration. If the move reduces the objective function, it is accepted. If the move increases the objective function, it is accepted or rejected based on the Boltzman distribution function.

Genetic methods [23] [24] are similar to simulated annealing. It maintains a large population of configurations during the iteration and each new configuration for the next generation is constructed from the two previous configurations.

3.3 Branch and Bound Technique

The general branch-and-bound algorithm can be applied to the placement problem. The solutions can be systematically searched according to certain cost evaluation which can be applied to either single solution or a solution set. If it is a single solution, the cost obtained represents the cost of the solution. If it is a solution set, the cost represents the lower bound of the solutions' costs.

The search can be actually represented by a tree structure called *decision tree*. The root of the decision tree represents the entire solution set. The nodes of the decision tree can be thought of as sets of solutions, and the children of a node n each represent a subset of the solutions that n represents. The leaves each represent single solutions.

In the example shown in Figure 3.8, leaves H, I, . . . , O represents feasible solutions, the root, node A, represents the entire feasible solution set {H, I, J, K, L, M, N, O}. The internal nodes represent a subset of solutions. For example, node C represents {L, M, N, O}, node F represents {L, M} and node G represents {N, O}

The selection of a solution is equivalent to traversing a branch of the tree and therefore this step is called *branch*. If at any node in decision tree a solution yields a cost which is greater than the currently known lowest, the search continues in another part of

the decision tree. This step is called *bound*. The algorithm actually prunes the decision tree which results in reduced computation. [2]

The search process can be illustrated by search trees. A search tree is part of the decision tree. Starting a tree with one node, which is the root of the decision tree, the search tree includes the children of a node once it is visited. Therefore, the search tree is updated whenever a node is visited. The search process finishes when the current search tree reaches a leaf node of the decision tree and this node has the lowest cost among the current search tree's leaf nodes. This node is considered to be the best solution. Figure 3.8 demonstrates the evolution of the search tree corresponding to the decision tree and search sequence shown in Figure 3.7.

Figure 3.7 Decision Tree

Figure 3.8 Search Trees

Chapter 4

Problem Definition

In this chapter, an optimization problem is formulated. The transistor placement problem is first reduced to a floorplanning problem and then is stated as a constrained combinational optimization problem.

4.1 Reduction of Transistor Placement Problem to Module Floorplanning Problem

The reduction of transistor placement problem to module placement problem is carried out by forming modules, which are rectangular objects, out of the transistors, and passing both geometrical and connectivity information to modules from their constituent devices.

Forming modules

Three kinds of modules are introduced: *basic modules*, *composite modules* and *fixed modules*. A basic module comprises a single device, while a composite module includes multiple devices. Both basic and composite modules may have flexible locations and shapes. A fixed module has both locations and shape fixed.

Fixed modules are formed by layout objects such as cell boundary, supply rails, feed through nets, etc. These modules have fixed positions already, hence they don't participate in the placement. However, they do affect the placement by their connection to modules participating in the placement.

Composite modules come from three sources: First, because of the possibility for MOS devices to share diffusion areas and the fact that they are usually closely connected, MOS transistors are merged in locally optimum chains, forming one module for each chain. Second, for those devices requiring close match, they are also merged to

a composite module, which is then treated as a flexible-shape object. Third, closely related devices are merged into a composite module in circuits with big numbers of elements to reduce the size of the placement problem.

All the other devices form basic modules by themselves.

Table 4.1 summarizes the three kinds of modules discussed above:

Name	# of devices	position	shape
basic module	1	flexible	fixed or flexible
composite module	>1	flexible	fixed or flexible
fixed module	0	fixed	fixed

Table 4.1

Summary of three kinds of modules

Extracting Information

Geometrical information includes shape function, minimum width and minimum height. For basic modules, they inherit all the geometrical information from the device they are formed from. For composite modules the geometrical information is defined in an optimistic way. For each width, the height of a composite module is defined to be the least height among the heights of all possible configurations of the devices inside that module. The minimum width (height) is defined to be the maximum of the minimum widths (height) of the devices inside. Fixed modules need not extract geometrical information, since they do not participate in the placement.

Connectivity information extraction involves all the modules. If two devices do not belong to the same module, each wire between them in the input netlist is mapped to a wire in the module connectivity graph. Otherwise, the wires between them are omitted in the module connectivity graph. Suppose a wire in the input netlist ends at a certain device, after it is mapped to the module connectivity graph, the wire will end at the module which this device belongs to. Figure 4.1 summarizes the reduction of a transistor placement problem to a floorplanning problem.

4.2 Definition for the Floorplanning Problem

The floorplanning problem can be formulated as a combinational optimization problem as follows:

Given three sets of modules: basic, composite and fixed, with:

- (i) shape functions for basic and composite modules,
- (ii) dimensions and locations for fixed modules,
- (iii) the connectivity graph $G(V, E)$ of all the modules.

Determine:

- (i) the shape for each basic or composite module,
- (ii) the exact position for each basic or composite module.

Such that:

- (i) the cost is minimized (see Section 4.2.1)
- (ii) all the constraints are satisfied (see Section 4.2.2)

4.2.1 Objective Cost Function

The true objective of the cell placement is to achieve 100% routing with minimum area. Such an objective is not mathematically well defined. Whether one placement is better than another is not known until actual routing is tried. For this reason the true goal is replaced by a simplified objective function which is easy to calculate by computer and provides a close approximation of result. While choosing the objective, it is expected that a solution to optimize the objective can lead to a minimum area but high routability.

In this thesis, a cost function is designed to act as the objective of the placement. The target of the placement algorithm is to minimize the area occupied by circuit. Therefore, the cost function reflects how much area will be covered by a placement. As far as

placement is concerned, area is covered in two ways. First, the modules are placed and occupy a certain amount of area. Second, the interconnecting wires between modules also cover a certain area, since they are usually routed between the modules, may not allow to intersect with each other, and require a minimum space in between.

Considering two sources that occupy area, the overall cost of a complete placement is a combination of area cost and wiring cost:

$$\text{Cost} = \text{AreaCost} + \lambda \text{ WeightedWirelength} \quad (\text{EQ 4.1})$$

where λ is a user specified constant. It can be considered as the constant that converts the weighted wire length to the amount of area that is taken by the wires.

AreaCost and *WeightedWireLength* for complete placements are explained in the rest of this section. For partial placements, both of them are close approximations and are explained in Section 5.2.

Area Cost

Area cost is equal to the area required by the modules alone, disregarding interconnection. In a standard cell layout, the height of the cell is given, the width of the cell can be calculated by calling the placement's shape function $SF_y(x)$.

$$\text{AreaCost} = \text{CellHeight} \times SF_y(\text{CellHeight}) \quad (\text{EQ 4.2})$$

$SF_y(x)$ of a complete placement can be constructed according to Section 3.1.2.

Weighted Wire Length

In order to get a better estimate of the area taken by wiring, different weights are put on the wire lengths.

$$\text{WeightedWireLength} = \sum \text{Weight} \times \text{WireLength} \quad (\text{EQ 4.3})$$

The weights give the user an option to specify what kind of wire are more preferable than the others. For example, while designing a standard cell layout, since the height of

the cell is given, it is much more desirable to have vertical wires than horizontal wires whenever possible, especially when the cell is wide.

There are four kinds of wiring receiving different weights. Suppose 1 unit length wire to *fixed modules* contributes as much cost as α units length of wire to *basic/composite modules*, and 1 unit length *horizontal* wire contributes as much cost as β units of *vertical* wire. The four kinds of wiring and the weights they receive are as follows:

type of wire	horizontal wire	vertical wire
wires to "fixed modules"	$\alpha\beta$	α
wires to "basic/composite modules"	β	1

Table 4.2 Weights on wire lengths

It is also possible to vary the weight of wires on a net by net basis in order to address critical nets.

Wire lengths are calculated optimistically, aiming at minimizing *WeightedWireLength*. The two ends of each wire are drawn as close as possible. As a result, all the wire ends are at the corners of modules.

