

2009

Placement techniques for the physical synthesis of nanometer-scale integrated circuits

Natarajan Viswanathan
Iowa State University

Follow this and additional works at: <https://lib.dr.iastate.edu/etd>



Part of the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Viswanathan, Natarajan, "Placement techniques for the physical synthesis of nanometer-scale integrated circuits" (2009). *Graduate Theses and Dissertations*. 10758.

<https://lib.dr.iastate.edu/etd/10758>

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

**Placement techniques for the physical synthesis of
nanometer-scale integrated circuits**

by

Natarajan Viswanathan

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:
Chris C.-N. Chu, Major Professor
Charles J. Alpert
Maria Axenovich
Degang Chen
Randall L. Geiger
Gi-Joon Nam
Akhilesh Tyagi

Iowa State University

Ames, Iowa

2009

Copyright © Natarajan Viswanathan, 2009. All rights reserved.

TABLE OF CONTENTS

LIST OF FIGURES	vi
LIST OF TABLES	ix
LIST OF ALGORITHMS	xii
ACKNOWLEDGEMENTS	xiii
ABSTRACT	xiv
CHAPTER 1. GENERAL INTRODUCTION	1
1.1 Integrated Circuit Placement	1
1.2 Dissertation Organization	5
<u>PART I PLACEMENT AS A POINT TOOL</u>	8
CHAPTER 2. GLOBAL PLACEMENT	9
2.1 Introduction	9
2.2 Key Contributions of This Work	12
2.3 Quadratic Placement Methodology	13
2.4 Density Aware Module Spreading	16
2.4.1 Spreading of Standard-cells	17
2.4.2 Spreading of Macro-blocks	19
2.4.3 Handling the Density Target Constraint	22
2.4.4 Addition of Spreading Forces	23
2.5 Placement Restructuring via Force-vector Modulation	25
2.5.1 Spreading Forces During Global Placement	26

2.5.2	Force-vector Modulation	27
2.5.3	Advantages of Modulation	29
2.5.4	Effect of Force-vector Modulation	30
2.6	Iterative Local Refinement	32
2.6.1	Bin Structure for Iterative Local Refinement	33
2.6.2	Iterative Local Refinement for Simultaneous Spreading and Wire Length Minimization	33
2.6.3	Iterative Local Refinement for Handling Placement Blockages	35
2.6.4	Score for Module Movement During Iterative Local Refinement	37
2.6.5	Iterative Local Refinement for Placement Congestion Control	38
2.7	Multilevel Global Placement Framework	38
2.7.1	Clustering for Placement	42
2.8	The FastPlace and RQL Global Placement Algorithms	45
CHAPTER 3. LEGALIZATION		49
3.1	Introduction	49
3.2	Overview of the Mixed-size Legalization Algorithm	50
3.3	Legalization of Macro-blocks	51
3.3.1	Iterative Clustering Algorithm	52
3.3.2	Macro-block Legalization by Simulated Annealing	54
3.3.3	Effect of Macro-block Legalization	57
3.4	Legalization of Standard-cells	59
3.4.1	Slice Aware Bin-based Cell Movement	59
3.4.2	Slice-based Cell Legalization	61
3.4.3	Advantages of Slice-based Legalization over Bin-based Legalization	63
CHAPTER 4. EXPERIMENTAL RESULTS		66
4.1	Benchmark Circuits	66
4.2	Placement Results on the ISPD-2005 Benchmarks	66
4.3	Placement Results on the ISPD-2006 Benchmarks	68

PART II PLACEMENT IN A PHYSICAL SYNTHESIS FLOW **72**

CHAPTER 5. CLOCK CONSTRAINT AWARE TIMING-DRIVEN PLACE-

MENT		73
5.1	Introduction	73
5.1.1	Previous Work on Local Clock Tree Synthesis Methodology	74
5.1.2	Previous Work on Handling Clock Constraints During Timing-driven Placement	79
5.2	Key Contributions of This Work	80
5.3	Overview of Clock Constraint Aware Timing-driven Placement	81
5.4	Handling Latch Clusters During Timing-driven Placement	84
5.4.1	Initial Netlist Processing	84
5.4.2	LCB to Latch Net-weight Modulation	85
5.4.3	LCB Legalization	86
5.4.4	Latch Legalization	87
5.4.5	Advantages of the Clock Constraint Aware Timing-driven Placement Algorithm	87
5.5	Experimental Results	90
5.5.1	Latch Cluster Placement, Wire Length and Design Timing After Timing-driven Placement	91
5.5.2	Wire Length, Design Timing and Global Routing Congestion Analysis at the End of Physical Synthesis	94
5.6	Key Findings and Observations	96

CHAPTER 6. INTEGRATED TIMING OPTIMIZATION AND PLACE-

MENT		98
6.1	Introduction	98
6.1.1	Global Timing-driven Placement Techniques	98
6.1.2	Incremental Timing-driven Placement Techniques	103
6.2	Key Contributions of This Work	104

6.3	Overview of Integrated Timing Optimization and Placement	106
6.4	Critical Path Smoothing	108
6.4.1	Slack-based Critical Path Threading	108
6.4.2	Incremental Timing-driven Placement	111
6.4.3	Tunneling to Handle Placement Blockages	113
6.5	Congestion Mitigation and Wire Length Recovery	115
6.6	Incremental Timing Optimization	118
6.7	Slack Histogram Compression	119
6.8	The ITOP Algorithm	119
6.9	Experimental Results	122
6.9.1	Effect of Placement During ITOP	122
6.9.2	Effect of Tunneling During Critical Path Smoothing	123
6.9.3	Effect of Periodic Slack Histogram Compression	125
6.9.4	Physical Synthesis Flows for Comparison of Results	125
6.9.5	Results on High Performance Industrial Designs	127
6.10	Key Findings and Observations	132
CHAPTER 7. GENERAL CONCLUSIONS		135
BIBLIOGRAPHY		137

LIST OF FIGURES

Figure 1.1	Mixed-size circuit with placement blockages and significant white space	3
Figure 1.2	An example nanometer-scale physical synthesis flow	7
Figure 2.1	Quadratic placement approach and its analogy to a spring system . . .	13
Figure 2.2	Clique and star models for a multi-pin net	14
Figure 2.3	Bin structure and utilization during Density Aware Module Spreading for standard-cells	18
Figure 2.4	Bin structure and cell distribution during Density Aware Module Spreading for standard-cells	20
Figure 2.5	Bin structure construction for macro-block spreading	21
Figure 2.6	Spreading force addition on a module during Density Aware Module Spreading by using an on-chip fixed-point	24
Figure 2.7	Spreading force magnitude for all the modules in a circuit during one of the iterations of global placement	26
Figure 2.8	Force-vector modulation by nullifying the top 10% of spreading forces .	28
Figure 2.9	Effect of modulation on the module locations and net wire length . . .	31
Figure 2.10	Eight tentative moves for score calculation during Iterative Local Re- finement	34
Figure 2.11	Handling placement blockages during Iterative Local Refinement . . .	36
Figure 2.12	Bin size progression during Iterative Local Refinement	39
Figure 2.13	Multilevel global placement framework employed within FastPlace . . .	40
Figure 2.14	Multilevel global placement framework employed within RQL	41

Figure 3.1	Minimum perturbation floorplan realization problem	51
Figure 3.2	Iterative Clustering algorithm for macro-block legalization on an example circuit	55
Figure 3.3	Iterative Clustering algorithm for macro-block legalization on an example circuit (continued)	56
Figure 3.4	Effect of macro-block legalization during placement	58
Figure 3.5	A row slice for standard-cell legalization	59
Figure 3.6	Slice aware bin-based cell movement	61
Figure 3.7	Cell movement during slice-based cell legalization	62
Figure 3.8	Disadvantage of a bin-based legalization technique – satisfying bin capacity does not guarantee a legal placement	64
Figure 3.9	Advantage of a slice-based legalization technique – satisfying slice capacity guarantees a legal placement	65
Figure 5.1	Traditional physical design flow employing clock tree synthesis	75
Figure 5.2	Reduction in the local clock interconnect by following an enhanced local clock tree synthesis methodology	76
Figure 5.3	Latch huddle around a local clock buffer	77
Figure 5.4	Enhanced clock tree synthesis methodology for optimized local clock network	78
Figure 5.5	Latch clustering and LCB duplication during local clock tree synthesis	78
Figure 5.6	Latch clusters after clock constraint aware timing-driven placement . .	79
Figure 5.7	High-level flow for clock constraint aware timing-driven placement . . .	83
Figure 5.8	Clock constraint aware timing-driven placement on an example circuit	88
Figure 6.1	A physical synthesis flow using net-based timing-driven placement . . .	99
Figure 6.2	Total wire length and routing congestion at various stages of a physical synthesis flow employing net-weight driven timing-driven placement . .	101

Figure 6.3	A physical synthesis flow incorporating Integrated Timing Optimization and Placement	106
Figure 6.4	High-level flow for Integrated Timing Optimization and Placement . .	107
Figure 6.5	Slack-based Critical Path Threading	110
Figure 6.6	Tunneling through fixed macros during critical path smoothing	114
Figure 6.7	Bin density target during Congestion Mitigation	117
Figure 6.8	Scheduling the number of paths to be optimized during Integrated Timing Optimization and Placement	120
Figure 6.9	Effect of incremental placement during Integrated Timing Optimization and Placement. Worst slack progression during the iterative flow . . .	123
Figure 6.10	Effect of incremental placement during Integrated Timing Optimization and Placement. Figure of Merit progression during the iterative flow .	124
Figure 6.11	Physical synthesis flows for comparison of results.	126
Figure 6.12	Final routing congestion on the placements obtained from the No timing-driven placement, Net-weighted timing-driven placement and Integrated Timing Optimization and Placement flows	133

LIST OF TABLES

Table 2.1	Effect of force-vector modulation on the half-perimeter wire length . . .	31
Table 4.1	Statistics for the ISPD-2005 and ISPD-2006 placement benchmarks . . .	67
Table 4.2	HPWL comparison of FastPlace and RQL with existing academic placers on the ISPD-2005 placement benchmarks	67
Table 4.3	HPWL comparison of RQL with the top performing academic placers during the ISPD-2005 placement contest	68
Table 4.4	Runtime comparison of RQL and FastPlace with existing academic placers on the ISPD-2005 placement benchmarks	69
Table 4.5	HPWL comparison of RQL and FastPlace with existing academic placers on the ISPD-2006 placement benchmarks	70
Table 4.6	Scaled HPWL (S_HPWL) comparison of RQL and FastPlace with existing academic placers on the ISPD-2006 placement benchmarks . . .	71
Table 5.1	Statistics for a set of high performance industrial designs to test the clock constraint aware timing-driven placement algorithm	90
Table 5.2	Comparison of the LCB to latch distance statistics between timing-driven placement and clock constraint aware timing-driven placement .	91
Table 5.3	Comparison of the HPWL between timing-driven placement and clock constraint aware timing-driven placement after the timing-driven placement step	93

Table 5.4	Comparison of the design timing between timing-driven placement and clock constraint aware timing-driven placement after the timing-driven placement step	93
Table 5.5	Comparison of the HPWL between timing-driven placement and clock constraint aware timing-driven placement based flows at the end of physical synthesis	95
Table 5.6	Comparison of the design timing between timing-driven placement and clock constraint aware timing-driven placement based flows at the end of physical synthesis	95
Table 5.7	Global routing congestion analysis results of the timing-driven placement and clock constraint aware timing-driven placement based flows at the end of physical synthesis	96
Table 6.1	Design timing at various stages of a physical synthesis flow employing net-weight driven timing-driven placement	102
Table 6.2	Statistics for a set of high performance industrial designs to test the Integrated Timing Optimization and Placement algorithm	122
Table 6.3	Effect of tunneling during critical path smoothing. Design timing at the end of Integrated Timing Optimization and Placement	124
Table 6.4	Effect of periodic slack histogram compression during the iterative flow. Design timing at the end of Integrated Timing Optimization and Placement	125
Table 6.5	Worst slack comparison between the No timing-driven placement, Net-weighted timing-driven placement and Integrated Timing Optimization and Placement flows	128
Table 6.6	Timing Figure of Merit comparison between the No timing-driven placement, Net-weighted timing-driven placement and Integrated Timing Optimization and Placement flows	129

Table 6.7	Number of negative paths at the end of the No timing-driven placement, Net-weighted timing-driven placement and Integrated Timing Optimization and Placement flows	129
Table 6.8	Total wire length comparison between the No timing-driven placement, Net-weighted timing-driven placement and Integrated Timing Optimization and Placement flows	130
Table 6.9	Global routing congestion analysis on the final placements obtained from the No timing-driven placement, Net-weighted timing-driven placement and Integrated Timing Optimization and Placement flows	131
Table 6.10	Runtime comparison between the No timing-driven placement, Net-weighted timing-driven placement and Integrated Timing Optimization and Placement flows	132

LIST OF ALGORITHMS

Algorithm 2.1	The quadratic placement algorithm	12
Algorithm 2.2	Quadratic placement algorithm with force-vector modulation	29
Algorithm 2.3	Best-choice clustering with placement information	44
Algorithm 2.4	The FastPlace algorithm	47
Algorithm 2.5	The RQL algorithm	48
Algorithm 3.1	Mixed-size legalization algorithm	50
Algorithm 3.2	The Iterative Clustering algorithm	52
Algorithm 3.3	Macro-block legalization using simulated annealing	57
Algorithm 6.1	The ITOP algorithm	121

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Dr. Chris Chu, for giving me the opportunity to work under his guidance at Iowa State University. As a friend and a mentor, his motivating and enlightening discussions, timely suggestions and constructive criticisms have been a major driving force during the course of my research.

I am also grateful to Dr. Charles Alpert and Dr. Gi-Joon Nam at IBM Austin Research Lab., who were always open to discussion, irrespective of the number of times I knocked on their door during the course of a day! Without a doubt, I have immensely benefitted from the discussions, feedback and their friendship. Furthermore, I am thankful to Dr. Maria Axenovich, Dr. Degang Chen, Dr. Randall Geiger and Dr. Akhilesh Tyagi for taking their time to serve on my Ph.D. committee and providing valuable feedback on my work.

Thanks are also due to Pam Myers for her administrative and technical support, and to my friends in our research group for all their help during the course of my stay at Iowa State University.

From infancy to adulthood, the friendship and guidance of my parents has been a key factor in determining where I am today. I would like to express my heartfelt gratitude for their constant love, support and encouragement in all my endeavors. I would also like to thank my father and mother-in-law for their affection and constant prayers. Last but not the least, to my wife - my sounding board, cheerleader and pillar of strength. The last few years would have been extremely difficult without you being constantly by my side.

ABSTRACT

Placement is a critical component in the physical synthesis of nanometer-scale integrated circuits. Placement of circuit modules determines to a large extent interconnect length and routing resource demand. Interconnect length has a direct impact on the interconnect delay, which has become the determining factor of circuit performance in nanometer-scale process technology. In addition, interconnect length has a direct impact on the circuit power. Hence, the quality of the placement significantly affects the ability of a physical synthesis tool or designer to achieve design closure.

In this work, efficient and high quality placement techniques have been developed for the physical synthesis of multi-million gate integrated circuits in the nanometer regime. The focus of these techniques are: (a) global placement and legalization of mixed-size circuits to minimize interconnect length, circuit power and routing resource demand, and (b) incremental physical synthesis via integrated timing optimization and placement to achieve timing closure.

The effectiveness of the techniques is demonstrated by: (a) comparing them with existing approaches that perform integrated circuit placement, and (b) embedding them within a state-of-the-art industrial physical synthesis tool that is used in the design of high performance integrated circuits in the $65nm$ and $45nm$ process technology nodes.

CHAPTER 1. GENERAL INTRODUCTION

1.1 Integrated Circuit Placement

The placement problem can be defined as follows: Given a chip layout and a circuit consisting of modules with input and output terminals that are connected in a specific manner. Determine the locations of the modules, such that there is no overlap among them and a specified objective is optimized. The inputs to the placement problem are: (a) the *module descriptions*, consisting of the shapes, sizes and terminal locations on the modules, (b) the *netlist*, describing the interconnection between the terminals of the modules, and (c) the *chip-layout*, specifying the dimension, orientation and spacing information of the feasible locations on the chip where the modules need to be placed. For two-dimensional placement, the output is a list of x - and y -coordinates for all the modules such that no two modules overlap in the final layout.

Placement is a critical step in the physical design of nanometer-scale integrated circuits. It is a key factor in determining the performance of the circuit. The reason being, placement of circuit modules determines to a large extent interconnect length and hence, delay. With semiconductor process technology advancing into the nanometer regime, interconnect delay has become the determining factor of circuit performance. Hence, the quality of the placement significantly impacts the performance of the circuit. Placement also impacts the subsequent routing stage. A poor placement can lead to an increase in routing resource demand, leading to a difficult or sometimes impossible routing task. Among other factors, a bad placement can significantly increase circuit power. Power dissipation can also causes excessive heat, which leads to other problems such as variability and cost of cooling. More importantly, circuit power affects the reliability of the design, with low-power designs typically being more reliable.

Some of the key challenges faced by placement algorithms in the nanometer regime are:

- **Circuit Size:** Circuits today contain millions of modules that need to be placed. Hence, in addition to obtaining high-quality solutions, placement algorithms have to be efficient and scalable with circuit size. Efficient algorithms lead to a fast turn-around time and more importantly, permit designers to make iterative improvements to layouts to achieve the desired performance.
- **Mixed-size Circuits:** With a steady increase in the reuse of pre-designed macro-blocks like IP cores etc., circuits often contain a combination of a large number of macro-blocks and millions of standard-cells that need to be placed simultaneously. This design style known as mixed-size design, complicates the placement step due to the large difference in the sizes of the modules (Figure 1.1).
- **Placement Blockages:** Often designers fix the positions of certain critical modules like embedded memories and analog blocks. These components appear as fixed macros or blockages during placement and they present a highly fragmented chip layout in which the movable modules need to be placed (Figure 1.1).
- **White Space:** Circuits today have a large amount of white space. It is quite common for large-scale circuits to have chip utilizations that are as low as 20–30 %, indicating a large amount of white space (Figure 1.1). This is required to provide room for timing optimization transforms like buffer insertion and gate sizing, which either add new gates to the circuit or increase the area of the existing gates in the circuit. White space is also required for the subsequent routing stage. Hence, in addition to satisfying the primary design objectives, placement algorithms need to perform white space management.
- **Physical Synthesis for Timing Closure:** Placement can no longer be considered as an independent step in the design of high performance integrated circuits. Due to the dominance of interconnect delay in the nanometer regime, placement algorithms need to closely interact with timing optimization transforms like buffer insertion, gate sizing, and logic re-synthesis to achieve timing closure. Hence, placement algorithms need to operate

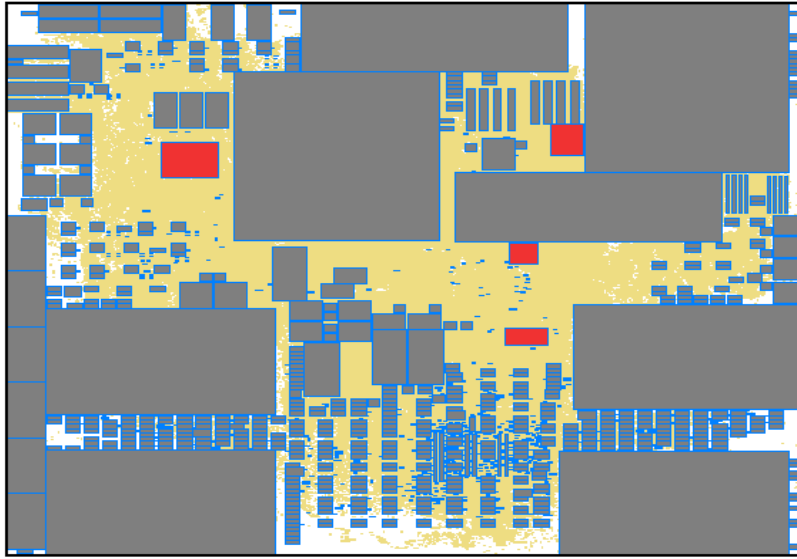


Figure 1.1 Mixed-size circuit with placement blockages and significant white space. The gray regions (medium shade) represent the placement blockages, the light-red boxes (dark shade) represent the movable macro-blocks, the light-brown boxes (pale shade) represent the standard-cells and the white regions represent the empty space in the design.

within a “physical synthesis” flow, where the placement and timing optimization steps are performed multiple times in an iterative manner until the circuit timing requirements are achieved (Figure 1.2).

- **Power Dissipation in Global Interconnect:** One of the major consumers of the dynamic (switching) power in nanometer-scale circuits is the global clock distribution network. The reason being, it usually switches during every clock cycle. The clock network is primarily determined by the placement of the clocked elements in the circuit (e.g., latches, flip-flops and other dynamic macros). Hence, in addition to handling the non-clocked (logic) modules in the circuit, placement algorithms need to incorporate special techniques to optimize the placement of the clocked elements. These techniques are required to minimize clock network wire length and consequently power, without degrading the clock skew or circuit timing.

Due to its complexity, placement is typically performed in three stages:

1. **Global Placement:** During global placement, the non-overlapping constraint among the modules is relaxed. The modules are then moved until they are reasonably distributed over the placement region. The output of global placement usually contains overlap among the modules that needs to be resolved.
2. **Legalization:** The legalization stage takes in as input a global placement solution and resolves the overlap among the modules to create a “legal” (overlap free) placement. Typically, legalization tries to perturb the modules by a small amount so as to preserve the characteristics of the input global placement solution.
3. **Detailed Placement:** Detailed placement tries to further optimize the placement objective (for e.g., wire length) by performing some local movement of the modules. The legality of the placement is usually preserved during this stage.

Among the three, global placement happens to be the most important stage of placement. The reason being, global placement determines the relative locations of the modules on a global scale. Whereas, the remaining stages of placement only make local changes to the module locations, thereby having a limited impact on the overall placement solution. Therefore, the final solution quality is heavily dependant on, and dictated by the quality of global placement. In addition, this is the most time-consuming stage of placement.

This dissertation describes the placement techniques that have been developed to address the key challenges of nanometer-scale integrated circuit placement. Specifically, the developed techniques are:

- Efficient and high quality wire length driven global placement techniques to handle multi-million gate mixed-size circuits containing numerous placement blockages in the chip layout. In addition to minimizing the wire length, these techniques also perform white space management to reserve space for the timing optimization and routing steps of physical synthesis.

- Effective legalization techniques for macro-block and standard-cell legalization during mixed-size circuit placement.
- Placement techniques that simultaneously optimize circuit timing (weighted wire length) and clock power (placement of clocked elements like latches and flip-flops) within a physical synthesis flow.
- An integrated timing optimization and placement technique to achieve timing closure within a nanometer-scale physical synthesis flow.

The global placement and legalization techniques outlined above have been implemented within two global placers namely, *FastPlace* [65,67,68] and *RQL* [66]. The remaining techniques have been embedded within a state-of-the-art industrial physical synthesis tool *PDS* [3,64].

1.2 Dissertation Organization

This section describes the organization of the remainder of this dissertation. This dissertation is essentially organized in two parts:

Part I: Placement as a point tool

This part describes the global placement and legalization techniques that are implemented within the *FastPlace* and *RQL* placement algorithms. In this part, placement is treated as a point tool. In other words, it is not combined with any of the other steps comprising a physical synthesis flow. The implemented techniques are evaluated on the primary objective for placement – total wire length of the circuit. Within a physical synthesis flow, the focus of this part is highlighted by the first shaded box in Figure 1.2. Specifically, Chapter 2 describes the global placement techniques that have been developed to handle multi-million gate mixed-size circuits. This is followed by Chapter 3 which describes the mixed-size legalization techniques to handle macro-block and standard-cell legalization. Finally, Chapter 4 compares the *FastPlace* and *RQL* algorithms with existing academic techniques that perform integrated circuit placement.

Part II: Placement in a physical synthesis flow

This part describes the following techniques that are implemented within a nanometer-scale physical synthesis flow: (a) placement techniques to handle clock network based placement constraints within a global timing-driven placement framework, and (b) techniques to achieve timing closure by integrating placement and timing optimization during physical synthesis. The focus of these techniques is highlighted by the second shaded box in Figure 1.2. Specifically, Chapter 5 describes a clock constraint aware timing-driven placement technique. This is followed by Chapter 6 that describes an integrated timing optimization and placement technique. This part also evaluates all the placement techniques described in this dissertation within a physical synthesis flow. It combines the techniques developed in Chapter 5 and Chapter 6 with wire length driven initial placement using *RQL*, and tests their effectiveness by performing physical synthesis on high performance integrated circuits in the *65nm* and *45nm* process technology nodes.

Finally, this dissertation concludes with some key insights and observations in Chapter 7.

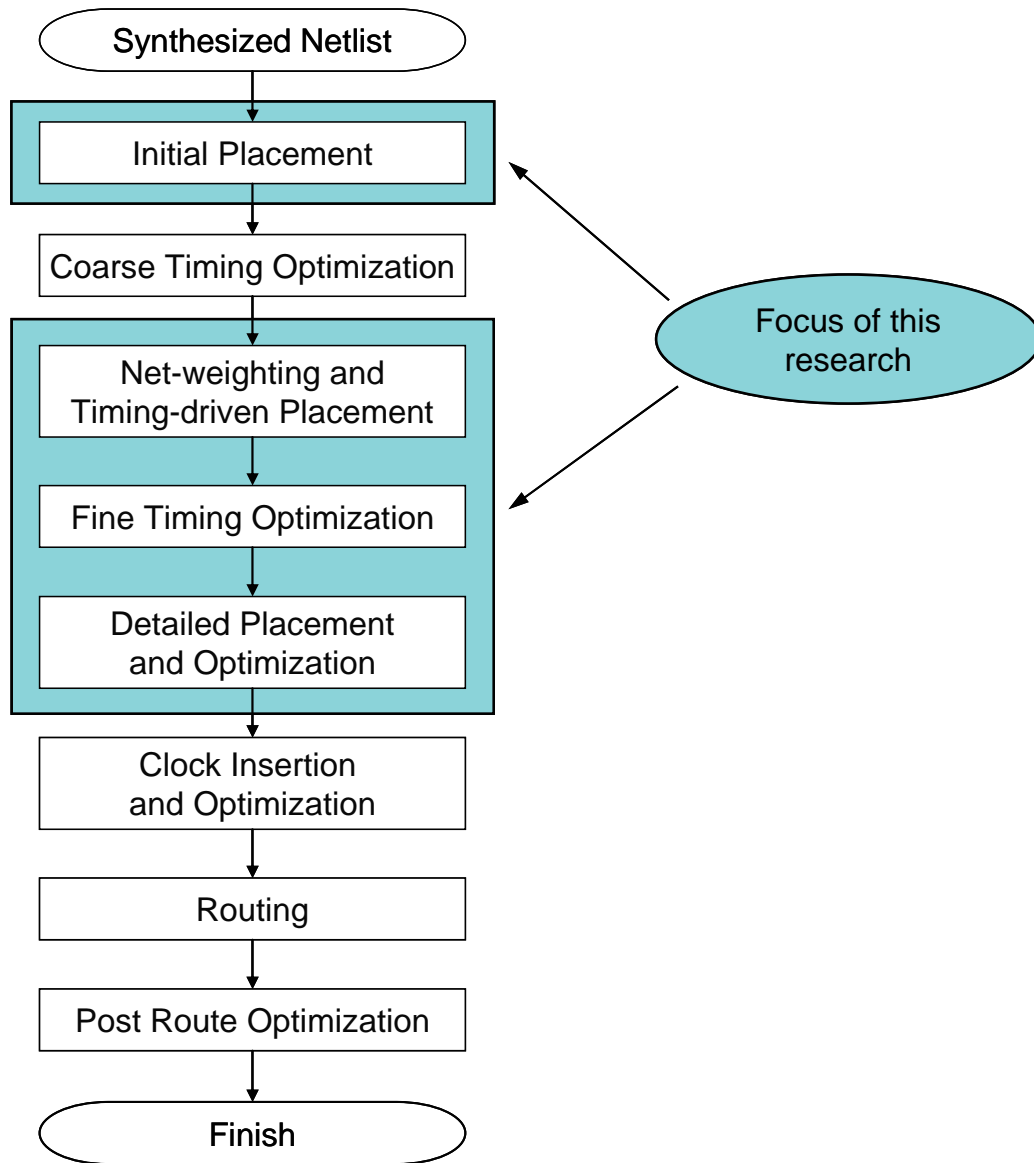


Figure 1.2 An example nanometer-scale physical synthesis flow.

PART I

PLACEMENT AS A POINT TOOL

CHAPTER 2. GLOBAL PLACEMENT

2.1 Introduction

Total wire length minimization happens to be the most common objective during global placement. Minimizing the total wire length typically has a good correlation with design routability as it can minimize the routing resource demand (though in some cases excessive packing of modules can cause routing hot-spots leading to inferior or sometimes impossible routing). It also has a direct impact on circuit performance. Since circuit delay in nanometer-scale integrated circuits is dominated by interconnect delay, minimizing the interconnect length leads to shorter wires with lesser delay. Finally, it also impacts circuit power. Circuit power is typically dominated by the dynamic power which is associated with the switching activity of the gates in the circuit. Minimizing interconnect length leads to a decrease in the capacitive load driven by the gates. This in turn decreases the dynamic power.

In addition to minimizing the total wire length of the circuit, global placement techniques also need to perform white space management. Circuits today have a large amount of white space that needs to be effectively distributed. The key reasons for doing so are to reserve space for timing optimization transforms like buffer insertion and gate sizing, and to improve the routability of the circuit by alleviating placement congestion. White space management is typically performed by setting a *density target* for the entire circuit. The density target is defined as follows: If we impose a regular grid structure over the placement region, then the density of a bin in this grid is defined as the ratio of the total area of the movable modules to the total available free-space within the bin. The density target essentially specifies the maximum permissible density within any bin in the regular grid structure. To satisfy the density target constraint, all the bins in the placement region must have a density less than or

equal to the density target.

This chapter describes efficient and high quality wire length driven global placement techniques to handle multi-million gate mixed-size circuits. In addition to minimizing placement wire length, these techniques also handle the density target constraint to alleviate placement congestion and effectively distribute the white space in the circuit.

Based on their approach, existing global placement techniques can be broadly classified as follows:

1. *Simulated Annealing*: The main advantage of simulated annealing is that it is a general heuristic that can be used to simultaneously optimize a combination of design objectives like wire length, timing, power, etc,. Simulated annealing based approaches typically obtain high-quality solutions, but placers using this approach normally suffer from extremely long runtimes. As a result, they cannot be used to place nanometer-scale circuits with millions of modules. TimberWolf [58,62] is a well known placer in this category.
2. *Top-down Partitioning*: Partitioning-based approaches recursively divide the circuit and the placement area using a min-cut objective to give a placement of the circuit. Recent placers in this category are Capo [6], Dragon [72], Fengshui [2] and NTUPlace2 [11]. Although partitioning-based placers are quite efficient, the min-cut objective does not accurately capture the real objective of minimizing the placement wire length, and this can lead to potentially inferior solutions.
3. *Hybrid Placement*: To handle mixed-size circuits, placers also use a hybrid approach where they combine partitioning, floorplanning and placement to simultaneously place the macro-blocks and standard-cells in the circuit. Representative examples of hybrid placers are Capo9.0 [1], Capo10.2 [57] and FLOP [75]. Since all these techniques primarily rely on partitioning to seed the subsequent floorplanning and/or placement steps, they can suffer from the same issues as the partitioning-based approaches described above.
4. *Analytical Placement*: The core of analytical placement approaches is an objective function that is minimized by methods of mathematical analysis. Based on their objective

function, analytical placers can be further divided into two categories:

- (a) *Nonlinear Objective*: Approaches in this category typically use an approximation of the linear wire length like the log-sum-exponential function [47], and nonlinear optimization techniques to perform placement. Examples in this category being APlace [32], mPL [8] and NTUPlace3 [12]. Results of the recently held placement contests [43,44] show that placers based on the log-sum exponential approximation can also obtain high-quality placement solutions, but they take a significant amount of runtime to converge to a solution.
- (b) *Quadratic Objective*: The objective function is quadratic and can hence be minimized by solving a system of linear equations. Representative examples of placers in this category are GORDIAN [36], Kraftwerk [21,61], BonnPlace [4], mFAR [28], FastPlace [65], FDP [35], hATP [45] and RQL [66].

Due to its inherent nature, quadratic placement is an attractive approach for designing efficient placement algorithms. The reason being, minimizing the quadratic objective is equivalent to solving a system of linear equations. This can be done quite efficiently using SOR or conjugate gradient based methods. However, quadratic placement faces two problems:

- The quadratic objective results in a placement with significant overlap among the modules. To resolve this overlap a spreading step has to be employed in conjunction with the quadratic solver. Thus, quadratic placement algorithms typically follow an iterative procedure comprising of: (a) the quadratic optimization step, which minimizes the wire length, (b) the spreading step, which resolves module overlap. This is shown in Algorithm 2.1. To overcome this drawback of quadratic placement, an efficient spreading technique is required that does not adversely impact the placement wire length.
- The quadratic objective is only an indirect measure of the linear wire length. Consequently, quadratic placement on its own does not yield the best possible result in terms of placement wire length. Hence, effective techniques are required to minimize the linear wire length within a quadratic placement framework.

Algorithm 2.1 The quadratic placement algorithm

- 1: solve initial quadratic program
 - 2: **repeat**
 - 3: spread modules to reduce overlap
 - 4: calculate spreading forces for all the modules
 - 5: add spreading forces to quadratic program formulation
 - 6: solve the quadratic program
 - 7: **until** (modules are distributed over the placement region)
-

This chapter describes the techniques that have been developed to address the key issues faced by quadratic placement algorithms. These techniques have been implemented in two state-of-the-art placers namely, *FastPlace* [65, 67, 68] an academic placer developed at Iowa State University and *RQL* [66] an industrial placer currently being used within the placement driven synthesis flow at IBM.

2.2 Key Contributions of This Work

The key contributions of this work in the area of global placement are:

- An efficient *Density Aware Module Spreading* technique to spread the modules during the early stages of global placement. This technique roughly maintains the relative ordering of the modules as obtained by solving the quadratic program in both the horizontal and vertical directions.
- An effective linearization technique called *Force-vector Modulation* that restructures the placement at a global scale to minimize wire length without sacrificing the degree of spreading.
- An *Iterative Local Refinement* technique to reduce the wire length based on the half-perimeter wire length measure. This technique is applied on a coarse global placement and is highly effective in simultaneously reducing the wire length and spreading the modules. It can also effectively handle placement blockages and placement congestion constraints.

- A *multilevel placement framework* using circuit clustering, that incorporates the above techniques during global placement so as to handle multi-million gate designs.

The rest of this chapter is organized as follows: Section 2.3 describes the generic quadratic placement methodology. This is followed by Sections 2.4 – 2.7 that describe the individual components in detail. Finally, Section 2.8 gives an overview of the *FastPlace* and *RQL* global placement algorithms.

2.3 Quadratic Placement Methodology

The quadratic placement approach uses the analogy of springs to model the connectivity between the modules of a circuit. This is depicted in Figure 2.1(a) which shows a circuit consisting of two movable modules i and j and one fixed module f . The circuit consists of two nets, net $\{f, i\}$ connecting modules f and i , and net $\{i, j\}$ connecting modules i and j . Figure 2.1(b) shows the corresponding spring system with the two-pin nets being replaced by the springs in the system.

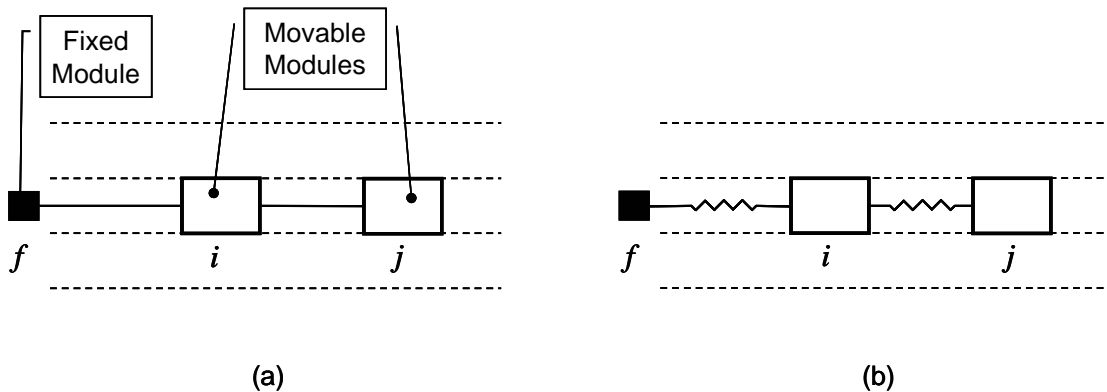


Figure 2.1 Quadratic placement approach and its analogy to a spring system. (a) Connections in a circuit (b) Corresponding spring system.

