Iowa State University
Digital Repository

Graduate Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

2016

Physical design algorithms for asynchronous circuits

Gang Wu
Iowa State University

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

Part of the Computer Engineering Commons

Recommended Citation

Wu, Gang, "Physical design algorithms for asynchronous circuits" (2016). *Graduate Theses and Dissertations*. 15840.
https://lib.dr.iastate.edu/etd/15840

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.

المنارة للاستشارات

www.manaraa.com

Physical design algorithms for asynchronous circuits

by

Gang Wu

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 Chu, Major Professor

Randall L. Geiger

Akhilesh Tyagi

Phillip H. Jones III

Lizhi Wang

Iowa State University

Ames, Iowa

2016

Copyright © Gang Wu, 2016. All rights reserved.

المنارة للاستشارات

www.manaraa.com

DEDICATION

I would like to dedicate this thesis to my families without whose help and support I would not have been able to complete this work.

المنارة للاستشارات

www.manaraa.com

TABLE OF CONTENTS

LIST OF TABLES . . . viii

LIST OF FIGURES . . . ix

ACKNOWLEDGEMENTS . . . xi

ABSTRACT . . . xii

CHAPTER 1. OVERVIEW . . . 1

1.1 Physical Design for Asynchronous Circuits . . . 2

1.1.1 Placement . . . 3

1.1.2 Gate Sizing and Repeater Insertion . . . 3

1.1.3 Slack Matching . . . 4

1.1.4 Timing Analysis . . . 5

1.2 Research Contributions . . . 6

1.3 Dissertation Organization . . . 7

CHAPTER 2. ASYNCHRONOUS CIRCUIT PLACEMENT USING LAGRANGIAN RELAXATION . . . 8

2.1 Introduction . . . 8

2.2 Timing for Asynchronous Circuits . . . 11

2.2.1 Performance for Asynchronous Circuits . . . 11

2.2.2 Timing Constraints . . . 12

2.3 Asynchronous Placement with Lagrangian Relaxation . . . 13

2.3.1 Problem Formulation . . . 14

2.3.2 Lagrangian Relaxation . . . 15

2.3.3 Simplification of $\mathcal{LRS}$ . . . 16

المنارة للاستشارات

www.manaraa.com

2.3.4 Solving $\mathcal{LRS}^*$ . . . 16

2.3.5 Solving $\mathcal{LDP}$ . . . 17

2.4 Asynchronous Placement Flow for QDI Pipeline Templates . . . 18

2.4.1 Asynchronous Design with Pre-Charged Half Buffer (PCHB) Templates 18

2.4.2 Asynchronous Placement Flow . . . 19

2.5 Experimental Results . . . 20

2.6 Conclusion . . . 22

CHAPTER 3. TIMING-DRIVEN PLACEMENT BY LAGRANGIAN RELAXATION . . . 23

3.1 Introduction . . . 23

3.2 Problem Formulations . . . 26

3.2.1 Wirelength-driven Placement Problem ($\mathcal{WDP}$) . . . 28

3.2.2 Timing-driven Placement Problems . . . 28

3.2.3 A General Timing-driven Placement Problem ($\mathcal{GTDP}$) . . . 32

3.3 Our Proposed Approaches on Solving $\mathcal{GTDP}$ . . . 34

3.3.1 Spreading-inside Approach . . . 34

3.3.2 Spreading-outside Approach . . . 38

3.3.3 Comparing our approaches with previous Lagrangian relaxation based $\mathcal{TDP}$ algorithms . . . 39

3.4 Detailed Implementation . . . 40

3.4.1 POLAR: a wirelength-driven placer based on quadratic and rough legalization techniques . . . 40