4.2.2 Constraints

There are four kinds of constraints. Some of them are basic constraints. The others are user specified constraints:

i) Non-overlapping constraint. Although overlapping is allowed between fixed modules and others, it is not allowed among basic and composite modules.

ii) Shape constraints. Each module has its own shape function. In a placement, the rectangle space assigned to a module has to satisfy the shape function of the module. The rectangle is said to have satisfied the shape function of a module if the point (*width,height*) in xy-plane belongs to the feasible region defined by the shape function. (see Figure 3.3).

iii) The height of the cell. All the placements for the cells from the same library must have the same height. The height may differ from one library to another. That's why the height of the cell is a user specified constraint.

iv) The pre-placement constraints. Some of the modules have to be placed in a particular region. The user can specify the pre-placement region for each of those modules.

Chapter 5

Algorithm Description

5.1 Algorithm Overview

In the placement context, a solution means a complete placement. A partial placement is a set of solutions since it includes a number of complete placements. The search space is represented as a decision tree shown in Figure 5.1. Each node of the decision tree is a slicing tree representing a slicing floorplan. The root of the decision tree is a slicing tree with only one node, which is a cluster of all the modules. Each internal node is an incomplete slicing tree representing a partial placement. Each leaf node is a complete slicing tree representing a complete placement. In the decision tree, as descending from a node to its children, one more cut is applied to one of the leaf node of the slicing tree of this decision tree node.

Following the guidance provided by cost evaluation of branch-and-bound technique, the placement algorithm explores the search space which consists of a set of possible decompositions of the initial participating module set (including basic and composite modules) into hierarchical slicing floorplans.

Starting from the root of the decision tree, three steps are carried out iteratively until the best solution is found. The three steps are: search tree node selection, completeness checking, and search tree node expansion. The algorithm is summarized in Figure 5.2.

Search Tree Node Selection

The problem of the search tree node selection can be formulated as follows. Given the current search tree, each leaf is considered as a candidate for expansion. Select the leaf node of the search tree with the minimum cost.

Figure 5.1 Branch-and-Bound Decision Tree

Figure 5.2 Branch and Bound Placement Algorithm

Cost evaluation is the main problem in this step. The cost of a complete placement is explained in Section 4.2.1. The cost of a partial solution is an optimistic approximation of the cost function of complete placements, and is described in detail in Section 5.2.

Completeness Checking

Once a node in search tree is selected, the slicing tree represented by this node is checked to see if it is complete. If it is a complete slicing tree, it is returned as the solution to the placement problem. Otherwise, it is expanded to generate more candidates.

The process of completeness checking is straight forward. If each leaf node contains only one module, the slicing tree is complete. Otherwise, it is an incomplete slicing tree.

Search Tree Node Expansion

When a node in the search tree is selected, and it is found that it is an incomplete slicing tree, it is expanded to a number of children which are slicing trees with one more level of partitioning either in the vertical or the horizontal direction. That process of generating new slicing trees from a partial solution is as follows. Only one slicing tree node is selected for expansion. It is selected on the basis of area estimation and the number of modules. This strategy is used to limit the number of search tree nodes generated at each level. Once a slicing tree leaf node is chosen, its expansion is carried out by generating all possible two-way partitions of its modules and applying either a horizontal or a vertical cut between them. Thus, the expansion process results in a large number of new candidate slicing trees. (See Figure 5.1)

Not all these candidate slicing trees are retained as search tree nodes. Candidate slicing trees may be discarded for two reasons. First, some of these solutions may violate one of the pre-placement constraints. Secondly, a candidate solution is discarded if the target cell height is not feasible for the placement.

The rest of this chapter gives details related to algorithm implementation. Section 5.2 describes the cost function approximation for partial placements. Section 5.3 provides details on how the new slicing trees are generated. Section 5.4 explains how a topology represented by a slicing tree is converted to a placement. Section 5.5 addresses the effects of pre-placement constraints.

5.2 Cost Approximation for Partial Placements

Many of the search tree leaf nodes are incomplete slicing trees, representing partial placements. Therefore, the cost function of partial placements is critical in the step of “search tree node selection”. The cost function of a partial placement must be an optimistic approximation (close lower bound) of the cost function for complete placements described in Chapter 4.

The overall cost function specified in (EQ 4.1) still applies to a partial placement, except that both *AreaCost* and *WeightedWireLength* are optimistic approximations.

Area Cost Approximation (Slicing tree Leaf Node Shape Function Approximation)

Similar to a complete placement, the area cost of a partial placement is calculated by:

$$\text{AreaCost} = \text{CellHeight} \times \text{SFy}(\text{CellHeight}) \quad (\text{EQ 5.3})$$

SFy(x) can be constructed as described in Section 3.1.2. The problem here is that some of the leaf node don't have a shape function available, since leaf nodes of an incomplete slicing tree may have more than 1 module.

In order to combine the needs for accuracy and efficiency, a two-tier approach to compute leaf node shape functions is adopted. Given the slicing tree of a partial placement to evaluate, the algorithm proceeds as follows. If the number of modules in the leaf node is small ($\leq$ a user specified parameter *exhaustive-threshold*, typically 4), an

exhaustive enumeration of the all possible configurations of that layout block is carried out. For each height h of the layout block, its width w is defined to be the minimum width among all the configurations when they are constrained to the height of h :

$$w = SFx(y)/_{y=h} = \text{MIN} \{ SFx(y)/_{y=h}, \text{for each placement} \} \quad (\text{EQ 5.4})$$

If the number of modules renders an exhaustive search infeasible, an optimistic approximation is used instead. The approximation of the shape function of that leaf node is taken as following:

$$xy = \sum_{\text{Each_Module_Included}} \text{Minimum_Area} \quad (\text{EQ 5.5})$$

which is not as close a approximation as (EQ 5.4). The size limitations for the layout block in both cases are:

$$\text{min_width} = \text{Max} \{ \text{min_width for each module} \} \quad (\text{EQ 5.6})$$

$$\text{min_height} = \text{Max} \{ \text{min_height for each module} \} \quad (\text{EQ 5.7})$$

In this way, an accurate shape function is defined when the cluster has a small number of modules and a not-so-accurate but optimistic shape function is defined when the module number is big and exhaustive search is infeasible.

[Example 5.1] Leaf node shape function:

A leaf node is a cluster of module A and B, whose shape functions are shown in Figure 5.3 (a). The shape function of the leaf node obtained by exhaustive enumeration is illustrated in (b). The shape function of the leaf node obtained by optimistic approximation is illustrated in (c)

Figure 5.3 Leaf Node Shape Function Definition

Weighted wire length Approximation

In weighted wire length approximation for partial placements, zero-length wires are first identified and ignored in the rest of wire length estimation. A zero-length wire is a wire whose length might be zero. Before identifying zero-length wires, each wire is broken into two sections, horizontal and vertical. A vertical wire is classified as a zero-length wire if it is connecting two modules whose vertical relative positions are uncertain (i.e. one module's bottom edge is not necessarily higher than the other's top edge). A horizontal wire is classified as a zero wire based on the horizontal relative positions of the modules. Then the optimized locations are assigned to wiring pins with consideration of those non-zero-length wires only.

Before locating a pin inside a module, the module itself needs to be located first. A module and its wiring pins are first taken as the same dot which can move around inside the rectangular space assigned to the leaf node of the slicing tree. The problem of finding optimized positions for these dots can be formulated as a linear programming problem. The positions of dots are treated as variables, and, these rectangular regions they can move around inside are treated as feasible regions of the variables. By omitting zero-length wires, the objective, the weighted wire length, can be formulated as a linear function of the variables. As the result, the positions of these dots are located at the corners of the rectangular spaces when the weighted wire length is minimized.