During quadratic placement, the total potential energy of the springs¹ is minimized to produce a placement solution. Minimizing the total potential energy of the springs results in

¹which is a quadratic function of their length

a force equilibrium state in the spring system. In this respect, quadratic placement is also commonly referred to as force-directed placement.

The circuit netlist that describes the connectivity between the modules is a weighted hypergraph $G = (V, E)$, where $V = \{v_1, \dots, v_m, v_{m+1}, \dots, v_n\}$ is the set of vertices representing the modules and $E = \{e_1, e_2, \dots, e_k\}$ is the set of hyperedges representing the connections or nets between the modules. In the netlist, modules v_1, \dots, v_m are assumed to be movable and modules v_{m+1}, \dots, v_n are assumed to be fixed. Also, each net $e_j \in E$ has a weight w_{e_j} that reflects the criticality of this net. Let, (x_i, y_i) represent the coordinates of the center of module $v_i \in \{v_1, \dots, v_m\}$. Then, a placement of the circuit is given by the two m -dimensional vectors $\mathbf{x} = (x_1, x_2, \dots, x_m)$ and $\mathbf{y} = (y_1, y_2, \dots, y_m)$.

In order to model the circuit by a spring system, the hypergraph needs to be transformed into a graph by using a suitable model. Circuits typically contain hyperedges of degree two or more (or in other words, two-pin and multi-pin nets). Hence, the transformation is equivalent to saying that each multi-pin net in the circuit needs to be transformed into a set of two-pin nets. This transformation can be performed using the traditional clique or star models (Figure 2.2) or the hybrid net model proposed in [65], where a clique model is used for two-pin and three-pin nets and a star model is used for nets with four or more pins. For the following discussion on quadratic placement, it is assumed that this transformation has been applied.

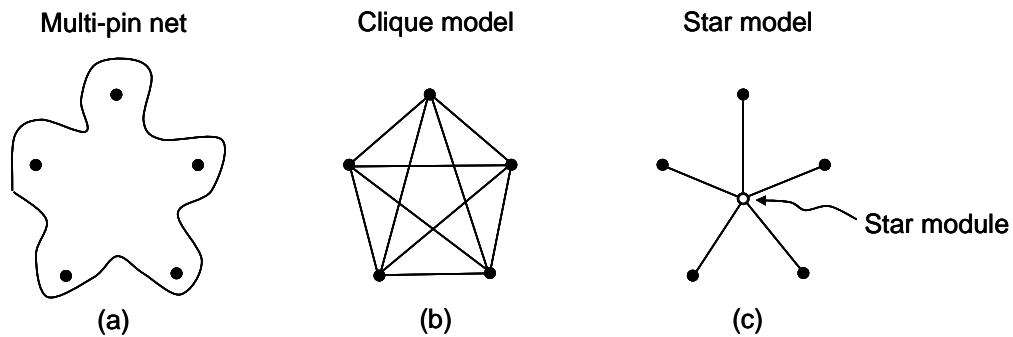


Figure 2.2 Clique and star models for a multi-pin net.

From Figure 2.1, consider the net between the movable modules i and j in the circuit. Let

W_{ij} be its weight. Then the cost of the net between the two modules is:

$$\frac{1}{2}W_{ij}\{(x_i - x_j)^2 + (y_i - y_j)^2\} \quad (2.1)$$

For the net between movable module i and fixed module f , the cost of the net is given by:

$$\frac{1}{2}W_{if}\{(x_i - x_f)^2 + (y_i - y_f)^2\} \quad (2.2)$$

The objective function that sums up the cost of all the nets can then be written as:

$$\Phi(x, y) = \frac{1}{2} \sum_{1 \leq i < j \leq n} W_{ij}\{(x_i - x_j)^2 + (y_i - y_j)^2\} \quad (2.3)$$

This can be written succinctly in matrix notation [23] as:

$$\Phi(x, y) = \frac{1}{2}x^T Cx + d_x^T x + \frac{1}{2}y^T Cy + d_y^T y + \text{constant} \quad (2.4)$$

where C is an $m \times m$ symmetric positive definite matrix and d_x, d_y are m -dimensional vectors. Since equation (2.4) is separable into $\Phi(x, y) = \Phi(x) + \Phi(y)$, only the x -dimension is considered for subsequent discussion, which is:

$$\Phi(x) = \frac{1}{2}x^T Cx + d_x^T x + \text{constant} \quad (2.5)$$

From expression (2.1), the cost in the x -direction between the two movable modules i and j is:

$$\frac{1}{2}W_{ij}(x_i^2 + x_j^2 - 2x_i x_j) \quad (2.6)$$

If c_{ij} is the entry in row i and column j of matrix C , then the first and second terms in expression (2.6) contribute W_{ij} to c_{ii} and c_{jj} respectively. The third term contributes $-W_{ij}$ to c_{ij} and c_{ji} .

From expression (2.2), the cost in the x -direction between movable module i and fixed module f is:

$$\frac{1}{2}W_{if}(x_i^2 + x_f^2 - 2x_i x_f) \quad (2.7)$$

The first term in expression (2.7) contributes W_{if} to c_{ii} . The third term contributes $-W_{if}x_f$ to the vector d_x at row i and the second term contributes to the constant part of equation (2.5).

The objective function (2.5) is then minimized by solving the system of linear equations represented by:

$$\frac{\partial \Phi}{\partial x} = Cx + d_x = 0. \quad (2.8)$$

Equation (2.8) thus gives the solution to the unconstrained problem of minimizing the quadratic objective function in (2.5).

2.4 Density Aware Module Spreading

Solving equation (2.8) essentially minimizes the quadratic objective function without considering the overlap among the modules. Therefore, the resulting placement has significant module overlap. To reduce the overlaps among the modules and spread them over the placement region, this work uses an efficient and effective *Density Aware Module Spreading* algorithm.

In [52] Ren *et al.* proposed a diffusion based spreading algorithm for placement legalization. Based on the density map of the placement region, their scheme uses the diffusion process to move the modules from high to low concentration regions. The technique proposed in [52] can be viewed as global diffusion, because modules will spread as long as there exists a density gradient among the bins in the current density map. However, using such an approach within global placement will cause excessive spreading and adversely impact placement wire length.

One method to control the degree of spreading within diffusion is to reduce the number of time steps [52] taken during module movement. In the limit, the module movement can be restricted to only its neighboring bins. Such a technique can be considered as a localized version of the diffusion algorithm. In principle, this technique is similar to one that equalizes the densities of adjacent bins by migrating the modules between them. This is the underlying principle of the *Density Aware Module Spreading* algorithm. By applying spreading over multiple iterations along with quadratic optimization, the modules are gradually distributed over the placement region.

During module spreading, an $M \times N$ (*rows* \times *columns*) regular bin structure (B) is initially imposed on the placement region. The area of each bin in the regular bin structure is such

that on average it can accommodate about four modules. Based on the current placement, for each bin i in B , the utilization of the bin (U_i) is then computed. U_i is defined as the ratio of the total area of all the modules overlapping with bin i to the bin area. Once the utilization has been determined, the modules are then spread over the placement region based on their respective bins and its current utilization. Module spreading is independent and similar in both the x - and y -dimensions. Hence, it is described by considering the case where the modules are shifted in the x -dimension. In addition, since circuits typically contain both standard-cells and macro-blocks, the subsequent discussion first describes module spreading as applied to standard-cells and then extends it to handle macro-blocks.

2.4.1 Spreading of Standard-cells

Module spreading for standard-cells is essentially a two step process: (a) construction of an irregular bin structure for each row in the original bin structure and (b) linear mapping of the cells from the regular to the irregular bin structure. Given below is a detailed description of these steps for one of the rows in the bin structure.

Consider a particular row in the regular bin structure. Let the utilization of all the bins in this row be as shown in Figure 2.3(a). Based on the utilization of all the bins in this row, an irregular bin structure reflecting the current bin utilization is constructed. This is as shown in Figure 2.3(b). To get the equation for the irregular bin structure, from Figure 2.3 let:

- U_i : Utilization of bin i in the regular bin structure.
- OB_i : Coordinate of the boundary of bin i in the regular bin structure.
- NB_i : Coordinate of the boundary of bin i in the irregular bin structure.

Then,

$$NB_i = \frac{OB_{i-1}(U_{i+1} + \delta) + OB_{i+1}(U_i + \delta)}{U_i + U_{i+1} + 2\delta} \quad (2.9)$$

Since the intuition behind module spreading is to equalize the utilization among adjacent bins, equation (2.9) changes the boundary of bin i such that it averages the utilization of bin i and bin $i + 1$. The reason for having the parameter δ is as follows: Let, $\delta = 0$ and $U_{i+1} = 0$,

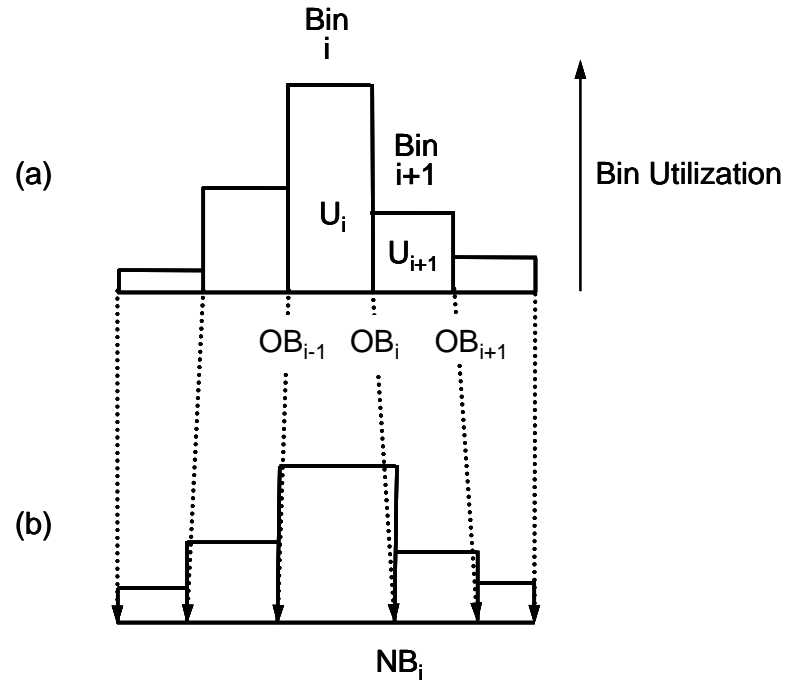


Figure 2.3 Bin structure and utilization during Density Aware Module Spreading for standard-cells. (a) Regular bin structure and initial bin utilization (b) Irregular bin structure and utilization after spreading.

then from equation (2.9) it can be seen that, $NB_i = OB_{i+1}$ and $NB_{i+1} = OB_i$, which results in a cross-over of the bin boundaries in the irregular bin structure. This in turn results in an improper mapping of the cells. To avoid this problem, the parameter δ is empirically set to a value of 1.5.

In the second step of module spreading, every cell present in a particular bin in the regular bin structure is then linearly mapped to the corresponding bin in the irregular bin structure. As a result of this mapping, cells in bins with a high utilization will spread out to reduce the utilization of the bin and the overlap among themselves. For performing the linear mapping of cells, if:

- x_j : x -coordinate of cell j in bin i before mapping.
- x'_j : x -coordinate of cell j in bin i after mapping.

Then,

$$x'_j = \frac{NB_i(x_j - OB_{i-1}) + NB_{i-1}(OB_i - x_j)}{OB_i - OB_{i-1}} \quad (2.10)$$

To have a tight control on the displacement of the cells during module spreading a *movement control* parameter α_x (< 1) is used. Once the coordinates of cell j have been obtained after mapping (equation 2.10), the actual distance moved by the cell is $\alpha_x |x'_j - x_j|$. During each iteration of global placement, the *movement control* parameter is inversely proportional to the maximum utilization among all the bins in the regular bin structure. It has a small value during the early stages of placement. As a result, cells will move by a very small distance during the initial placement iterations. When the placement is reasonably spread out (as reflected by the maximum bin utilization), the cells will not have a tendency to shift over large distances. The *movement control* parameter can then take a larger value to accelerate convergence.

Module spreading for standard-cells is illustrated in Figure 2.4, which shows the distribution of the cells before and after spreading for a particular row in the regular bin structure. From Figure 2.4(b), it can be seen that the cells in bin i have spread out to reduce the overlap amongst themselves and also decrease the bin utilization.

To spread the cells in the y -dimension an irregular bin structure is constructed for each column in the regular bin structure. This is followed by the linear mapping of the cells in the y -dimension.

2.4.2 Spreading of Macro-blocks

Typically, the area of a standard-cell is smaller than the area of a bin in the regular bin structure. Hence, the movement of a cell has an influence on the utilization of only its adjacent bins. Therefore, for the purpose of spreading, a cell is assumed to be completely contained by a bin if the center of the cell is located within the bounds of the bin. On the other hand, macro-blocks span multiple bins in the regular bin structure. Hence, the movement of a macro will influence the utilization of all the bins spanned by the macro. Therefore, to move a macro during module spreading, a larger region proportional to the size of the macro should be considered for constructing the irregular bin structure and subsequent linear mapping.

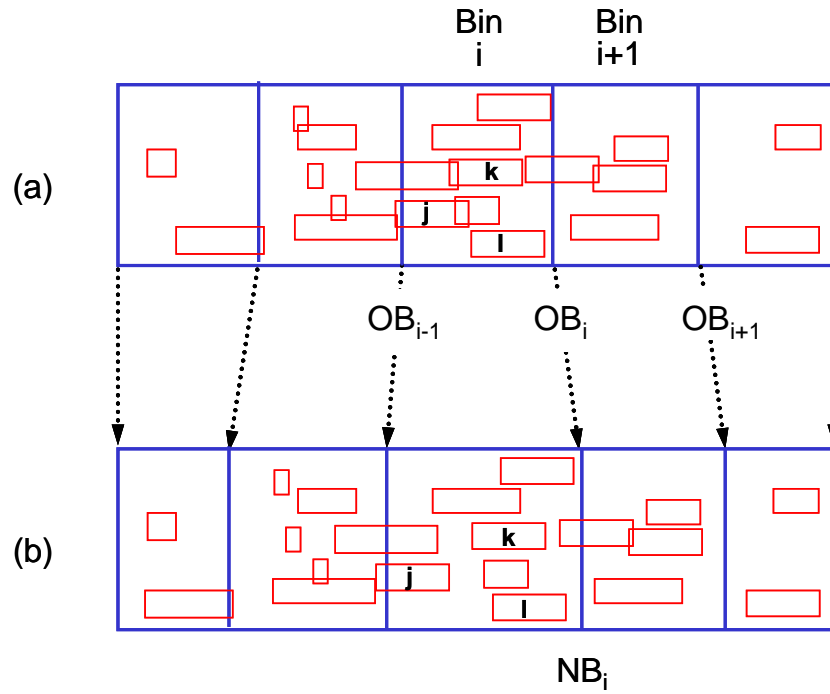


Figure 2.4 Bin structure and cell distribution during Density Aware Module Spreading for standard-cells. (a) Regular bin structure and cell distribution before spreading (b) Irregular bin structure and cell distribution after spreading.

Spreading of the macro-blocks follows the same two-step process as the standard-cells. The only difference being the construction of the irregular bin structure. Figure 2.5 illustrates the construction of the irregular bin structure for spreading in the x -dimension. From Figure 2.5(a), for the regular bin structure, let:

- x_span : Total number of columns spanned by the macro.
- y_span : Total number of rows spanned by the macro.
- N : Total number of bins spanned by the macro ($= x_span \times y_span$).
- OB_L : x -coordinate of the left boundary of the leftmost set of bins spanned by the macro.
- OB_R : x -coordinate of the right boundary of the rightmost set of bins spanned by the macro.

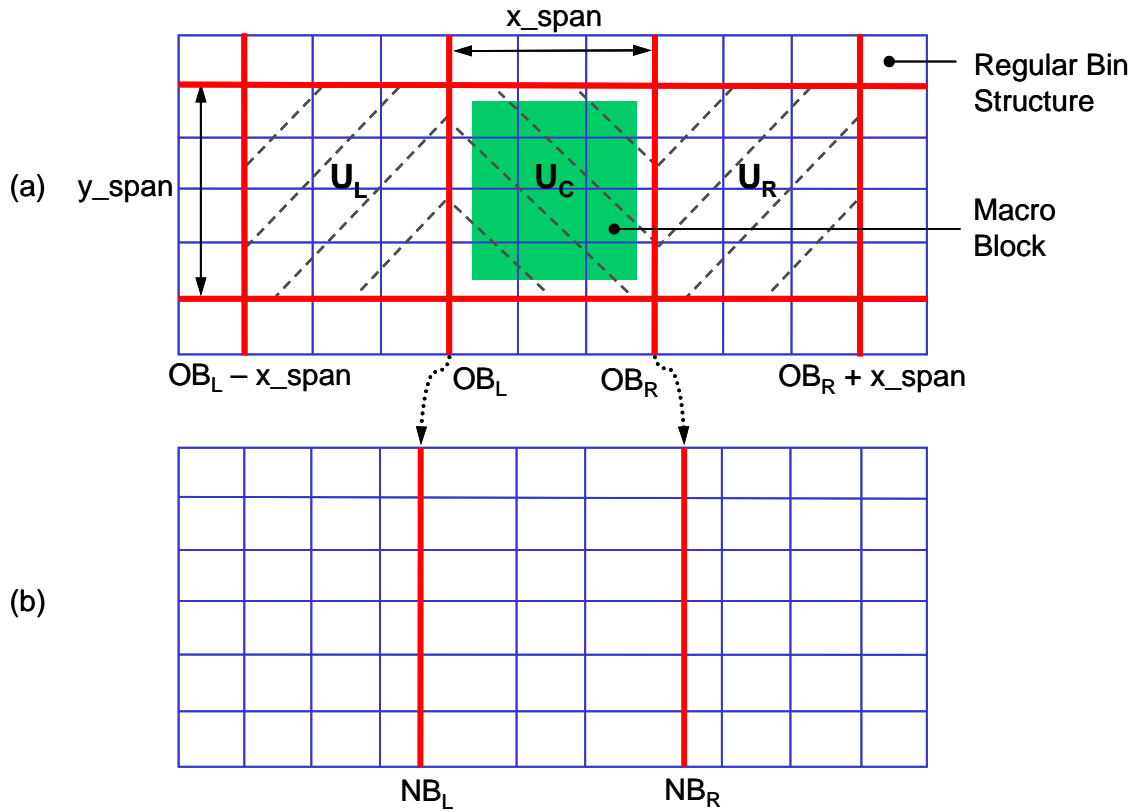


Figure 2.5 Bin structure construction for macro-block spreading. (a) Regular bin structure and utilization calculation based on the bins spanned by a macro-block (b) New bin boundaries for macro-block spreading.

- U_C : Sum of the utilizations of the N bins spanned by the macro (shaded region with lines to the bottom right).
- U_L : Sum of the utilizations of N bins to the left of the macro. (shaded region with lines to the bottom left).
- U_R : Sum of the utilizations of N bins to the right of macro. (shaded region with lines to the bottom left).

From Figure 2.5(b), for the unequal bin structure, let:

- NB_L : x -coordinate of the left boundary of the leftmost set of bins spanned by the macro.

- NB_R : x -coordinate of the right boundary of the rightmost set of bins spanned by the macro.

Then,

$$NB_L = \frac{(OB_L - x_span)(U_C + \delta) + OB_R(U_L + \delta)}{U_L + U_C + 2\delta} \quad (2.11)$$

$$NB_R = \frac{OB_L(U_R + \delta) + (OB_R + x_span)(U_C + \delta)}{U_R + U_C + 2\delta} \quad (2.12)$$

For performing the linear mapping, if:

- x : x -coordinate of the macro before mapping.
- x' : x -coordinate of the macro after mapping.

Then,

$$x' = \frac{NB_R(x - OB_L) + NB_L(OB_R - x)}{OB_R - OB_L} \quad (2.13)$$

2.4.3 Handling the Density Target Constraint

To handle the density target constraint and prevent excessive spreading, the module spreading algorithm uses an effective bin blocking scheme during the linear mapping step. Bin blocking is performed as follows: For every bin i in the regular bin structure a scaled density value s_i is determined. The scaled density is defined as the average density of an $x \times y$ window of bins around the bin under consideration. If s_i is lower than the density target for the bin, it is blocked during module spreading. Otherwise, the entire $x \times y$ window of bins is considered for spreading.

Please note, bin blocking does not alter the construction of the irregular bin structure. It only prevents modules from being mapped to the irregular bin structure during the linear mapping step of module spreading. Bin blocking effectively prevents unnecessary spreading in regions that are already below the density target. This in turn improves the placement wire length.

2.4.4 Addition of Spreading Forces

It should be noted that module spreading does not physically move a module to a new location. It only provides a “target location” for the module based on the current utilization map of the placement region. The new location for a module is actually determined during the subsequent quadratic optimization step. To enable module movement to the target location as determined by module spreading, a spreading force needs to be added to the module during each iteration of global placement.

A spreading force is added to a module by connecting it to an associated fixed-point via a pseudo-net having an appropriate net-weight. Fixed-point and pseudo-net addition for a module i in the circuit is depicted in Figure 2.6. The figure shows module i being connected to three other modules in the circuit netlist. The empty boxes in the figure represent the locations of the modules before spreading and the shaded box for module i represents its location after module spreading. From Figure 2.6, for module i if:

- $NF(i)$: The *Native Force* on i due to its connections with the other modules in the netlist.
- $SF(i)$: The *Spreading Force* imposed on i to move it to its target location obtained from module spreading.

The *Spreading Force* vector on the module is equal in magnitude and opposite in direction to the *Native Force* vector experienced by the module in its target location.

The location of the fixed-point and the weight on the pseudo-net are key elements that affect the stability of the spreading and the placement wire length. If the fixed-point is too close to the module, then the spreading force will dominate the native force during quadratic optimization. This will result in extremely slow spreading and also severely degrade the wire length. Adding the fixed-point to the chip boundary for large designs makes the spreading force behave as a “constant-force”. Constant forces are hard to control and may lead to a “blow-up” of the placement - creating “donuts” or empty regions in parts of the placement.

Therefore, an on-chip fixed-point is used (shown by the hollow circle in Figure 2.6 whose

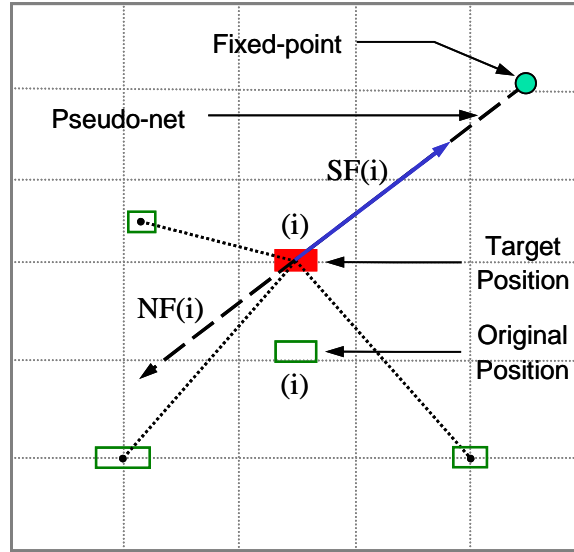


Figure 2.6 Spreading force addition on a module during Density Aware Module Spreading by using an on-chip fixed-point.

location is in between the two cases discussed above. The fixed-point to module distance is set proportional to the distance moved by the module during module spreading. Specifically, the fixed-point distance $D_{FP} = K1 + K2 \times f(module_displacement)$, where $K1 = K2 = 0.25(chip_diagonal)$ and $module_displacement$ is the distance between the original and target location of a module as shown in Figure 2.6. If the fixed-point falls outside the chip boundary, it is added at the intersection of the spreading force vector with the chip boundary. This ensures that modules are always placed within the placement region during the subsequent quadratic optimization.

The reason behind a constant and variable term in the fixed-point distance is two-fold: (a) if a module does not move during spreading, the constant term ensures that there is still a force on the module to hold it at its current location and (b) if a module moves by a large distance during spreading, then the distance of the fixed-point from the module should be proportional to this movement, so as to enable this large displacement during the subsequent quadratic optimization step.

Since a new fixed-to-movable connection is added to the netlist via the fixed-point, we

know from expression (2.2) and the subsequent analysis in Section 2.3, that only the diagonal of matrix C and the d_x and d_y vectors need to be updated for each module. Hence, it takes only a single pass of $O(m)$ time (where m is the total number of movable modules in the circuit) to regenerate the connectivity matrix for the next quadratic optimization step.

2.5 Placement Restructuring via Force-vector Modulation

As mentioned in Section 2.1, force-directed placement follows an iterative procedure that comprises of a non-linear optimization step which minimizes wire length, followed by a spreading step that adds “spreading forces” to reduce module overlap. One of the key problems with force-directed placement is the fact that spreading forces strictly enforce the relative ordering of the modules as obtained by solving the non-linear program. In other words, if a module is placed to the right of another module, the spreading forces will not allow the two to flip sides, even if it results in an improvement in the half-perimeter wire length. Although this property simplifies spreading, it can adversely impact the wire length. The reason being, relative cell ordering is often established during the early stages of placement, starting from the initial solution to the unconstrained non-linear program. This is when there exists significant module overlap. As a result, spreading forces are often arbitrary and are likely to do the most damage to wire length.

To offset this disadvantage, force-directed placers use a variety of local optimization techniques that change the relative ordering of the modules to improve the half-perimeter wire length. Representative examples of these are Local Cost Optimization [28], BoxPlace [35, 70] Relaxation Based Local Search [29, 30]. By nature, such techniques do not have a global view of the placement and can only improve the wire length in a local region. In addition, careful engineering is required to interleave these techniques with the non-linear optimization and spreading steps during global placement.

In contrast, this work uses a novel *Force-vector Modulation* technique to reorder the modules and restructure the placement at a global scale. Force-vector modulation directly optimizes the half-perimeter wire length during the non-linear optimization step and does not rely on

indirect methods (e.g., local optimization) that are interleaved with global optimization. In addition, force-vector modulation is a general technique that can be used within any force-directed placer, and is independent of the objective function.

2.5.1 Spreading Forces During Global Placement

In force-directed placement, a module is being acted upon by two conflicting forces: (a) the *Native Force* that attracts it to the other modules in the circuit due to the connections in the netlist, and (b) the *Spreading Force* that tries to pull it to the sparse areas within the placement region. During force-directed placement, a careful balance between the native and spreading forces is required to achieve a good trade-off between the twin objectives of spreading and wire length minimization.

To better understand the nature of the spreading forces, and their relationship with the placement wire length, the spreading force magnitude for all the modules in the circuit was tracked over successive iterations of global placement. Figure 2.7 shows a plot of the spreading force magnitude for all the modules in the circuit during one of the iterations of global placement.

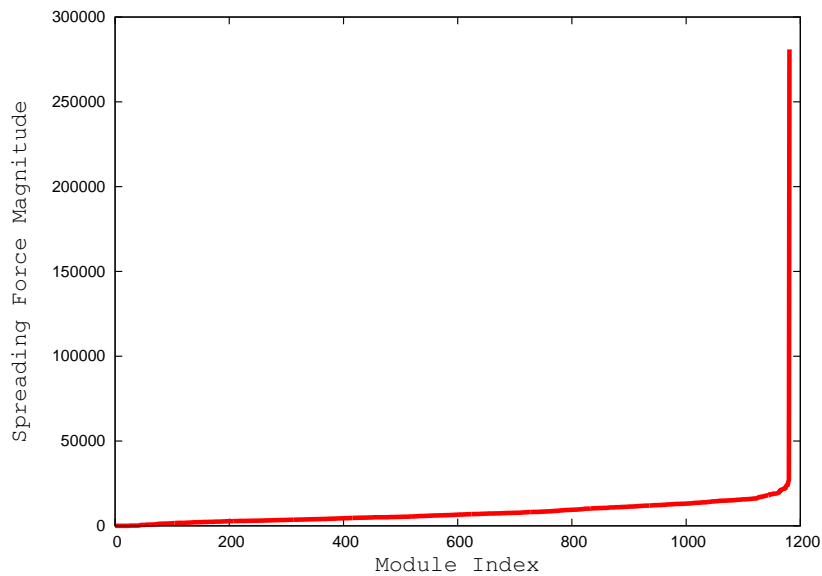


Figure 2.7 Spreading force magnitude for all the modules in a circuit during one of the iterations of global placement.

From the plot, it can be seen that a small fraction of the modules have an extremely high spreading force magnitude. From Section 2.4.4, we know that the spreading force on a module is directly proportional to the native force acting on the module. Hence, a high spreading force on a module could imply one or more of the following: (a) the module belongs to a large number of nets in the circuit, (b) it could belong to net(s) that have a high net-weight and are hence critical, or (c) the module is connected to distant modules like boundary IOs. In all these cases, this module has a significant attraction to one or more of the other modules in the circuit.

It was empirically observed that retaining the full spreading force magnitude for the modules with an extremely high spreading force resulted in a placement with a very high wire length. Instead, by controlling the spreading force magnitudes for these modules, a substantial decrease in the wire length was observed without sacrificing the degree of spreading.

2.5.2 Force-vector Modulation

The force-vector modulation technique, modulates the spreading force vectors within force-directed placement. Modulation of spreading forces results in a modified distribution of the spreading force magnitudes. As a result, the modules no longer move to the locations that were originally dictated by the spreading forces. In fact, by carefully modulating the spreading forces, the modules can be moved to their optimal locations as determined by the objective function being used during global placement.

One of the methods for modulation, that is also used in this work, is as follows:

- Sort the modules in a non-decreasing order of their spreading force magnitude during each iteration of global placement.
- Pick a fraction of the modules having a very high spreading force.
- Nullify the spreading force on these modules for the subsequent non-linear optimization.

This is shown in Figure 2.8 that depicts a plot of the modified spreading force magnitudes due to nullification during one of the iterations of global placement. Typically, only a small

fraction of around 5 – 10% of the modules are picked for nullification during each iteration of global placement. The reason being, nullifying the forces on a large number of modules could significantly increase module overlap due to the modules being placed very close to each other (or for that matter even on top of each other) during the non-linear optimization step. As the placement progresses, the number of modules picked for nullification can be varied to achieve a good trade-off between placement wire length and spreading.

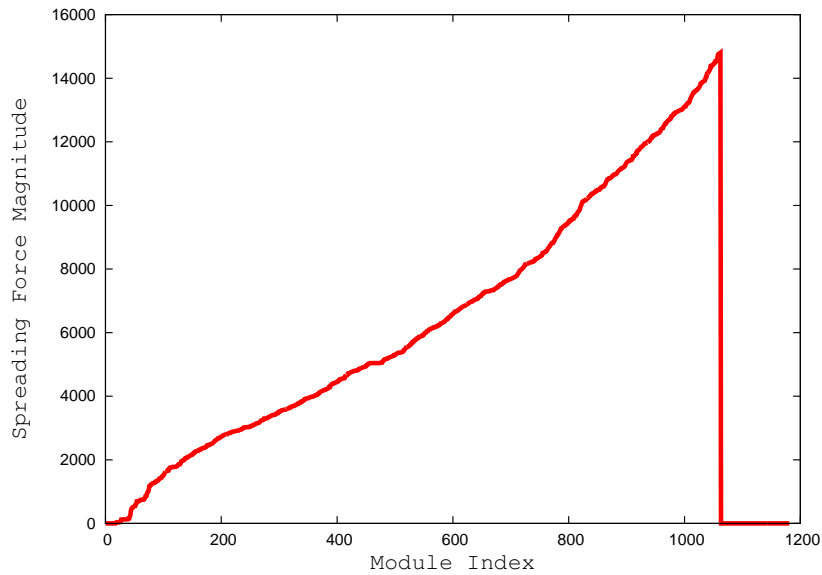


Figure 2.8 Force-vector modulation by nullifying the top 10% of spreading forces.

Due to the nullification of the spreading force on a small fraction of the modules, the only force acting on the modules is the native force. As a result, the locations of these modules after the subsequent non-linear optimization step will be determined solely due to their connections to the other modules in the netlist. This is equivalent to optimizing the wire length objective with no spreading constraints on these modules. As a result, they will be placed in their optimal locations as determined by the objective function used during global placement.

Since this work uses a quadratic objective function for wire length minimization, Algorithm 2.2 now gives the modified version of the quadratic placement algorithm (Algorithm 2.1) with force-vector modulation. As it can be seen, the only difference between the two algorithms is

the addition of steps 5 and 6 in Algorithm 2.2.

Algorithm 2.2 Quadratic placement algorithm with force-vector modulation

- 1: solve initial quadratic program
 - 2: **repeat**
 - 3: spread modules to reduce overlap
 - 4: calculate spreading forces for all the modules
 - 5: rank modules based on the spreading force magnitude
 - 6: modulate spreading forces for top $x\%$ of modules
 - 7: add spreading forces to quadratic program formulation
 - 8: solve the quadratic program
 - 9: **until** (modules are distributed over the placement region)
-

2.5.3 Advantages of Modulation

The key advantages of using force-vector modulation are:

- Modulation of spreading forces results in a placement solution with a better wire length. The reason being, the “modulated” modules are placed at their optimal locations as determined by the wire length objective used during global placement.
- Modulation of spreading forces results in a reordering of the modules at a global scale. Since the non-linear optimization step is used to perform the reordering, it occurs on a global scale. As a result, modules no longer retain their relative ordering as obtained after the initial non-linear optimization step. The significant decrease in the wire length of the placement can be attributed to this change in the relative ordering of the modules.
- Modulation does not impact the degree of spreading. Since only a small fraction of the modules are picked for modulation, the unmodulated modules will still shift towards their target locations as determined during module spreading. In doing so, they contribute towards spreading the placement.
- Modulation can be used within any force-directed placer, irrespective of the wire length objective used during placement.

2.5.4 Effect of Force-vector Modulation

Figure 2.9 illustrates the effect of force-vector modulation on an example circuit that consists of a single net connecting five movable modules. The arrows in Figure 2.9(a) depict the spreading forces on the modules during the current global placement iteration. It can be seen that the shaded box has a very high associated spreading force (depicted by the thick arrow heading outward from the box). The bold hatched rectangle represents the bounding box of the net connecting the five modules. Initially, the half-perimeter wire length of the net is: $HPWL = 3 + 2 = 5 \text{ units}$, with 3 and 2 units corresponding to the width and height of the bounding box. Figures 2.9(b) and 2.9(c) show the locations of the modules after the subsequent quadratic optimization step. Figure 2.9(b) depicts the case with no modulation and Figure 2.9(c) depicts the case where the spreading force on the shaded box has been nullified for the quadratic optimization step. From Figure 2.9(b), the shaded module is pulled all the way to the top of the chip. As a result the wire length of the net is: $HPWL = 4 + 4 = 8 \text{ units}$. On the other hand, from Figure 2.9(c) since the spreading force on the shaded box is nullified, it is placed in its quadratically optimal location with respect to its connections to the other modules. This results in a wire length of: $HPWL = 4 + 3 = 7 \text{ units}$, which is lesser than the case without modulation. In addition, modulation does not impact the degree of spreading as the other modules still spread based on their spreading forces.

Finally, Table 2.1 shows the effect of force-vector modulation on the placement wire length for two industrial ASIC designs from the ISPD-2005 benchmark suite [44]. In the table: *Case I* corresponds to running the generic quadratic placement algorithm (Algorithm 2.1) and *Case II* corresponds to running the quadratic placement algorithm with force-vector modulation (Algorithm 2.2). In both cases, the placer was run to satisfy the same density target, and all placement results are after final placement legalization. From column four of Table 2.1, it can be seen that force-vector modulation can achieve upto $1.95\times$ reduction in the legalized wire length without impacting the degree of spreading.

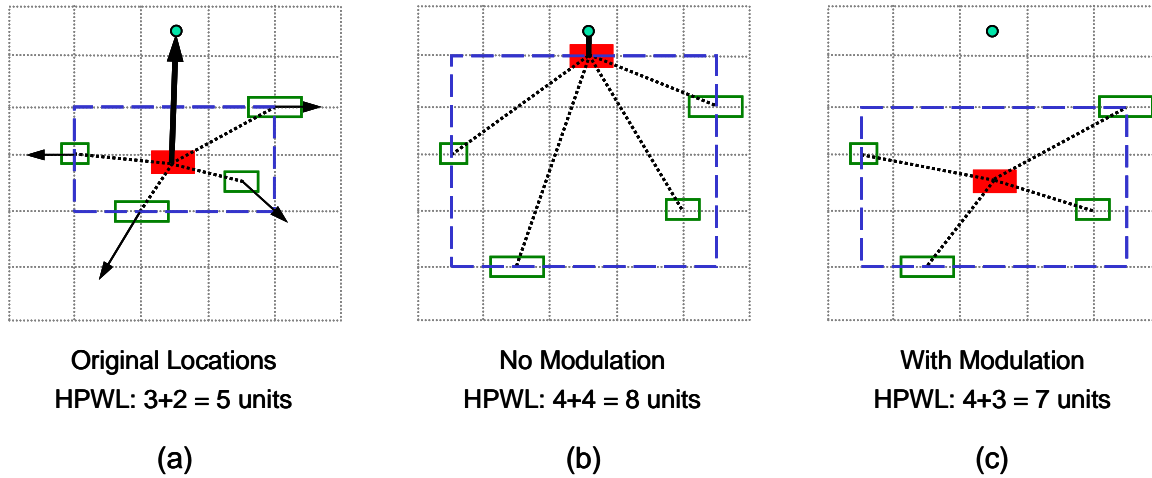


Figure 2.9 Effect of modulation on the module locations and net wire length. (a) Original module locations and net wire length (b) Module locations and net wire length without modulation (c) Module locations and net wire length with modulation.