3.4.2 TD-POLAR: a general timing-driven placement tool . . . 42

3.4.3 Timing-driven Detailed Placement . . . 43

3.5 Experiments . . . 45

3.6 Conclusion . . . 52

3.7 Acknowledgments . . . 52

المنارة للاستشارات

www.manaraa.com

CHAPTER 4. DETAILED PLACEMENT ALGORITHM FOR VLSI DESIGN WITH DOUBLE-ROW HEIGHT STANDARD CELLS 53

4.1 Introduction 53

4.2 Overview 56

4.3 Detailed Placement Approach 57

4.3.1 Matching Graph Construction 57

4.3.2 Edge Weight Calculation 59

4.3.3 Maximum Weighted Matching 60

4.3.4 Matching Pair Selection 61

4.3.5 Unpair and Refinement 62

4.4 Experimental Results 62

4.5 Conclusions and Future Work 67

CHAPTER 5. GATE SIZING AND VTH ASSIGNMENT FOR ASYNCHRONOUS CIRCUITS USING LAGRANGIAN RELAXATION 68

5.1 Introduction 68

5.2 Preliminaries 71

5.2.1 Gate Selection Techniques for Synchronous Circuits 71

5.2.2 Full Buffer Channel Net Model 71

5.2.3 Asynchronous Performance Analysis 72

5.2.4 Asynchronous Timing Constraints 73

5.2.5 Asynchronous Gate Sizing and Vth Assignment 73

5.3 Simultaneous Gate Sizing and Vth Assignment by Lagrangian Relaxation . . . 75

5.3.1 Lagrangian Relaxation Subproblem ($\mathcal{LRS}$) 75

5.3.2 Simplified Lagrangian Relaxation Subproblem ($\mathcal{LRS}^*$) 75

5.3.3 Solving $\mathcal{LRS}^*$ 76

5.3.4 Lagrangian Dual Problem ($\mathcal{LDP}$) 77

5.3.5 Solving $\mathcal{LDP}$ 78

المنارة للاستشارات
www.manaraa.com

5.4 Static Timing Analysis for Asynchronous Circuits 79

5.4.1 Iterative Slew Update Approach 80

5.4.2 Convergence of the proposed approach 81

5.4.3 Extension to a tight lower bound 82

5.5 Asynchronous Gate Selection Flow 82

5.6 Experiments 83

5.7 Conclusions 85

CHAPTER 6. SIMULTANEOUS SLACK MATCHING, GATE SIZING AND REPEATER INSERTION FOR ASYNCHRONOUS CIRCUITS 88

6.1 Introduction 88

6.2 A Motivating Example 91

6.3 Optimization Framework Overview 93

6.4 Lagrangian Relaxation Framework 93

6.5 Simultaneous Gate Sizing, Repeater Insertion and Pipeline Buffer Insertion . . 95

6.5.1 Constructing Candidate Pipeline Buffer Location 95

6.5.2 Constructing Look-up Tables for Fast Repeater Insertion 97

6.5.3 Solving $\mathcal{LDP}$ 98

6.5.4 Solving $\mathcal{LRS}^*$ 99

6.6 Experiments 100

6.7 Conclusions 103

CHAPTER 7. A FAST INCREMENTAL MAXIMUM CYCLE RATIO ALGORITHM 105

7.1 Introduction 105

7.2 Preliminaries 108

7.2.1 Maximum cycle ratio problem 108

7.2.2 Traditional maximum cycle ratio algorithms 109

7.3 Our Incremental Cycle Ratio Algorithm 110

7.3.1 Considering incremental changes on an edge 111

www.manaraa.com

7.3.2 Considering incremental changes on a node . . . 112
7.3.3 Considering HOW and KO incrementally . . . 113
7.3.4 An overview of our incremental MCR algorithm . . . 114
7.3.5 Cycle detection . . . 114
7.3.6 Local upward search . . . 118
7.3.7 Global downward search . . . 119
7.4 Timing-driven detailed placement . . . 120
7.5 Experiments . . . 124
7.6 Conclusions . . . 126
BIBLIOGRAPHY . . . 127