After the dot representing a module is placed at a corner of the rectangular space, it is replaced by the actual module, i.e. the actual module is placed at the same corner as the dot, inside the rectangular space. The remaining problem is to locate pins inside the modules. For each non-zero-length wire, separate optimizations are carried out, with the objectives of minimizing the current wire's length. The two ends of each wire, are drawn as close as possible. As a result, all the wire ends are at the corners of modules. The shapes of the modules are adjusted to their limits after pins are located in order to reduce the wire length further. Figure 5.4 summarizes the process of wire length approximation.

Figure 5.4 Wire Length Estimation

5.3 Details of Search Tree Node Expansion

Since search tree node expansion means generate the child nodes of a search tree node by applying cuts to the leaf nodes of the slicing tree of this search tree node, and generate a set of new slicing trees, two questions arises: "which leaf node to choose to apply the cut?" and "how to apply the cut once the leaf node is chosen?" The following discussion is focused on solutions to these two questions.

Choosing Leaf

Only one slicing tree leaf node is selected for expansion. This strategy is used to limit the number of search tree nodes generated at each level.

Various targets can be adopted as criteria in choosing the leaf node to expand. The criteria does not affect the final solution, but it does affect the searching sequence. In order to be efficient, the target in this thesis is that the new slicing trees generated by expanding this slicing tree leaf node give closer cost estimation than the ones generated by expanding other slicing tree leaf nodes. The closer the cost estimation, the more nodes in the decisions remain unexplored before the best solution is found.

From the description about the cost approximation in Section 5.2, one can see that the leaf node with bigger number of modules, or larger area covered by modules, contributes more inaccuracy in the cost approximation of the partial placement. Therefore, the slicing tree node is selected on the basis of area estimation and the number of modules.

$$\text{Let: } \text{Score} = \alpha N + \beta S$$

where N is the number of modules in the cluster represented by the leaf node, S is the minimum area taken by the modules in the cluster. As the result, only the leaf with the highest score is chosen to apply cuts on. α and β are constants.

Expanding Leaf

Once a slicing tree node (a cluster) is chosen, its expansion is carried out by generating all possible two-way partitions of its modules and applying either a horizontal or a vertical cut between them. (Figure 5.1)

Suppose a node (a cluster) with n modules is chosen to be the leaf to expand, let M be the cluster represented by that node. In other words, M is the set of the n modules which belong to the cluster. The process of expanding leaf is as follows.

Step 1: Enumerate all the possible combinations of k modules out of M , for all k satisfying $1 \leq k \leq \lfloor n/2 \rfloor$. Let set *Comb* include all the combinations enumerated.

Step 2: For each combination of modules, say set A of modules, let set B to be its complement, i.e. $A \cup B = M$.

Step 3: for $1 \leq k < n/2$, generate four new slicing trees. Two of them apply a horizontal cut, with different positions of set A modules with respect to set B modules, i.e. Set A modules on the top or Set B modules on the top. The other new slicing trees applied a vertical cut, with different positions of set A modules with respect to set B modules, i.e. set A modules on the left or set B modules on the left.

Step 4: if n is an even number, $k=n/2 = \lfloor n/2 \rfloor$ is not covered by Step 3. In this case, for $k=n/2$, two new slicing trees are generated. One of them applied a horizontal cut to the cluster M , with set A modules on the top. The other new slicing tree applied a vertical cut to the cluster M , with set A modules on the left.

The process needs to treat the cases where n is even or odd differently in order that no identical new slicing trees are generated. If n is an odd number, B can not be a member of set *Comb*, since B has module number greater than $n/2$ and set *Comb* only includes combinations of k modules where $1 \leq k \leq \lfloor n/2 \rfloor$. If n is an even number, and

$k=n/2$, B is a member of set $Comb$, since B has $n/2$ modules (because A has $n/2$ modules and $A \cup B = M$). In this case, switching positions when a cut is applied will generate redundant slicing trees

[Example 5.2]

Given a slicing tree shown in Figure 5.5 (a) and a node selected to be the leaf node to expand. The children of the slicing tree have the structure shown in Figure 5.5 (b). The node selected has 4 modules: a , b , c and d . In this case, $n=4$ and $1 \leq k \leq \lfloor n/2 \rfloor = 2$. The possible combinations are enumerated appended to set $Comb$:

$$\{(a)(b)(c)(d)(ab)(ac)(ad)(bc)(bd)(cd)\}$$

For (a) , (b) , (c) and (d) , each combination generates four new slicing trees. For $(ab)(ac)(ad)(bc)(bd)$ and (cd) , each combination generates two slicing trees, because n is an even number. The expanded nodes from combination (a) and (ab) are shown in Figure 5.6.

In order to show the possibility of generating redundant slicing trees, Let's take the example of combination (ab) . According to Step 2, set $A = (ab)$ set $B = (cd)$. If it is treated equally and generated four slicing trees, the four expanded nodes will be:

1.	cut: horizontal	top: (ab)	bottom: (cd)
2.	cut: horizontal	top: (cd)	bottom: (ab)
3.	cut: vertical	left: (ab)	right: (cd)
4.	cut: vertical	left: (cd)	right: (ab)

Then when it comes to combination (cd) another four slicing trees are generated, the four expanded nodes will be:

5.	cut: horizontal	top: (cd)	bottom: (ab)
6.	cut: horizontal	top: (ab)	bottom: (cd)
7.	cut: vertical	left: (cd)	right: (ab)

8. cut: vertical left: (a b) right: (c d)

case 1 and 6, 2 and 5, 3 and 8, 4 and 7 are exactly identical.

5.4 Topology to Placement Conversion

A slicing tree only gives the topology of the placement. The locations and shapes of the modules in the placements represented by the new slicing trees also need to be specified in order to evaluate the cost.

If the slicing tree is incomplete, the placements is defined in a optimistic way. For leaves which represent clusters of modules, their positions and the shapes are assigned optimistically according to leaf node shape function definition described in Section 5.2. To start with, the shape function $SF_x(y)$ of the root of the slicing tree is constructed, and the width of the cell is obtained by:

$$CellWidth = SF_x(CellHeight) \quad (EQ\ 5.8)$$

Name the root to be *CurrentNode*. After defining the shape functions for the leaf nodes and *CurrentNode*'s dimensions and left bottom corner's position $H=CellHeight$, $W=Cellwidth$, $X_POS=0$, $Y_POS=0$, following recursive procedure is carried out until the all the leafnodes are reached:

Step 1: Shape function construction for child nodes: If *CurrentNode* is not a leaf node, name the two child node of *CurrentNode* to be *LeftChild* and *RightChild*. Construct the shape functions $SF_{left}(x, y)$ and $SF_{right}(x, y)$ for *LeftChild* and *RightChild* according to Section 5.5.2, based on leafnodes' shape functions which are already available. ($SF_{left}(x, y)=0$ can be expressed as $y=SF_{y_{left}}(x)$ or $x=SF_{x_{left}}(y)$, and same with $SF_{right}(x, y)=0$)

Step 2: Take different treatments based on the cut: If the cut represented by *CurrentNode* is horizontal cut, goto Step 3; If it is a Vertical goto Step 4; If it is nil, which means this is the leaf node of the slicing tree, goto Step 5.