Table 2.1 Effect of force-vector modulation on the half-perimeter wire length.

Circuit	Half-Perimeter Wire Length ($\times e6$)		
	No Modulation (Case I)	With Modulation (Case II)	$\frac{\text{Case I}}{\text{Case II}}$
adaptec1	252.11	128.84	1.95
bigblue1	168.74	114.47	1.47

2.6 Iterative Local Refinement

As mentioned in Section 2.1, one of the key attractions of quadratic placement is the fact that the quadratic objective can be efficiently minimized by solving a set of linear equations. But, the quadratic objective is only an indirect measure of the linear (or half-perimeter) wire length. For small circuits, the inaccuracy of quadratic wire length is not substantial and quadratic placement techniques can still produce very competitive wire length. However, for larger circuits, this inaccuracy becomes a major bottleneck in the quality of quadratic placement techniques.

To offset this disadvantage, previous approaches to quadratic placement modify the quadratic objective function to approximate the linear wire length. For example, *GORDIAN-L* [60] uses an iterative net-weight adjustment technique to achieve linearization during quadratic placement. Within *GORDIAN-L*, during each iteration of quadratic optimization, the quadratic objective between any two modules is weighted by a constant additional net-weight. This weight is inversely proportional to an approximation of the linear net-length connecting the two modules during the immediately preceding iteration. This weight behaves as a variable spring constant that increases with decreasing net-length, thereby serving to effectively minimize the wire length. *Kraftwerk* [61] uses a BoundingBox net model which, when combined with the wire length linearization technique of *GORDIAN-L* can theoretically model the half-perimeter wire length in an accurate manner during quadratic placement. Since the net-weights rely on the actual module coordinates, which are constantly changing, these approaches have two main disadvantages:

- Separate copies of the connectivity matrix (Section 2.3) need to be maintained to solve the problem in the x - and y -dimensions.
- The net-weights need to be determined by an iterative technique, which in practice requires some approximations to provide numerical stability to the solver. This can potentially hurt both the runtime and solution quality of the solver.

Instead of modifying the quadratic objective, this work uses an efficient and highly effective

technique called Iterative Local Refinement (ILR) to directly minimize the half-perimeter wire length during placement. The ILR technique is a greedy heuristic that is used to refine a coarse global placement (typically obtained after a few iterations of quadratic placement). It iteratively improves the placement wire length while simultaneously spreading the modules over the placement region. It has the ability to move the modules by a relatively large distance, and hence can be used to modify the placement on a global scale. In addition, it can seamlessly handle placement blockage and placement congestion constraints.

2.6.1 Bin Structure for Iterative Local Refinement

To estimate the module utilization in a given area and move the modules over the placement region, the ILR technique employs a regular bin structure similar to the one used during module spreading. During the first step of ILR the width and height of each bin is set to $5\times$ that of the bin used during module spreading. Such large bins are constructed to have a global view of the current placement, and enable modules to move over long distances. This is done to minimize the wire length of long nets that might span a large part of the placement region. During subsequent steps, the width and height of the bins are gradually brought down to the values used during module spreading. As a result, the movement of the modules gets progressively localized.

2.6.2 Iterative Local Refinement for Simultaneous Spreading and Wire Length Minimization

During each iteration of ILR, once the placement region has been binned, the *source* bin for all the movable modules is determined. A source bin for a module is one in which it is originally present before movement during the current ILR iteration. Each module is then tentatively moved from its source bin to its eight neighboring bins. These neighboring bins are denoted as the *target* bins for the module under consideration. For each tentative move, one score is computed. For calculating the score, it is assumed that a module is moving from its current position in a source bin to the same relative position in the target bin. This is shown in

Figure 2.10, where the box in the center (with the solid outline) represents the source bin for a particular module. The boxes with the dotted outlines represent the eight tentative locations that are considered for score calculation during ILR. The score for each move is a weighted sum of a wire length component and a utilization component. If all eight scores are negative, the module will remain in its source bin. Otherwise, it is moved to the target bin with the highest score.

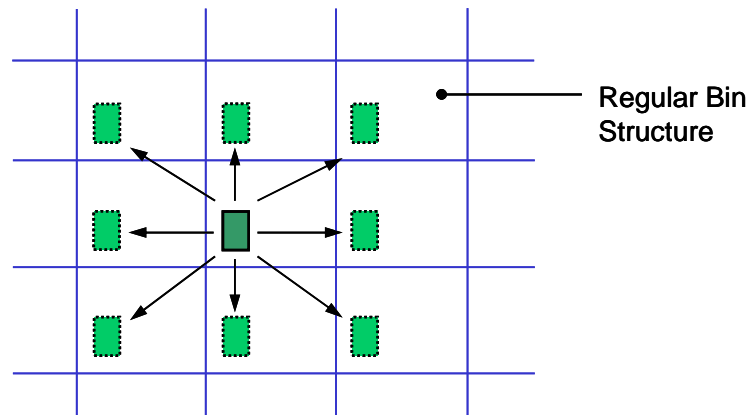


Figure 2.10 Eight tentative moves for score calculation during Iterative Local Refinement.

The wire length component of the score is the total change in the half-perimeter wire length (HPWL) of all the nets connected to the module. Since it directly takes the HPWL into account, it is more accurate than the quadratic objective. The utilization component is the difference in an exponential function of the source and target bin utilizations. It encourages a move from a bin with a high utilization to one with lower utilization. For the utilization component to accurately reflect the placement distribution, a utilization weight is defined for each bin in the placement region. This weight is a function of the bin utilization and is constantly updated based on the current placement distribution. As a result, a sparse bin will have a low utilization weight so that more modules can be moved into it, whereas a dense bin will have a higher utilization weight so that modules can be moved out of the bin. As the weights are a function of the bin utilization, they are dynamically updated, and this prevents

oscillations in terms of the movement of the modules.

During one iteration of ILR, all the movable modules in the circuit are visited and the above steps are followed for module movement. Subsequently, this iteration is repeated until there is no significant improvement in the wire length.

2.6.3 Iterative Local Refinement for Handling Placement Blockages

Circuits today contain numerous placement blockages in the form of fixed macros. As shown in Figure 2.11(a) and Figure 2.11(b), fixed macros fragment the placement image which in turn disrupts the smooth spreading of modules during placement. The reason being, there will be abrupt transitions in the utilization map at the boundaries of the fixed macros. This in turn inhibits the modules to pass over the macros to find better locations in the placement region. As a result, analytical placement techniques often place a lot of movable modules on top of the fixed macros. In addition, it is possible for modules that are connected to each other to be placed on opposite sides of a fixed macro. Along with increasing the wire length, this can restrict the ability of a physical synthesis tool to close timing, as techniques like buffer-insertion can no longer be performed on the straight-line path connecting the modules. This situation can also lead to severe routing congestion around the edges of the macro.

To avoid the problems associated with fixed macros and enable smooth spreading during placement, the ILR technique uses a smoothing transform to remove the abrupt transitions in the utilization map at the boundaries of the fixed macros. Initially, using the current ILR bin structure, a utilization map is constructed that considers only the fixed macros in the placement region. This is shown in Figure 2.11(b) in which a value of 1 for a bin implies that the bin is completely covered by a fixed macro. A 3×3 Laplacian matrix is then applied as a smoothing filter over the entire utilization map for a pre-defined number of iterations. This transformation smoothes the sharp edges in the original utilization map creating the modified map as shown in Figure 2.11(c). During the initial steps of ILR, the smoothing transform is run for a large number of iterations so that modules can easily cross over a fixed macro if required. During the final steps of ILR, the smoothing iterations are gradually reduced so that

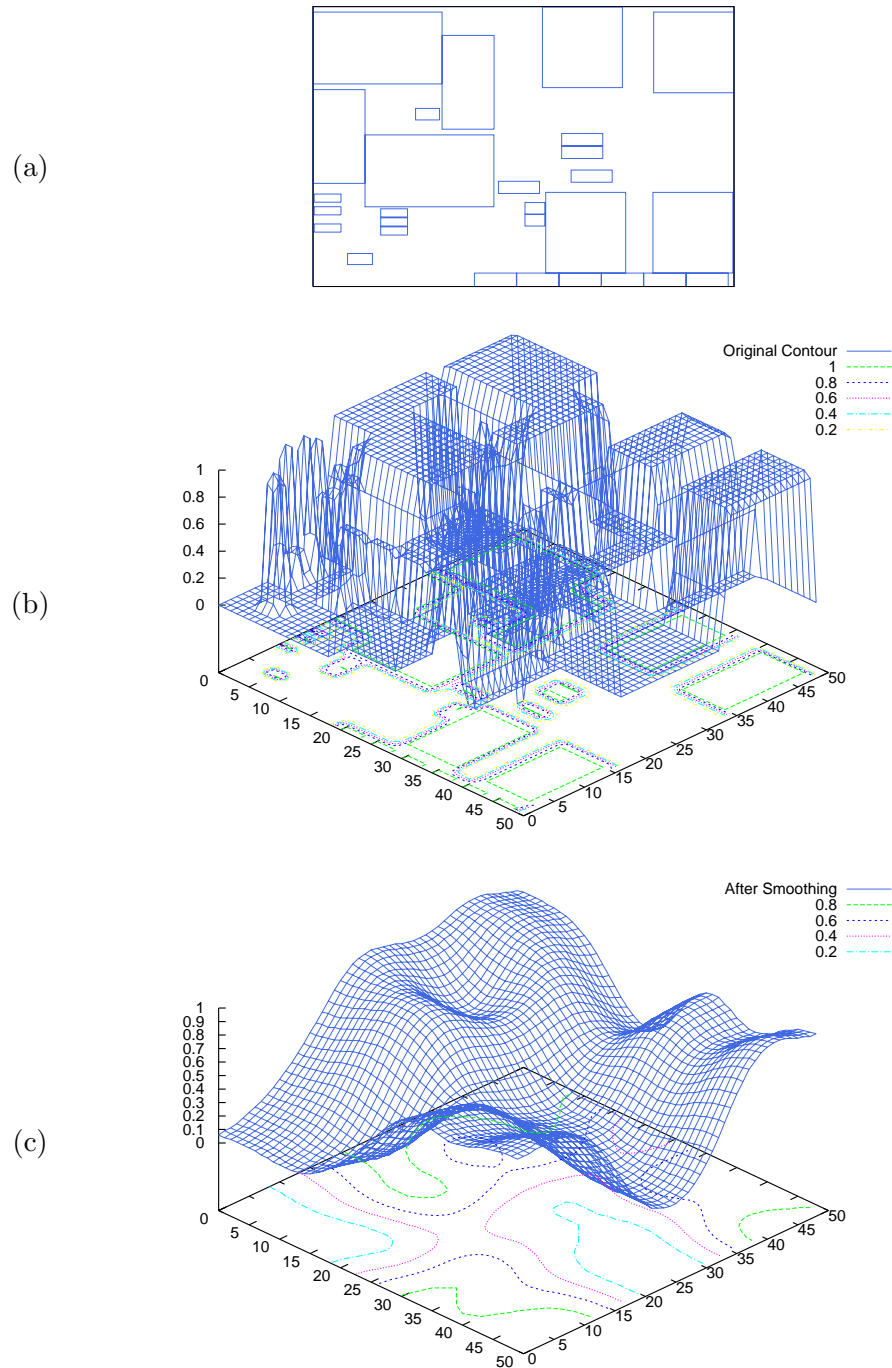


Figure 2.11 Handling placement blockages during Iterative Local Refinement. (a) Placement blockage profile for a circuit (b) Initial contour plot of the placement blockages showing a fragmented placement image (c) Contour plot after smoothing that aids in module spreading during Iterative Local Refinement.

the edges get progressively steeper. This forces the modules to slide down the slope of the macro and be effectively pushed out of the macro.

As a result of the smoothing, each bin will now have a contour height associated with it. The contour height is added to the movement score described in Section 2.6.2 to handle fixed macros during ILR.

2.6.4 Score for Module Movement During Iterative Local Refinement

Based on the discussion in Sections 2.6.2 and 2.6.3, to calculate the score for moving a module i from source bin m to target bin n , let:

- α : Weight for the wire length component.
- $wl_i(m)$: HPWL of all the nets connected to module i when it is in bin m .
- $wl_i(n)$: HPWL of all the nets connected to module i when it is in bin n .
- $\beta(m)$: Weight of the utilization component for bin m .
- $\beta(n)$: Weight of the utilization component for bin n .
- $U(m)$: Utilization function for bin m .
- $U(n)$: Utilization function for bin n .
- γ : Weight for the contour component.
- $C(m)$: Contour height within bin m .
- $C(n)$: Contour height within bin n .

Then the consolidated score for the move from bin m to bin n accounting for wire length, bin utilization and placement blockage is given by:

$$s_i(m, n) = \alpha\{wl_i(m) - wl_i(n)\} + \{\beta(m)U(m) - \beta(n)U(n)\} + \gamma\{C(m) - C(n)\}$$

2.6.5 Iterative Local Refinement for Placement Congestion Control

To reduce congestion, designers often enforce a *density target* constraint during the placement step. For each bin within a pre-defined bin grid, the density target specifies the maximum allowed movable area (or density) within the bin. Where, the *density* of a bin is defined as the ratio of the total movable area to the total free space within the bin. Satisfying the density target constraint implies that the density of all the bins in the pre-defined bin grid should be less than or equal to the density target value.

To handle the density target constraint, a density-bin based ILR (*d-ILR*) is used along with the regular ILR iterations. Score computation and module movement during a d-ILR iteration follows the same procedure as a regular ILR iteration. The key differences between the two techniques are as follows: (a) to perform placement refinement, d-ILR employs the pre-defined bin grid used for density target computation, (b) a higher weight is given to the bin utilization component to enforce spreading.

Within the *FastPlace* global placement framework, the two ILR techniques are combined as follows: Each phase of refinement initially runs the d-ILR technique. This is followed by the regular ILR steps, during which the bin sizes are changed as described in Section 2.6.1. The interaction between the d-ILR and regular ILR techniques during each refinement phase is shown in Figure 2.12, which depicts the change in the bin sizes during the various ILR steps.

2.7 Multilevel Global Placement Framework

Recent years have seen a continued increase in the circuit size that needs to be handled during placement. Today, it is quite common for circuits to have millions of modules that need to be placed simultaneously, and this number is steadily increasing. Therefore, in addition to producing high quality solutions, a modern placer should be scalable with circuit size. In such a scenario, a flat placement methodology may not be effective in producing a good quality solution within a reasonable amount of time. Hence, for efficient and scalable placement algorithm design, a hierarchical approach is beneficial. To this effect, many placers follow a hierarchical or multilevel approach to global placement [7, 8, 10, 13, 28, 32, 45, 62, 63, 69].

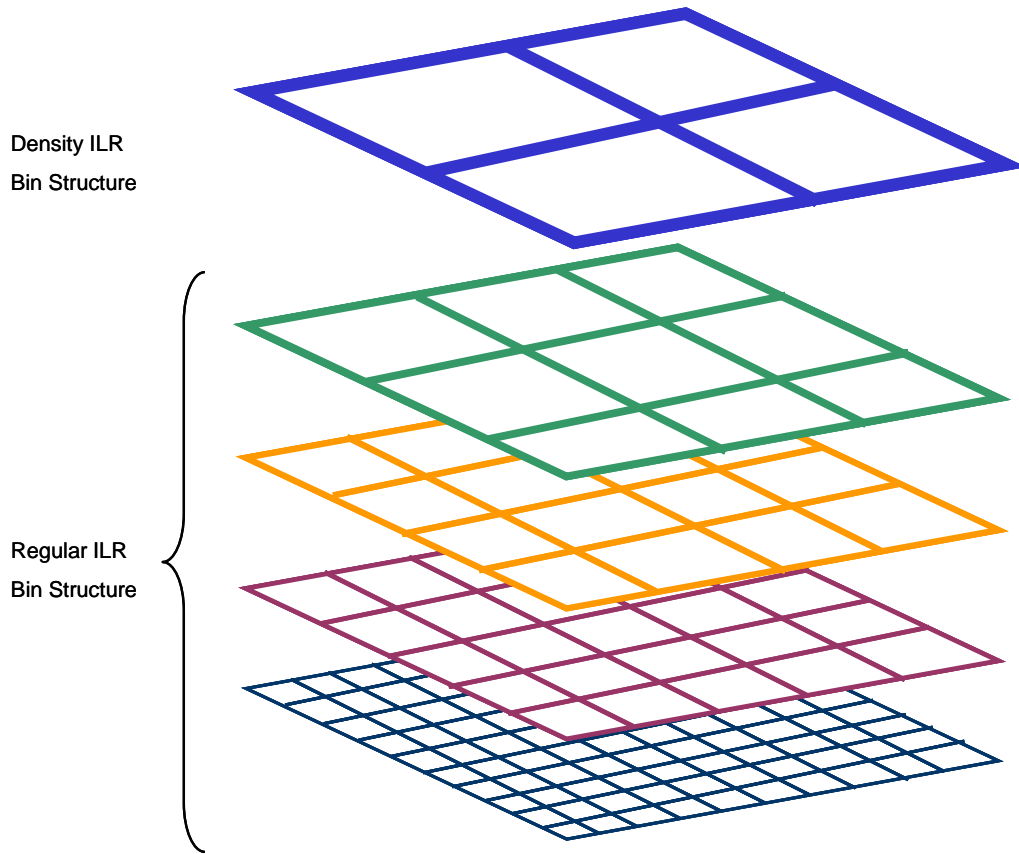


Figure 2.12 Bin size progression during Iterative Local Refinement.

Multilevel global placement algorithms typically use circuit clustering to reduce the placement problem size for large-scale designs. If clustering is performed in a careful manner, it can also yield better wire length along with faster runtime as compared to flat placement approaches. The reason being that clustering also has an indirect linearization effect. Clustered modules are often placed in close proximity and this helps reduce the wire length of the placement. In the spirit of the placement algorithms mentioned above, this work also employs circuit clustering to develop a multilevel framework for global placement. Two separate multilevel placement frameworks have been developed over the course of this research. These are outlined in Figure 2.13 and Figure 2.14, and are employed within the *FastPlace* [68] and *RQL* [66] global placement algorithms respectively.

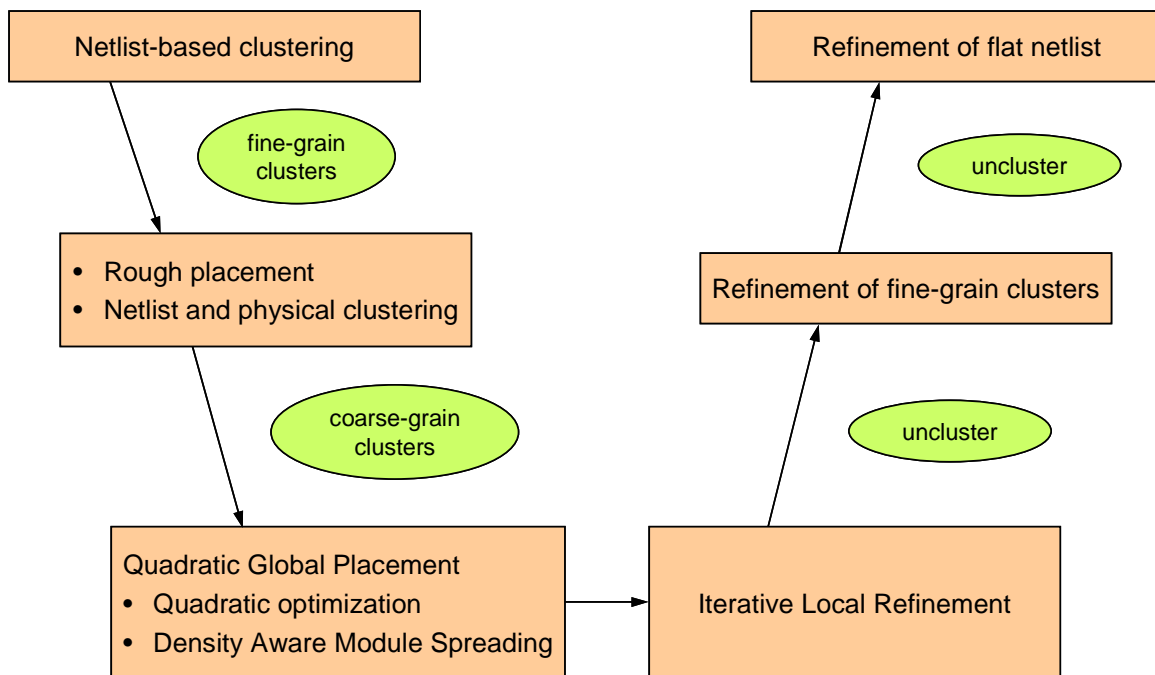


Figure 2.13 Multilevel global placement framework employed within Fast-Place.

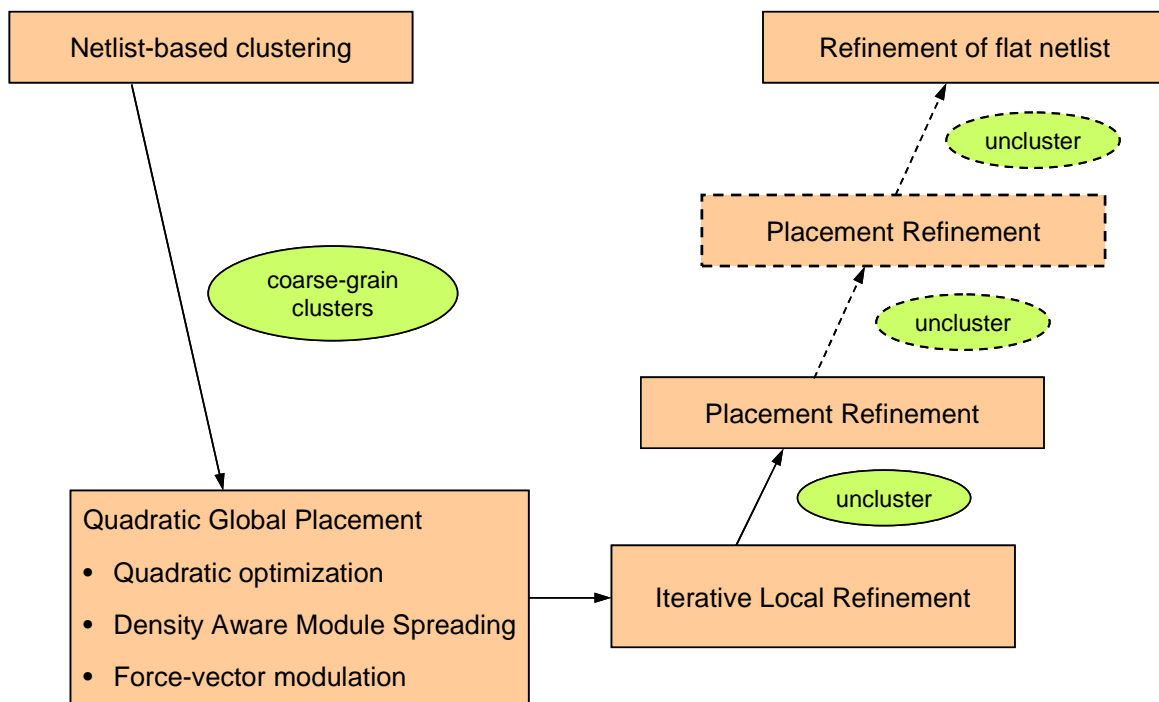


Figure 2.14 Multilevel global placement framework employed within RQL.

2.7.1 Clustering for Placement

In the implemented multilevel global placement frameworks, clustering is used in a *semi-persistent* context as defined by [45]. As in, clustering is used at the beginning of the placement flow to pre-process the input netlist so as to reduce the placement problem size. This is followed by a placement of the coarse-grain clustered netlist. The clusters are then gradually dissolved, and the placement progressively refined to finally obtain a placement of the original flat netlist. In this respect, the multilevel frameworks employed within *FastPlace* and *RQL* have two key differences:

1. The method used to construct the coarse-grain clustered netlist – *FastPlace* uses a two-level netlist and physical-based clustering approach, whereas *RQL* uses only netlist-based clustering.
2. The method used to refine the placement once a solution has been obtained for the coarse-grained clustered netlist – *FastPlace* uses a fixed, two-step unclustering and refinement flow, whereas *RQL* uses a gradual unclustering and refinement flow.

The rest of this section describes the two-level clustering approach employed within *FastPlace* (Figure 2.13). The netlist-based clustering within *RQL* is essentially the same as the first level of clustering within *FastPlace*, and differs only in the amount of clustering performed to reduce the placement problem size.

The first and second levels of clustering employed within *FastPlace* are named *fine-grain clustering* and *coarse-grain clustering* respectively. During fine-grain clustering, the size of each cluster is such that on average, it contains about two to three modules of the original circuit netlist. This clustering is solely based on the connectivity information between the modules in the netlist. Since it is performed at the beginning of placement, it is restricted to building fine-grain clusters to minimize any loss in placement quality due to incorrect clustering. In fact, it was demonstrated in [27] that fine-grain clustering can improve placement efficiency with negligible loss in placement quality.

A fast initial placement of the fine-grain clusters is then performed. The purpose of this step is to get some placement information for the next clustering level. Since each cluster in the first level has only around two to three modules, the initial placement of the clusters closely resembles an initial placement of the original flat netlist. Following initial placement, coarse-grain clusters are created by performing a second level of clustering (coarse-grain clustering). To perform clustering, coarse-grain clustering uses both, the connectivity information between the clusters and their physical locations as obtained from the initial placement.

A modified version of the *Best-choice* clustering algorithm using lazy-update speed-up technique [45] is employed for the two-level clustering scheme described above. The key feature of the best-choice clustering algorithm is that it maintains a global view of the circuit netlist and always picks the pair of modules having the highest score, to be clustered together. In addition, it is quite efficient to be used as a pre-processing step during placement.

Algorithm 2.3 outlines the modified version of the best-choice clustering algorithm that is employed within the two-level clustering scheme. Please note, in Algorithm 2.3, the “closest module” k to any given module j is the one that has the highest clustering score with j . In addition, $a(j)$ refers to the area of module j .

From Algorithm 2.3 there are four key parameters associated with the clustering scheme:

- α : The clustering ratio, defined as the ratio of the number of modules before and after clustering.
- $s(j, k)$: The netlist-based clustering score between modules j and k .
- $max_cluster_area$: The upper-bound on the cluster area.
- $distance_threshold$: The distance threshold used for physical clustering.

In the two-level clustering scheme, for each level of clustering, α is set to a value of 2 resulting in a $4\times$ reduction in the number of movable modules in the coarse-grain clustered netlist.

Algorithm 2.3 Best-choice clustering with placement information

```

1:  $m \leftarrow$  number of movable modules in the design
2:  $\alpha \leftarrow$  clustering ratio
3:  $target\_number\_of\_modules \leftarrow m/\alpha$ 
4: Phase 0: Construct Initial Priority-queue (PQ)
5:   for ( $j \leftarrow 1$  to  $m$ ) do
6:     find closest module  $k$  and clustering score  $s(j, k)$ 
7:     insert triple  $(j, k, s)$  into PQ with  $s$  as the key
8:   end for
9: end
10: Phase 1: Generate Clusters
11:   while ( $number\_of\_modules > target\_number\_of\_modules$ ) do
12:     pick top triple  $(j, k, s)$  from PQ
13:     if ( $j$  is marked invalid) then
14:       recalculate closest module  $k'$  and clustering score  $s'(j, k')$ 
15:       insert triple  $(j, k', s')$  into PQ
16:     else
17:       if (netlist-based clustering) then
18:         if ( $a(j) + a(k) < max\_cluster\_area$ ) then
19:           cluster  $j$  and  $k$  into new module  $j'$ 
20:         end if
21:       else if (netlist and physical clustering) then
22:         calculate  $d(j, k)$  the distance between  $j$  and  $k$ 
23:         if ( $d(j, k) < distance\_threshold$  and  $a(j) + a(k) < max\_cluster\_area$ ) then
24:           cluster  $j$  and  $k$  into new module  $j'$ 
25:         end if
26:       end if
27:       update circuit netlist based on the clustering
28:       for module  $j'$  find closest module  $k'$  and clustering score  $s'(j', k')$ 
29:       insert triple  $(j', k', s')$  into PQ with  $s'$  as the key
30:       mark neighbors of  $j'$  as invalid
31:     end if
32:   end while
33: end

```

The netlist-based clustering score between two modules j and k is given by:

$$s(j, k) = \frac{\sum_{\nu \in N} w_{\nu}}{(a_j + a_k)^{\beta}}$$

where N is the set of nets connecting the two modules and w_{ν} is the weight of the edge connecting the two modules on net ν . The parameter β is used to control the relative areas of the clusters in the clustered netlist. Within *FastPlace*, $w_{\nu} = 1/k$ and within *RQL*, $w_{\nu} = 2/k - 1$, where k is the degree of net ν . In addition, β is set to a value of 1 and 2 within *FastPlace* and *RQL* respectively.

Controlling the area of the clusters is highly imperative. Otherwise, a cluster can get progressively larger by absorbing smaller clusters around it. This is often detrimental and leads to bad solution quality. Having an area term in the denominator of the clustering score biases the clustering technique to pick modules that will not result in forming huge clusters. In addition, an upper-bound on the cluster area is also imposed using the *max_cluster_area* parameter. Within the two-level clustering scheme, *max_cluster_area* is set to $5 \times \text{average_cluster_area}$. Where, $\text{average_cluster_area} = \sum_{i=1}^m a(i) / \text{target_number_of_modules}$. This results in the formation of balanced clusters.

It is quite possible that two modules having a very high connectivity score do not actually end up being close to each other in the final placement. This can happen due to the influence of the other nets or modules connected to them. Hence, during coarse-grain clustering, even though the modules are ranked and picked based on their connectivity score, they are clustered only if the distance between them, as obtained from the initial global placement is within a certain threshold. In the clustering scheme employed, the *distance_threshold* is empirically set to 10% of the maximum chip dimension.

2.8 The FastPlace and RQL Global Placement Algorithms

This section outlines the two global placement algorithms that have been developed over the course of this research. Algorithm 2.4, first outlines the *FastPlace* algorithm. This is followed by Algorithm 2.5, which outlines the *RQL* algorithm. The key differences between the two algorithms are as follows:

- *RQL* uses the force-vector modulation technique to minimize the linear wire length whereas *FastPlace* does not have this feature.
- To add the spreading forces during quadratic placement, *RQL* uses on-chip fixed-points whereas *FastPlace* uses boundary fixed-points.
- *RQL* uses only netlist-based clustering, whereas *FastPlace* follows a two-level netlist and physical-based clustering approach.
- *RQL* uses a gradual unclustering and refinement flow, whereas *FastPlace* uses a fixed, two-step unclustering and refinement flow.
- To handle the density target constraint *FastPlace* uses the density-bin based ILR technique, whereas *RQL* handles the constraint by running more iterations of the regular ILR technique.
- The macro-blocks are legalized at a much earlier stage in *RQL* (at the end of coarsened netlist placement), whereas in *FastPlace* they are legalized at the end of global placement. It was observed that legalizing the macro-blocks early on in the placement flow lead to better spreading of the standard-cells and also aided the standard-cell legalization step within *RQL*.

Algorithm 2.4 The FastPlace algorithm

```

1: Phase 0: Initial Placement
2:   construct fine-grain clusters using Best-choice netlist-based clustering
3:   solve initial quadratic program
4:   repeat
5:     perform regular Iterative Local Refinement on fine-grain clusters
6:   until (the placement is roughly even)
7: end
8: Phase 1: Coarse Global Placement
9:   construct coarse-grain clusters using Best-choice netlist and physical clustering
10:  re-initialize the cluster coordinates to the center of the placement region
11:  solve the quadratic program
12:  repeat
13:    perform Density Aware Module Spreading on coarse-grain clusters
14:    add spreading forces to the quadratic program formulation
15:    solve the quadratic program
16:  until (the placement is roughly even)
17:  repeat
18:    perform density-based Iterative Local Refinement on coarse-grain clusters
19:    perform regular Iterative Local Refinement on coarse-grain clusters
20:    perform Density Aware Module Spreading on coarse-grain clusters
21:  until (the placement is quite even)
22: end
23: Phase 2: Refinement of fine-grain clusters
24:  uncluster coarse-grain clusters
25:  perform density-based Iterative Local Refinement on fine-grain clusters
26:  perform regular Iterative Local Refinement on fine-grain clusters
27: end
28: Phase 3: Refinement of flat netlist
29:  uncluster fine-grain clusters
30:  perform density-based Iterative Local Refinement on flat netlist
31:  perform regular Iterative Local Refinement on flat netlist
32: end

```

Algorithm 2.5 The RQL algorithm

```

1: Phase 0: Clustering
2:   initial_number_of_modules  $\leftarrow$  module_count_in_flat_netlist
3:   while (number_of_modules > target_number_of_modules) do
4:     cluster netlist using the Best-choice clustering algorithm
5:   end while
6: end
7: Phase 1: Coarsened Netlist Placement
8:   solve initial quadratic program
9:   while (max_bin_util > target_util_threshold) do
10:    perform Density Aware Module Spreading
11:    calculate spreading forces for all the modules
12:    rank modules based on the spreading force magnitude
13:    modulate the spreading force for the top  $x\%$  of modules
14:    add spreading forces to quadratic program formulation
15:    solve the quadratic program
16:  end while
17:  repeat
18:    perform regular Iterative Local Refinement
19:    perform Density Aware Module Spreading
20:  until (max_bin_util  $\leq$  1.0)
21:  uncluster movable macro-blocks
22:  legalize and fix movable macro-blocks
23: end
24: Phase 2: Refinement
25:  while (number_of_modules < initial_number_of_modules) do
26:    uncluster netlist s.t. number_of_modules =  $2 \times$  number_of_modules
27:    perform regular Iterative Local Refinement
28:  end while
29: end

```

CHAPTER 3. LEGALIZATION

3.1 Introduction

To manage the complexity of placement, the non-overlapping constraints among the modules are typically ignored during the global placement stage. Once the modules have been reasonably distributed over the placement region, the aim of the legalization stage is to resolve the overlaps among the modules and assign them to legal locations in the placement region.

Due to the widespread use of pre-designed macro-blocks like IP cores etc., modern circuits often contain a large number of placeable macro-blocks along with the standard-cells (mixed-size circuits). In such a scenario, traditional standard-cell based legalization techniques like Diffusion [53] and network-flow [5] can no longer be used. Since the movement of the macro-blocks can significantly impact the placement wire length, particular attention should be paid during the legalization of macro-blocks within mixed-sized circuits.

Based on their approach, existing mixed-size legalization techniques can be broadly classified into two categories:

1. *Single-phase Legalization*: Single-phase legalization techniques do not differentiate between the standard-cells and macro-blocks in the circuit, and simultaneously legalize all the modules in the circuit. Examples of single-phase legalization are the techniques employed within [34] and [2] that extend the legalization algorithm in [25] to perform mixed-size legalization. Yet another single-phase legalization technique is described in [76], which uses the technique of zone refinement to perform mixed-size legalization and detailed placement.
2. *Two-phase Legalization*: These techniques distinguish between the macro-blocks and

standard-cells, and perform mixed-size legalization as follows: (a) in the first phase, the overlaps among the macro-blocks are resolved and they are assigned to legal locations within the placement region, (b) in the second phase, the legalized macro-blocks are treated as fixed macros (placement blockages) and the standard-cells are legalized in the presence of the fixed macros. Representative examples of two-phase legalization are the techniques employed within [18, 19, 71]. To legalize the macro-blocks in the circuit, [71] uses a three-step approach: (a) it first determines all pairs of macros that overlap with each other, (b) for each pair of overlapping macros, it uses a branch and bound algorithm to find the non-overlapping constraint that causes the least perturbation of the macros from their global placement locations, (c) finally, it uses linear programming (LP) to determine the locations of the macro-blocks satisfying the non-overlapping constraints determined in the previous step. On the other hand, [18, 19] use constraint graphs to represent the overlaps among the macro-blocks. A constraint graph pre-processing step is performed to ensure that all the macros can be placed with the placement region once they are legalized. This is followed by an LP to determine the locations of the macro-blocks.

3.2 Overview of the Mixed-size Legalization Algorithm

To perform mixed-size legalization, this work uses the two-phase legalization technique as outlined in Section 3.1. The top-level flow for mixed-size legalization is given in Algorithm 3.1.