Figure 5.5 Slicing Tree Generation

(1)~(4): subtrees expanded by combination (a)
 (5)~(6): subtrees expanded by combination (a b)

Figure 5.6 Leaf Node Expansion

Step 3: When it is a horizontal cut,

Step 3-1: set:

$$H_{left} = SFy_{left}(W),$$

$$X_POS_{left} = X_POS,$$

$$Y_POS_{left} = Y_POS,$$

Step 3-2: Start another two rounds of this recursive procedure with following settings:

(1) $CurrentNode = LeftNode$

$$X_POS = X_POS_{left},$$

$$Y_POS = Y_POS_{left}$$

$$H = H_{left}$$

$$W = W \text{ (inherit from its parent node)}$$

Let $(W_{left_act}, H_{left_act})$ be the actual dimensions returned for Left-Child

Step 3-3 set:

$$H_{right} = SFy_{right}(W),$$

$$X_POS_{right} = X_POS$$

$$Y_POS_{right} = Y_POS + H_{left_act}$$

Step 3-4 Start another two rounds of this recursive procedure with following settings:

$CurrentNode = RightNode$

$$X_POS = X_POS_{right},$$

$$Y_POS = Y_POS_{right}$$

$$H = H_{right}$$

$$W = W \text{ (inherit from its parent node)}$$

Let $(W_{right_act}, H_{right_act})$ be the actual dimensions returned for RightChild

Step 3-5: *Return $[\max (W_{right_act}, W_{left_act}), (H_{left_act} + H_{right_act})]$ the actual size of CurrentNode*

Step 4: *When it is a vertical cut*

Step 4-1: *set:*

$$W_{left} = SFx_{left}(H),$$

$$X_POS_{left} = X_POS,$$

$$Y_POS_{left} = Y_POS,$$

Step 4-2: *Run another round of the recursive procedure with following initial settings:*

$$CurrentNode = LeftNode$$

$$X_POS = X_POS_{left},$$

$$Y_POS = Y_POS_{left}$$

$$H = H \text{ (inherit from its parent node)}$$

$$W = W_{left}$$

Let $(W_{left_act}, H_{left_act})$ be the actual dimensions returned for Left-Child.

Step 4-3: *set:*

$$W_{right} = SFx_{right}(H),$$

$$X_POS_{right} = X_POS + W_{left_act}$$

$$Y_POS_{right} = Y_POS$$

Step 4-4: *Run another round of the recursive procedure with following initial settings:*

$$CurrentNode = RightNode$$

$$X_POS = X_POS_{right},$$

$$Y_POS = Y_POS_{right}$$

$$H = H \text{ (inherit from its parent node)}$$

$$W = W_{right}$$

Let $(W_{right_act}, H_{right_act})$ be the actual dimensions returned for RightChild.

Step 4-5: Return $[(W_{right} + W_{left}), \max(H_{left_act}, H_{right_act})]$ the actual size of CurrentNode.

Step 5: When it is a leaf node, assign its position (left bottom corner) to be $(X_POS$ and $Y_POS)$, and dimensions to be width= W , height= H . and return (W, H) as the actual dimensions of CurrentNode

5.5 The Effects of Pre-placement Constraints

This section analyze the constraints imposed on placement boundaries by pre-placement constraints, thereby explaining their effects on the prediction of shape functions, the conversion of topology to placement, and the validation of a new slicing tree to be a search tree node.

5.5.1 Constraints on Placement Boundaries Imposed by Pre-placement

Let the four edges of the placement boundary be *top*, *bottom*, *left*, *right*. If there is no pre-placement constraint, the possible location of these four edges can be anywhere as long as size limitations are met. But, it is a different situation when at least one module has pre-placement constraints.

[Example 5.3]

Two modules, A and B, having minimum width w_{amin} and w_{bmin} respectively, minimum height h_{amin} and h_{bmin} respectively, x direction pre-placement region $[x_{amin},$

x_{amax}] and $[x_{bmin}, x_{bmax}]$ respectively, y direction pre-placement regions $[y_{amin}, y_{amax}]$ and $[y_{bmin}, y_{bmax}]$ respectively.

For the placement edge *top*, its lowest possible location y_{min_top} and its highest possible location y_{max_top} are:

$$y_{min_top} = \max [(y_{amin} + h_{amin}) (y_{bmin} + h_{bmin})] \quad (EQ 5.9)$$

Similarly, we have:

$$y_{max_bottom} = \min [(y_{amax} - h_{amin}) (y_{bmax} - h_{bmin})] \quad (EQ 5.10)$$

$$x_{min_right} = \max [(x_{amin} + w_{amin}) (x_{bmin} + w_{bmin})] \quad (EQ 5.11)$$

$$x_{max_left} = \min [(x_{amax} - w_{amin}) (x_{bmax} - w_{bmin})] \quad (EQ 5.12)$$

Figure 5.7 illustrates the effect of pre-placement constraints on the limitation of feasible range of placement edges *top* and *bottom*. From the graph it is obvious that the placement containing module A and module B has minimum height $\geq (y_{min_top} - y_{max_bottom})$. Similarly the placement will also have minimum width $\geq (x_{min_right} - x_{max_bottom})$. The height and width limitations of the placements resulted from pre-placement constraints are called *PP_min_height* and *PP_min_width*. They may have negative values, which indicates that these limitations are not active. Generally,

$$PP_min_height = MAX \{ (y_{min} + h_{min}) \text{ for all the modules contained} \} \\ - MIN \{ (y_{max} - h_{amin}) \text{ for all the modules contained} \} (EQ 5.13)$$

$$PP_min_width = MAX \{ (x_{min} + w_{min}) \text{ for all the modules contained} \} \\ - MIN \{ (x_{max} - w_{amin}) \text{ for all the modules contained} \} (EQ 5.14)$$

Figure 5.7 Impact of Pre-placement Constraints on Placement Boundary

5.5.2 Pre-placement Constraints in Shape Function Construction

When dealing with placements with pre-placement constraints, the shape functions constructed are modified. The modification is as follows:

Suppose $PP_min_height = h_1$, $PP_min_width = w_1$, and the shape function before modification is:

$$y(x) = \begin{cases} \text{Max}[F(x), h_2] & \text{if } x \geq w_2, \\ \infty & \text{if } x < w_2, \end{cases}$$

The new shape function will be:

$$y = \text{Max} \left\{ SFy(x), \frac{\text{Max}(h_1, h_2) \times \text{Max}(w_1, w_2)}{x} \right\} \quad (EQ 5.15)$$

The modification of the curve is also explained in Figure 5.8

Figure 5.8 Impact of Pre-Placement Constraints on Shape Functions

5.5.3 Pre-placement Constraints in Conversion of Topology to Placement

The recursive process described in Section 5.4 is modified if there is pre-placement constraint present. The sizes and locations of any part of the placements are bounded by the pre-placement limitations. Any height (width) computed less than its PP_min_height (PP_min_width) is raised to PP_min_height (PP_min_width). Any location computed

(left bottom corner of a layout block) whose y-coordinate is less than that of the lowest point of the pre-placement regions is raised the same y-coordinate as that point. Any location computed whose x-coordinate is less than that of the left-most point of the pre-placement regions is raised the same x-coordinate as that point.

5.5.4 Pre-placement Constraints in Slicing Tree Validation

In order to be validated as a search tree node, a slicing tree has to satisfy all the pre-placement constraints for each module. The test is a comparison between the module's pre-placement region and the space assigned to a module or to the leaf node the module belongs to. Let A be the space assigned, Top_A , $Bottom_A$, $Left_A$ and $Right_A$ be the four edges of the rectangular space A . Pre-placement limitations on edges of module's placement region are calculated as (refer to Section 5.5.1):

$$ymin_top = y_{min} + h_{min}$$

$$ymax_bottom = y_{max} - h_{min}$$

$$xmin_right = x_{min} + w_{min}$$

$$xmax_left = x_{max} - w_{min}$$

The slicing tree pass the test if all of following are satisfied:

$$Top_A \geq ymin_top$$

$$Bottom_A \leq ymax_bottom$$

$$Left_A \leq xmax_left$$

$$Right_A \geq xmin_right$$

Chapter 6

Experimental Results

This chapter illustrates the behavior of the software on some small examples. Results are controlled by two groups of parameters. The first group of parameters are called result-control parameters. They reflect user requirements and affect the resulting placement. Result-control parameters are summarized in Table 6.1. The second group, search-control parameters control the behavior of the software, the searching process. When the optimal search is not degraded to a sub-optimal search, the search control parameters do not change the resulting placement. But different placements may be led to when an optimal search is degraded to a sub-optimal search because of the setting of one of these parameters: **blk_number_difference_bound**. Table 6.2 summarizes two most important such parameters. Table 6.3 gives the typical values for each of the parameters in Table 6.1 and Table 6.2

Experimental results on three circuits are demonstrated. Each example illustrates a different aspect of the software. For 4-input BiCMOS nor gate, the impact of the height of the cell and other parameters are illustrated. For I/O buffer (narrow configuration), the software's capability of dealing with pre-placement constraints and overlapping permission are demonstrated. 2-input ECL nor/or gate shows how the software handle circuits with relatively bigger numbers of modules.

parameter	explanation	default value
cellHeight	the height of the standard cell	nil
wire_length_to_area	the constant converts wiring cost to area cost	200
bus_penalty	larger value makes the wires to the "fixed units" cost more. If equals to 1, those wires are treated equally	1
horizontal_penalty	larger value makes the horizontal wires cost more. if =1, horizontal and vertical wires are treated equally	1

Table 6.1 Parameters of Group I: user requirements

parameter	explanation	default value
blk_number_difference_bound	the upper bound of the difference between the numbers of the modules of the two nodes when a cut is applied between them	1
exhaustive_threshold	the node shape function estimation will be carried out by exhaustive search when the number of the modules inside the leaf node $\leq$ *exhaustive_threshold*	1

Table 6.2 Parameters of Group II: software behavior control

parameter	value	explanation
bus_penalty	2	wires to the power rails cost twice as much as the other wires
horizontal_penalty	1	horizontal and vertical wires are treated equally
wire_length_to_area	200	In BATMIOS technology, 1 micron wire occupies about 2 micron ² area, i.e. 100 unit (1/100 micron) occupies about 20,000 unit ² , which results in *wire_length_to_area*=20,000/100=200
cellHeight	4000	40 micron
blk_number_difference_bound	4	in a 6-module circuit, this means the search will explore all the possible partitions)
exhaustive_threshold	3	if a slicing tree's leaf contains 3 or more modules, exhaustive search will be carried out in cost evaluation

Table 6.3 Typical settings for some of the parameters

6.1 4-input BiCMOS nor gate

6.1.1 Circuit Description

The schematic of the circuit is given in Figure 6.1 (a). The cell comprises four PMOS transistors, nine NMOS transistors, two NPN bipolar devices and one diode. From this initial netlist, a number of modules are formed. All PMOS transistors are merged to form a single rigid module called "pmos", whereas the NMOS transistors are merged into two chains "nmos1" and "nmos2". The diode and the two NPN transistors each form a module by themselves. The connectivity graph for this example is shown in Figure 6.1 (b). The shape functions of the modules are shown in Figure 6.1(c).

6.1.2 Impact of control parameters

cellHeight

As described before, this is a result-control parameter restricting the height of the standard cell. In this example, the software successfully gives the optimal placement according to the height given. When setting the other parameters to their typical values in Table 6.3, different placements result from different **cellHeight** values are obtained. (See Figures 6.1)

Figure 6.1 Problem formulation for 4-input BiCMOS nor gate

(a) *cellHeight*= 2820

(b) *cellHeight*= 3400

(c) *cellHeight*= 4000

(d) *cellHeight*= 4800

(e) *cellHeight*= 5600

(f) *cellHeight*= 7280

Figure 6.2 Impact of *cellHeight*: Placements Achieved for 4-input BiCMOS Nor Gate

****bus_penalty* and *wire_length_to_area****

These are two other result-control parameters. **bus_penalty** reflects the weight put on the wiring to *fixed modules* (power rails in this example). **wire_length_to_area** reflects the weight put on overall wiring length with respect to area cost. Although they are not active in the context of a small example, their controlling effect can still be seen. After setting the other parameters to their typical values, and gradually change the value of **bus_penalty**, following are the results. When **bus_penalty**=1,2,3,or 4, the placements remains the same as the one shown in Figures 6.1 (c) (see Figure 6.2 (a)). When **bus_penalty** ≥ 5 , a different placement is resulted (see Figures 6.2 (b)). The new placement reduces the wiring length to power rails by switching the positions of “nmos1” and “diode1”. Detailed reports on costs is listed under the placements in Figures 6.2. Bigger **bus_penalty** reduces wire length to power rails at the cost of either increasing placing area or increasing the wires among *basic or composite modules*. The cost reports for (a) and (b) indicate that increasing area cost is the case here.

Similar experiment is done to **wire_length_to_area**. When **wire_length_to_area**=50, 100,150, 200, 250 the placement remains the same as Figures 6.1 (c) (see Figure 6.2 (c)). When **wire_length_to_area** ≥ 300 , the same new placement is achieved as shown in Figures 6.2 (d). From the detailed cost report listed under the placement, the overall wiring length is reduced at the cost of increasing the placing area.

****horizontal_penalty****

horizontal_penalty gives the preference to vertical wires over horizontal_penalty. While the other parameters are set to their typical values, **horizontal_penalty** is set from 1 to 100. Three placements are achieved. (See Figure 6.3). Related parameter settings and the a detailed report is listed beside each placement. From the reports, it is obvious that a bigger **horizontal_penalty** reduces the horizontal wire length at the cost of either increasing the area cost or increasing vertical wire length.

****exhaustive_threshold****

****exhaustive_threshold**** is a search-control parameter. The setting of ****exhaustive_threshold**** saves memory at the expense of increasing the execution time, but it does not change the resulting placement. Therefore, this parameter can be used to balance the utilization of resources memory and CPU time.

After setting ****exhaustive_threshold**** to $n \in [1, 4]$ and the other parameters to typical values shown in Table 6.3, the same placement achieved is shown in Figure 6.1 (c). The relationship of ****exhaustive_threshold**** to execution time and memory requirement are shown in Figure 6.4.

****blk_number_difference_bound****

****blk_number_difference_bound**** is another search-control parameter. It controls the scale of the search. When ****blk_number_difference_bound**** $\geq (\text{total_module_number} - 2)$, the whole search space is covered. Otherwise, only part of the space is covered. In this way this optimal search may be degraded to sub-optimal search and a lot memory and execution time is saved. In this example, the same optimal solution (Figure 6.1 (a) (c) (d)) is achieved even when limiting ****blk_number_difference_bound**** to 1, but different placements result for Figure 6.1 (b) (e) (f), showing the parameter's effect of degrading optimal search to sub-optimal search. Figure 6.6 and Figure 6.7 illustrate how the parameter ****blk_number_difference_bound**** setting leads to a sub-optimal solution.

Figure 6.8 shows the relationship ****blk_number_difference_bound**** to execution time and memory requirement. When ****blk_number_difference_bound**** increases, the algorithm generally explores a bigger scale of searching space. Therefore, both execution time and memory generally increase when ****blk_number_difference_bound**** increases until it reaches the limit. Exceptions exist when the parameter increases, both execution time and memory stay the same. For example, in Figure 6.8, when the parameter increases from 2 to 3, the execution time and memory do not increase. The

reason is that for a 6-module cluster, the cut may separate 3 modules from the other 3 modules, or 2 modules from the other 4 modules, or 1 module from the other 5 modules. The difference between the numbers of modules are 0, 2 and 4 respectively. Increasing the parameter from 2 to 3 does not increase the scale of searching scale at all.

Figure 6.9 shows the layout of the 4-input BiCMOS nor gate designed manually and independently from the placement result obtained by the software. The placement of the layout is the same as the placement achieved in Figure 6.1 (c). This indicates that the placements achieved by the software for small circuits are very comparable to the placement designed manually.