Algorithm 3.1 Mixed-size legalization algorithm

- 1: **Phase 1: Macro-block Legalization**
 - 2: legalize movable macro-blocks using the Iterative Clustering algorithm
 - 3: fix the movable macro-blocks and transform them into placement blockages
 - 4: **end**
 - 5: **Phase 2: Standard-cell Legalization**
 - 6: construct row slices for all the circuit rows in the placement region
 - 7: perform slice aware bin-based cell movement
 - 8: move cells among slices to satisfy slice capacities
 - 9: legalize cells within slices
 - 10: **end**
-

The rest of this chapter is organized as follows: Section 3.3 describes the macro-block legalization algorithm in detail. This is followed by Section 3.4 that describes the standard-cell legalization algorithm.

3.3 Legalization of Macro-blocks

Compared to standard-cells, the macro-blocks are much larger in size, with sizes occasionally comparable to the chip dimensions. Hence, the movement of the macro-blocks has a significant impact on the placement wire length. Therefore, the aim of the macro-block legalization phase is to maintain the global placement locations of the macros as much as possible. If we denote the global placement location of a macro as its *target* location. Then the macro-block legalization problem is to resolve overlap among all the macros and assign them to legal locations in the placement region, while minimizing the total perturbation of the macros from their target locations.

This problem is formulated as a minimum perturbation fixed-outline floorplanning problem. The sequence-pair [42] is used to represent the floorplan and enforce the non-overlapping constraints among the macros. Please note, the developed approach is not restricted to a floorplan representation by a sequence-pair. It is a general technique that can incorporate other floorplanning representations as well. Formally, the minimum perturbation floorplanning problem is described in Figure 3.1.

Minimum Perturbation Floorplan Realization (MPFR) Problem:

Given: N macro-blocks with target coordinates (x_i^*, y_i^*) for $i = 1, \dots, N$ and a sequence-pair (\mathbf{p}, \mathbf{q}) .

Determine: Legalized coordinates (x_i, y_i) s.t. $\sum_{i=1}^N |x_i - x_i^*| + |y_i - y_i^*|$ is minimized.

Figure 3.1 Minimum perturbation floorplan realization problem.

For a given sequence-pair, the *MPFR* problem is solved by using a novel *Iterative Clustering* algorithm. This is described in Section 3.3.1. This is followed by a description of the top-level

flow for macro-block legalization using simulated annealing, which is given in Section 3.3.2. Since the horizontal and vertical non-overlapping constraints can be handled independently, only the horizontal problem is considered for further discussion.

3.3.1 Iterative Clustering Algorithm

As suggested by the name, the basic idea behind the Iterative Clustering algorithm is to progressively cluster the macros that overlap with each other, when the macros are assigned or moved to their target locations during legalization. Initially, every macro in the circuit is assigned to a separate cluster. Subsequently, to determine which macros need to be grouped in the same cluster, all the clusters are shifted to their optimal locations that satisfy the minimum perturbation objective function. In doing so, if there is any overlap among the clusters, then these clusters are merged to form a larger cluster. Within each cluster, the macros are abutted to each other preserving their left-to-right relationship as defined by the current sequence-pair. As a result, there is no overlap among the macros within the clusters. By progressively clustering the macros and moving the clusters to their optimal locations, a legalized placement solution of the macros is obtained. The pseudo-code for the Iterative Clustering algorithm to legalize the macro-blocks in the horizontal direction is given in Algorithm 3.2.

Algorithm 3.2 The Iterative Clustering algorithm

- 1: from the current sequence-pair (\mathbf{p}, \mathbf{q}) determine the horizontal constraint graph
 - 2: find the immediate left and right neighbors of all the macro-blocks from the horizontal constraint graph
 - 3: **for** $(i \leftarrow 1 \text{ to } N)$ **do**
 - 4: place macro p_i in its target location
 - 5: Let \mathbf{C} be a new cluster consisting of p_i
 - 6: **while** (\mathbf{C} overlaps with other clusters) **do**
 - 7: merge \mathbf{C} with the closest cluster on its left
 - 8: let \mathbf{C} be the new cluster that is formed
 - 9: shift \mathbf{C} to its optimal location
 - 10: **if** (macro $p_j \in \mathbf{C}$ is at its target location) **then**
 - 11: detach p_j from \mathbf{C} if possible and goto step 9
 - 12: **end if**
 - 13: **end while**
 - 14: **end for**
-

The individual steps of the Iterative Clustering algorithm are now explained in more detail with the help of an example given in Figure 3.2 and Figure 3.3. Figure 3.2(a) shows an example circuit with six macro-blocks in which macros 3, 5 and 6 overlap in the horizontal direction. Traversing the list of the macros from left to right, the overlaps among the macros are as follows: (a) macro 5 overlaps with macro 6 by d_1 units (b) macro 3 overlaps with macro 6 by $d_1/2$ units, and (c) macro 3 overlaps with macro 5 by $d_2 - d_1/2$ units. From Figure 3.2(a), it can be seen that the minimum horizontal displacement required to yield an overlap-free placement is: $d_1 + (d_2 - d_1/2) = d_1/2 + d_2$ units.

As mentioned before, the macro-block legalization algorithm uses the sequence-pair to represent the current floorplan. The sequence-pair is essentially a pair of N -dimensional vectors (\mathbf{p}, \mathbf{q}) that represent the horizontal and vertical relationships between the macro-blocks. For the circuit given in Figure 3.2(a), the corresponding sequence-pair is given by $(\mathbf{p}, \mathbf{q}) = (412653, 615342)$ [42]. From Algorithm 3.2, for horizontal placement, initially the horizontal constraint graph is constructed from the current sequence-pair. This is shown in Figure 3.2(b). The horizontal constraint graph gives the immediate left and right neighbors of all the macro-blocks in the circuit. These neighbors are associated with the non-transitive edges in the horizontal constraint graph and can be found in $O(N^2)$ time.

The macros are then placed one at a time from left to right according to the horizontal sequence \mathbf{p} . In case, there is no overlap between a macro and an existing cluster, the macro is placed at its target location. This is shown in Figure 3.2(c) in which the macros have been placed in the order $4 \rightarrow 1 \rightarrow 2 \rightarrow 6$ given by the horizontal sequence. In case a macro overlaps with an existing cluster, then the clustering is updated according to steps 6–13 (Figure 3.2(d)–Figure 3.3(g)). The condition in step 6 and the closest cluster in step 7 can be determined by considering the constraints of the immediate left neighbors of the macros in \mathbf{C} .

The distance to shift a cluster in step 9 is determined according to the following lemma:

Lemma 1 *For a cluster \mathbf{C} , its location is optimal if the number of macros perturbed to the left from their target locations is equal to the number perturbed to the right.*

Since the macros are added from left to right in the horizontal sequence \mathbf{p} , if there is

an overlap between a macro and an existing cluster, the macro will always be abutted to the right of the cluster (Figure 3.2(d) and Figure 3.3(f)). Hence, newly added macros will always be placed to the right of their target locations. As a result, the clusters will always shift left to move the macros closer to their target locations (Figure 3.3(e) and Figure 3.3(g)). This makes it relatively simple to find the correct shift amount for the newly formed clusters. Let the displacement of a macro to the right of its target location be defined as a positive displacement and to left of its target location as negative. Then the shift amount for a cluster can be determined by looking at all its constituent macros that have a positive displacement value. In accordance with Lemma 1, if a cluster has an even number of macros, then it is shifted to the left by half the minimum positive displacement value among its constituent macros (Figure 3.2(d)). If the cluster has an odd number of macros, then it is shifted to the left by the minimum positive displacement value among its constituent macros (Figure 3.3(g)).

In step 10, after shifting a cluster \mathbf{C} , a macro $p_j \in \mathbf{C}$ may potentially reach its target location. If p_j does not have any right neighbors that belong to cluster \mathbf{C} , then it is detached from the cluster. Otherwise, it will move along with the cluster during the subsequent iterations and will not be placed in its optimal location. The condition to detach p_j can be checked by looking at its immediate right neighbors.

Although the while loop in steps 6–13 looks complicated, it can be shown with careful implementation and analysis that the runtime complexity of the Iterative Clustering algorithm is $O(N^2)$.

3.3.2 Macro-block Legalization by Simulated Annealing

The aim of the top-level simulated annealing framework is to obtain a sequence-pair such that the corresponding placement obtained from the Iterative Clustering algorithm will resolve overlaps among the macros with minimum perturbation from their global placement locations. Another factor to be considered during placement is that the macros have to be placed in legal locations within the placement region. Hence, the cost function for simulated annealing is a weighted sum of the total perturbation of the macros, along with a penalty for being out of

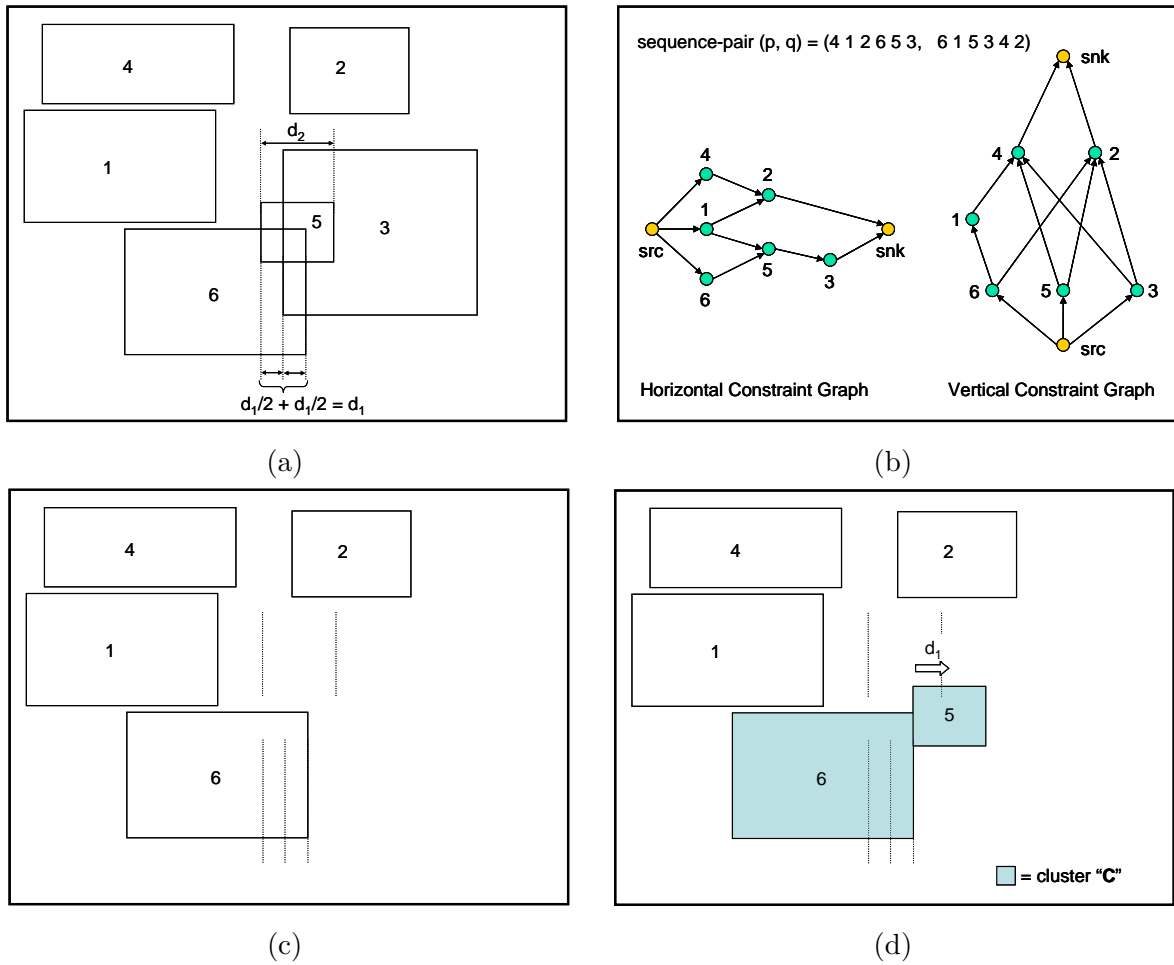


Figure 3.2 Iterative Clustering algorithm for macro-block legalization on an example circuit with six macro-blocks. (a) Initial placement of macros with horizontal overlap (b) Sequence-pair from initial placement (c) Go through horizontal sequence and add macros from left to right (d) Shift macro 5 from target location because of overlap and form cluster C .

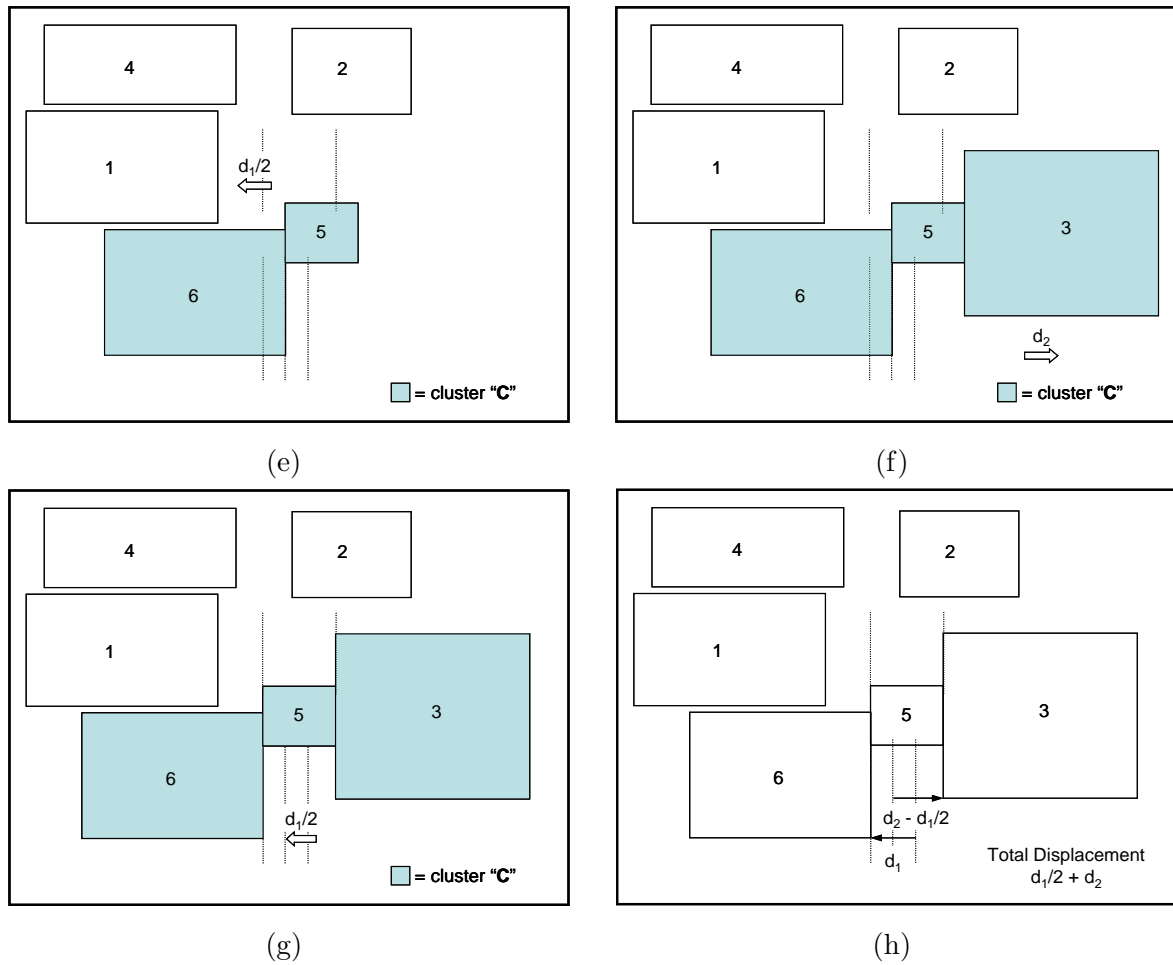


Figure 3.3 Iterative Clustering algorithm for macro-block legalization on an example circuit with six macro-blocks (continued). (e) Shift cluster C to the left towards its optimal location (f) Shift macro 3 from target location because of overlap and form cluster C (g) Shift cluster C to the left towards its optimal location (h) Final locations of macros with no overlap.

bounds of the placement region.

If (\mathbf{p}, \mathbf{q}) represents the sequence-pair. Then, the initial sequence for \mathbf{p} and \mathbf{q} are generated by sorting the macros in a non-decreasing order according to the Manhattan distance from the upper-left and lower-left corner to their target locations respectively. This sequence-pair closely corresponds to the original placement and is usually quite good. Hence, a low-temperature annealing is sufficient to generate a good result. Additionally, each annealing move is restricted to an exchange of two adjacent macros in one of the two sequences so as to not disturb the current solution significantly. Algorithm 3.3, now gives the pseudo-code for the top-level flow for macro-block legalization using simulated annealing.

Algorithm 3.3 Macro-block legalization using simulated annealing

```

1: get initial sequence-pair (SP) from the global placement of the macro-blocks
2: set initial temperature  $T$  and iteration  $I$ 
3: while ( $T > T_{terminate}$  and  $I < I_{terminate}$ ) do
4:    $step \leftarrow 0$ 
5:   evaluate SP and modify if necessary
6:   while ( $step < Number\_of\_steps\_per\_temperature$ ) do
7:      $placement \leftarrow \text{ITERATIVE\_CLUSTERING\_ALGORITHM}(\text{SP})$ 
8:      $\text{EVALUATE}(placement)$ 
9:      $\text{MODIFY}(\text{SP})$ 
10:     $step \leftarrow step + 1$ 
11:  end while
12:   $T \leftarrow T \times cooling\_rate$ 
13:   $I \leftarrow I + 1$ 
14: end while

```

3.3.3 Effect of Macro-block Legalization

Finally, Figure 3.4 illustrates the solution quality of the macro-block legalization algorithm. The circuit shown in Figure 3.4 happens to be a difficult instance for macro-block legalization. The reason being, the entire circuit consists of only macro-blocks and it has less than 20% of white-space. This greatly limits the flexibility of a macro-block legalization algorithm. Figure 3.4(a) shows the global placement solution of *FastPlace*. It can be seen that there is significant module overlap in and around the center of the placement region. Figure 3.4(b) shows the

placement after macro-block legalization. It can be seen that the macros have moved by a very small amount during legalization.

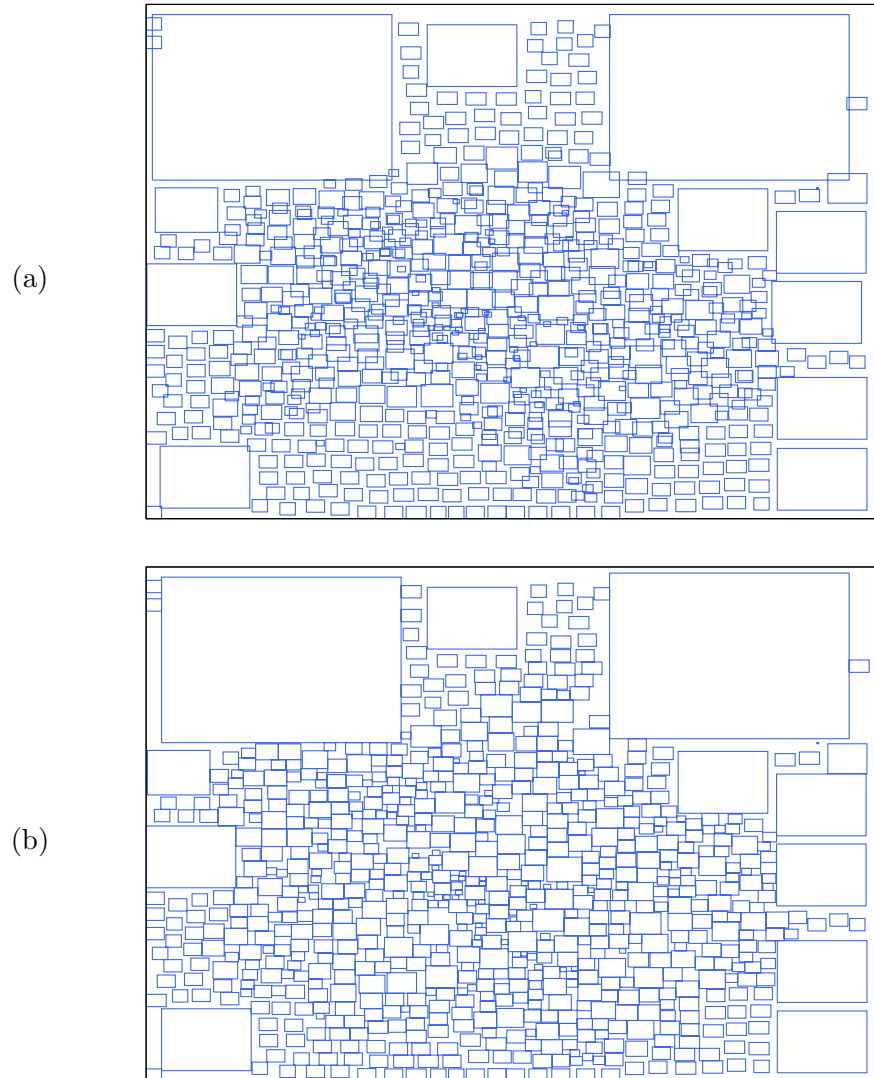


Figure 3.4 Effect of macro-block legalization during placement. (a) Locations of macro-blocks before legalization (b) Locations of macro-blocks after legalization.

3.4 Legalization of Standard-cells

The aim of the standard-cell legalization phase is to resolve overlaps among the cells in the presence of the fixed modules or placement blockages in the circuit. Fixed modules overlapping with the circuit rows in the placement region essentially fragment the rows into a number of sub-rows. Therefore, the problem of the standard-cell legalization phase is to place the cells in legal locations within these sub-rows. If we consider a single row in the placement region, then a “row slice” happens to be one of the sub-rows within which the cells need to be placed. In other words, a row slice is the maximal part of a circuit row that is not covered by a placement blockage. The construction of a row slice is shown in Figure 3.5. In the figure, the horizontal dashed lines represent the standard-cell circuit rows, and the solid boxes represent the fixed modules overlapping with the circuit rows. The regions with the diagonal lines represent two of the row slices in the circuit.

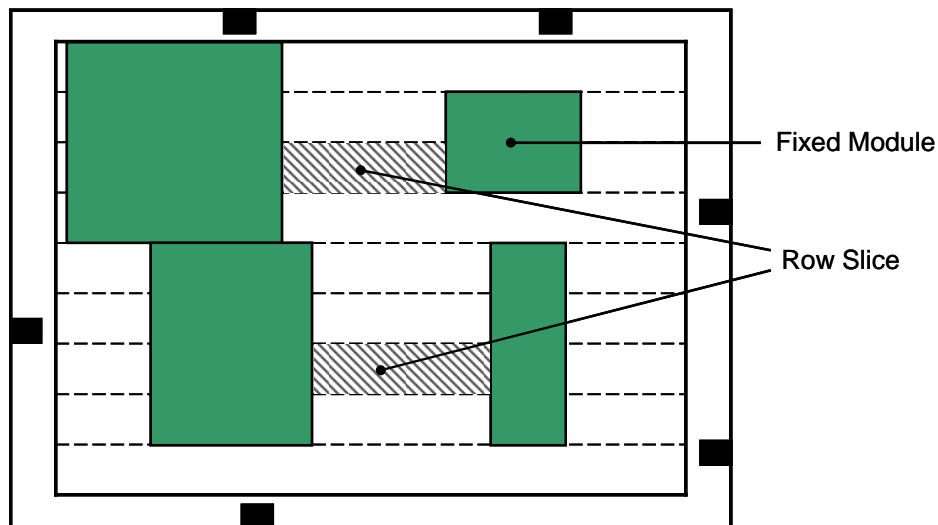


Figure 3.5 A row slice for standard-cell legalization.

3.4.1 Slice Aware Bin-based Cell Movement

The aim of the slice aware bin-based cell movement step is to satisfy slice capacities as well as placement congestion constraints. It achieves this dual objective by using a bin-based cell

movement scheme that is aware of the row slice capacity requirements.

To perform bin-based cell movement, a regular bin structure (B) is initially imposed on the placement region. The height of each bin in the regular bin structure is equal to the circuit row height and its width is equal to around $4\times$ the average cell width. Based on the current placement, the utilization of each bin in B and each slice in the placement region is then computed. The utilization of a bin is defined as the ratio of the total area of all the modules overlapping with a bin to the bin area. The utilization of a slice is defined as the total width of all the standard-cells within the slice. If the slice utilization is greater than the slice width, it is considered to be above capacity.

Based on the slice utilizations and placement blockages, a move map (M) is constructed that has the same dimensions as the regular bin structure. A bin in M has a value of either 1 for allowing movement of the cells into or out of this bin, or 0 otherwise. Bins that completely overlap blockages are assigned a value of 0 as we do not want cells to be moved on top of the blockages. If the utilization of a particular slice is greater than its width, then a small region of bins in and around the current slice is assigned a value of 1. This is to allow cell movement to be performed only on these bins. This is depicted in Figure 3.6 where there are two slices that are above capacity (shown by the diagonal lines). Then, bin-based cell movement is turned on for a small set of bins in and around the slices (shown by the lightly shaded bins in the figure).

A modified version of the ILR technique is then used to move the cells among the bins. The difference being that the score for a move during legalization is a weighted sum of three components: (a) the half-perimeter wire length reduction for the move, (b) a function of the utilization of the source and target bins, (c) a weighted difference of the move map values for the source and target bins. Since the legalization technique is mainly used to even out the placement and satisfy slice capacities, a higher weight is assigned to the second and third components.

The key advantages of the slice aware bin-based cell movement technique are that it does not significantly perturb the global placement solution. Secondly, it distributes the cells evenly within the slices. This helps to satisfy placement congestion constraints.

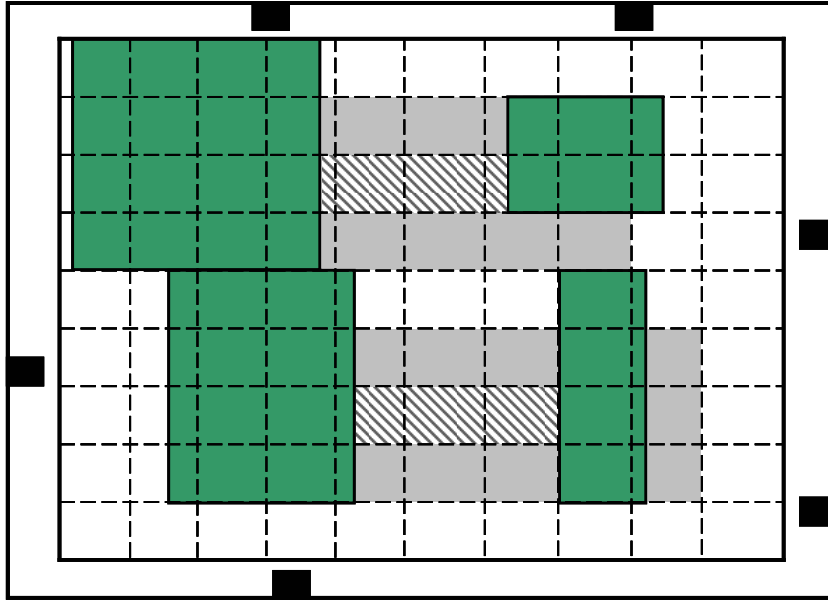


Figure 3.6 Slice aware bin-based cell movement.

3.4.2 Slice-based Cell Legalization

Following the slice aware bin-based cell movement, an explicit slice-based cell legalization step is performed. To legalize the cells within the slices, the cells are first moved among the slices to satisfy the slice capacities. Once all the slices in the placement region are under capacity, the cells are assigned to legal locations within the slices.

To satisfy the capacity requirements of all the slices, an iterative approach is followed, where the number of over-occupied slices is gradually reduced during each iteration. To perform cell movement among the slices, for every cell present in an over-occupied slice, eight scores are computed that correspond to tentatively moving the cell to its nearest eight neighboring slices. This is shown in Figure 3.7. The cell is then assigned a *movement score* that is equal to the highest value among the eight scores computed in the previous step.

For calculating the score, it is assumed that the cell is moving from its current location in a *source* slice to the nearest possible location in the *target* slice. Each score is a weighted sum of two components: (a) the change in the half-perimeter wire length associated with the move,

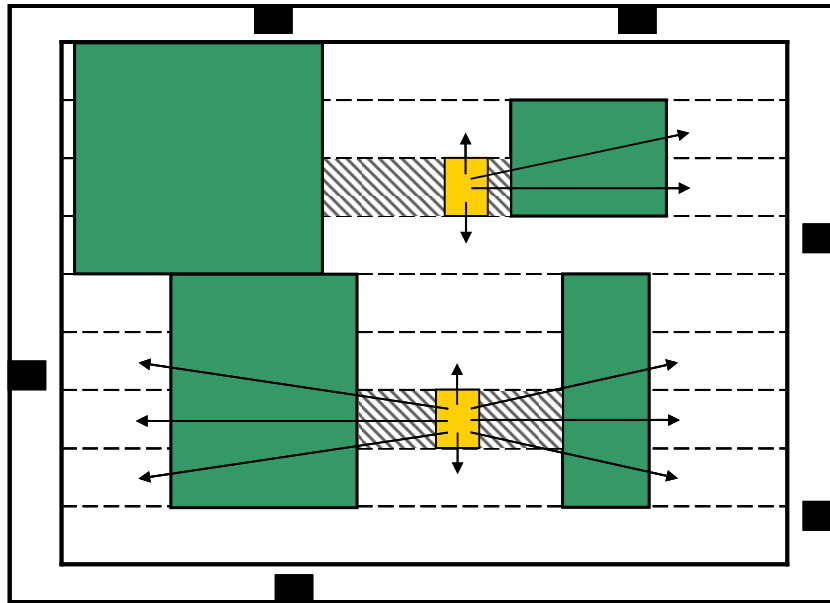


Figure 3.7 Cell movement during slice-based cell legalization.

(b) a function of the utilization of the source and target slices. Since the legalization technique is mainly used to even out the placement and satisfy the capacity requirements associated with the slices, a higher weight is assigned to the second component. In addition, each score is augmented with a *displacement weight* that is inversely proportional to the intended displacement of the cell from its original location (location prior to legalization). The displacement weight is required to prevent the cell from moving by a large distance during legalization.

Once the movement score has been determined for all the cells in a slice, the cells are moved out of the slice in the decreasing order of their movement scores. During one legalization iteration, we traverse through all the slices that are above capacity and follow the above steps for cell movement. Subsequently, this iteration is repeated until all the slices satisfy their capacity requirements.

Please note that any given objective can be used during cell movement to satisfy the slice capacity requirements. In addition, the search space for the target slice need not be restricted to the nearest eight neighboring slices. For a given source slice, any number of target slices can be considered to move a cell to satisfy the legalization objective.

3.4.3 Advantages of Slice-based Legalization over Bin-based Legalization

Traditional standard-cell legalization techniques like diffusion [52,53] and network flow [38] typically follow a bin-based cell movement approach. During legalization, these techniques divide the placement region into equal sized bins and determine the utilization of each bin. They then move the cells among the bins to satisfy the capacity requirement associated with each bin. Bin-based schemes have two major drawbacks:

- They cannot effectively handle fractured spaces which are very common in current integrated circuits. As an example, during legalization, it might be required to move a cell across a fixed module for it to be legalized. Bin-based techniques typically do not allow cell movement on top of fixed modules. As a result, they might not be able to find a legal placement solution.
- Satisfying bin capacity does not guarantee a legalized placement. This is shown in Figure 3.8(a) and Figure 3.8(b). These figures show a placement region with two large fixed modules, four small fixed modules (both depicted by the dark shaded boxes) and eight standard-cells (depicted by the light shaded boxes). One of the bins in the regular bin structure employed by the legalization technique is shown by the thick solid lines. In Figure 3.8(a) the total area of the cells in the bin is equal to the capacity of the bin. But, once the cells are aligned and legalized within the circuit rows, it can be seen from Figure 3.8(b), that cells 3 and 5 overlap with the adjacent fixed modules.

One method to overcome these drawbacks is to increase the number of bins in the regular bin structure. But this would significantly increase the runtime of the legalization algorithm. Secondly, such a technique might not be able to legalize the modules with minimum perturbation from their original locations. Hence, bin-based algorithms are not very effective in highly fragmented spaces.

In contrast, the key advantages of a slice-based legalization technique are:

- Since the slice boundaries are aligned to the boundaries of the fixed modules, cells in a slice-based legalization technique have the ability to jump over fixed modules if required.

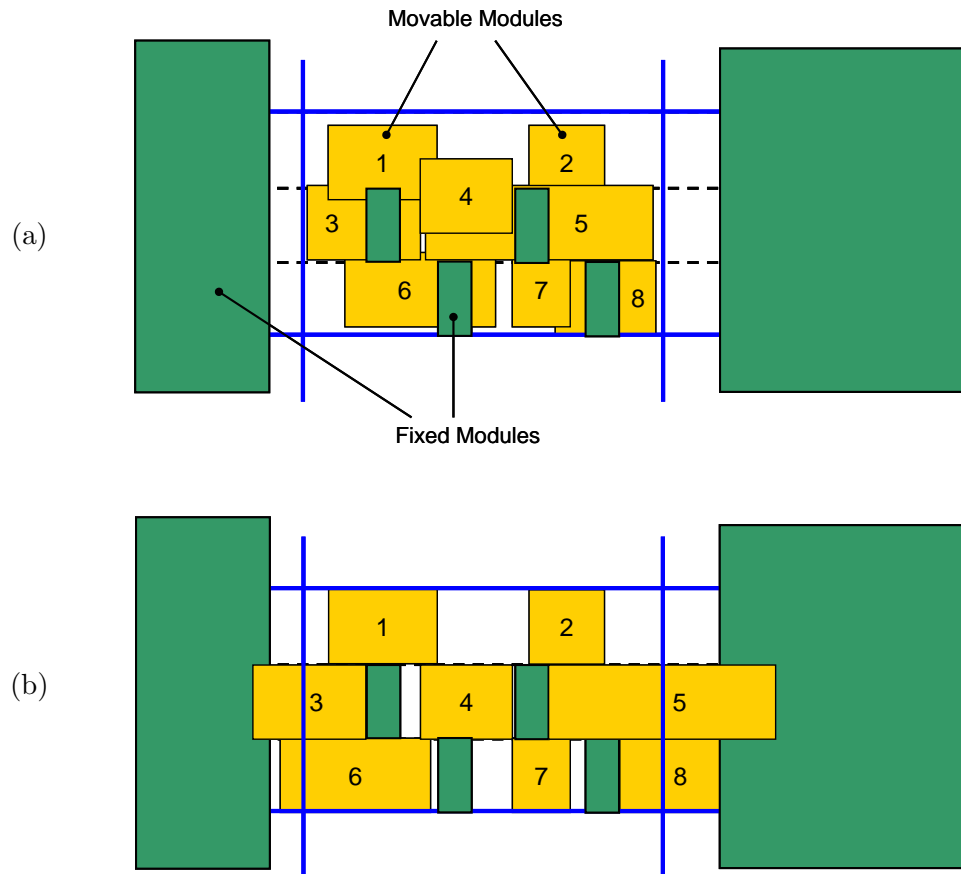


Figure 3.8 Disadvantage of a bin-based legalization technique – satisfying bin capacity does not guarantee a legal placement. (a) Total area of all the cells within the bin is equal to the capacity of the bin. Bin-based techniques stop when this criteria is met (b) Subsequent alignment and final legalization of the cells, leading to an overlapping placement.

- Slice-based legalization techniques can effectively incorporate bin-based cell movement techniques to satisfy slice capacities as well as bin density target requirements. This is shown by the slice aware bin-based cell movement technique.
- If the placement region is divided into slices, then satisfying slice capacities guarantees a legal placement solution (Figure 3.9).

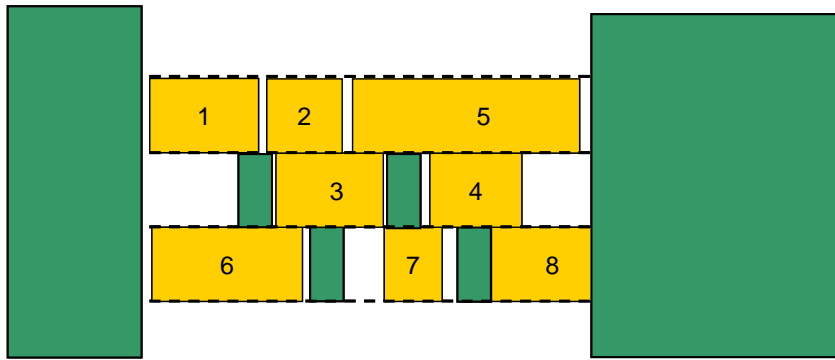


Figure 3.9 Advantage of a slice-based legalization technique – satisfying slice capacity guarantees a legal placement.

CHAPTER 4. EXPERIMENTAL RESULTS

This chapter describes the placement results of FastPlace and RQL on the ISPD-2005 [44] and ISPD-2006 [43] placement contest benchmarks. It compares the two algorithms with other state-of-the-art academic placement algorithms in terms of the half-perimeter wire length and the runtime for placement. Since RQL is the more recent of the two placement algorithms, it is used as the basis for all the comparisons. The placement runtimes are reported on a 2.6 GHZ AMD Opteron 252 machine with 8 GB RAM.

4.1 Benchmark Circuits

The ISPD-2005 and ISPD-2006 benchmarks have been derived from industrial ASIC designs and have circuit sizes ranging from 210k to 2.48M movable modules. In addition, the ISPD-2006 benchmarks are placement congestion constrained, with a specific density target assigned to each circuit. Therefore, in addition to obtaining a good wire length, the placement algorithms need to satisfy the density target associated with each circuit of the benchmark suite. Statistics for the two benchmark suites are shown in Table 4.1.

4.2 Placement Results on the ISPD-2005 Benchmarks

Table 4.2 compares the half-perimeter wire length (HPWL) of RQL and FastPlace with other state-of-the-art academic placers. For this experiment all the placers were run in their default mode. From Table 4.2, in default mode, **RQL obtains the best HPWL on all eight circuits of the ISPD-2005 benchmark suite.** In particular, RQL obtains an average wire length improvement of 3.7%, 2.8%, 8.5%, 14.6%, 3.2% and 5.4% versus FastPlace, mPL6, APlace2.0, Capo10.2, NTUPlace3 and Kraftwerk respectively.

Table 4.1 Statistics for the ISPD-2005 and ISPD-2006 placement benchmarks.

Circuit	Total Modules	Movable Modules	Fixed Modules	Nets	Design Utilization (%)	Density Target (%)
ISPD-2005 Benchmarks						
adaptec1	211447	210904	543	221142	57.34	–
adaptec2	255023	254457	566	266009	44.32	–
adaptec3	451650	450927	723	466758	33.66	–
adaptec4	496045	494716	1329	515951	27.23	–
bigblue1	278164	277604	560	284479	44.67	–
bigblue2	557866	534782	23084	577235	37.94	–
bigblue3	1096812	1095519	1293	1123170	56.68	–
bigblue4	2177353	2169183	8170	2229886	44.35	–
ISPD-2006 Benchmarks						
adaptec5	843128	842482	646	867798	49.98	50
newblue1	330474	330137	337	338901	70.69	80
newblue2	441516	330239	1277	465219	61.66	90
newblue3	494011	482833	11178	552199	26.31	80
newblue4	646139	642717	3422	637051	46.45	50
newblue5	1233058	1228177	4881	1284251	49.56	50
newblue6	1255039	1248150	6889	1288443	38.78	80
newblue7	2507954	2481372	26582	2636820	49.31	80

Table 4.2 HPWL ($\times e6$) comparison of FastPlace and RQL with existing academic placers – mPL6 (MP), APlace2.0 (AP), Capo10.2 (CP), NTUplace3 (NP) and Kraftwerk (KW), on the ISPD-2005 placement benchmarks. [*avg of 6 circuits]

Circuit	Placer (Default Mode Runs)						
	RQL	FastPlace	MP [9]	AP [32]	CP [57]	NP [12]	KW [61]
adaptec1	77.54	77.82	77.91	78.35	91.28	80.93	NA
adaptec2	88.51	93.33	91.96	95.70	100.75	89.95	93.84
adaptec3	210.96	212.89	214.05	218.52	228.47	214.20	NA
adaptec4	188.86	197.05	194.23	209.28	208.35	193.74	199.75
bigblue1	94.98	95.62	96.79	100.02	108.60	97.28	99.61
bigblue2	150.03	153.42	152.33	153.75	162.92	152.20	155.19
bigblue3	323.09	362.73	344.37	411.59	398.49	348.48	339.20
bigblue4	797.66	831.29	829.35	871.29	965.30	829.16	857.09
Average	1.000	1.037	1.028	1.085	1.146	1.032	1.054*

Table 4.3 compares the wire length results of RQL in default mode with the wire length obtained by the top three placers during the ISPD 2005 placement contest. It should be noted that during the contest, all the placers were given the circuits in advance. There was no limit on the CPU time and the placers were allowed to have *separate* parameters for each individual circuit to obtain the best possible wire length. **RQL in default mode obtains an average HPWL improvement of 1.5% as compared to APlace, which generated the best HPWL results during the ISPD 2005 placement contest.** Till date, the APlace contest wire length (reproduced in column three of Table 4.3) was the best reported results in the literature on these circuits.

Table 4.3 HPWL ($\times e6$) comparison of RQL with the top performing academic placers during the ISPD-2005 placement contest.

Circuit	Default Mode	Placer is tuned for each individual circuit			
	RQL	APlace [32]	mFAR [28]	Dragon [63]	mPL [8]
adaptec2	88.51	87.31	91.53	94.72	97.11
adaptec4	188.86	187.65	190.84	200.88	200.94
bigblue1	94.98	94.64	97.70	102.39	98.31
bigblue2	150.03	143.82	168.70	159.71	173.22
bigblue3	323.09	357.89	379.95	380.45	369.66
bigblue4	797.66	833.21	876.28	903.96	904.19
Average	1.000	1.015	1.079	1.098	1.105

Table 4.4 compares the runtime of RQL and FastPlace with mPL6, APlace 2.0 and Capo10.2. On average, RQL is 3.09 times, 10.22 times and 6.99 times faster than mPL6, APlace2.0 and Capo10.2 respectively, but it is about 2.0 times slower than FastPlace.

4.3 Placement Results on the ISPD-2006 Benchmarks

Table 4.5 compares the HPWL of RQL and FastPlace with existing academic placers. The results for NTUPlace3 [12] are reported from the associated publication. The results for all other placers are from the ISPD 2006 placement contest. From Table 4.5, on average RQL is 12%, 5% and 4% better in HPWL as compared to Kraftwerk, mPL6 and NTUPlace2 respectively. These algorithms were the top three placers during the ISPD 2006 placement

Table 4.4 Runtime comparison of RQL and FastPlace with existing academic placers – mPL6, APlace2.0 and Capo10.2, on the ISPD-2005 placement benchmarks.

Circuit	Placer				
	RQL (sec)	$\frac{FastPlace}{RQL}$	$\frac{mPL6}{RQL}$	$\frac{APlace2.0}{RQL}$	$\frac{Capo10.2}{RQL}$
adaptec1	626	0.42	3.22	9.42	6.57
adaptec2	1039	0.40	2.17	8.83	5.44
adaptec3	2004	0.62	3.58	11.11	6.31
adaptec4	1747	0.47	3.87	14.39	6.59
bigblue1	967	0.46	2.83	8.81	6.93
bigblue2	1884	0.58	4.14	10.64	7.06
bigblue3	4053	0.74	2.59	9.31	9.38
bigblue4	10342	0.46	2.33	9.30	7.62
Average		0.52×	3.09×	10.22×	6.99×

contest. The most current results of NTUPlace3, shows an average improvement of 1% in HPWL over RQL.

Finally, Table 4.6 compares the scaled HPWL of RQL and FastPlace with the other academic placers. The scaled HPWL (S_HPWL) is a weighted function of the wire length and placement congestion and is defined as [43]:

$$S_HPWL = HPWL \times (1 + density_overflow_penalty)$$

where the *density_overflow_penalty* penalizes the placers that do not meet the density target requirements. From Table 4.6, on average RQL is 7%, 1% and 2% better in terms of S_HPWL as compared to Kraftwerk, mPL6 and NTUPlace2 respectively. The average S_HPWL of NTUPlace3 is comparable to that of RQL, but looking at individual results, RQL obtains better S_HPWL on five out of the eight circuits as compared to NTUPlace3.

Table 4.5 HPWL ($\times e6$) comparison of RQL and FastPlace with existing academic placers on the ISPD-2006 placement benchmarks.

	adaptec5	newblue1	newblue2	newblue3	newblue4	newblue5	newblue6	newblue7	Average
RQL	405.73	64.21	196.74	269.13	268.07	473.14	494.30	1031.33	1.00
FastPlace	432.96	78.56	201.51	292.58	284.54	530.12	539.44	1124.55	1.10
Aplace3	449.61	73.26	197.42	273.63	377.55	545.90	522.58	1098.26	1.12
mFAR	448.43	77.36	211.65	303.58	307.73	567.65	527.36	1135.80	1.13
Dragon	500.24	80.76	259.95	524.41	340.70	613.34	572.19	1408.97	1.36
mPL6	425.12	66.90	197.53	283.80	294.43	530.67	510.40	1070.33	1.05
Capo	491.60	98.35	308.64	361.21	358.28	657.40	668.33	1518.49	1.40
NTUPlace2	404.98	62.40	201.95	291.14	284.99	494.57	504.39	1116.86	1.04
Kraftwerk	444.07	78.29	205.87	279.94	311.09	555.48	537.32	1139.17	1.12
DPlace	463.95	102.37	324.07	379.19	305.78	600.11	674.39	1398.85	1.37
NTUPlace3	378.56	60.74	198.76	278.87	274.48	474.84	484.81	1056.78	0.99

Table 4.6 Comparison of the Scaled HPWL (S_HPWL) ($\times e6$) which includes the HPWL and density target based overflow penalty, on the ISPD-2006 placement benchmarks.

	adaptec5	newblue1	newblue2	newblue3	newblue4	newblue5	newblue6	newblue7	Average
RQL	443.28	64.43	199.60	269.33	308.75	537.49	515.69	1057.79	1.00
FastPlace	517.56	78.75	202.98	294.77	325.06	633.21	546.72	1139.24	1.11
APlace3	520.97	73.31	198.24	273.64	384.12	613.86	522.73	1098.88	1.10
mFAR	476.28	77.54	212.90	303.91	324.40	601.27	535.96	1153.76	1.10
Dragon	500.74	80.77	260.83	524.58	341.16	614.23	572.53	1410.54	1.29
mPL6	431.14	67.02	200.93	287.05	299.66	540.67	518.70	1082.92	1.01
Capo	494.64	98.48	309.53	361.25	362.40	659.57	668.66	1518.75	1.33
NTUPlace2	432.58	63.49	203.68	291.15	305.79	517.63	532.79	1181.30	1.02
Kraftwerk	457.92	78.60	208.41	280.93	315.53	569.36	545.94	1170.85	1.07
DPlace	572.98	102.75	329.92	380.14	364.45	752.08	682.87	1438.99	1.40
NTUPlace3	448.58	61.08	203.39	278.89	301.19	509.54	521.65	1099.66	1.00

PART II

PLACEMENT IN A PHYSICAL SYNTHESIS FLOW

CHAPTER 5. CLOCK CONSTRAINT AWARE TIMING-DRIVEN PLACEMENT

5.1 Introduction

In nanometer-scale technology, power dissipation is one of the most important concerns for high-performance circuit design. Power dissipation can also cause excessive heat, which leads to other problems such as variability and cost of cooling. More importantly, power dissipation impacts the reliability of the design, with low-power designs typically being more reliable [46].

Usually, the dynamic (or switching) power is the major component of the overall design power. It is especially a key concern for high-performance designs like microprocessors. For microprocessors, it is a well-known fact that the clock distribution network including the clocked elements (e.g. latches, flip-flops and macros) consumes a majority of the total dynamic power, because the clock network usually switches during every clock cycle [22, 50]. Studies [20, 55] have shown that the leaf level of the clock tree, i.e., nets driving the clocked elements (hereafter referred to as latches), is a major power consumer among the entire clock distribution network. Hence, to minimize clock network power, it is crucial to minimize the local clock interconnect which connects from the local clock buffers (LCBs) to the latches. In other words, the distance between the LCBs and their driven latches needs to be minimized.

In a conventional physical synthesis flow, a tight LCB/latch placement is accomplished by imposing an explicit maximum allowable distance constraint between an LCB and its driven latch. This is called the *LCB to latch distance constraint*. The benefits of an LCB to latch distance constraint are two fold: (a) it reduces the local clock network wire capacitance leading to significant reduction in the clock power consumption, (b) the maximum clock skew of latches is naturally minimized leading to less usage of buffers.

5.1.1 Previous Work on Local Clock Tree Synthesis Methodology

In recent years, latch clustering has emerged as an important technique to reduce the total capacitance of the local clock network. To achieve this reduction, clustering techniques typically attempt to minimize the total interconnect capacitance from an LCB to the latches.

Lu *et al.* [39] use the concept of *register anchors* to reduce the clock net wire length in a timing-driven quadratic placement framework. A register anchor is essentially the center of gravity of a subset of latches in the design. The register anchors indicate preferred locations for the latches and are appended to a normal quadratic placement formulation as additional constraints. Experimental results show a significant reduction in the clock net wire length with a negligible increase in the wire length of the signal nets. The technique is also expanded to address zero-skew placement of latches. Although the weighted wire length can serve as an approximation for the timing, it is an extremely crude metric to evaluate the timing quality of the generated placement.

A more sophisticated power-aware placement algorithm is proposed in Cheon *et al.* [15] using activity-based latch clustering and net-weighting techniques. The work shares the same goal of minimizing lower-level clock power in a design by reducing the capacitance of the clock nets. Although the paper reports significant reduction in the power on a set of real designs, it primarily focuses on clock power reduction. As a result, it might increase the signal net timing and power while minimizing the same between the LCBs and latches (a fact also admitted by the work). Based on the signal net activity rate and the switching rate of the latches, the work determines a bound for each group (the size of the bounding box where all associated latches need to be placed). It then determines a set of net-weights, which are kept constant during the subsequent placement. A key disadvantage with constant net-weights is that it does not allow the placement algorithm to dynamically trade-off between the conflicting objectives of clock net and signal net minimization. In addition, in high performance designs, tightly packed latch clusters are considered more preferable solutions, mainly due to clock skew problems, which may significantly increase the number of inserted clock buffers.

Shelar [59] proposed a latch clustering algorithm to be used within a traditional clock tree

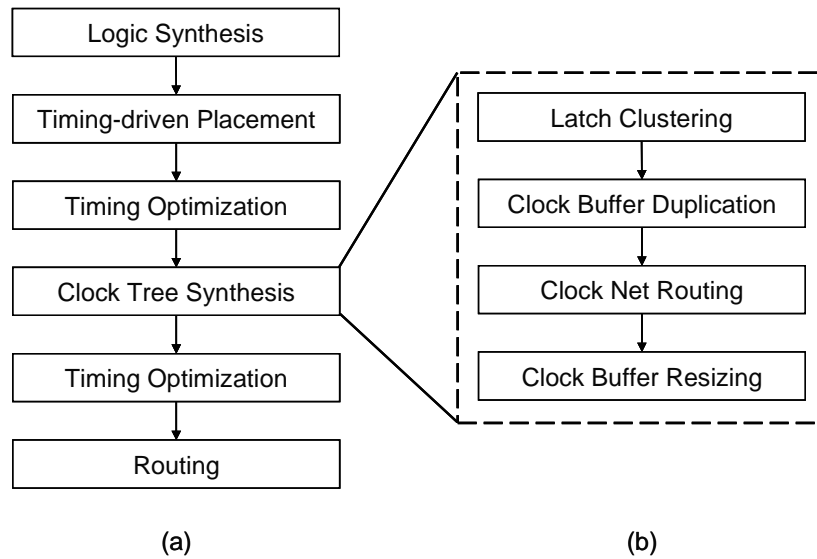


Figure 5.1 (a) Traditional physical design flow (b) Clock tree synthesis methodology employed within the physical design flow.

synthesis methodology, depicted in Figure 5.1. His work minimizes the interconnect capacitance in the clock tree by using the minimum spanning tree (MST) metric as an estimation of the interconnect capacitance. The benefits of the proposed technique are shown by embedding the latch clustering within an industrial physical synthesis flow and performing more rigorous timing and power measurements. However, the clustering method proposed in [59], and the flow in Figure 5.1 have a vital limitation, namely, the latches are fixed in place before performing local clock tree synthesis (LCTS). This methodology suffers from the following issues:

- The reduction in the local clock interconnect is bounded by the locations of the latches. Since the latches are fixed, irrespective of the clustering, we need to connect from the clock spine to the LCBs, and in turn, to the latches. As a result, the local clock interconnect can still be quite large. This issue is illustrated in Figure 5.2(a), which shows twelve latches (rectangles) within the block boundary (dotted lines). Assume an LCB (square) can drive upto six latches. Then, the latches can be partitioned into two clusters, with each cluster being driven by an LCB placed at the centroid of the cluster. The lines between

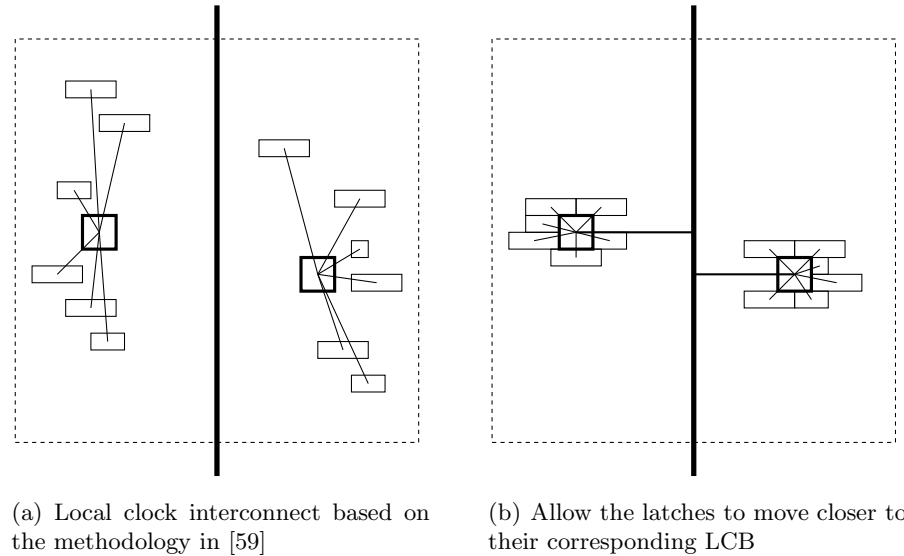


Figure 5.2 Reduction in the local clock interconnect by following an enhanced local clock tree synthesis methodology.

the LCB and the latches represent the clock routes that will be tuned for skew/slew by a clock router after the clustering and LCB duplication steps in Figure 5.1(b). However, if we are allowed to move the latches, as shown in Figure 5.2(b), the local clock interconnect can be *truly* minimized.

- The timing-driven placement of latches is based on an ideal clock. Due to the inaccuracy in timing analysis by using ideal clock arrival times, the resulting placement can lead to degraded timing when the clock trees are actually routed. In the traditional flow, since the latches are fixed, the subsequent steps (clock routing, LCB placement and sizing) are constrained to use these potentially incorrect latch locations. This can lead to situations where even post-LCTS optimization may not be able to fix all the timing problems.

In [50], Puri *et al.* proposed a design style where the latches are placed quite close to their driving LCB and form a “latch huddle”. This is illustrated in Figure 5.3, which shows one LCB (long red rectangle in the center), with a cluster of latches (small red rectangles) placed right next to the LCB. The black lines from the center of the LCB show the latches that are

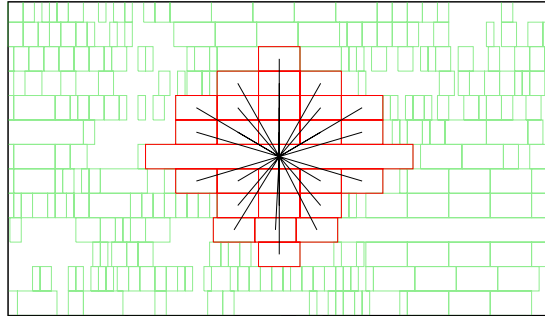


Figure 5.3 Latch huddle around a local clock buffer.

being driven by this LCB. Results in [50] show that this design style can significantly reduce the capacitive load on the clock signals driven by the LCBs, and also improve timing.

However, the design style proposed in [50] requires the latches to be moved after clustering and LCB duplication. This requires a more sophisticated clock tree synthesis methodology as shown in Figure 5.4. In this flow, once timing-driven placement has placed the latches in their ideal locations with respect to signal net (i.e., non-clock related net) timing; clustering and LCB duplication is performed. An example clustering followed by LCB duplication is shown in Figure 5.5, where the rectangles and lines represent the same information as Figure 5.3. After clustering, a timing analysis is performed and timing-driven net-weights are generated to reflect the timing on the signal nets. Subsequently, a new timing-driven placement with explicit LCB to latch clustering constraints is run to obtain the latch huddles shown in Figure 5.6. From Figure 5.4(b), a new timing-driven placement is required for the following reasons:

- After clustering, it is possible for a cluster to overlap with other clusters in the design (for example, the cluster in the top-right corner of Figure 5.5). This can happen due to the presence of multiple clock domains in the design, which require dedicated LCBs per domain. It can also happen due to inferior solutions being produced by the placement/clustering algorithm. Hence, this overlap among the clusters needs to be resolved.
- The previous issue leads to certain clusters having a very large spread. As a result, the latches may need to move by a significant distance.

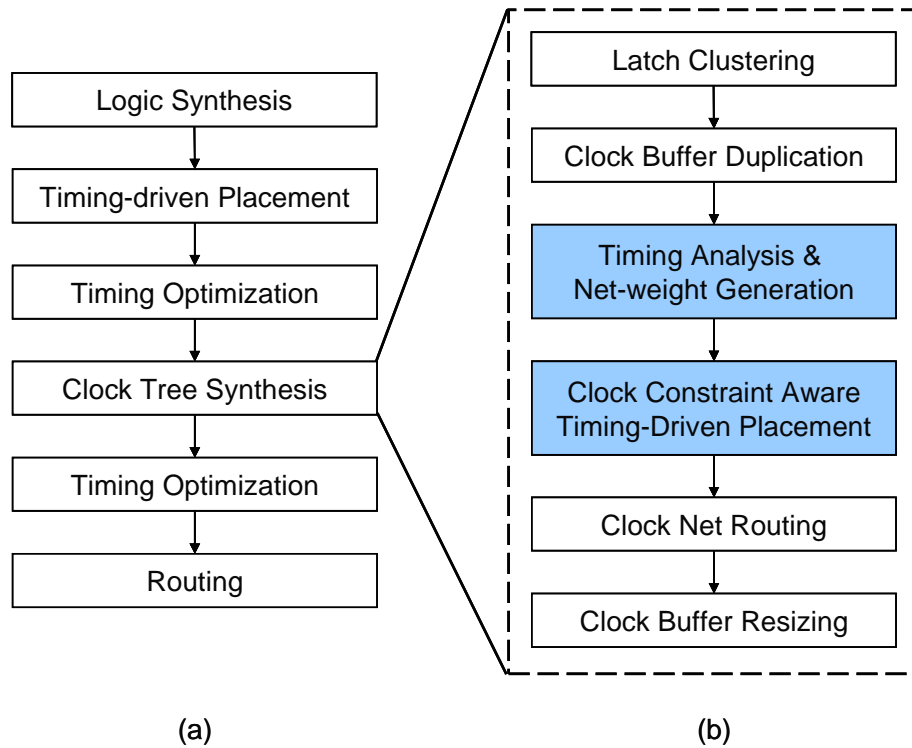


Figure 5.4 (a) Physical design flow for optimized local clock network (b) Enhanced clock tree synthesis methodology (the shaded boxes represent the enhancements).

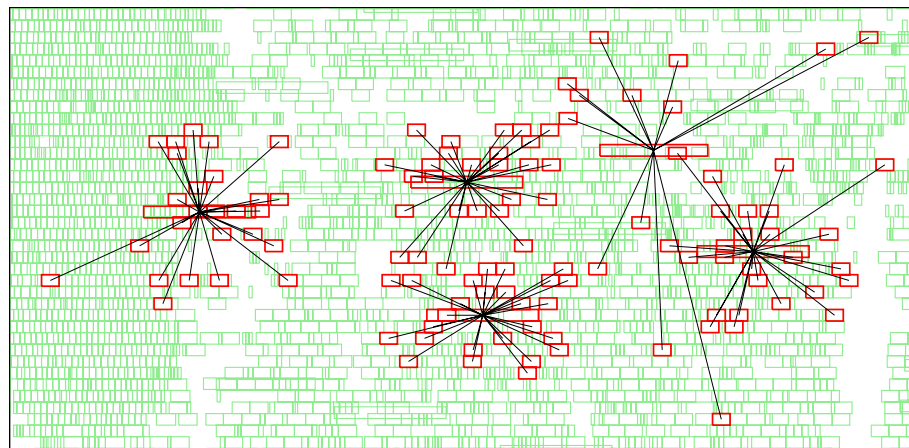


Figure 5.5 Latch clustering and LCB duplication during local clock tree synthesis.

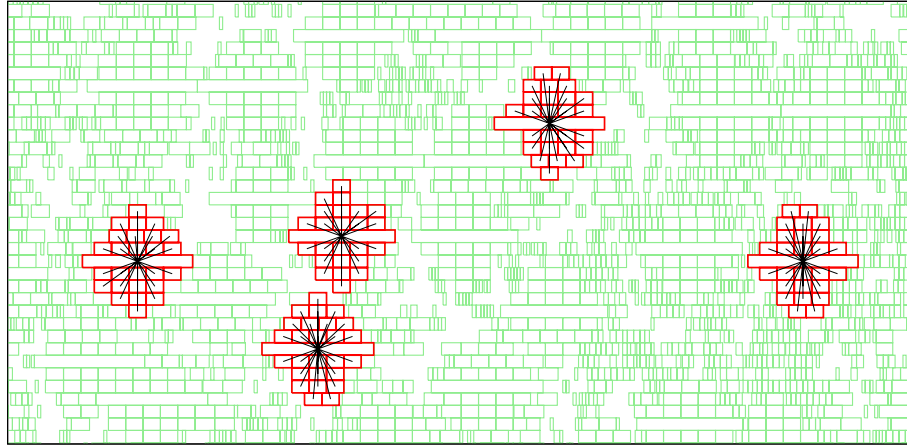


Figure 5.6 Latch clusters after clock constraint aware timing-driven placement.

- Movement of the latches affects the non-clocked modules and signal net timing. The non-clocked modules need to react to the movement of the latches.

Therefore, to meet timing requirements while satisfying the *LCB to latch distance constraint*, a separate clock constraint aware timing-driven placement is required after clustering. Please note, from herein, a “latch cluster” refers to an LCB and all its driven latches.

5.1.2 Previous Work on Handling Clock Constraints During Timing-driven Placement

State-of-the-art placers [6, 9, 12, 32, 61, 66] typically handle the weighted wire length minimization problem, and do not pay any attention to clocking structures like latch clusters.

Typically, the input to a timing-driven global placement algorithm is a weighted netlist hypergraph, where the individual nets in the input netlist are assigned weights based on their timing criticalities. The rationale behind this approach is that critical nets get higher weights during placement and hence get minimized at the expense of the non-critical nets. The objective of the placement algorithm is to minimize the weighted total wire length so as to improve the overall timing of the design. Although timing-driven net-weighting is a crude model to

represent design timing, in practice, it serves well to find a globally good placement solution by minimizing the weighted wire length objective.

To obtain tight latch clusters in such a scenario, designers typically set an artificially high net-weight between an LCB and its driven latches. Hence, during timing-driven placement, there are two kinds of interacting net-weights: (a) the “true” timing-driven net-weights, (b) the “artificial” net-weights between an LCB and its driven latches.

The key drawback with this approach is in the interaction between these two sets of net-weights. Typically, the LCB to latch net-weights dominate the timing-driven net-weights. As a result, the latches cannot freely migrate to their natural locations as dictated by the timing-driven net-weights. This in turn adversely impacts the timing characteristics of the design. Another issue with this approach is that the latches are not differentiated by their timing criticalities while handling the LCB to latch placement (distance) constraint. Since the timing criticality of the latches are reflected by the timing-driven net-weights (albeit in an indirect manner), it would be beneficial to use the more critical latches in the design to guide the placement of their associated latch cluster.

Another potential approach to obtain tight latch clusters is to consider the entire latch cluster as a single pseudo-object (or cluster) during global placement. The key drawback with this approach is in determining the dimensions of the pseudo-object during placement. Incorrect dimensions might lead to a bad global placement. This can in turn lead to problems during legalization, wherein the individual modules (LCB and latches) within the pseudo-object need to be assigned to legal locations in the placement region. Legalization can potentially move the LCB and/or the latches by a large distance and severely degrade the latch cluster placement.

5.2 Key Contributions of This Work

This section outlines the placement techniques that have been developed to simultaneously optimize timing (weighted wire length) while satisfying the LCB to latch distance constraint. These techniques have been embedded within the *RQL* global placement algorithm to yield an

effective force-directed Clock Constraint Aware Timing-driven Placement (CCATP) algorithm. The effectiveness of the CCATP algorithm is demonstrated by experimental results on high-performance industrial designs in the $45nm$ process technology.

The key contributions of this work in the development of a Clock Constraint Aware Timing-driven Placement algorithm are:

- Nominal LCB to latch clock net-weights that do not dominate the timing-driven net-weights during placement.
- Selective modulation of the LCB to latch net-weights during placement to minimize LCB to latch distance without degrading the total weighted wire length.
- A variable net-weight for each LCB to latch connection, where the weight is set proportional to the timing criticality of the latch. This ensures that the more timing-critical latches in the design can guide the physical placement of the LCBs.
- An intermediate legalization scheme to preserve tight latch cluster placement.
- A greedy swapping technique among the latches within a cluster to minimize the number of wires that cross on top of the LCB. This technique helps to improve wire length and routability.

The rest of this chapter is organized as follows: Section 5.3 gives an overview of the CCATP algorithm. Section 5.4 describes the key components of the CCATP algorithm in detail. Experimental results are reported in Section 5.5, followed by the key observations in Section 5.6.

5.3 Overview of Clock Constraint Aware Timing-driven Placement

Figure 5.7 presents the high-level flow for Clock Constraint Aware Timing-driven Placement (CCATP) as used within the enhanced clock tree synthesis methodology of Figure 5.4(b). The input to the placement algorithm are: (a) a weighted netlist, wherein the individual nets are weighted according to timing criticality, (b) a set of LCBs and corresponding latches that are identified by the latch clustering and LCB duplication steps. The shaded boxes in Figure

5.7 represent the enhancements to the force-directed global placement framework of *RQL* to handle latch cluster placement. From Figure 5.7, the key steps during CCATP are:

1. Initial Netlist Processing: This step determines the latch criticalities based on the timing-driven net-weights. It then sets nominal LCB to latch net-weights for the coarse global placement step (Section 5.4.1).
2. Circuit Clustering: As mentioned in Section 2.7 and Algorithm 2.5, circuit clustering is performed to improve the efficiency, scalability and solution quality of the placer. Since the latch clusters are processed in a special manner, the LCBs and latches are excluded from the circuit clustering phase and hence, are not clustered with any other module in the design.
3. Coarse Global Placement: This phase comprises of three steps: (a) Quadratic placement with force-vector modulation (b) Density-aware module spreading (c) Iterative Local Refinement.
4. LCB to Latch Net-weight Modulation: During placement, the LCB to latch net-weights are selectively increased for clusters that have a large spread (Section 5.4.2).
5. Placement Refinement: During this phase, a series of unclustering and Iterative Local Refinement steps are performed until a placement of the flat netlist is obtained.
6. LCB Legalization: Once the modules in the design have been reasonably spread over the placement region, and the latches are in close proximity to their driving LCBs, the LCBs are assigned to legal locations within the placement region (Section 5.4.3).
7. Latch Legalization: After LCB legalization, another round of placement refinement is performed to allow the latches to adjust to the legalized locations of the LCBs. Following which, the latches are legalized around their driving LCBs. During this process, a greedy swapping technique is applied among the latches within a cluster to reduce the number of wires that cross on top of the LCBs (Section 5.4.4).

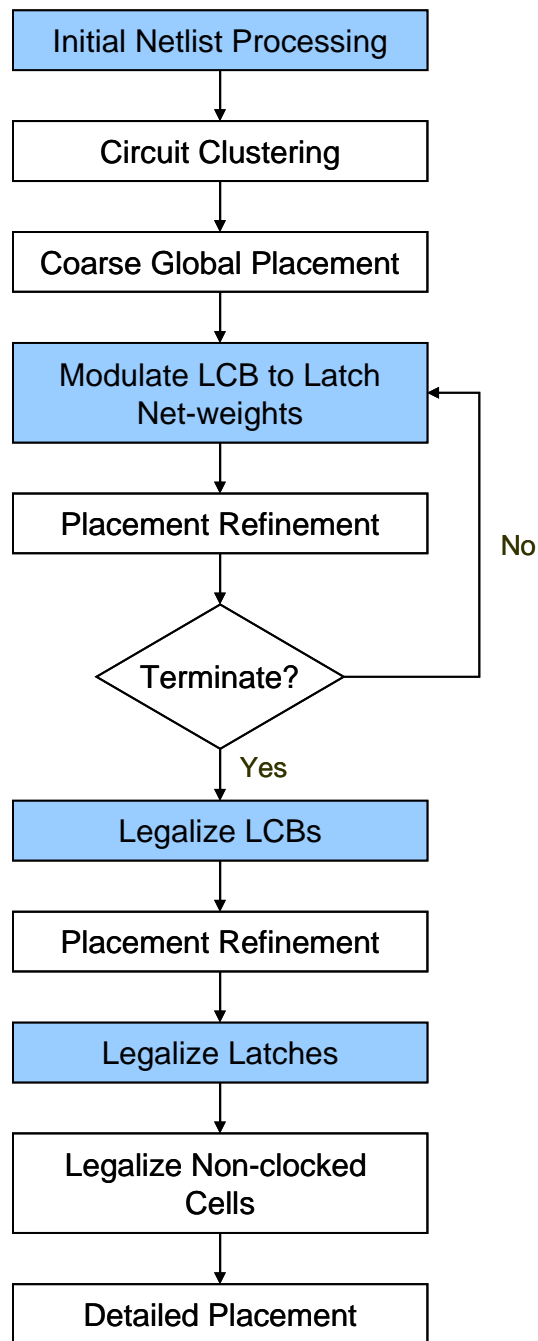


Figure 5.7 High-level flow for Clock Constraint Aware Timing-driven Placement (CCATP).

5.4 Handling Latch Clusters During Timing-driven Placement

This section describes the key components (shaded boxes in Figure 5.7) of the CCATP algorithm with the help of an example given in Figure 5.8. Figure 5.8(a) shows an example circuit with two LCBs (solid squares), each driving four latches (empty rectangles). The clock nets between an LCB and its driven latches are depicted by the dashed lines and the signal nets in the circuit are depicted by the solid lines. For the sake of simplicity, all the signal nets in the circuit are two-pin nets and connect directly to the latches.

As shown in Figure 5.4(b), before placement, a timing analysis is performed and weights are assigned to the signal (non-clock) nets reflecting their timing criticalities. To perform timing-driven net-weighting, the sensitivity guided net-weighting scheme of Ren *et al.* [54] is used.

5.4.1 Initial Netlist Processing

The purpose of the initial netlist processing step is two-fold: (a) determine the relative criticalities of the latches in the design, (b) assign the initial net-weights to the clock nets between an LCB and its driven latches.

In addition to the LCB, a latch belonging to a latch cluster, is connected to a set of non-clocked modules via the signal nets. As mentioned before, the signal nets are weighted according to their timing criticalities determined from a timing analysis step. The relative criticalities of the latches are determined based on the sum of all the timing-driven net-weights on the signal nets that connect the latches to the non-clocked modules in the design. For any two latches in the design, the one with a higher total signal net-weight is deemed more critical.

For each clock net between an LCB and its driven latch, the net-weight comprises of two components: (a) a constant, *base_weight*, which is equal to the nominal net-weight of a non-critical signal net, (b) a variable, *criticality_weight*, which reflects the timing criticality of the latch. To bias the latch cluster towards the more critical latch(es) within the cluster, the *criticality_weight* component is given a value that is proportional to the signal net-weights used in determining the relative latch criticalities (as outlined before).

With respect to a single latch cluster, one method to set the net-weights on the clock nets between the LCB and its driven latches is as follows:

- Step 1: Assign a nominal net-weight to the *base_weight* component.
- Step 2: For each latch within the cluster, determine the sum of the net-weights on the signal nets connected to the latch. Let this be denoted as *latch_signal_net-weight*.
- Step 3: Set the *criticality_weight* component for the clock net on the latch with the lowest *latch_signal_net-weight* to be *minimum_criticality_weight*.
- Step 4: Set the *criticality_weight* component for the clock net on the latch with the highest *latch_signal_net-weight* to be *maximum_criticality_weight*.
- Step 5: Use linear scaling to determine the *criticality_weight* component for the clock nets on all the other latches in the cluster.
- Step 6: For each latch, set the LCB to latch clock net-weight as:

$$clock_net_weight = criticality_weight \times base_weight.$$

Within CCATP, the *minimum_criticality_weight* is assigned a value of 0.1, and the *maximum_criticality_weight* is assigned a value of 1.0.

The initial netlist processing step is depicted in Figure 5.8(a), where the criticality of the various signal nets are reflected by the thickness of the lines depicting the nets. As seen from the figure, the latches connected to the PIs/POs have low timing-driven net-weights, whereas, the two latches that are connected to each other have a very high timing-driven net-weight. Based on the net-weights on the signal nets, for each LCB to latch clock net, the appropriate *criticality_weight* and corresponding *clock_net_weight* is determined using the steps outlined above.