(a) and (c)

(b) and (d)

(a) {
 cellHeight = 4000
 bus_penalty = 2
 wire_length_to_area = 200
 Area_Cost = 12920000
 Wire_Length_to_Power_Rails = 2870

(b) {
 cellHeight = 4000
 bus_penalty ≥ 5
 wire_length_to_area = 200
 Area_Cost = 13120000
 Wire_Length_to_Power_Rails = 2756

(c) {
 cellHeight = 4000
 bus_penalty = 2
 wire_length_to_area = 200
 Area_Cost = 12920000
 Weighted_Wire_Length = 14730

(d) {
 cellHeight = 4000
 bus_penalty = 2,
 wire_length_to_area ≥ 300
 Area_Cost = 13120000
 Weighted_Wire_Length = 14000

(a) → (b) *bus_penalty* ↗ Area_Cost ↗ Wire_length_to_Power_Rails ↘

(c) → (d) *wire_length_to_area* ↗ Area_Cost ↗ Weighted_wire_length ↘

Figure 6.3 Impact of *bus_penalty* and *wire_length_to_area*

(a) *BUS_PENALTY* = 2
 HORIZONTAL_PENALTY = 1
 WIRE_LENGTH_TO_AREA = 200
 AREA_COST = 12920000
 WEIGHTED_WIRE_LENGTH = 14730
 BUS_WIRE = 2870
 HORIZONTAL_WIRE = 1800
 VERTICAL_WIRE = 12930
 OVERALL_COST = 15866000

(b) *BUS_PENALTY* = 2
 HORIZONTAL_PENALTY = 3
 WIRE_LENGTH_TO_AREA = 200
 AREA_COST = 13120000
 WEIGHTED_WIRE_LENGTH = 14480
 BUS_WIRE = 2756
 HORIZONTAL_WIRE = 240
 VERTICAL_WIRE = 13760
 OVERALL_COST = 16016000

(c) *BUS_PENALTY* = 2
 HORIZONTAL_PENALTY = 7
 WIRE_LENGTH_TO_AREA = 200
 AREA_COST = 13600000
 WEIGHTED_WIRE_LENGTH = 13008
 BUS_WIRE = 3422
 HORIZONTAL_WIRE = 0
 VERTICAL_WIRE = 13008
 OVERALL_COST = 16201600

(a) → (b) → (c):

horizontal_penalty

horizontal_wire

Figure 6.4 Impact of *horizontal_penalty*

(a) "execution time" vs. **exhaustive_threshold** (b) "number of nodes" (memory) vs. **exhaustive_threshold**Figure 6.5 Impact of **exhaustive_threshold**

(a) $\begin{cases} *cellHeight=3400 \\ *blk_number_difference_bound*=2 \\ Cost=17789200 \end{cases}$

(b) $\begin{cases} *cellHeight=3400 \\ *blk_number_difference_bound*=1 \\ Cost=18468400 \end{cases}$

blk_number_difference_bound setting may degrade an optimal search to a sub-optimal search: (a) $\rightarrow$ (b) *blk_number_difference_bound* $\nearrow$ Cost $\nearrow$

Figure 6.6 Impact of *blk_number_difference_bound* (1)

blk_number_difference_bound setting may degrade an optimal search to a sub-optimal search: (a) → (b) *blk_number_difference_bound* ↘ Cost ↗

Figure 6.7 Impact of *blk_number_difference_bound* (2)

(a) “execution time” vs. *blk_number_difference_bound**

(b) “number of nodes” (memory) vs. *blk_number_difference_bound**

Figure 6.8 Impact of *blk_number_difference_bound**

Width = 41.1um

Height= 40um

The layout is designed manually by the author. The placement achieved in Figure 6.1 (c) has the same placement as the one in this layout.

Figure 6.9 A Manual Layout of 4-input BiCMOS nor gate

6.2 I/O Buffer (Narrow Pad Configuration)

The schematic of the I/O Pad circuit is given in Figure 2.6 on Page 12. The circuit is comprised of a PMOS transistor (forming module “pmos”), a NMOS transistor (“nmos”), a n-well Resistor (“Res”), a p-type diode (“p-diode”), three logic blocks (“logic”), a pad (“pad”), a lateral silicon controlled rectifier (“LSCR”), a snap back transistor (“SBT”). Among these modules, “pmos”, “nmos”, “Res” and “p-diode” are flexible modules, whose shape functions are generated by a model generator, while “logic”, “pad”, “LSCR” and “SBT” have fixed dimensions. In addition, there are five power rails running through the cell. They form five *fixed modules* by themselves: “VddCore”, “VssCore”, “VddIO”, “VssIO”, and “VssIO2”. As described in Chapter 5, those modules all have fixed dimensions and fixed locations which may overlap other modules. They only affect the placement by their connections to the modules. The connectivity graph for the circuit is shown in Figure 6.10. Figure 6.11 contains the shape functions for the modules. The locations and dimensions for the fixed modules are listed in Table 6.4.

Figure 6.10 Connectivity Graph $G(V, E)$ for the I/O Buffer circuit

Figure 6.11 Shape functions for the modules in the I/O buffer

fixed unit	width	height	center_position_x	center_position_y
VddCore	nil	3000	nil	1500
VssCore	nil	3000	nil	9500
VddIO	nil	6200	nil	15100
VssIO	nil	6200	nil	22600
VssIO2	nil	6200	nil	58500

Table 6.4 Locations and dimensions for fixed modules in I/O buffer

module	x_min	x_max	y_min	y_max
pad	nil	nil	28200	52900
LSCR	nil	nil	53500	59300

Table 6.5 Pre-placement restrictions (1/100 micron)

In order to avoid undesirable overlapping and to enforce the necessary overlapping with power rails, there are pre-placement restrictions on two modules: “pad” and “LSCR”. For “pad”, it must be placed between VssIO1 and VssIO2, with the minimum separation of 25 micron to the power rails. For “LSCR”, half of it must overlap with VssIO2. The pre-placement restrictions are summarized in Table 6.5

After setting the parameters to their typical values, the placement shown in Figure 6.12 is obtained. The placement successfully places the two modules with pre-placement requirements, “LSCR” and “pad”, inside the rectangular spaces assigned. The power rails are overlapped by other modules.

Figure 6.13 is a layout of the I/O buffer (narrow configuration) extracted from [43], whose placement and routing are done manually. Except “SBT” and “pdiodel”, each module has similar position and dimensions in the two placements. The positions of “pdiodel” and “SBT” are different in the two placements because the following two facts are not considered in the placement generated automatically:

- 1) Devices for ESD protection need to be placed as close as possible to the pad opening. “pdiodel” is one of such devices.
- 2) “SBT” is an N type MOS transistor, which can share diffusion area with the “nmos1” to save area.

It is expected that, after these two facts are considered, the software can achieve a placement which is closer to the placement designed manually.

The placement has following special attributes:

- (1) Pre-placement constraints are satisfied
- (2) Power rails are overlapped by the other modules

Figure 6.12 Placement Achieved for I/O buffer (narrow configuration)

The layout is extracted from [43], whose placement is designed manually. Comparing the placement in Figure 6.12 and the placement in this layout, most modules are placed very similarly in both placements, except "SBT" and "pdiode".

Figure 6.13 A layout for I/O Buffer (narrow configuration)

6.3 2-input ECL nor/or Circuits

Figure 6.14 Schematic of 2-input ECL nor/or circuit

The schematic of the 2-input ECL nor/or circuit is given in Figure 6.14. The circuit is composed of a number of bipolar transistors and 5 resistors. The numbers of devices in the circuits are relatively bigger, therefore, a number of composite modules are formed in order to reduce the number of objects handled. Transistors $Q1_ip_1$, $Q1_ip_2$ and $Q1_Vref_1$ form one composite module called $Q1_ip1_fblk$. $Q2_opb$ and $Rref_opb$, $Q2_op$ and Ref_op , $Q1_Vref_3$ and R_s form composite modules $Q2_opb_fblk$, $Q2_op_fblk$, and $Q1_vref3_fblk$ respectively. The module formulation and the connectivity graph are shown in Figure 6.15.

All the devices in the circuits are treated as rigid blocks but with two possible orientations. The feasible dimensions of the transistors and resistors are listed in Table 6.6. Note that x dimension can be either dimension1 or dimension2, while y is the other dimension that is not chosen by x . This notation is caused by the fact that all the devices are allowed to be placed in two orientations. The dimensions and locations for fixed modules are listed in Table 6.7. Figure 6.16 give the same information as Table 6.7 graphically.