5.4.2 LCB to Latch Net-weight Modulation

To tighten specific latch clusters in the design, LCB to latch net-weight modulation is used at various stages in the placement flow. To identify the latch clusters with a large span,

initially, for each latch cluster, the average and maximum distance of the latches from the center of their driving LCBs is determined. A weighted function of the average and maximum distance values is then calculated. The latch clusters with a large span are identified as the ones that have a function value greater than a specified threshold. To optimize the placement of these latch clusters (reduce their span), the *base_weight* component of the LCB to latch clock net-weight is then increased for all the latches within these clusters. This is followed by a set of placement refinement steps wherein the span of these latch clusters is gradually reduced.

The LCB to latch net-weight modulation step is shown in Figure 5.8(c) where the *base_weight* component of all the clock nets is increased by a factor of two. Figure 5.8(d) then shows the locations of the latch clusters after the subsequent placement refinement step.

LCB to latch net-weight modulation is a critical step in the overall CCATP algorithm. Since the net-weights on the clock nets are increased in a gradual manner, the latch cluster placement is optimized over several iterations of refinement. This in turn allows the non-clocked modules to adjust their locations in a gradual manner based on the movement of the latches, thereby optimizing the signal net wire length. As a result, there is minimal impact to the total wire length of the design.

5.4.3 LCB Legalization

Once the modules in the design have been reasonably spread over the placement region and the latches are sufficiently close to the LCBs, the LCBs are assigned to legal locations in the placement region. The LCB legalization step ignores all the other movable modules in the design (both clocked and non-clocked cells). It legalizes the LCBs only in the presence of the fixed macros or placement blockages in the design. LCB legalization is a three step process: (a) in the first step, each LCB is artificially inflated to a size that is proportional to the number of latches being driven by the LCB, (b) this is followed by the legalization step wherein the overlap among the LCBs and/or placement blockages is resolved and the LCBs are assigned to legal locations within the placement region, (c) finally, the LCBs are deflated back to their original sizes. Once the LCBs have been legalized, they are fixed in place and

behave as placement blockages for all the subsequent steps of global placement.

Inflation of the LCBs ensures that the subsequent latch legalization step has sufficient room to accommodate the latches around the LCBs. In addition, considering only the LCBs and placement blockages during LCB legalization ensures that the LCBs are legalized with minimum perturbation from their original locations.

5.4.4 Latch Legalization

LCB legalization is followed by another round of placement refinement for the latches to react and adjust to the legalized locations of the LCBs. Each latch within a cluster is then placed at the center of its driving LCB and legalized by using a *spiral*-based legalization technique. The advantage of a spiral-based legalization technique is that it ensures that the latches are relocated by the minimum possible distance from the center of their driving LCB.

Although legalization ensures that each latch is placed in close proximity to its driving LCB, the spiral-based legalization technique is unaware of the relative locations of the latches before the legalization step. In other words, the initial (before legalization) relative locations of the latches with respect to their driving LCB and the non-clocked modules to which they are connected via the signal nets, is not respected during legalization. This can potentially create a “rat’s nest” of wiring around, and on top of the LCB. Hence, after latch legalization, for each cluster, a greedy swapping is performed among all its latches. The objective of this swapping step is to minimize the number of wires that cross on top of the LCB. This ensures that the latches in the cluster end up being in the correct relative locations with respect to their connections with the LCB and the non-clocked modules in the design.

5.4.5 Advantages of the Clock Constraint Aware Timing-driven Placement Algorithm

The CCATP algorithm incorporates a number of techniques to specifically handle latch clusters within a timing-driven placement framework. The key advantages of the developed techniques are:

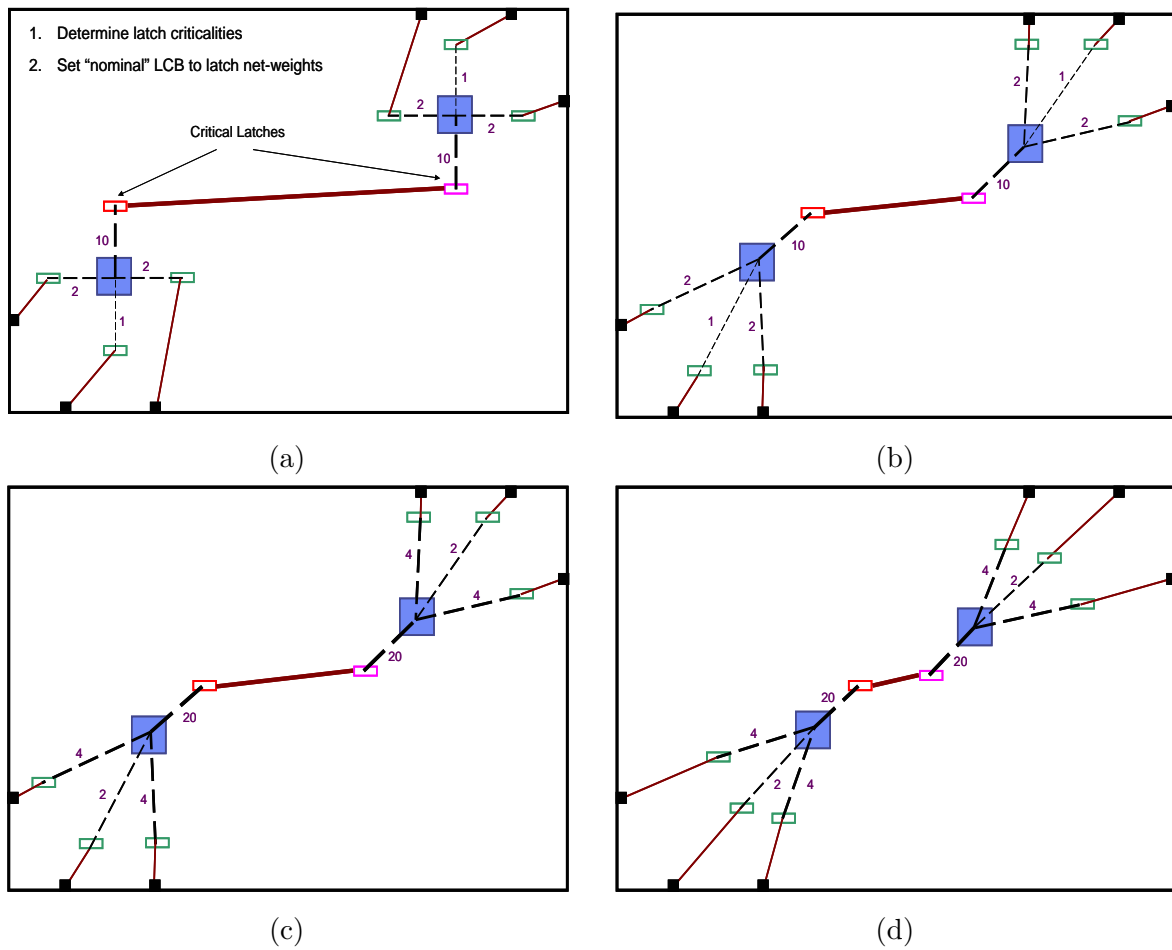


Figure 5.8 Clock constraint aware timing-driven placement on an example circuit. (a) Connections in the circuit and initial netlist processing (b) Locations of the latch clusters after coarse global placement (c) LCB to latch net-weight modulation (d) Locations of the latch clusters after placement refinement.

1. Setting nominal LCB to latch net-weights and selective net-weight modulation does not significantly impact the global placement optimization of the signal nets. This is verified by the experimental results in Table 5.3 and Table 5.4.
2. The conventional approach of setting a constant, high net-weight between an LCB and its driven latches, keeps the latches close to the LCB throughout the placement step. The net effect is that the LCB is not guided by the latches. Rather, the LCB pulls all its driven latches. On the other hand, the effect of low net-weights and gradual modulation within CCATP allows the LCBs to be guided by the latches.
3. Latch criticality based variable net-weights ensure that the LCBs are further guided by the most timing critical latches in the cluster. This improves the overall timing of the design.
4. Intermediate legalization of the LCBs and latches without considering the non-clocked modules ensures tight latch cluster placement satisfying the LCB to latch distance constraints.
5. The inflation of the LCBs during legalization ensures that there is enough space around the LCBs to accommodate their associated latches, and prevent any unexpected spiralling of the latches by a large distance.
6. In addition, a spiral-based legalization technique ensures that the latches are placed at the minimum possible distance from the center of their driving LCB. The subsequent greedy swapping among the latches ensures that the latches end up being in the proper relative location with respect to their driving LCB and their signal net connections.
7. Finally, the developed techniques are general enough to be embedded within any global placement framework.

5.5 Experimental Results

The CCATP algorithm is implemented within the PDS industrial physical synthesis framework [3, 64]. PDS is a state-of-the-art physical synthesis framework that has been used in the design of many high performance integrated circuits. All the results presented in this section are on high performance industrial designs in the 45nm process technology. The design statistics are given in Table 5.1 in terms of the number of modules, the number of clocked elements (Latches) and the number of local clock buffers (LCBs).

Table 5.1 Statistics for a set of high performance industrial designs to test the CCATP algorithm.

Ckt	Modules	Latches	LCBs
ckt_1	69k	19981	1000
ckt_2	103k	8796	387
ckt_3	141k	20209	948
ckt_4	200k	31543	1816
ckt_5	11k	3878	160
ckt_6	232k	48234	2424
ckt_7	102k	18406	1057

To demonstrate the effectiveness of the developed techniques, the following algorithms are compared:

- TP: Default timing-driven placement with high and constant LCB to latch net-weights.
- CCATP: Clock Constraint Aware Timing-driven Placement, that includes: (a) variable net-weights (b) LCB to latch net-weight modulation (c) intermediate LCB and latch legalization followed by greedy swapping of the latches.

For both the algorithms, the sensitivity guided net-weighting scheme of Ren *et al.* [54] was used to generate the timing-driven net-weights for the signal nets. The metrics for comparison are the LCB to latch distance statistics, half-perimeter wire length (HPWL), design timing and global routing congestion at various stages of the physical synthesis flow.

5.5.1 Latch Cluster Placement, Wire Length and Design Timing After Timing-driven Placement

Table 5.2 compares the latch cluster placement statistics between the two algorithms on the following distance metrics:

- Max_D: The maximum LCB to latch distance among all the latch clusters in the design.
- Max(Avg_D): For each latch cluster, let Avg_D represent the average LCB to latch distance within the cluster. Then, Max(Avg_D) gives the maximum value of Avg_D over all the latch clusters in the design.
- D_100: The number of latches that are placed at a distance greater than 100 placement units from the LCB. (As a measure of comparison, a circuit row in these designs is of height 12 placement units).

Table 5.2 Comparison of the LCB to latch distance statistics between timing-driven placement (TP) and clock constraint aware timing-driven placement (CCATP). Max_D: Maximum LCB to latch distance. Max(Avg_D): Maximum value of the average LCB to latch distance within a cluster. D_100: Number of latches placed at a distance greater than 100 placement units from the LCB.

Ckt	Max_D			Max(Avg_D)			D_100	
	TP	CCATP	$\frac{CCATP}{TP}$	TP	CCATP	$\frac{CCATP}{TP}$	TP	CCATP
ckt_1	169	96	0.57	95.11	49.24	0.52	56	0
ckt_2	111	100	0.90	52.49	51.58	0.98	1	1
ckt_3	100	89	0.89	58.82	49.54	0.84	5	0
ckt_4	120	84	0.70	61.39	52.17	0.85	16	0
ckt_5	120	78	0.65	54.65	44.07	0.81	4	0
ckt_6	115	100	0.87	58.59	50.50	0.86	10	1
ckt_7	100	79	0.79	84.85	50.00	0.59	5	0
Avg			0.77			0.78		

As seen from Table 5.2, in spite of using nominal net-weights, CCATP is able to produce superior latch cluster placement as compared to the default timing-driven placement algorithm

using constant, high net-weights. In particular, on average, CCATP is able to reduce the maximum LCB to latch distance (Max.D) by 23% as compared to the default algorithm.

The reasons for this substantial decrease in the LCB to latch distance are as follows:

- In the default flow, during global placement, the high net-weights between the LCB and its driven latches tend to clump the latches on top of the LCB for the entire duration of placement. As a result, it appears as if there is enough space around an LCB to place other modules (clocked as-well-as non-clocked cells). In reality, this is not the case as the latches need to be placed around the LCB in the final solution. Due to the presence of other modules close to the LCB (especially other LCBs), the latches tend to move by a significant amount during legalization. In contrast, CCATP does not set a high LCB to latch net-weight and uses net-weight modulation to guide the placement of the latch clusters. Hence, the latches never end up being on top of the LCB during global placement. This presents a more realistic picture of the placement utilization to the spreading techniques within global placement. As a result, there is enough space around an LCB for the latches to be legalized without significant movement during legalization.
- In addition, CCATP performs the following steps to explicitly reserve the space around an LCB for its associated latches: (a) LCB inflation prior to legalization - this separates the LCBs during legalization thereby creating more space around the LCBs for their associated latches (b) after LCB legalization, an additional placement refinement step is performed which enables the latches to adjust to the legalized locations of the LCBs.

Table 5.3 and Table 5.4 compare the two algorithms after the timing-driven placement step. Table 5.3 shows the total HPWL of the design and Table 5.4 shows the worst slack and Figure of Merit (FOM) of the design. In Table 5.4, the FOM is a weighted sum of all the negative path slacks in the design, and is a measure of the overall slack histogram of the design.

From column four of Table 5.3, on average the CCATP algorithm obtains more than 50% reduction in the total HPWL of the design as compared to the TP algorithm. In addition, columns four and seven of Table 5.4 show a 53% improvement in the worst slack and 69% improvement in the Figure of Merit (overall design timing) for the CCATP algorithm.

Table 5.3 Comparison of the HPWL ($\times e6$) between timing-driven placement (TP) and clock constraint aware timing-driven placement (CCATP) after the timing-driven placement step.

Ckt	TP	CCATP	$\frac{CCATP}{TP}$
ckt_1	32.67	12.68	0.39
ckt_2	26.26	18.05	0.69
ckt_3	62.64	32.76	0.52
ckt_4	106.08	40.42	0.38
ckt_5	35.22	18.44	0.52
ckt_6	167.63	57.11	0.34
ckt_7	52.26	21.14	0.40
Avg			0.46

Table 5.4 Comparison of the design timing (worst slack and timing figure of merit) between timing-driven placement (TP) and clock constraint aware timing-driven placement (CCATP) after the timing-driven placement step.

Ckt	Worst Slack (ns)			Figure of Merit (ns)		
	TP	CCATP	%Improv	TP	CCATP	%Improv
ckt_1	-1.81	-0.19	89.04	-670	-14	97.91
ckt_2	-100.32	-8.93	91.10	-14225	-1949	86.30
ckt_3	-6.18	-5.99	2.95	-2987	-1622	45.70
ckt_4	-3.84	-1.41	63.24	-8444	-295	96.51
ckt_5	0.09	0.11	19.57	0	0	0.00
ckt_6	-12.66	-8.82	30.38	-20833	-6954	66.62
ckt_7	-2.84	-0.69	75.41	-991	-79	92.03
Avg			53.10			69.29

The results in Tables 5.2 – 5.4 show that not only does the CCATP algorithm obtain better latch cluster placement, it also obtains significantly better placement solutions in terms of wire length and timing. These results empirically demonstrate the effectiveness of the techniques incorporated within the CCATP algorithm.

5.5.2 Wire Length, Design Timing and Global Routing Congestion Analysis at the End of Physical Synthesis

As mentioned before, the CCATP algorithm is embedded within the PDS physical synthesis framework. Hence, after placement, a host of timing optimization transforms like buffer insertion, gate sizing, logic restructuring, etc., are run to further improve the design timing. It is quite possible for the gains in HPWL and design timing that were obtained after timing-driven placement, to be lost by the end of physical synthesis. To show that the superior placement results obtained by the CCATP algorithm persist till the end of physical synthesis, Table 5.5 – Table 5.7 give the *final* total HPWL, design timing and global routing congestion analysis results at the end of the physical synthesis flow. In Table 5.7, the routing congestion of a design is measured by invoking an industrial strength global router. The numbers reported in the table give the average congestion number (in percentage) of the worst 20% global routing tiles in the design. A value above 100 implies that the design is unroutable. In practice, it is desirable to have a low value for this metric to make the design more routable.

Table 5.5 – Table 5.7 show that the superior placement results of CCATP are maintained till the end of physical synthesis. This further indicates that CCATP is able to find globally better solutions in terms of HPWL and design timing. In particular, from column four of Table 5.5, the CCATP-based flow is able to obtain more than 50% reduction in the total HPWL at the end of physical synthesis as compared to the TP-based flow. In addition, columns four and seven of Table 5.6 show a 40% improvement in the worst slack and about 6% improvement in the timing Figure of Merit of the design. Finally, from Table 5.7, in terms of routability, the CCATP-based flow yields more than 30% improvement in the global routing congestion analysis results versus the TP-based flow. In fact, on four out of the seven designs, the TP-based flow

Table 5.5 Comparison of the HPWL ($\times e6$) between timing-driven placement (TP) and clock constraint aware timing-driven placement (CCATP) based flows at the end of physical synthesis.

Ckt	TP	CCATP	$\frac{CCATP}{TP}$
ckt_1	33.72	13.95	0.41
ckt_2	28.41	19.85	0.70
ckt_3	67.53	36.09	0.53
ckt_4	109.44	43.69	0.40
ckt_5	37.70	20.48	0.54
ckt_6	172.51	62.51	0.36
ckt_7	54.34	22.84	0.42
Avg			0.48

Table 5.6 Comparison of the design timing (worst slack and timing figure of merit) between timing-driven placement (TP) and clock constraint aware timing-driven placement (CCATP) based flows at the end of physical synthesis.

Ckt	Worst Slack (ns)			Figure of Merit (ns)		
	TP	CCATP	%Improv	TP	CCATP	%Improv
ckt_1	-0.026	-0.025	3.85	-2	-2	0.00
ckt_2	-0.027	-0.025	7.41	-7	-5	28.57
ckt_3	-0.048	-0.026	45.83	-10	-4	60.00
ckt_4	-0.023	-0.023	0.00	-1	-2	-100.00
ckt_5	-0.018	0.014	177.78	0	0	0.00
ckt_6	-0.060	-0.051	15.00	-21	-10	52.38
ckt_7	-0.021	-0.014	33.33	0	0	0.00
Avg			40.46			5.85

Table 5.7 Global routing congestion analysis results of the timing-driven placement (TP) and clock constraint aware timing-driven placement (CCATP) based flows at the end of physical synthesis.

Ckt	TP	CCATP	%Improv
ckt_1	102.015	67.349	33.98
ckt_2	82.412	73.404	10.93
ckt_3	113.148	80.822	28.57
ckt_4	137.199	76.354	44.35
ckt_5	77.274	62.783	18.75
ckt_6	162.184	83.301	48.64
ckt_7	98.964	70.014	29.25
Avg			30.64

generates unroutable results. This effectively invalidates the physical synthesis solution from the TP-based flow on these designs. The improved routability in the CCATP-based flow can be attributed to the following two reasons:

- Lower total HPWL which typically translates to lesser usage of routing resources.
- Better latch placement within the individual latch clusters (greedy latch swapping within the clusters alleviates the routing congestion around the LCBs by placing the latches in better locations with respect to their connections to the non-clocked modules).

5.6 Key Findings and Observations

Latch clustering and tight latch cluster placement are essential in reducing the local clock network power in high performance designs. Once the latch clustering is performed, it is the job of the global placement algorithm to place the latches in a tight huddle around their associated LCBs. Simultaneously, the placement algorithm should not degrade the total wire length and overall timing of the design. Currently the top-performing academic placers handle the wire length minimization problem, and do not pay any attention to clocking structures like latch clusters during placement. This chapter shows that using the traditional approach of imposing constant, high net-weights between an LCB and its associated latches to obtain tight latch

clusters, adversely impacts the total wire length, timing and routability of the design.

Alternately, several techniques have been developed and incorporated within force-directed global placement, to yield a Clock Constraint Aware Timing-driven Placement algorithm. The CCATP algorithm simultaneously optimizes timing (weighted wire length) while satisfying tight LCB to latch distance constraints. Experimental results on high-performance industrial designs show that the CCATP algorithm can obtain high-quality latch cluster placement without degrading the total wire length, design timing and routability.

CHAPTER 6. INTEGRATED TIMING OPTIMIZATION AND PLACEMENT

6.1 Introduction

Timing closure happens to be one of the primary objectives of a physical synthesis tool. In this respect, timing-driven placement is a critical step in any physical synthesis flow. The quality of timing-driven placement significantly impacts the ability of the physical synthesis tool or designer to achieve timing closure.

Existing timing-driven placement techniques can be broadly classified into two categories: (a) global timing-driving placement techniques [21, 24, 26, 29, 33, 37, 51, 54, 56, 74], and (b) incremental timing-driven placement techniques [14, 16, 17, 40, 41, 73].

6.1.1 Global Timing-driven Placement Techniques

Global timing-driven placement techniques place the entire circuit netlist and redo the placement from the beginning, ignoring the module locations that were obtained from any of the previous stages within a physical synthesis flow. They typically use a net-based approach, where they try to minimize the wire length of the “nets” on the critical paths (also referred to as “critical nets”). The rationale being that optimizing the wire length of the critical nets would implicitly minimize critical path lengths, leading to better critical path delay for the design. To guide the global placement algorithm, these techniques transform the timing constraints or specifications into net specifications, that appear as either net-weights [21, 33, 37, 54, 56, 74], or net-length constraints [24, 29, 51].

To generate reasonably accurate net specifications (net-weights or net-length constraints), these techniques usually follow the physical synthesis flow shown in Figure 6.1. Initially, a

pure wire length driven global placement is performed to obtain some physical information for all the modules in the design. Since the initial placement is timing unaware, this stage is followed by a coarse timing optimization stage, which brings the design to a reasonable electrical and timing state. Based on a timing analysis after the coarse timing optimization stage, net specifications are generated to reflect the timing criticality of the nets in the design. These specifications are used to guide the subsequent timing-driven placement stage, which optimizes the critical nets to improve design timing.

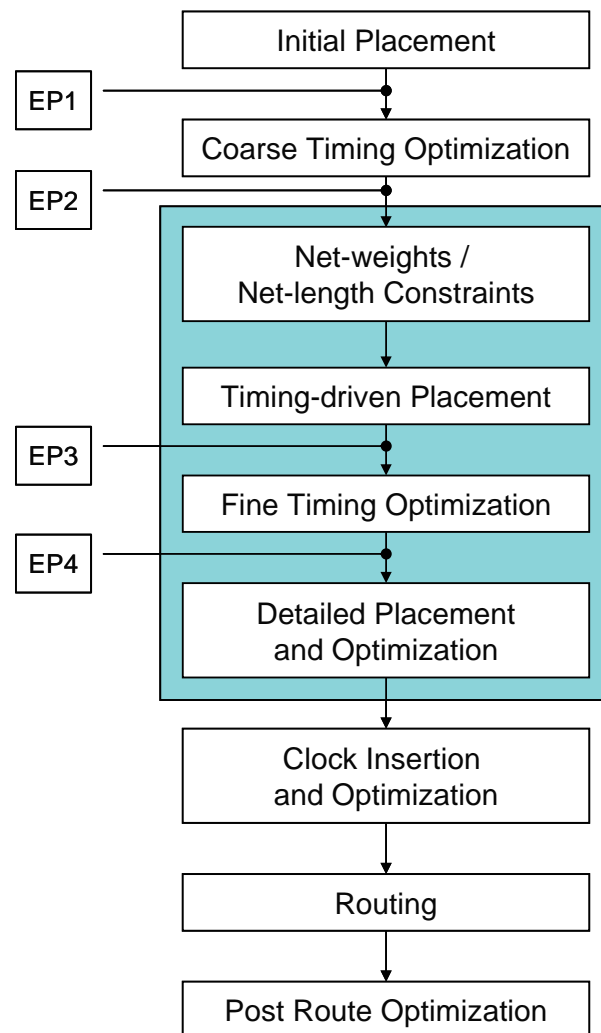


Figure 6.1 A physical synthesis flow using net-based timing-driven placement. To reflect the timing criticality of the nets, net specifications can be in the form of net-weights or net-length constraints.

The key drawbacks of global timing-driven placement techniques are as follows:

- Global timing-driven placement techniques can result in significant degradation in the total wire length of the design. The reason being, there is no effective method to come up with a good set of net specifications to effectively trade-off total wire length and design timing. Since timing-driven placement minimizes the wire length of the critical nets at the expense of the non-critical nets, inferior net specifications can overly optimize the critical nets leading to a substantial increase in the total wire length. Increase in wire length can also cause severe routing congestion. This is depicted in Figure 6.2 which shows the total wire length and routing congestion at various stages of the physical synthesis flow given in Figure 6.1 on a high performance industrial design. The regions colored pink and purple have more than 100% global routing resource usage - indicating unroutable regions. It can be seen that the global timing-driven placement stage significantly increased total wire length and routing congestion (Figure 6.2 (EP 3)) as compared to the coarse timing optimization stage (Figure 6.2 (EP 2)).
- Global timing-driven placement techniques do not have any interaction with timing optimization transforms like buffer insertion, gate sizing, etc., for the entire duration of placement. Purely minimizing the critical net wire length without performing any timing optimization in between, will in most cases, degrade the design timing after placement. This is shown in Table 6.1 which gives the design timing at various stages of the physical synthesis flow shown in Figure 6.1 on two of the benchmark designs considered in Section 6.9. As seen from Table 6.1, although the design timing at the end of fine timing optimization (EP 4), is better than what is obtained after coarse timing optimization (EP 2), there is a significant degradation at the end of timing-driven placement (EP 3). In addition, a close interaction with timing optimization can potentially re-buffer a net or re-size a gate instead of the corresponding net getting over-optimized during placement at the expense of the other nets in the design. This can potentially lead to significant savings in total wire length, a fact that is demonstrated by the experimental results in Section 6.9.

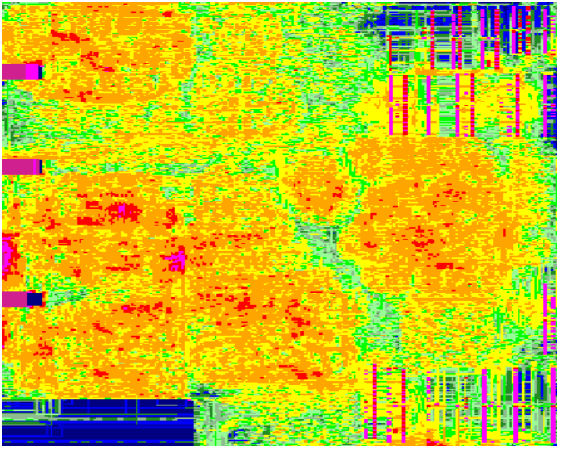
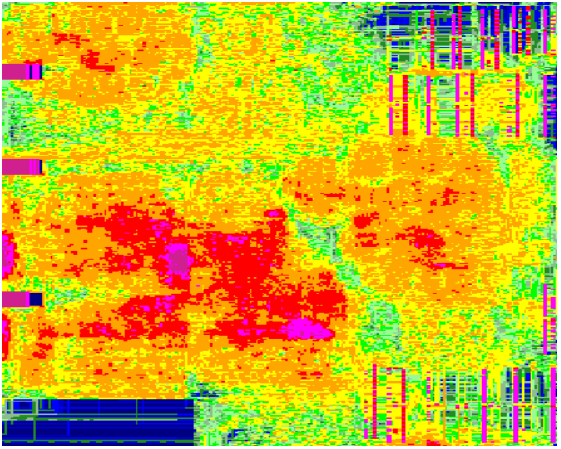
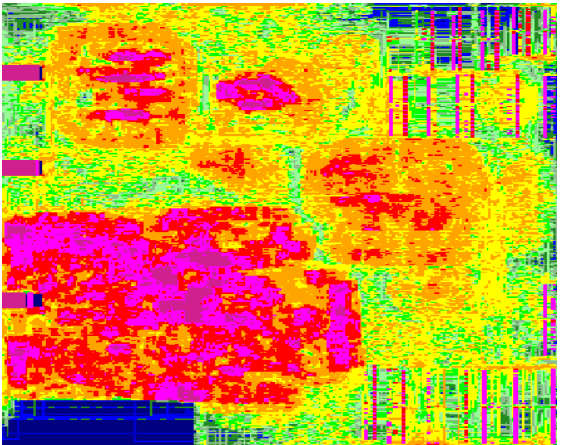
Stage	Steiner Wire Length ($\times e6$)	Routing Congestion
EP 1	140.29	
EP 2	146.94	
EP 3	160.10	

Figure 6.2 Total wire length and routing congestion at various stages of the physical synthesis flow given in Figure 6.1. (EP 1) After initial placement (EP 2) After coarse timing optimization (EP 3) After timing-driven placement. (Regions colored pink and purple have more than 100% global routing resource usage).

Table 6.1 Design timing at various stages of the physical synthesis flow shown in Figure 6.1. Although the design timing at the end of fine timing optimization (EP 4), is better than what is obtained after coarse timing optimization (EP 2), there is a significant degradation at the end of timing-driven placement (EP 3).

Design	Evaluation Point	Worst Slack (ns)	Number of Negative Paths
ckt_5	EP 1	-246.99	21377
	EP 2	-0.71	10203
	EP 3	-4.03	12110
	EP 4	-0.26	2578
ckt_8	EP 1	-976.79	73524
	EP 2	-1.14	6610
	EP 3	-24.40	24464
	EP 4	-0.38	1167

- In addition, net specifications are just an indirect measure of the actual timing constraints. It is extremely difficult to come up with a good set of net specifications that can effectively optimize design timing within placement.
- For most of the global placement techniques, net specifications are generated at the beginning of timing-driven placement, and are kept constant during the placement stage. This introduces an additional level of inaccuracy. Placement changes can invalidate the timing upon which the net specifications were initially generated. For example, nets that were critical in the beginning could become non-critical very early on during placement. Over-optimizing these nets could cause other non-critical nets to become critical.
- Dynamic updation of net specifications during global placement have been proposed (e.g., [21, 24, 56]). But in most cases, to maintain placement efficiency, the net specifications are updated using inaccurate timing models. Even if net specifications are updated using accurate timing from a static timing engine, they are based on illegal locations of the modules, as there will be significant module overlap during global placement. In addition, dynamic updation of net specifications during global placement can potentially cause oscillations during placement, resulting in issues with timing convergence.

6.1.2 Incremental Timing-driven Placement Techniques

Incremental timing-driven placement techniques place a subset of the circuit netlist, retaining the locations of a majority of the modules as obtained from the previous stage within a physical synthesis flow. They typically use a path-based approach, wherein they model the various physical properties like gate delay, interconnect delay etc., during placement, and try to directly optimize the timing critical paths in the design. Although many flavors exist, a majority of these techniques use the approach of linear programming [31], to perform timing-driven incremental placement.

The key drawbacks of incremental timing-driven placement techniques are:

- To perform module movement, these techniques often rely on inaccurate or crude models for the various physical properties like gate delay, interconnect delay etc. Typically, they use a linear model for the interconnect delay, which breaks down when the modules need to move by a large distance. As a result, they constrain the movement of the modules to a local region (e.g., [16, 17, 40]).
- Since incremental placement techniques rely on computationally intensive mathematical programming techniques, they are limited in their scope in terms of the number of paths that can be considered during placement. Hence, they cannot be used during the early stages of physical synthesis where a large number of paths need to be simultaneously optimized.
- As with global placement techniques, they typically do not interact with timing optimization transforms during placement. As an exception, certain techniques for considering optimization during placement have been proposed (e.g., [14]), but again, they rely on inaccurate and simple delay models that do not reflect the complexities of nanometer-scale integrated circuit design.
- Incremental timing-driven placement techniques typically ignore module overlap constraints during the solution of the mathematical program. To resolve the overlaps among the modules, these techniques rely on a subsequent legalization step which is often timing

unaware. This can lead to a degradation in design timing as there is no guarantee for the legalization step to preserve the design timing as seen at the end of the critical path optimization step.

- The legalization issue is particularly magnified in modern designs that contain numerous fixed macros which appear as placement blockages. This results in a highly fragmented placement region in which the modules need to be placed. Incremental placement techniques do not explicitly model and account for placement blockages during critical path optimization.
- Finally, they do not address other placement issues like placement density constraints, which are required to provide space for timing optimization and routing. As with global placement techniques, excessive packing of modules within a local region during incremental placement can cause routing congestion issues.

6.2 Key Contributions of This Work

This work is motivated by the following observations: (a) a robust and high-quality timing closure flow requires a close coupling between placement and timing optimization (e.g., buffer insertion and gate sizing), and (b) both steps should rely on accurate timing information from a state-of-the-art timing analysis tool which can model the complexities of nanometer-scale VLSI design.

In this respect, this chapter describes an Integrated Timing Optimization and Placement (ITOP) algorithm to achieve timing closure on nanometer-scale VLSI designs. In contrast to existing approaches, ITOP does not rely on a net specification based timing-driven placement, or on delay modeling followed by computationally intensive mathematical programming. To achieve timing closure within a physical synthesis flow, it uses an incremental approach with tight integration between placement and timing optimization, to gradually improve design timing without degrading wire length and routability.

The key contributions of this work in the development of an Integrated Timing Optimiza-

tion and Placement algorithm are:

- A simple, yet effective netlist transformation technique to model the critical paths in the design, so that they can be effectively optimized during placement. To identify the critical paths, this technique uses accurate timing from a timing analysis engine.
- An efficient incremental placement technique that directly optimizes critical path lengths while considering placement blockages during critical path optimization. It should be noted, none of the existing path-based approaches to timing-driven placement explicitly model placement blockages during critical path optimization.
- A tight integration of placement with incremental timing optimization, which in turn uses accurate timing information from a static timing engine to perform buffer insertion and gate sizing on the design.
- An iterative flow that includes periodic congestion mitigation, wire length recovery and slack histogram compression, to improve design timing without sacrificing total wire length and routability.

Since ITOP has been primarily developed to improve design timing during the early stages of physical synthesis (in other words, have a global impact on the design timing), it is embedded within a physical synthesis framework as shown in Figure 6.3. Please note, the incremental nature of ITOP also makes it attractive to perform fine-grained critical path optimization. Hence, it can also be employed within the detailed placement and optimization stage of physical synthesis if required (Figure 6.1).

The rest of this chapter is organized as follows: Section 6.3 provides an overview of the ITOP algorithm. This is followed by Sections 6.4 – 6.7 that describe the individual components of the algorithm in detail. Section 6.8 gives the detailed algorithm for ITOP as employed within an industrial physical synthesis framework. Experimental results on industrial designs in the 65nm and 45nm process technology nodes are reported in Section 6.9. The chapter concludes with some key observations in Section 6.10.

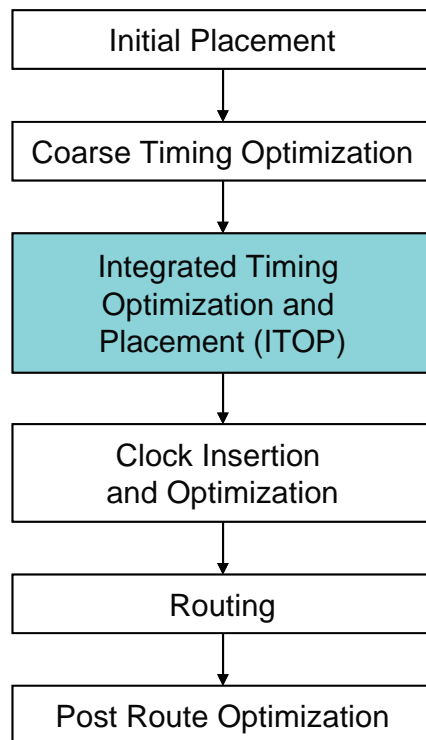


Figure 6.3 A physical synthesis flow incorporating Integrated Timing Optimization and Placement.

6.3 Overview of Integrated Timing Optimization and Placement

Figure 6.4 gives a high-level view of the ITOP algorithm. The input to ITOP is a placed and coarsely optimized design. ITOP then uses an iterative approach to improve design timing in an incremental manner. From Figure 6.4, the key steps during ITOP are:

1. **Critical Path Smoothing:** The goal of this step is to smooth (or straighten) the critical paths in the design and minimize critical path lengths. It comprises of two steps: (a) Slack-based Critical Path Threading – to identify, and model the critical paths in the design (Section 6.4.1), (b) Incremental Timing-driven Placement – to re-place the modules on the critical paths so as to straighten the critical paths and minimize critical path lengths (Section 6.4.2).
2. **Congestion Mitigation and Wire Length Recovery:** The goal of this step is to preserve the

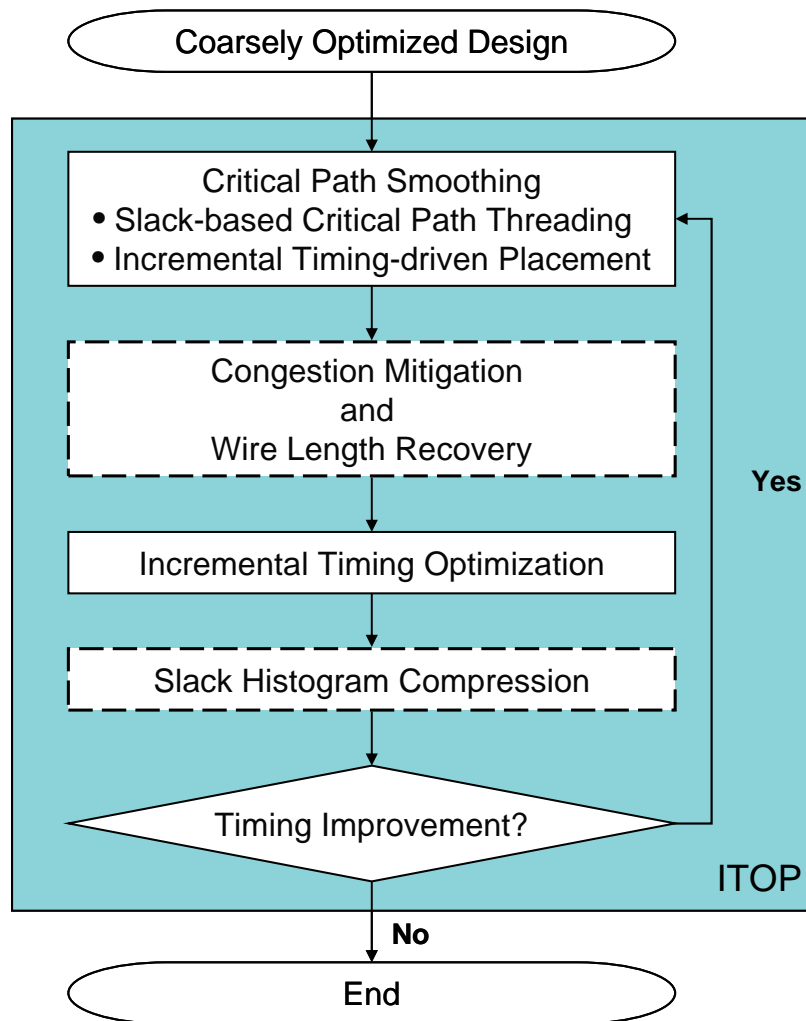


Figure 6.4 High-level flow for Integrated Timing Optimization and Placement (ITOP).

routability and wire length characteristics of the incoming placement to ITOP. This is achieved by periodically performing placement density management by moving the non-critical modules in the design to maintain the density profile of the incoming placement. In addition, detailed placement transforms like module swapping, flipping, etc., are run to recover wire length (Section 6.5). Please note, this step is not performed during every iteration of ITOP. Hence, it is represented by the dashed box in Figure 6.4.

3. Incremental Timing Optimization: This step uses accurate timing information from a static timing engine to perform quick optimization (e.g., buffering and gate sizing) to improve the timing on the critical paths in the design (Section 6.6).
4. Slack Histogram Compression: Periodically in the iterative flow, a global timing optimization is also performed to compress the entire slack histogram (Section 6.7). Please note, this step is not performed during every iteration of ITOP. Hence, it is represented by the dashed box in Figure 6.4

6.4 Critical Path Smoothing

The key objectives of the critical path smoothing step are: (a) straighten or smooth the critical paths in the design, subject to a maximum perturbation constraint on the modules forming the paths, (b) distribute the modules evenly along their respective paths. These objectives are achieved by using a two-step approach that comprises of first, modeling the critical paths in the design, and second, optimizing them during placement. The remainder of this section describes these steps in detail.

6.4.1 Slack-based Critical Path Threading

During each iteration of ITOP, this step uses the timing report from a state-of-the-art static timing engine to identify the critical paths in the design. For each iteration, the critical paths are identified as follows: To satisfy the timing requirements for the design, circuit designers set an overall *slack threshold* value (normally “zero”) for the design. Usually, all the paths that

have a slack value below the slack threshold (“negative slack”) are considered critical. Since ITOP uses an incremental approach to gradually improve design timing, only a small subset of the paths with a negative slack are considered to be critical during each iteration. These happen to be the paths with the most negative slack values among the negative slack paths in the design.

Once the critical paths have been identified, for each path, the modules on the path are linked or “threaded” together via additional two-pin nets. These nets are assigned a higher weight than the default net-weight for a non-critical net so that the subsequent placement step can effectively minimize the two-pin net lengths. Minimizing the lengths of the two-pin nets forming the path consequently minimizes the total path length. Within ITOP, these two-pin nets are termed as “path-threading attractions” and their net-weight is set to be $10\times$ the default net-weight. As an example, Figure 6.5(a) shows a sample netlist, with the bold arrows representing the two critical paths in the design. Figure 6.5(b) shows the transformed netlist, with the modules on the critical paths being threaded using additional two-pin nets that have a high net-weight (solid lines linking the modules on the critical paths). Please note, if the same set of modules appear in multiple critical paths, all the attractions between any two modules are coalesced and only a single attraction is added between them. For example, from Figure 6.5(a), the modules *A*, *B*, and *C* are shared by both the critical paths. In Figure 6.5(b), only a single attraction is added between modules *A* and *B*, and between modules *B* and *C*.

During the early stages of the flow there can be thousands of paths that do not meet their timing requirements. To obtain a tight coupling between placement and timing optimization, during each ITOP iteration, only a few hundred paths are considered for threading and placement refinement.

To set the net-weights on the two-pin nets, critical path threading does not use complex schemes that rely on path counting [37] or slack and delay sensitivity [40, 54, 74]. Instead, it uses a simple technique wherein it sets a constant weight on all the attractions which is independent of the path slacks. The main objective of critical path threading is to identify the modules that need to be re-placed during incremental placement. In addition, the modules

are not moved by a large distance during placement. Therefore, this relatively simple net-weighting scheme coupled with incremental placement is adequate to significantly improve the overall design timing. This is validated by the experimental results in Section 6.9.

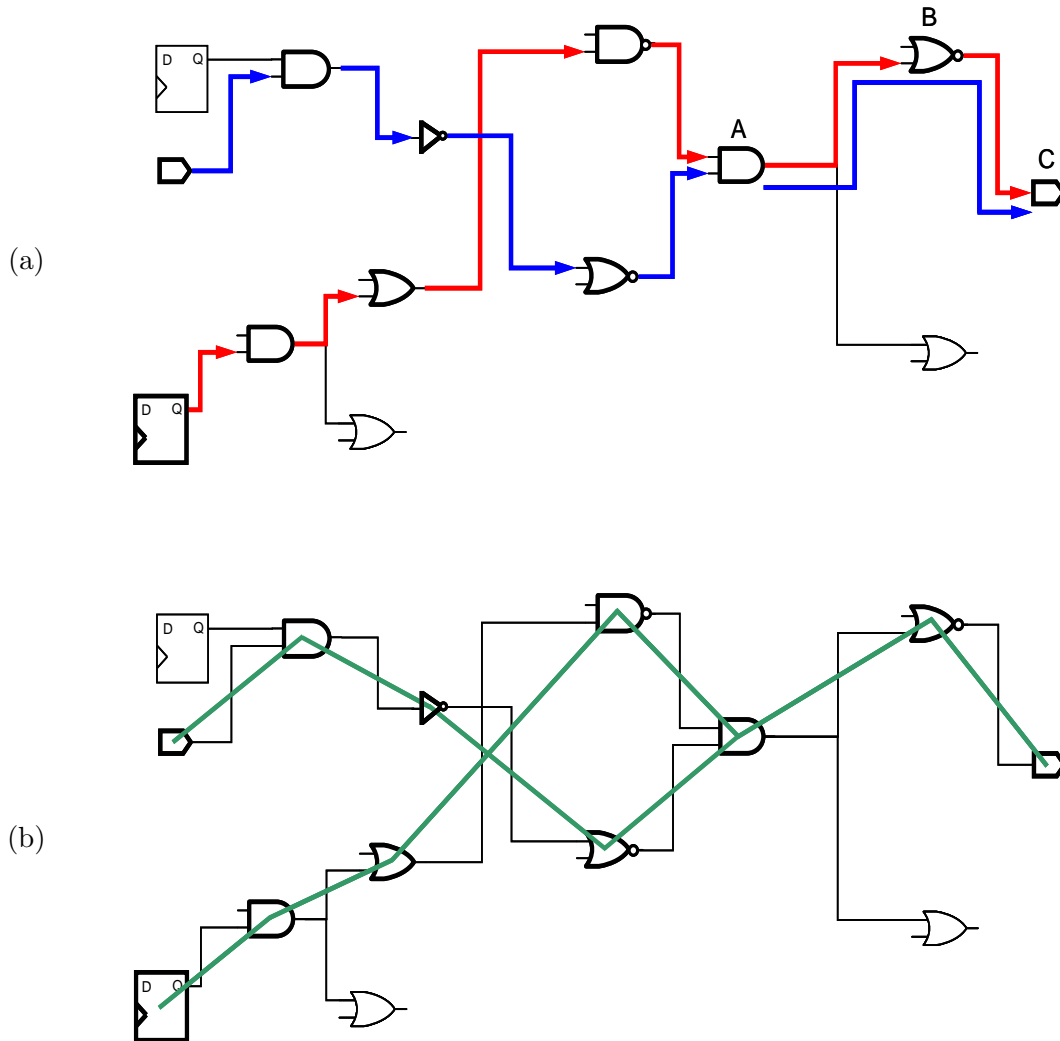


Figure 6.5 Slack-based Critical Path Threading. (a) An example netlist. The bold arrows represent the critical paths in the design (b) Critical path threading via additional two-pin nets with a high net-weight.

6.4.2 Incremental Timing-driven Placement

The goal of the incremental timing-driven placement step is to optimize the critical paths in the design by re-placing the movable modules constituting the paths. Previous approaches typically use either quadratic programming [21, 54, 56] or linear programming [16, 17, 31, 40, 49, 73] to optimize the critical paths during placement. In contrast, ITOP uses a greedy heuristic that relies on iterative movement of the modules in a local neighborhood to perform critical path optimization. In principle, this approach is similar to the Iterative Local Refinement technique described in Section 2.6. The key differences between the two techniques are: (a) the wire length objective that is used for score calculation, (b) the technique employed to handle placement blockages during critical path optimization, (c) handling of placement density during critical path optimization. The key advantages of the incremental timing-driven placement algorithm employed within ITOP are:

- It is efficient in nature and has the ability to simultaneously optimize a large number of paths.
- It explicitly models and accounts for placement blockages during critical path optimization.

To facilitate the subsequent discussion, we define:

- **Critical module:** A critical module is one that belongs to one or more critical paths that are currently being refined. It includes the latches/flip-flops in the design, which typically are the end-points of the path(s) being optimized.
- **Path length:** For a given path, the total net-length of the two-pin path-threading attractions forming the path.

To perform module movement, initially, a regular bin grid is constructed over the placement region and the current or *source* bin for all the critical modules is determined. For each critical module, eight *movement scores* are computed that correspond to tentatively moving the module to its eight neighboring bins. To calculate the score, it is assumed that a module is moving from

its current location in a source bin to the same relative location in the neighboring bin. Since the primary objectives during placement are path length minimization via path smoothing, the score for each move is solely based on the reduction in the weighted *quadratic* wire length of all the nets connected to the module. If all eight scores are negative, the location of the module remains unchanged. Otherwise, it is moved to the neighboring bin with the highest positive score. During a single iteration of refinement, the above steps are performed on all the critical modules. It is then repeated until there is no improvement in the total path length over all the critical paths being optimized. Please note:

- During this stage of placement, the locations of the non-critical modules remain unchanged. It is only the critical modules in the design that are moved.
- The stopping criterion during placement refinement is not based on the improvement in the total wire length of the design. Rather, it is based on the improvement in the total path length over all the critical paths being currently optimized. In fact, this stopping criterion marginally increases the total wire length as a subset of the paths in the design (consequently nets) are being optimized at the expense of others.

To ensure a tight coupling between placement and optimization, the critical modules are not moved by a large distance during placement. This is ensured by setting a maximum displacement constraint on the critical modules during each ITOP iteration. The maximum displacement constraint is relaxed only when the critical modules need to cross over any of the placement blockages in the image. This is explained in more detail in Section 6.4.3.

Placement refinement is followed by legalization, where the overlaps among the modules are resolved and they are assigned to legal locations within the placement region. To ensure that the critical modules are not perturbed by a large distance from their locations after refinement, legalization follows a two-step approach:

1. In the first step, all the non-critical modules are ignored and the critical modules are legalized in the presence of the placement blockages in the design. The critical modules are then fixed and transformed into placement blockages.

2. In the next step, the non-critical modules are legalized in the presence of all the placement blockages in the design.

During critical path smoothing, a high attraction net-weight coupled with weighted quadratic wire length minimization ensures that: (a) the lengths of the critical paths are effectively minimized, and (b) the critical modules are evenly distributed along their respective paths.

6.4.3 Tunneling to Handle Placement Blockages

To the best of our knowledge, none of the existing techniques on incremental timing-driven placement explicitly model and account for placement blockages (fixed macros) during critical path optimization. Previous techniques typically follow a two-step approach: (a) in the first step they ignore the fixed macros and solve the mathematical program (quadratic or linear) to optimize the critical paths, (b) in the second step they rely on a “timing-unaware” legalization to resolve the overlaps between the fixed macros and critical modules. Modern mixed-size designs contain thousands of fixed macros in the placement region, and ignoring them during critical path optimization can lead to significant overlaps between the fixed macros and the critical modules. As a result, the above two-step approach can potentially lead to severe degradation in the design timing after legalization.

To consider fixed macros during critical path optimization, ITOP uses the concept of tunneling while moving the critical modules during placement refinement. As mentioned before, to perform critical module movement, the placement refinement technique uses a bin-based approach and evaluates a *movement score* in a local neighborhood of bins. If a neighboring bin overlaps with a fixed macro, then instead of landing on top of the fixed macro, it is assumed that the critical module tunnels through the fixed macro in the general direction of the move. As a result, to evaluate the *movement score* the closest bin(s) adjacent to the macro are considered instead of the bin overlapping with the macro. During tunneling, the maximum displacement constraint on the “tunneled module” is ignored. This ensures that additional candidate locations can be considered for score evaluation.

The concept of tunneling is illustrated in Figure 6.6. In the figure, the dark shaded box

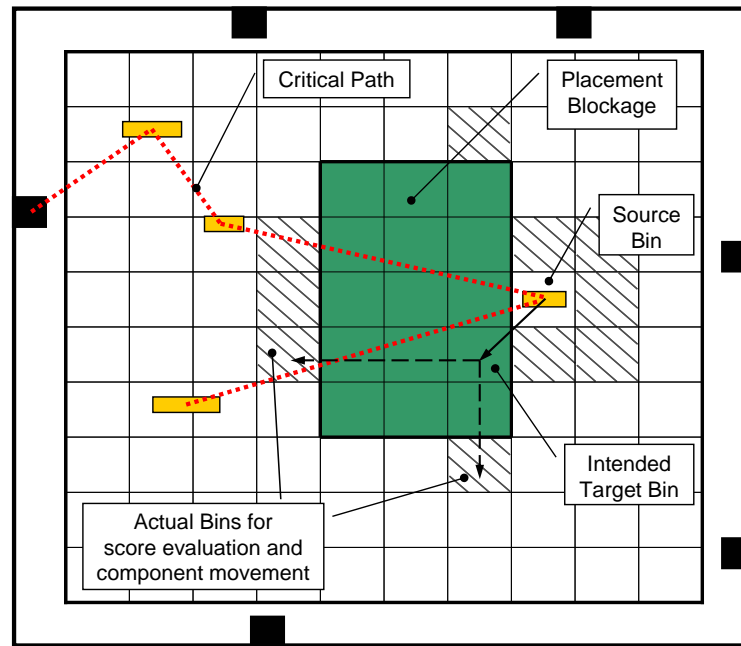


Figure 6.6 Tunneling through fixed macros during critical path smoothing.

represents a fixed macro. There are four movable modules (lightly shaded boxes) forming the critical path. One of these modules is placed on the right-hand side of the fixed macro and needs to be moved across the macro to optimize the critical path. As shown in Figure 6.6, the lower-left neighboring bin of the module overlaps with the fixed macro. To consider additional candidate locations, it is assumed that the module tunnels through the fixed macro during placement refinement. As a result, the *movement score* is evaluated in the closest bins adjacent to the left and bottom boundaries of the macro in the direction of the move. In all, this module has three neighboring bins that overlap with the fixed macro. In Figure 6.6, the bins with the hatched lines are the ones that are ultimately used for score evaluation to move this module.

Please note, the maximum number of bins that need to be considered on account of tunneling is limited to twelve¹, which is not much higher than the number of bins being considered without tunneling (eight). Also, determining adjacent bins to a fixed macro from a blockage

¹This scenario occurs when a movable module is surrounded by fixed macros on all sides.

map is quite efficient. Hence, on the whole, tunneling has negligible impact on the runtime of the algorithm.

Tunneling through fixed macros during critical path smoothing has two key advantages:

- Tunneling ensures that there are no overlaps between the critical modules and fixed macros at the end of the placement refinement stage. Since the first step of legalization considers only the critical modules, they are not perturbed by a large distance during legalization. This preserves the timing characteristics of the design as obtained after the placement refinement stage.
- In addition, it is quite possible for greedy local search techniques to get stuck in a local minima because the critical modules fall into “alleys” between the large fixed macros. Tunneling ensures that such a case does not happen. Ignoring the maximum displacement constraint for the “tunneled modules” enables the placement refinement technique to further optimize the critical paths.

Experimental results in Section 6.9 on a set of industrial designs show that tunneling is an essential component for the timing convergence of the incremental flow.

6.5 Congestion Mitigation and Wire Length Recovery

During critical path smoothing, the objective function for module movement considers only the weighted quadratic wire length and ignores placement density or placement congestion constraints. As a result, movement of the critical modules can greatly increase the placement congestion in certain parts of the placement region. The impact of increasing the placement congestion is two-fold: (a) the subsequent timing optimization step might not have enough space to perform buffer insertion or gate sizing, which in turn limits its performance and can lead to degraded quality of results, (b) a large number of pins being present in a local region can also cause severe routing congestion. Therefore, to alleviate placement congestion, a congestion mitigation step is performed after periodic intervals in the iterative flow. This interval is determined by the variable *itop_transition_iteration* in Algorithm 6.1.

The goal of the congestion mitigation step is to maintain the density profile of the incoming placement to ITOP. The intuition being that preserving the incoming placement distribution will yield a final placement that has the same degree of routability as the incoming placement. In addition, there will be a reasonable amount of free-space for the timing optimization transforms. To prevent excessive spreading of the modules and have an accurate view of the local placement distribution, congestion mitigation attempts to satisfy a “bin density target” as opposed to a global “density target” for the entire design. During congestion mitigation, the non-critical modules are locally perturbed to satisfy the bin density target constraints that are determined from the incoming placement.

To determine the bin density target, an $M \times N$ bin-grid, (B) is initially imposed over the placement region. The density of each bin in B is then calculated from the incoming placement. The density of a bin is defined as the ratio of the total movable module area to the total free-space within the bin. The density target for each bin is then a function of the bin density and the global density target for the design. If,

- G_{DT} : The global density target for the design.
- d_b : Incoming density of bin b in B .
- DT_b : Density target for bin b .
- δ : Overfill factor (a fixed, additional density allowed within bins for which $d_b \leq G_{DT}$).
- d_{max} : Maximum incoming bin density above which no overfill is allowed.

Then, the bin density target is given by the following piecewise linear function which is graphically shown in Figure 6.7.

$$DT_b = \begin{cases} (1 + \delta)G_{DT} & d_b \leq G_{DT} \\ (1 - \frac{\delta G_{DT}}{d_{max} - G_{DT}})d_b + \frac{\delta G_{DT} d_{max}}{d_{max} - G_{DT}} & G_{DT} < d_b < d_{max} \\ d_b & d_b \geq d_{max} \end{cases}$$

The intuition behind using the piecewise linear function for calculating the bin density target is as follows:

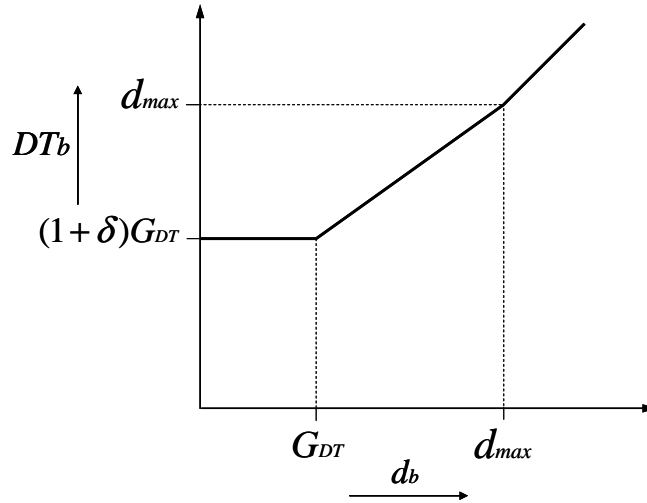


Figure 6.7 Bin density target (DT_b) during Congestion Mitigation.

- For all bins with an incoming density below a maximum density threshold (d_{max}), the bin density target is set to be higher than the incoming bin density. This allows for the bins to get marginally overfilled during ITOP. This in turn aids in the timing convergence and runtime of the flow as the non-critical modules do not have to be moved during each ITOP iteration to satisfy the bin density targets.
- Bins that have an incoming density equal and above d_{max} are highly congested to start with. Increasing the logic area in these bins might potentially impact the performance of the timing optimization step and also cause congestion and routing issues. Hence, the bin density target is set equal to the incoming bin density to prevent any overfilling of these bins during ITOP.
- In addition, bins with $d_b < d_{max}$ are further divided into two categories:
 1. Those with $d_b \leq G_{DT}$. These bins happen to be quite sparse to begin with. Hence, the amount of excess area added to these bins can be higher than the other bins in the regular bin structure. As a result, these bins are allowed to get overfilled until their final densities are a fixed value above the global density target.

2. Those with $G_{DT} < d_b < d_{max}$. If the initial density of a bin happens to be greater than the density target, then with an increase in the initial bin density, the amount of additional area added to a bin is progressively decreased, until no overfilling is allowed for bins with density greater than d_{max} .

To spread the modules from the over-congested bins, an enhanced version of the ILR technique is used. The enhancements being: (a) rather than moving all the modules in the design during each iteration of ILR, a bin-blocking mechanism is employed which prevents module movement from any bin with a density less than its bin density target, (b) the modules in an over-congested bin are sorted by their *movement scores* and are moved out of the bin in the decreasing order of their benefit for the move.

In addition to increasing the placement congestion, movement of the critical modules can also increase the total wire length of the design. The reason being, during each iteration of placement, only a small number of critical nets are being optimized at the expense of all the non-critical nets in the design. As a result, the increase in the wire length of the non-critical nets can be much larger than the decrease in the wire length of the critical nets. Hence, after each congestion mitigation step, a detailed placement step is performed to recover placement wire length. To perform detailed placement, this step uses a variety of techniques like module swapping, flipping, etc., in the spirit of the techniques described in [48].

6.6 Incremental Timing Optimization

An important component in the ITOP flow is the incremental timing optimization step that is performed during each iteration. Once placement has optimized a subset of the negative paths in the design (critical paths for the current iteration), a timing analysis followed by incremental optimization is performed to further improve the design timing. Since buffering and gate sizing are the most common and powerful optimizations during the early stages of physical synthesis, the incremental optimization step employs these transforms to improve design timing. Please note, other optimization transforms, such as multi-threshold vt tuning, wire sizing, layer assignment, etc., can also be easily embedded within the ITOP flow if required.

Since the goal of ITOP is to gradually improve design timing, only the top-most critical paths are considered for buffering and gate sizing during each iteration. To obtain high quality of results during optimization, the state-of-the-art buffering and resizing algorithms outlined in [3] are used. In addition, to avoid any inaccuracy and the resulting degradation in the quality of results, all the transforms rely on accurate timing from an industrial static timing engine to perform optimization.

6.7 Slack Histogram Compression

Periodically, after a certain number of iterations in the ITOP flow (given by the variable *itop_transition_iteration* in Algorithm 6.1), a slack histogram compression is performed on the entire design. Slack histogram compression comprises of buffering and repowering a larger set of paths in the design. This is done for the following two reasons:

- Movement of the modules on the critical paths over multiple iterations can accumulate and impact the timing on a large set of the off-critical paths. Periodically increasing the scope of the optimization can recover any degradation in the timing on the off-critical paths. This in turn can help the critical path optimization steps of ITOP to find the correct set of critical paths to be optimized.
- Since timing optimization is performed on a larger set of paths, periodic slack histogram compression also improves the efficiency of the overall algorithm.

6.8 The ITOP Algorithm

Since ITOP has been primarily developed to improve the design timing on a global scale, a large number of paths are considered for simultaneous placement optimization during the early stages of the iterative flow. The scope of timing improvement is gradually reduced by progressively decreasing the number of paths being optimized during placement. This is shown in Figure 6.8, where the change in the number of paths being optimized during placement is represented by the change in the size of the non-shaded boxes. At each transition point in the

iterative flow, congestion mitigation, wire length recovery and slack histogram compression are performed. This is shown by the shaded box in Figure 6.8. Please note, in between each transition point, the number of paths being optimized during placement is kept constant.

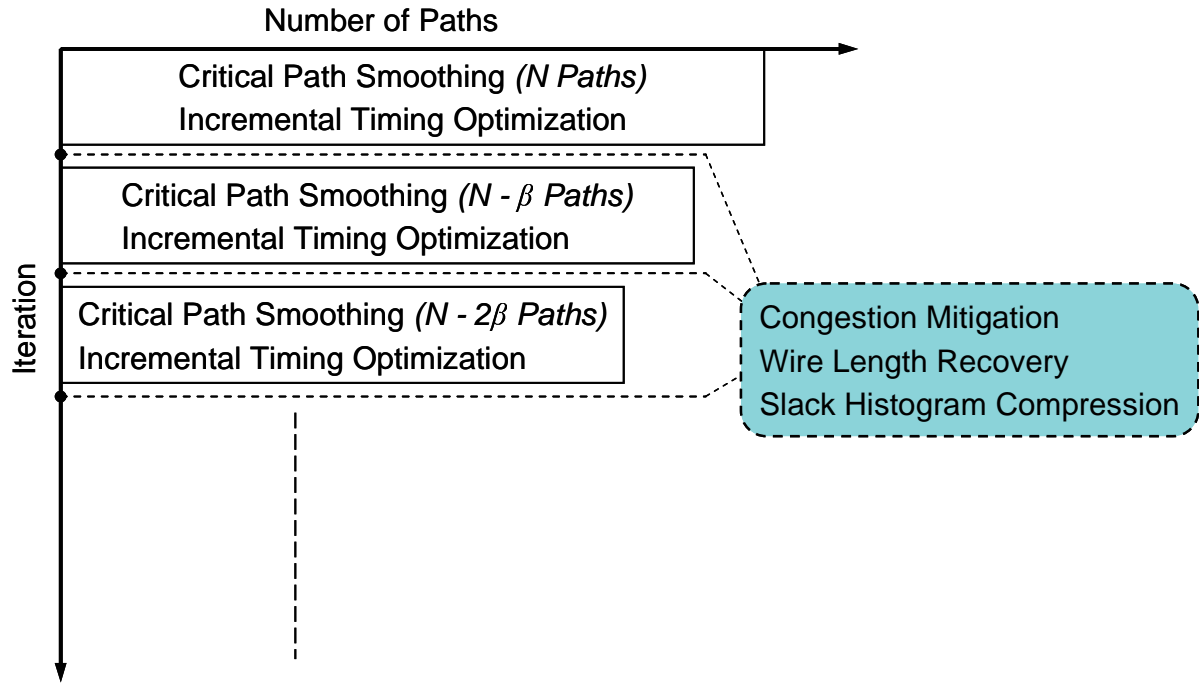


Figure 6.8 Scheduling the number of paths to be optimized during ITOP.

Finally, Algorithm 6.1 gives the overall ITOP algorithm, incorporating the techniques described in Sections 6.4 – 6.7. From Algorithm 6.1 and the discussion in the previous sections, the values for the key ITOP parameters used in practice are:

- Overfill factor (δ): 0.25 (Allow 25% overfilling of bins below the global density target).
- Maximum bin density upto which overfill is allowed (d_{max}): 0.90.
- Maximum displacement for critical modules ($D1$): 1% of the chip diagonal.
- Maximum displacement for non-critical modules ($D2$): $5 \times$ circuit row height.
- Transition iteration ($itop_transition_iteration$): 20.

Algorithm 6.1 The ITOP algorithm

```

1: Phase 0: Initial Setup
2:    $D1 \leftarrow \text{max\_displacement\_for\_critical\_modules}$ 
3:    $D2 \leftarrow \text{max\_displacement\_for\_non\_critical\_modules}$ 
4:   impose an  $M \times N$  bin-grid ( $B$ ) over the placement region
5:   determine the placement density ( $d_b$ ) for each bin  $b \in B$ 
6:   determine the density target ( $DT_b$ ) for each bin  $b \in B$ 
7: end
8:  $iteration \leftarrow 1$ 
9: repeat
10:  Phase 1: Critical Path Smoothing
11:    perform static timing analysis
12:    identify the critical paths
13:    perform slack-based critical path threading
14:    repeat
15:      move critical modules to minimize critical path lengths
16:      (subject to displacement constraint  $D1$ )
17:    until (no improvement in total path length over all critical paths)
18:    legalize and fix the critical modules
19:    legalize the non-critical modules
20:  end
21:  Phase 2: Congestion Mitigation and Wire Length Recovery
22:    if ( $iteration \% itop\_transition\_iteration == 0$ ) then
23:      move non-critical modules to satisfy  $DT_b$  for each bin  $b \in B$ 
24:      (subject to displacement constraint  $D2$ )
25:      legalize the non-critical modules
26:      perform detailed placement to recover wire length
27:      (subject to displacement constraint  $D2$ )
28:    end if
29:  end
30:  Phase 3: Incremental Timing Optimization
31:    perform static timing analysis
32:    identify the critical paths
33:    perform quick buffer insertion and gate sizing on the critical paths
34:  end
35:  Phase 4: Slack Histogram Compression
36:    if ( $iteration \% itop\_transition\_iteration == 0$ ) then
37:      perform static timing analysis
38:      perform slack histogram compression
39:    end if
40:  end
41:   $iteration \leftarrow iteration + 1$ 
42: until (no timing improvement)

```

6.9 Experimental Results

The ITOP algorithm is implemented within the PDS industrial physical synthesis framework [3, 64]. PDS is a state-of-the-art physical synthesis framework that has been used in the design of many high performance integrated circuits. To demonstrate the effectiveness of ITOP, this section presents experimental results on a set of high performance industrial designs in the 65nm and 45nm process technology nodes. The design statistics in terms of the number of modules and the number of nets are given in Table 6.2. The runtimes reported in this section are on a Intel Xeon CPU running at 2.93GHz. Finally, all the timing numbers are generated on legalized placements using an industrial static timing engine (EinsTimer).

Table 6.2 Statistics for a set of high performance industrial designs to test the ITOP algorithm.

Design	Modules	Nets
ckt_1	77K	61K
ckt_2	102K	104K
ckt_3	125K	124K
ckt_4	143K	145K
ckt_5	171K	177K
ckt_6	434K	441K
ckt_7	451K	465K
ckt_8	476K	491K
ckt_9	554K	562K
ckt_10	951K	961K
ckt_11	1034K	1056K

The results presented in this section are divided into two parts: (a) the first part demonstrates the timing impact of the key components within ITOP, (b) the second part compares the results of ITOP with two representative flows embedded within the PDS framework.

6.9.1 Effect of Placement During ITOP

Figure 6.9 and Figure 6.10 show the progression of the worst slack and Figure of Merit (FOM) respectively during ITOP. In Figure 6.10 the FOM is a weighted sum of all the negative

path slacks in the design, and is a measure of the overall slack histogram. In both the figures, the solid (red) line represents the case when both placement and timing optimization are run during each iteration of ITOP (default flow). The dotted (blue) line represents the case when only timing optimization is run during each iteration. From Figures 6.9 and 6.10, it can be seen that the timing improvement obtained during ITOP is not because of solely running multiple iterations of timing optimization on the design. To improve design timing, incremental placement is an essential component within ITOP, and is required to prevent timing optimization from getting stuck in a local minima.

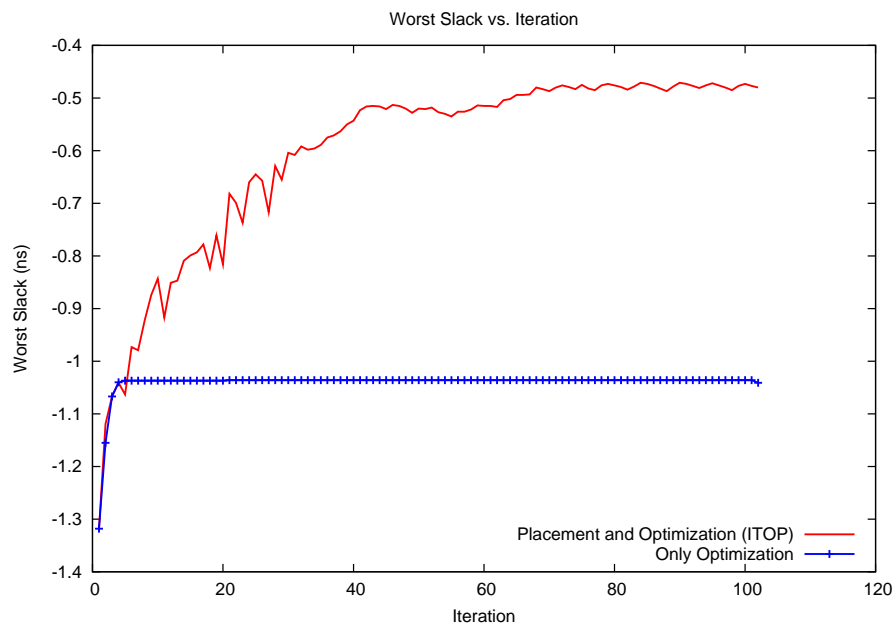


Figure 6.9 Effect of incremental placement during ITOP. Worst slack progression during the iterative flow. The red (solid) line depicts the default ITOP flow. The blue (dotted) line depicts the case when placement is skipped during each iteration.

6.9.2 Effect of Tunneling During Critical Path Smoothing

Table 6.3 shows the impact of tunneling on the design timing at the end of ITOP. The table compares two experiments: (a) No Tunneling: ITOP with no tunneling during critical path smoothing, (b) With Tunneling: ITOP with tunneling during critical path smoothing

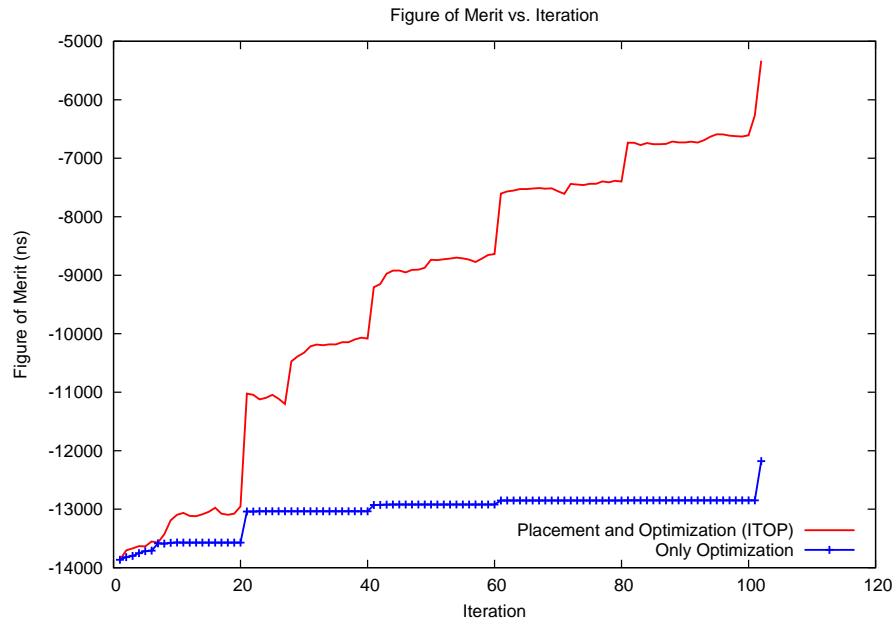


Figure 6.10 Effect of incremental placement during ITOP. Figure of Merit progression during the iterative flow. The lines correspond to the same cases as Figure 6.9.

(default flow). From Table 6.3, it can be seen that tunneling aids in significantly improving the worst slack and Figure of Merit of the design. In particular, from columns four and seven, tunneling obtains more than 35% improvement in the worst slack and up to 20% improvement in the Figure of Merit of the design respectively.