After merging elements to composite modules, the shape functions of the objects that participate in the placement are shown in Figure 6.17. Figure 6.18 (a) is the placement containing composite modules, (b) is the detailed placement where all composite modules have been replaced by basic modules, and (c) illustrates the same placement as in (b) along with the wiring net. Figure 6.19 is the hand-crafted layout of the circuit from the cell library.

(a) Module Formulation

(b) Connectivity Graph $G(V, E)$

Figure 6.15 Problem Formulation for 2-input ECL nor/or Gate

module	dimension1	dimension2
Q2_opb	1240	1270
Q2_op	1240	1270
Q1_ip ₁	880	1270
Q1_ip ₂	880	1270
Q1_Vref ₁	880	1270
Ref_opb	550	1760
Ref_op	550	1760
R1_ip	3000	7550
R1_Vref ₁	3000	7550
Rs	2920	3000

Table 6.6 Dimensions of the devices

fixed module	width	height	x	y
Vdd	nil	2890	nil	16505
Vss	nil	2890	nil	5805
Vref1	nil	1000	nil	2620
Vref2	nil	2000	nil	1000
Vref3	nil	1000	nil	3740

Table 6.7 Dimensions and locations for fixed modules

Figure 6.16 ECL standard cell specification in BATMOS

Figure 6.17 Shape functions for modules in ECL 2-input nor/or Gate

Figure 6.18 Placement for 2-input ECL nor/or Gate

Width=133.9 μm

Height=179.5 μm

The layout is from BATMOS cell library. Comparing to the placement in Figure 6.18 with the placement in this layout, the former places all the devices (excluding "R1"s) between the two "R1"s, bringing the "Rs" to the bottom, "Q1"s to the middle, "Q2" and "Ref"s to the top.

Figure 6.19 Layout for 2-input ECL nor/or gate

Chapter 7

Conclusions and Recommendations for Further Research

7.1 Summary

A new application of an old algorithm has been presented in the context of the placement of transistors inside of logic and interface circuits composed of a mixture of MOS and bipolar devices. The principal objective of the algorithm is to minimize the area of synthesized layouts. The approach proposed in this thesis adapts earlier techniques initially used to solve the layout floorplan design problem. MOS devices are merged to form locally optimum chains of transistors which are then treated as rigid modules. Other devices such as diodes, bipolar transistors, resistors etc. are treated as flexible modules which are characterized by a shape function.

As to the placement of the modules, the algorithm is optimal under certain size restrictions and in terms of an accurate composite cost function which considers the area of the modules as well as the connectivity between them. The algorithm is made very efficient by the application of a close bounding function based on a locally exhaustive enumeration when such an enumeration is feasible.

Although the target is to develop an efficient algorithm for small scale circuits, the algorithm is not limited to small circuits. It can produce reasonable results for larger circuits by controlling the extent of search space exploration, or merging elements into composite modules. The new feature of the algorithm is that it can handle certain types of overlapping, and it can take pre-placement constraints specifying a rectangular region that certain object must be placed within.

A main limitation of this algorithm is that there is no detection of duplicate slicing trees generated by different slicing tree expansion processes. Another limitation is that

the algorithm is non-optimal even for circuits with small numbers of elements when the whole transistor placement is considered, since module creation is done quite randomly.

7.2 Contributions

The contribution of this thesis consists of:

- i) An existing method is adapted to a new problem. Branch-and-Bound method is adapted to the placement problem.
- ii) Topology construction and optimization in the floorplanning are merged. They are done in a parallel manner. Therefore the drawback in most floorplan algorithms caused by carrying out optimization only on the decided topology is avoided.
- iii) The definition of the shape function is adjusted so that it serves the purpose of floorplanning better. The properties of shape functions in propagating geometrical information between a slicing-structure placement and its constituent objects is discovered. They are summarized systematically and applied efficiently in the algorithm.
- iv) An optimistic but close cost function for a slicing structured placement is defined.
- v) The algorithm is successfully implemented as a stand alone placement software with reasonable user-interface and graphical outputs.

7.3 Recommendations for Future Research

Further research can be conducted to achieve a closer cost approximation by evaluating the optimistic wiring cost along with area cost during locally exhaustive enumerations. Also, a mechanism can be designed to locate modules to their optimal positions within the rectangular regions that are assigned to them in the final placement.

2 of/de 2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

The objective of the algorithm is to minimize the area. There may be other objectives such as minimizing timing delay, minimizing number of vias and contacts, etc. The adaptation of the new objective to the algorithm can be done very easily. Only formulating a new cost function and make the branch-and-bound algorithm evaluate the cost according to the new cost function is needed.

In the step of "Search Tree Node Expansion", the incumbent solution need to be added. The incumbent solution is a known but not necessarily the best solution. A slicing tree will not be validated as the search tree node for the third additional reason: its cost is greater than the incumbent solution. This adjustment will save execution time and memory, especially when a fairly good solution is known before running the program.

Partition of the netlist is another opportunity. Currently, the input netlist is partitioned manually to form modules. An automatic partitioning will definitely improve the speed and results.

Routing space estimation will be another useful addition, and it will significantly improve the placement by giving a better start point for any router. A rough idea is to leave a space along any cut applied to the slicing structure. The space left for a cut is proportional to the sum of the number of wires between the two new clusters formed by the cut, and the number of wires between the modules on two sides of current clusters (left and right sides if it is a horizontal cut, top and bottom if it is a vertical cut).

The freedom of overlapping in a placement problem can be extended by dividing participating objects into several groups. Inside each group, no overlapping is allowed. But it is free for objects to overlap with others from different groups. When a placement is evaluated, the connectivity among all the objects are considered. By doing this overlap can occur between any two devices if not prohibited by putting them into the same group. One of these two devices doesn't have to have fixed dimensions and locations as required in this thesis.

Glossary of Terms and Symbols

Aspect ratio	The ratio of the height and width of a module
CMOS	Complementary Metal Oxide Semiconductor.
Cluster	A group of modules
Connectivity Graph	A weighted undirected graph. The vertex set V contains one element for each modules. Each edge in the set E is weighted with the number of connections between two vertices u and v in V .
BiCMOS	VLSI technology with both options of bipolar and CMOS devices.
Branch-and-bound	An optimization technique that assumes all the feasible solutions are known. The solutions can be systematically searched according to certain cost evaluation which can be applied to either single solution or a solution set. If it is a single solution, the cost obtained represents the cost of the solution. If it is a solution set, the cost represents the lower bound of the solutions' costs.
Block placement	Placement with fixed-shape modules
Decision tree	Tree structure that denotes the whole searching space for a problem in Branch-and-bound algorithm.
Diffusion	Process of integrated circuit technology by which atoms move through the crystal lattice.
ECL	Emitter Coupled Logic.
ESD	Electrostatic Discharge -- Transfer of electrostatic charge between bodies at different electrostatic potentials caused by direct contact or induced by an induced field.
Floorplan	Placement of flexible-shape modules.
IC	Integrated circuit
Latch-Up	Phenomenon whereby parasitic bipolar transistors are introduced with a presence of a CMOS bulk. The result of this effect is the shorting of the V_{DD} and V_{SS} lines.