Table 6.3 Effect of tunneling during critical path smoothing. Design timing (worst slack and timing figure of merit) at the end of ITOP.

Design	Worst Slack (ns)			Figure of Merit (ns)		
	No Tunneling	With Tunneling	%Improv	No Tunneling	With Tunneling	%Improv
ckt_1	-1.78	-1.22	31.46	-1507	-1337	11.28
ckt_3	-0.48	-0.34	29.17	-963	-865	10.18
ckt_5	-0.34	-0.22	35.29	-456	-365	19.96

6.9.3 Effect of Periodic Slack Histogram Compression

Table 6.4 shows the impact of periodically running optimizations targeted toward slack histogram compression at each transition point of the ITOP flow. The table compares two experiments: (a) No Compression: ITOP flow with no periodic slack histogram compression, (b) With Compression: ITOP flow with slack histogram compression at each transition point (default flow). Since compression targets the overall slack histogram and not particularly the worst slack path in the design, Table 6.4 shows the Figure of Merit and the total number of negative paths in the design at the end of ITOP. From columns four and seven of Table 6.4, running periodic slack histogram compression obtains more than 77% improvement in the Figure of Merit and more than 54% improvement in the number of negative paths at the end of ITOP. In Table 6.4, the designs are listed in ascending order of the design size (number of modules in the design). From column four, it can be seen that the improvement in design timing due to slack histogram compression increases dramatically with the design size.

Table 6.4 Effect of periodic slack histogram compression during the iterative flow. Design timing (timing figure of merit and number of negative paths) at the end of ITOP.

Design	Figure of Merit (ns)			Number of Negative Paths		
	No Compression	With Compression	%Improv	No Compression	With Compression	%Improv
ckt_3	-912	-865	5.15	4426	3395	23.29
ckt_5	-549	-365	33.52	7125	5665	20.49
ckt_8	-164	-37	77.44	1652	751	54.54

6.9.4 Physical Synthesis Flows for Comparison of Results

To test its effectiveness, ITOP is compared with two representative flows that are shown in Figure 6.11. These are:

- NO-TDP: Physical synthesis flow that only performs a single wire length driven global placement followed by coarse timing optimization and fine timing optimization.

- TDP: Physical synthesis flow that augments the NO-TDP flow with a net-weight driven timing-driven placement step. In this flow, timing-driven placement is performed between the coarse and fine timing optimization steps. Please note, the TDP flow happens to be the representative flow for a majority of the industrial physical synthesis frameworks.

To perform global placement, all the flows use the *RQL* global placement algorithm. In addition, to perform net-weighting, the TDP flow uses the sensitivity guided net-weighting scheme [54].

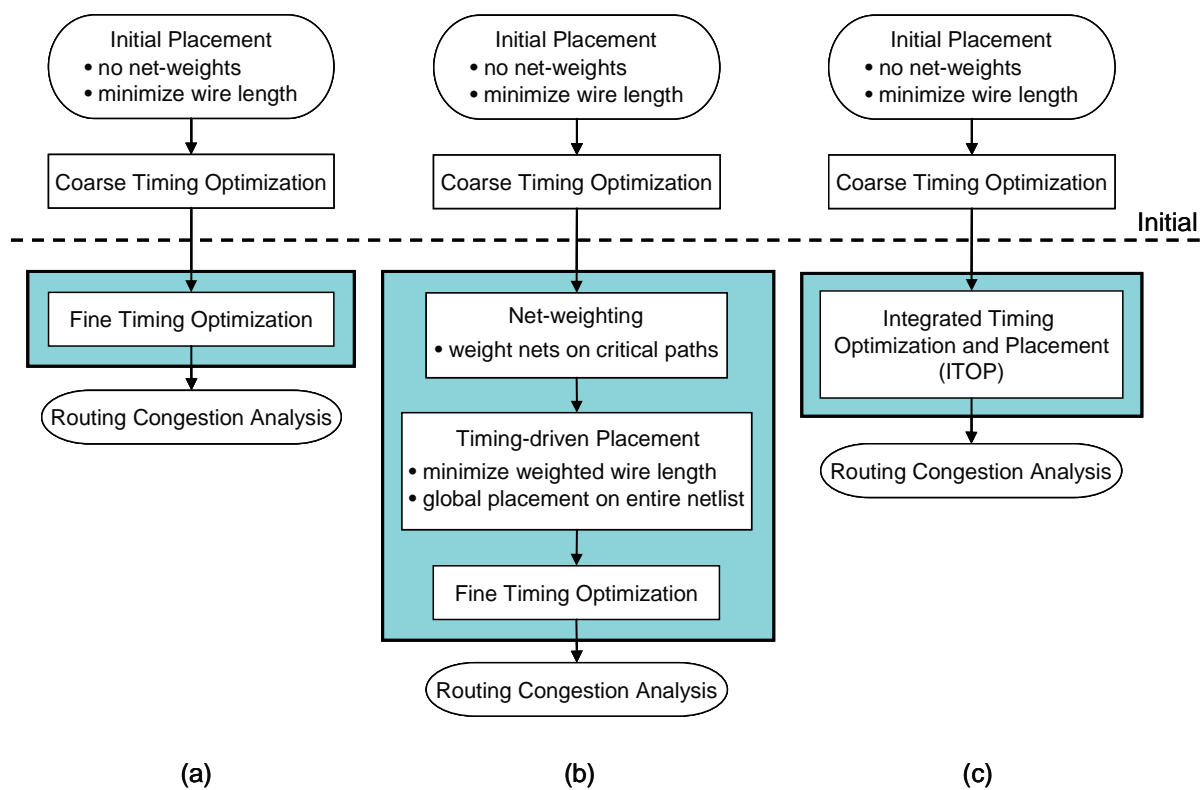


Figure 6.11 Physical synthesis flows for comparison of results. (a) NO-TDP: No timing-driven placement (b) TDP: Net-weighting and global timing-driven placement (c) ITOP: Integrated Timing Optimization and Placement. (Note: Although not depicted, detailed placement is performed within the fine timing optimization stage for the NO-TDP and TDP flows).

Before presenting the comparison results, the following observations are made regarding the NO-TDP and TDP flows. These observations are also validated by the experimental results presented later.

- Since the NO-TDP flow performs only a single wire length driven global placement (without any net-weighting), it should have better wire length and potentially better routing congestion as compared to the TDP flow.
- On the other hand, the TDP flow should have better overall design timing as compared to the NO-TDP flow since it includes an additional net-weight driven timing-driven placement step.

In this respect, the goal of ITOP is to obtain wire length and routing congestion numbers that are at least comparable to the NO-TDP flow and design timing that is at least comparable to the TDP flow.

6.9.5 Results on High Performance Industrial Designs

This section presents the comparison results between the three flows shown in Figure 6.11. The metrics for comparison are: (a) the design timing in terms of the worst slack, Figure of Merit and the number of negative paths, (b) the Steiner wire length, which is a more accurate estimation of the final routed wire length as compared to the half-perimeter wire length, (c) the global routing congestion, and (d) the overall runtime of the flows. In all the tables presented in this section, the values given along the *Initial* column are the numbers at the end of the coarse timing optimization stage in Figure 6.11.

6.9.5.1 Design Timing

Table 6.5, Table 6.6 and Table 6.7 compares the worst slack, Figure of Merit and number of negative paths at the end of the three flows. In the three tables, for each design, the improvement obtained by the respective flows is evaluated with respect to the “initial” timing obtained at the end of the coarse timing optimization stage. In addition, for each design, the numbers in bold represent the best timing result among the three flows.

From column eight of Tables 6.5 – 6.7, it can be seen that, on average, **ITOP obtains the best timing results among the three flows**. In particular, in terms of the worst slack, from Table 6.5, ITOP obtains an average improvement of 45.62% and 35.14% above the NO-TDP and TDP flows respectively. In terms of the Figure of Merit, from Table 6.6, ITOP obtains an average improvement of 37.75% and 12.2% above the NO-TDP and TDP flows respectively. From Table 6.7, on average, ITOP reduces the number of negative paths by 35.45% and 15.59% over the NO-TDP and TDP flows respectively. Looking at individual results, from Table 6.5, on nine out of the eleven designs, ITOP obtains the best result in terms of the worst slack of the design. In fact, ITOP closes timing (obtains positive worst-slack) on three of the designs. From Table 6.6, ITOP obtains the best timing Figure of Merit on seven out of the eleven designs.

Table 6.5 Worst slack comparison between the No timing-driven placement (NO-TDP), Net-weighted timing-driven placement (TDP) and Integrated Timing Optimization and Placement (ITOP) flows.

Design	Worst Slack (ns)						
	Initial	NO-TDP	%Improv	TDP	%Improv	ITOP	%Improv
ckt_1	-1.89	-1.82	3.54	-3.88	-105.39	-1.22	35.48
ckt_2	-0.24	-0.12	50.00	-0.20	19.26	0.03	111.89
ckt_3	-1.13	-1.01	10.83	-1.00	11.54	-0.34	69.57
ckt_4	-0.22	-0.15	28.37	-0.01	96.74	0.04	116.28
ckt_5	-0.71	-0.59	17.21	-0.27	62.20	-0.22	69.53
ckt_6	-1.32	-1.05	20.18	-0.69	47.95	-0.48	63.58
ckt_7	-1.33	-1.14	14.58	-0.63	52.52	-1.22	8.64
ckt_8	-1.14	-1.03	9.80	-0.66	42.52	-0.15	87.14
ckt_9	-0.43	-0.22	47.54	-0.21	51.29	0.07	116.86
ckt_10	-1.55	-1.24	19.90	-1.16	25.11	-0.85	45.59
ckt_11	-3.54	-3.51	0.96	-2.33	34.40	-3.54	0.11
Average			20.26		30.74		65.88

6.9.5.2 Design Wire Length

Table 6.8 compares the total Steiner wire length of the designs at the end of the three flows. From Table 6.8 it can be seen that on average, both the NO-TDP and ITOP flows

Table 6.6 Timing Figure of Merit comparison between the No timing-driven placement (NO-TDP), Net-weighted timing-driven placement (TDP) and Integrated Timing Optimization and Placement (ITOP) flows.

Design	Figure of Merit (ns)						
	Initial	NO-TDP	%Improv	TDP	%Improv	ITOP	%Improv
ckt_1	-1699	-1535	9.65	-1519	10.62	-1338	21.28
ckt_2	-103	-22	79.02	-29	71.50	-2	98.06
ckt_3	-1371	-1378	-0.48	-1444	-5.36	-865	36.88
ckt_4	-67	-36	45.55	-7	89.35	0	99.49
ckt_5	-1537	-1052	31.57	-200	86.99	-365	76.24
ckt_6	-13865	-11210	19.15	-5391	61.12	-5331	61.55
ckt_7	-1860	-1276	31.37	-395	78.75	-441	76.28
ckt_8	-1595	-746	53.21	-87	94.58	-37	97.68
ckt_9	-52	-49	6.05	-38	26.33	-1	97.60
ckt_10	-100277	-74968	25.24	-48093	52.04	-54479	45.67
ckt_11	-11332	-10082	11.03	-8337	26.43	-9535	15.86
Average			28.30		53.85		66.05

Table 6.7 Number of negative paths at the end of the No timing-driven placement (NO-TDP), Net-weighted timing-driven placement (TDP) and Integrated Timing Optimization and Placement (ITOP) flows.

Design	Number of Negative Paths						
	Initial	NO-TDP	%Improv	TDP	%Improv	ITOP	%Improv
ckt_1	2554	2002	21.61	2218	13.16	1928	24.51
ckt_2	1159	274	76.36	443	61.78	128	88.96
ckt_3	3525	3490	0.99	3306	6.21	3395	3.69
ckt_4	421	336	20.19	208	50.59	59	85.99
ckt_5	10180	8688	14.66	3061	69.93	5665	44.35
ckt_6	39853	37710	5.38	28705	27.97	33662	15.53
ckt_7	7090	5287	25.43	3033	57.22	3328	53.06
ckt_8	6612	3695	44.12	1232	81.37	751	88.64
ckt_9	371	757	-104.04	663	-78.71	133	64.15
ckt_10	148648	148426	0.15	121329	18.38	147215	0.96
ckt_11	30106	33810	-12.30	29178	3.08	26299	12.65
Average			8.41		28.27		43.86

improve the total wire length as compared to the initial wire length obtained at the end of the coarse timing optimization step. On the other hand, the TDP flow increases the total wire length by about 3% on average. In fact, for some cases (e.g., ckt_1 and ckt_7) the increase in the total wire length is in excess of 5%. From Table 6.8, it can be seen, that **ITOP obtains an average wire length improvement of 5% as compared to the TDP flow, and is comparable in wire length to the NO-TDP flow.**

Table 6.8 Total wire length comparison between the No timing-driven placement (NO-TDP), Net-weighted timing-driven placement (TDP) and Integrated Timing Optimization and Placement (ITOP) flows.

Design	Steiner Wire Length ($\times e6$)						
	Initial	NO-TDP	NO-TDP/Init	TDP	TDP/Init	ITOP	ITOP/Init
ckt_1	94.63	94.12	0.99	101.75	1.08	95.01	1.00
ckt_2	16.59	15.63	0.94	17.48	1.05	15.86	0.96
ckt_3	26.87	26.18	0.97	26.19	0.97	26.31	0.98
ckt_4	28.21	27.03	0.96	27.94	0.99	27.45	0.97
ckt_5	43.84	43.39	0.99	45.71	1.04	44.23	1.01
ckt_6	176.47	174.11	0.99	183.97	1.04	178.71	1.01
ckt_7	146.94	142.29	0.97	160.38	1.09	143.55	0.98
ckt_8	128.55	124.02	0.96	129.44	1.01	126.66	0.99
ckt_9	141.59	134.26	0.95	139.43	0.98	134.34	0.95
ckt_10	342.25	327.58	0.96	347.50	1.02	332.68	0.97
ckt_11	284.16	275.88	0.97	288.80	1.02	276.02	0.97
Average			0.97		1.03		0.98

6.9.5.3 Global Routing Congestion

Table 6.9 gives the routing congestion analysis results that are obtained by invoking an industrial strength global router on the final placement solutions of the three flows. The metric that is used to measure the congestion after global routing is the number of nets above 100% congestion. This is defined as the number of nets that are assigned to a global routing edge which is using 100% or more of its routing resources. In practice, it is desirable to have a low value for this metric to make the design more routable.

From Table 6.9 it can be seen that **the global routing congestion for the ITOP flow is comparable to that of the NO-TDP flow**. In fact, for most designs, ITOP preserves the routability obtained at the end of the coarse optimization stage. This demonstrates the effectiveness of the incremental path smoothing and congestion mitigation techniques in preserving the original routability of the design.

Table 6.9 Global routing congestion analysis on the final placements obtained from the No timing-driven placement (NO-TDP), Net-weighted timing-driven placement (TDP) and Integrated Timing Optimization and Placement (ITOP) flows.

Design	Global Routing Congestion (#Nets \geq 100%)						
	Initial	NO-TDP	Inc.	TDP	Inc.	ITOP	Inc.
ckt_1	76	75	-1	75	-1	119	43
ckt_2	1915	1216	-699	3302	1387	1537	-378
ckt_3	241	259	18	301	60	583	342
ckt_4	20	57	37	298	278	153	133
ckt_5	1024	953	-71	1537	513	974	-50
ckt_6	864	864	0	884	20	864	0
ckt_7	8708	9319	611	56051	47343	9066	358
ckt_8	971	1211	240	2034	1063	1333	362
ckt_9	402	347	-55	407	5	638	236
ckt_10	1301	6098	4797	161229	159928	7436	6135
ckt_11	6850	5204	-1646	14752	7902	4982	-1868
Average			294		19863		483

Figure 6.12 graphically shows the global routing congestion for the three flows on the design ckt_7. As seen from Figure 6.12, the TDP flow produces significant routing congestion, whereas the NO-TDP and ITOP flows have much lower congestion. The high routing congestion for the TDP flow can be attributed to the following reasons: (a) net-weighted global timing-driven placement can significantly increase the total wire length of the design (in this case 9%, from Table 6.8), which in turn increases the global routing resource demand, (b) net-weighting can pack quite a few modules in a local region, leading to routing hot-spots. On the other hand, the superior routing congestion for the ITOP flow can be attributed to the following reasons: (a) ITOP does not perform a global timing-driven placement and relies on an incremental

flow to achieve timing closure, (b) during the incremental flow it performs periodic congestion mitigation and wire length recovery leading to improved total wire length and routability of the final solution.

6.9.5.4 Runtime

Finally, Table 6.10 gives the runtimes of the three flows for all the designs. Since the NO-TDP flow performs only fine timing optimization, its runtime is going to be significantly lower than the other two flows across all the designs. Comparing the TDP and ITOP flows, it can be seen that **ITOP scales well with circuit size**. This can be seen from columns three and four of Table 6.10 where the runtime for ITOP is lower than that of TDP for the five largest designs in the benchmark suite.

Table 6.10 Runtime comparison between the No timing-driven placement (NO-TDP), Net-weighted timing-driven placement (TDP) and Integrated Timing Optimization and Placement (ITOP) flows.

Design	Runtime (sec)		
	NO-TDP	TDP	ITOP
ckt_1	898	2118	2905
ckt_2	561	1194	827
ckt_3	581	887	1805
ckt_4	652	1339	2244
ckt_5	1062	1966	2576
ckt_6	3303	6304	6624
ckt_7	3284	8654	7397
ckt_8	2915	7375	7210
ckt_9	2825	9021	8059
ckt_10	8966	29638	23099
ckt_11	6278	16517	15427

6.10 Key Findings and Observations

Timing-driven placement is a critical step in the physical synthesis of nanometer-scale VLSI designs. To improve the design timing on a global scale, net-weighting followed by a global

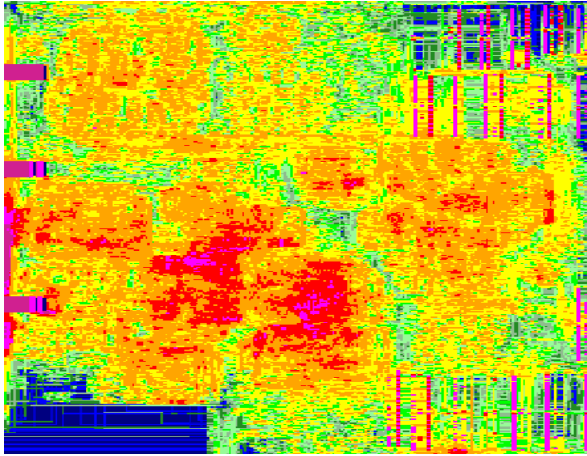
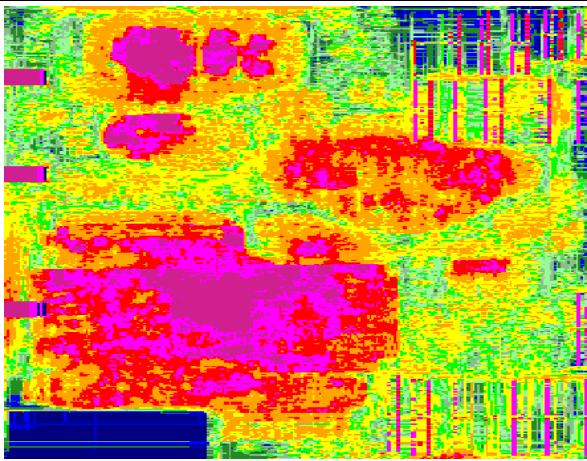
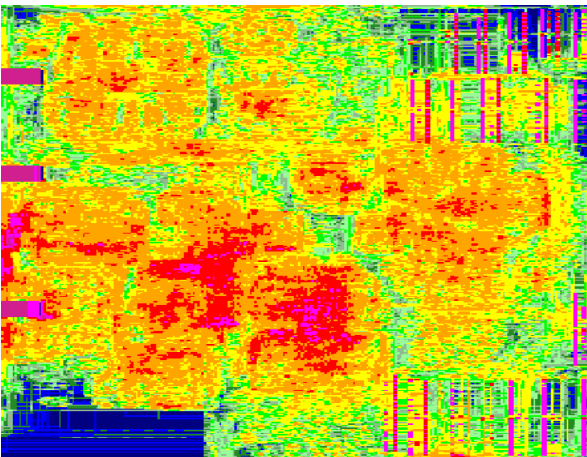
(a)	Num. Congested Nets: 9319	
(b)	Num. Congested Nets: 56051	
(c)	Num. Congested Nets: 9066	

Figure 6.12 Final routing congestion. (a) No timing-driven placement flow (NO-TDP) (b) Net-weighted timing-driven placement flow (TDP) (c) Integrated Timing Optimization and Placement (ITOP). (The regions colored pink and purple have more than 100% global routing resource usage).

timing-driven placement happens to be the most popular approach. This chapter demonstrates that such an approach can improve timing, but significantly degrade the wire length and routability of the design. In addition, it empirically demonstrates that there needs to be a tight coupling between timing optimization and placement, and that both these steps should rely on accurate timing from a state-of-the-art static timing engine. This in turn highlights the drawbacks of existing incremental placement approaches that use mathematical programming techniques based on inaccurate delay models to perform timing-driven placement.

Alternately, an Integrated Timing Optimization and Placement (ITOP) approach has been developed to achieve timing closure on nanometer-scale VLSI designs. ITOP uses an incremental approach with tight integration between placement and timing optimization to gradually improve design timing without degrading the wire length and routability. In addition, ITOP relies on accurate timing from a static timing engine to perform placement and optimization. Experimental results on high performance industrial designs show that ITOP can significantly improve design timing with negligible impact to the total wire length and routability of the design.

CHAPTER 7. GENERAL CONCLUSIONS

Placement is a fundamental and still highly relevant problem in VLSI CAD. The quality of the placement significantly impacts the ability of a physical synthesis tool or designer to achieve design closure. An industry-strength placement algorithm must be reasonably fast, yet still obtain high-quality solutions.

In addition, placement can no longer be considered as an independent step in the design of nanometer-scale integrated circuits. Due to the dominance of interconnect delay in the nanometer regime, placement algorithms need to closely interact with timing optimization transforms like buffer insertion, gate sizing, and logic re-synthesis to achieve timing closure.

This dissertation presents novel techniques that have been developed to address the key challenges faced during nanometer-scale integrated circuit placement. The first part of this dissertation presents efficient and high-quality techniques to perform force-directed global placement and legalization on multi-million gate mixed-size circuits. The second part of the dissertation presents placement techniques that have been developed to address the issues of clock power minimization and timing closure within a physical synthesis flow.

The key contributions of this work in the area of global placement and legalization are:

- An efficient *Density Aware Module Spreading* technique to spread the modules during the early stages of global placement. This technique roughly maintains the relative ordering of the modules as obtained by solving the quadratic program in both the horizontal and vertical directions.
- An effective linearization technique called *Force-vector Modulation* that restructures the placement at a global scale to minimize wire length without sacrificing the degree of spreading.

- An *Iterative Local Refinement* technique to reduce the wire length based on the half-perimeter wire length measure. This technique is applied on a coarse global placement and is highly effective in simultaneously reducing the wire length and spreading the modules.
- A *multilevel placement framework* using circuit clustering, that incorporates the above techniques during global placement to handle multi-million gate mixed-size circuits.
- An *Iterative Clustering Algorithm* to perform macro-block legalization, while guaranteeing minimum perturbation of the macros from their global placement locations.
- An efficient *Slice-based Cell Movement* technique to perform standard-cell legalization in the presence of placement blockages.

To minimize clock power in high performance integrated circuits several techniques have been developed and incorporated within a force-directed global placement framework. This yields a *Clock Constraint Aware Timing-driven Placement* (CCATP) algorithm. The CCATP algorithm simultaneously optimizes the timing by minimizing the weighted total wire length, and clock power by optimizing the placement of the clocked elements (latches, flip-flops etc.,) within a circuit.

Finally, to achieve timing closure on nanometer-scale VLSI designs, an *Integrated Timing Optimization and Placement* (ITOP) algorithm is presented. ITOP uses an incremental approach with tight integration between placement and timing optimization to gradually improve design timing without degrading the wire length and routability. In addition, ITOP relies on accurate timing from a static timing engine to perform placement and optimization.

The effectiveness of the techniques is demonstrated by: (a) comparing them with existing approaches to integrated circuit placement, and (b) embedding them within an industrial physical synthesis tool that is used in the design of high performance integrated circuits in the 65nm and 45nm process technology nodes.

BIBLIOGRAPHY

- [1] S. N. Adya, S. Chaturvedi, J. A. Roy, D. Papa, and I. L. Markov. Unification of partitioning, floorplanning and placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 550–557, 2004.
- [2] A. R. Agnihotri, S. Ono, C. Li, M. C. Yildiz, A. Khatkhate C.-K. Koh, and P. H. Madden. Mixed block placement via fractional cut recursive bisection. *IEEE Transactions on Computer-Aided Design*, 24(5):748–761, May 2005.
- [3] C. J. Alpert, S. Karandikar, Z. Li, G.-J. Nam, S. T. Quay, H. Ren, C. N. Sze, P. G. Villarrubia, and M. Yildiz. Techniques for fast physical synthesis. *Proceedings of IEEE*, 95(3):573–599, March 2007.
- [4] U. Brenner and M. Struzyna. Faster and better global placement by a new transportation algorithm. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 591–596, 2005.
- [5] U. Brenner and J. Vygen. Legalizing a placement with minimum total movement. *IEEE Transactions on Computer-Aided Design*, 23(12):1597 – 1613, December 2004.
- [6] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Can recursive bisection produce routable placements. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 477–482, 2000.
- [7] T. Chan, J. Cong, T. Kong, and J. Shinnerl. Multilevel optimization for large-scale circuit placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 171–176, 2000.

- [8] T. Chan, J. Cong, and K. Sze. Multilevel generalized force-directed method for circuit placement. In *Proceedings of the ACM International Symposium on Physical Design*, pages 185–192, 2005.
- [9] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie. mPL6: Enhanced multilevel mixed-size placement. In *Proceedings of the ACM International Symposium on Physical Design*, pages 212–214, 2006.
- [10] C. C. Chang, J. Cong, and X. Yuan. Multi-level placement for large-scale mixed-size IC designs. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 325–330, 2003.
- [11] T.-C. Chen, T.-C. Hsu, Z.-W. Jiang, and Y.-W. Chang. NTUplace: A ratio partitioning based placement algorithm for large-scale mixed-size designs. In *Proceedings of the ACM International Symposium on Physical Design*, pages 236–238, 2005.
- [12] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. A high-quality mixed-size analytical placer considering preplaced blocks and density constraints. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 187 – 192, 2006.
- [13] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design*, 27(7):1228–1240, July 2008.
- [14] W. Chen, C.-T. Hsieh, and M. Pedram. Simultaneous gate sizing and placement. *IEEE Transactions on Computer-Aided Design*, 19(2):206–214, February 2000.
- [15] Y. Cheon, P.-H. Ho, A. B. Kahng, S. Reda, and Q. Wang. Power-aware placement. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 795–800, 2005.
- [16] W. Choi and K. Bazargan. Incremental placement for timing optimization. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 463–466, 2003.

- [17] A. Chowdhary, K. Rajagopal, S. Venkatesan, T. Cao, V. Tiourin, Y. Perasuram, and B. Halpin. How accurately can we model timing in a placement engine? In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 801–806, 2005.
- [18] J. Cong and M. Xie. A robust detailed placement for mixed-size ic designs. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 188–194, 2006.
- [19] J. Cong and M. Xie. A robust mixed-size legalization and detailed placement algorithm. *IEEE Transactions on Computer-Aided Design*, 27(8):1349–1362, August 2008.
- [20] David E. Duarte, N. Vijaykrishnan, and Mary Jane Irwin. A clock power model to evaluate impact of architectural and technology optimizations. *IEEE Transactions on VLSI Systems*, 10:844–855, 2002.
- [21] H. Eisenmann and F. Johannes. Generic global placement and floorplanning. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 269–274, 1998.
- [22] Michael K. Gowan, Larry L. Biro, and Daniel B. Jackson. Power considerations in the design of the alpha 21264 microprocessor. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 726–731, 1998.
- [23] K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17:219–229, 1970.
- [24] B. Halpin, C. R. Chen, and N. Sehgal. Timing driven placement using physical net constraints. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 780–783, 2001.
- [25] D. Hill. Method and system for high speed detailed placement of cells within an integrated circuit design. US Patent 6370673, April 2002.
- [26] W. Hou, X. Hong, W. Wu, and Y. Cai. A path-based timing-driven quadratic placement algorithm. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 745–748, 2003.

- [27] B. Hu and M. Marek-Sadowska. Fine granularity clustering for large scale placement problems. In *Proceedings of the ACM International Symposium on Physical Design*, pages 67–74, 2003.
- [28] B. Hu and M. Marek-Sadowska. Multilevel fixed-point-addition-based VLSI placement. *IEEE Transactions on Computer-Aided Design*, 24(8):1188–1203, August 2005.
- [29] S. Hur, T. Cao, K. Rajagopal, Y. Perasuram, A. Chowdhary, V. Tiourin, and B. Halpin. Force directed Mongrel with physical net constraints. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 214–219, 2003.
- [30] S.-W. Hur and J. Lillis. Mongrel: Hybrid techniques for standard cell placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 165–170, 2000.
- [31] M. A. B. Jackson and E. S. Kuh. Performance-driven placement of cell based ics. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 370–375, 1989.
- [32] A. B. Kahng, S. Reda, and Q. Wang. Architecture and details of a high quality, large-scale analytical placer. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 890–897, 2005.
- [33] A. B. Kahng and Q. Wang. An analytical placer for mixed-size placement and timing-driven placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 565–572, 2004.
- [34] A. B. Kahng and Q. Wang. Implementation and extensibility of an analytic placer. *IEEE Transactions on Computer-Aided Design*, 24(5):734–747, May 2005.
- [35] A. Kennings and K. P. Vorwerk. Force-directed methods for generic placement. *IEEE Transactions on Computer-Aided Design*, 25(10):2076–2087, October 2006.

- [36] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design*, 10(3):356–365, March 1991.
- [37] T. Kong. A novel net weighting algorithm for timing-driven placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 172–176, 2002.
- [38] Z. M. Kurzum et al. Method for legalizing the placement of cells in an integrated circuit layout. US Patent 7089521, August 2006.
- [39] Y. Lu, C. N. Sze, X. Hong, Q. Zhou, Y. Cai, L. Huang, and J. Hu. Register placement for low power clock network. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 588–593, 2005.
- [40] T. Luo, D. Newmark, and D. Z. Pan. A new LP based incremental timing driven placement for high performance designs. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 1115–1120, 2006.
- [41] M. D. Moffitt, D. A. Papa, Z. Li, and C. J. Alpert. Path smoothing via discrete optimization. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 8–13, 2008.
- [42] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI module placement based on rectangle-packing by the sequence pair. *IEEE Transactions on Computer-Aided Design*, 15(12):1518–1524, December 1996.
- [43] G.-J. Nam. ISPD 2006 placement contest: Benchmark suite and results. In *Proceedings of the ACM International Symposium on Physical Design*, pages 167–167, 2006.
- [44] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz. The ISPD2005 placement contest and benchmark suite. In *Proceedings of the ACM International Symposium on Physical Design*, pages 216–220, 2005.

- [45] G.-J. Nam, S. Reda, C. J. Alpert, P. G. Villarrubia, and A. B. Kahng. A fast hierarchical quadratic placement algorithm. *IEEE Transactions on Computer-Aided Design*, 25(4):678–691, April 2006.
- [46] Sani Nassif. Delay variability: Sources, impacts and trends. In *IEEE International Solid-State Circuits Conference*, pages 368–369, 2000.
- [47] W. Naylor et al. Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer. US Patent 6301693, October 2001.
- [48] M. Pan, N. Viswanathan, and C. Chu. An efficient and effective detailed placement algorithm. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 48–55, 2005.
- [49] D. A. Papa, T. Luo, M. D. Moffitt, C. N. Sze, Z. Li, G.-J. Nam, C. J. Alpert, and I. L. Markov. RUMBLE: An incremental timing-driven physical-synthesis optimization algorithm. *IEEE Transactions on Computer-Aided Design*, 27(12):2156–2168, December 2008.
- [50] Ruchir Puri, Leon Stok, and Subhrajit Bhattacharya. Keeping hot chips cool. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 285–288, 2005.
- [51] K. Rajagopal, T. Shaked, Y. Perasuram, T. Cao, A. Chowdhary, and B. Halpin. Timing driven force directed placement with physical net constraints. In *Proceedings of the ACM International Symposium on Physical Design*, pages 60–66, 2003.
- [52] H. Ren, D. Z. Pan, C. J. Alpert, and P. Villarrubia. Diffusion-based placement migration. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 515–520, 2005.
- [53] H. Ren, D. Z. Pan, C. J. Alpert, P. G. Villarrubia, and G.-J. Nam. Diffusion-based placement migration with application on legalization. *IEEE Transactions on Computer-Aided Design*, 26(12):2158 – 2172, December 2007.

- [54] H. Ren, D. Z. Pan, and D. S. Kung. Sensitivity guided net weighting for placement-driven synthesis. *IEEE Transactions on Computer-Aided Design*, 24(5):711–721, May 2005.
- [55] P. J. Restle, T. G. McNamara, D. A. Webber, P. J. Camporese, K. F. Eng, K. A. Jenkins, D. H. Allen, M. J. Rohn, M. P. Quaranta, D. W. Boerstler, C. J. Alpert, C. A. Carter, R. N. Bailey, J. G. Petrovick, B. L. Krauter, and B. D. McCredie. A clock distribution network for microprocessors. *IEEE Journal of Solid-State Circuits*, 36(5):792–799, May 2001.
- [56] B. M. Riess and G. G. Ettl. SPEED: Fast and efficient timing driven placement. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 377–380, 1995.
- [57] J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov. Min-cut floorplacement. *IEEE Transactions on Computer-Aided Design*, 25(7):1313–1326, July 2006.
- [58] C. Sechen and A. L. Sangiovanni-Vincentelli. TimberWolf 3.2: A new standard cell placement and global routing package. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 432–439, 1986.
- [59] Rupesh Shelar. An efficient clusternig algorithm for low power clock tree synthesis. In *Proceedings of the ACM International Symposium on Physical Design*, pages 181–188, 2007.
- [60] G. Sigl, K. Doll, and F.M. Johannes. Analytical Placement: A linear or a quadratic objective function. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 427–431, 1991.
- [61] P. Spindler and F. M. Johannes. Fast and robust quadratic placement combined with an exact linear net model. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 179 – 186, 2006.
- [62] W.-J. Sun and C. Sechen. Efficient and effective placement for very large circuits. *IEEE Transactions on Computer-Aided Design*, 14(5):349–359, March 1995.

- [63] T. Taghavi, X. Yang, B.-K. Choi, M. Wang, and M. Sarrafzadeh. Dragon2005: Large-scale mixed-size placement tool. In *Proceedings of the ACM International Symposium on Physical Design*, pages 245–247, 2005.
- [64] L. Trevillyan, D. Kung, R. Puri, L. N. Reddy, and M. A. Kazda. An integrated environment for technology closure of deep-submicron IC designs. *IEEE Design and Test of Computers*, 21(1):14–22, January 2004.
- [65] N. Viswanathan and C. C.-N. Chu. FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. *IEEE Transactions on Computer-Aided Design*, 24(5):722–733, May 2005.
- [66] N. Viswanathan, G.-J. Nam, C. J. Alpert, P. Villarubia, H. Ren, and C. Chu. RQL: Global placement via relaxed quadratic spreading and linearization. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 453–458, 2007.
- [67] N. Viswanathan, M. Pan, and C. Chu. Fastplace 2.0: An efficient analytical placer for mixed-mode designs. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 195–200, 2006.
- [68] N. Viswanathan, M. Pan, and C. Chu. Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 135–140, 2007.
- [69] K. Vorwerk and A. Kennings. An improved multi-level framework for force-directed placement. In *Proceedings of the Conference on Design Automation and Test in Europe*, pages 902–907, 2005.
- [70] K. Vorwerk, A. Kennings, and A. Vannelli. Engineering details of a stable force-directed placer. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 573–580, 2004.
- [71] J. Vygen. Algorithms for large-scale flat placement. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 746–751, 1997.

- [72] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 260–263, 2000.
- [73] Q. B. Wang, J. Lillis, and S. Sanyal. An LP-based methodology for improved timing-driven placement. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 18–21, 2005.
- [74] Z. Xiu and R. A. Rutenbar. Timing-driven placement by grid-warping. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 585–590, 2005.
- [75] J. Z. Yan, N. Viswanathan, and C. Chu. Handling complexities in modern large-scale mixed-size placement. In *Proceedings of the ACM/IEEE Design Automation Conference*, page To Appear, 2009.
- [76] B. Yao, H. Chen, C.-K. Cheng, N.-C. Chou, L.-T. Liu, and P. Suaris. Unified quadratic programming approach for mixed mode placement. In *Proceedings of the ACM International Symposium on Physical Design*, pages 193–199, 2005.