LSCR	Lateral Silicon Control Rectifier -- Layout structure used as ESD protection device for I/O pad circuit.
MOS	Metal Oxide Semiconductor.
MOSFET	Metal Oxide Semiconductor Field Effect Transistor.
NMOS	N-type MOSFET.
Placement	A part of layout design automation, where objects are assigned specific locations and shapes.
Pre-placement	User-specifications about restrictions of the placement for certain modules.
PMOS	P-type MOSFET.
Search tree	A part of decision tree in branch-and-bound algorithm. It includes all the nodes the search has reached so far.
Shape function	Function summarizing feasible shapes of rectangular space assigned to a module.
Slicing floorplan	a floorplan which can be obtained by recursively cutting a rectangle into two parts by either a vertical line or a horizontal line.
Slicing tree	A binary tree structure representing a slicing floorplan. Each node represents a rectangle in the floorplan. Non-leaf nodes also represents the cut applied between its child nodes.
Topology	Relative positions of objects.
VLSI	Very Large Scale Integrated circuit

References

- [1] J. Soukup, "Circuit layout," *Proc. IEEE*, vol. 69, Oct. 1981, pp.1281-1304
- [2] Naveed A. Sherwani "Algorithms for VLSI Physical Design Automation," Kluwer Academic Publishers, ISBN 0-7923-9294-9, 1993, pp159-190
- [3] S. Sahni and A. Bhatt, "The complexity of design automation problems," *Proc. 17th Design Automat. Conf.*, Jun 1980, pp. 402-411
- [4] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," San Francisco, CA: Freeman, 1979
- [5] M. Lefebvre, D. Skoll, "PicassoII: A CMOS Leaf Cell Synthesis System," *Proc. of MCNC International Workshop on Layout Synthesis*, May 1992
- [6] S. Kang, "Linear ordering and application to placement," *Proc. 20th Design Automat. Conf.*, 1983, pp. 457-463
- [7] C. M. Kyung, J. M. Widder, and D. A. Mlynski, "Adaptive cluster growth (ACG): A new algorithm for circuit packing in rectilinear region," *Proc. European Design Automat. Conf.*, Mar. 1990, pp. 191-195
- [8] M. Razaz and J. Gan, "Fuzzy set based initial placement for IC layout," *Proc. European Design Automat. Conf.*, Mar. 1990, pp. 655-659
- [9] M. Breuer, "Mincut placement," *J. Des. Automat. Fault Tolerant Comput.*, Oct. 1977, pp. 343-363
- [10] U. Lauther, "A min-cut placement algorithm for general cell assemblies based on a graph representation," *Proc. 16th Design Automat. Conf.*, June 1979, pp. 1-10
- [11] A. E. Dunlop and B. W. Kernighan, "A procedure for placement of standard-cell VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, 1985 pp. 92-98
- [12] P. R. Suaris and G. Kedem, "Quadrisection: A new approach to standard cell layout," *Proc. IEEE Int. Conf. Computer-Aided Design*, 1987. pp. 473-477
- [13] N. R. Quinn and M. A. Breuer, "A force-directed component placement procedure for PCB's," *IEEE Trans. Circuits Syst.*, vol. CAS 26, June 1979, pp. 377-388
- [14] C. Cheng and E. S. Kuh, "Module placement based on resistive network optimization," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, July 1984, pp. 218-225
- [15] L. Sha and R. W. Dutton, "An analytical algorithm for placement of arbitrarily sized rectangular blocks," *Proc. 22nd Design Automat. Conf.*, 1985, pp. 602-608
- [16] J. P Blanks, "Near-optimal placement using a quadratic objective function," *Proc 22nd Design Automat. Conf.* 1985. pp. 609-615

- [17] K. M. Just, J. M. Kleinhans and F. M. Johannes, "On the relative placement and the transportation problem for standard-cell layout," *Proc. 23rd Design Automat. Conf.*, 1986, pp. 308-313
- [18] J. M. Kleinhans, G. Sigl, and F. M. Johannes, "GORDIAN: A new global optimization / rectangle dissection method for cell placement," *Proc. IEEE Int. Conf. Computer-Aided Design*, 1988, pp. 506-509
- [19] S. Kirkpatrick, C. C. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 229, 1983, pp. 671-680
- [20] C. Sechen and K. Lee, "An improved simulated annealing algorithm for row-based placement," *Proc. IEEE Int. Conf. Computer-Aided Design*, 1987, pp. 478-481
- [21] L.K. Grover, "Standard cell placement using simulated sintering," *Proc. 24th Design Automat. Conf.*, 1987, pp. 56-59
- [22] R. M. Kling and P. Banerjee, "ESP: Placement by simulated evolution", *IEEE Trans. Computer-Aided Design*, vol. 8, Mar. 1989, pp. 245-256
- [23] K. Shahookar and P. Mazumder, "A genetic approach to standard cell placement using meta-genetic parameter optimization," *IEEE Trans. Computer-Aided Design*, vol. 9 pp. 500-511
- [24] James P. Cohoon, Shailesh U. Hegde, Worthy N. Martin and Dana S. Richards "Distributed genetic algorithms for the floorplan design problem," *IEEE Trans. on Computer-Aided Design*, Vol. 10 NO.4, April 1991, pp483-491
- [25] D. F. Wong and C. L. Liu "A new algorithm for floorplan design," *Proc. IEEE/ACM 23rd Design Automation Conf.*, 1986, pp. 101-107
- [26] E. Berkcan and E. Kinnen "IC layout planning and placement by dimension relaxation," *Proc. of 1985 IEEE International Conf. on Computer Design*, 1985, pp.449-452
- [27] L. Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," *IEEE Trans. on Computer-Aided Design*, Vol. 59, 1989, pp. 91-101
- [28] David P. La Potin, Stephen W. Director "Mason: a global floorplanning approach for VLSI design," *Information and Control*, Vol. 59 1983, pp. 91-101
- [29] Ting-Chi Wang and D. F. Wong "An optimal algorithm for floorplan area optimization," *Proc. 27th ACM/IEEE Design Automation Conf.*, 1990, pp. 180-186
- [30] Shmuel Wimer, Israel Koren and Israel Cederbaum, "Optimal aspect ratios of building blocks in VLSI," *Proc. 25th ACM/IEEE Design Automation Conf.*, 1988, pp. 66-72
- [31] Masato Mogaki, Chihei Miura and Hidekazu Terai "Algorithm for block placement with size optimization technique by the linear programming approach," *Proc. International Conf. on Computer-Aided Design*, 1987, pp. 80-83

- [32] Sai-keung Dong, Jingsheng Cong and C. L. Liu "Constrained Floorplan Design for Flexible Blocks," *Proc. International Conf. on Computer-Aided Design*, 1989, pp. 488-491.
- [33] R. H. J. M. Otten, "Layout Structures," *Proc. IEEE Large Scale Systems Symp.* 1982
- [34] R. H. J. M. Otten, "Automatic Floorplan Design," *Proc. 19th ACM/IEEE Design Automation Conf.* 1982 pp261-267
- [35] R. H. J. M. Otten, "Efficient Floorplan Optimization," *Proc. ICCD 83*, 1983, pp499-502
- [36] M. Hanan and J. M. Kurtzberg, "Placement techniques," *Design Automation of Digital Systems: Theory and Techniques*, M. A. Breuer, Ed. Englewood, NJ: Prentice-Hall, 1972, pp. 213-282
- [37] B. T. Preas and P. G. Karger, "Automatic placement: a review of current techniques," *Proc. 23rd Design Automat. Conf.* Jun 1986, pp. 622-629
- [38] S. Goto and T. Matsuda, "Partitioning, assignment and placement," *Layout Design and Verification*, T. Ohtsuki, Ed. New York: North-Holland, 1986, pp 55-97
- [39] G. De Micheli, A. Sangiovanni-Vincentelli, and P. Antognetti, Eds. Dordrecht, *The Netherlands: Martinus Nijhoff Publishers*, 1987, pp. 113-195
- [40] Sung-Soo Kim and Chong-Min Kyung "Circuit Placement on Arbitrarily Shaped Regions Using the Self-Organization Principles," *IEEE Trans. on Computer Aided Design*, Vol. 11 NO. 7 July 1992, pp844-854
- [41] Yoji Nishio, Fumio Murabayashi, and Shoichi Kotoku, "A Feedback-Type BiCMOS Logic Gate," *IEEE Journal of Solid-state circuits*, VOL. 24, NO. 5, Oct. 1989, pp.1360-1362
- [42] Lefebvre M.C., *Micronet/CMC BiCMOS Design Environment* - Chapter 10, 1993
- [43] K. M. Nguyen, "Module Generators for the Layout Synthesis of BiCMOS Input/Output Buffers" Thesis, October 1993, Carleton University

END

'21-07-94

FIN