المنارة للاستشارات
www.manaraa.com

LIST OF TABLES

2.1 Comparison on asynchronous benchmarks 21

3.1 Comparison on non-timing-driven placement flow and commercial timing-driven placement flow 44

3.2 Statistics of the Circuits 46

3.3 Comparison on DP algorithms using spreading-inside approach 50

3.4 Comparison on DP algorithms using spreading-outside approach 51

4.1 Comparison on asynchronous benchmarks 65

4.2 Comparison on synchronous benchmarks 66

5.1 Comparison on transformed ISCAS89 benchmarks 86

5.2 Comparison on asynchronous benchmarks 87

6.1 Comparison on transformed ISCAS89 benchmarks 101

6.2 Comparison on asynchronous benchmarks 102

7.1 Comparison on small size random graphs 121

7.2 Analysis on medium and large size random graphs 122

7.3 Comparison on ISPD 2005 benchmarks 123

المنارة للاستشارات

www.manaraa.com

LIST OF FIGURES

2.1 Asynchronous ALU. . . . 11
2.2 Marked graph representation for Asynchronous ALU. . . . 12
2.3 PCHB pipeline template. . . . 19
2.4 QDI PCHB placement flow. . . . 20

3.1 Modified timing graph for $\mathcal{STDP}$. . . . 33
3.2 (a) The spreading-inside approach. (b) The spreading-outside approach. 35
3.3 (a) Cell overlaps after quadratic placement. (b) An almost legal placement obtained by rough legalization. (c) Use of the roughly legal placement to guide the spreading force generation. . . . 41
3.4 PCHB pipeline template. . . . 45
3.5 (a) The convergence of s38417 by SI. (b) The wirelength and cycltime trend of s38417 by SI. (c) The convergence of s38417 by SO. (d) The wirelength and cycltime trend of s38417 by SO. . . . 47

4.1 Detailed Placement Flow 57
4.2 Construct Matching Graph 58
4.3 (a) Pair up cells with large width difference (b) Pair up cells with small width difference 60
4.4 Experiment on adaptec2_dr benchmark 63
4.5 Experiment on bigblue1_dr benchmark 64

5.1 Asynchronous ALU. . . . 72
5.2 Marked graph representation for Asynchronous ALU. . . . 72

المنارة للاستشارات

www.manaraa.com

5.3 Asynchronous Gate Selection Flow. 82

5.4 (a) The convergence sequence of s38417. (b) Cycle time and leakage power trends of s38417. 84

6.1 (a) Three-stage pipeline. (b) FBCN model. 91

6.2 (a) Stall fixed by inserting pipeline buffers. (b) Stall fixed by gate sizing or repeater insertion. 92

6.3 High-level View of Our Framework. 93

6.4 A three-stage PCHB pipeline. 95

6.5 Pre-inserted pipeline buffer: (a) Transparent mode (b) Opaque mode . 96

6.6 Look-up table at each pin. 97

6.7 Lagrangian dual problem solver. 98

7.1 Finding the maximum cycle ratio in a graph. 108

7.2 An example of Karp and Orlin's algorithm 111

7.3 $e \in c^*$ and $w(e)$ is increased. 113

7.4 Overview of our incremental MCR algorithm. 114

7.5 (a) Before the cost shifting. (b) After the cost shifting. 115

7.6 (a) Before breaking at v. (b) After breaking at v. 116

7.7 The distance bucket data structure. 117

7.8 Timing-driven detailed placement. 120

المنارة للاستشارات

www.manaraa.com

ACKNOWLEDGEMENTS

I would like to thank my advisor, Prof. Chris Chu, for his guidance, patience and great support throughout my research. I would also like to thank my committee members for their efforts and valuable feedbacks to this work: Dr. Randall L. Geiger, Dr. Akhilesh Tyagi, Dr. Phillip H. Jones III and Dr. Lizhi Wang.

المنارة للاستشارات

www.manaraa.com

ABSTRACT

Asynchronous designs have been demonstrated to be able to achieve both higher performance and lower power compared with their synchronous counterparts [50] [51] [19]. It provides a very promising solution to the emerging challenges in advanced technology. However, due to the lack of proper EDA tool support, the design cycle for asynchronous circuits is much longer compared with the one for synchronous circuits. Thus, even with many advantages, asynchronous circuits are still not the mainstream in the industry. In this thesis, we provides several algorithms to resolve the emerging issues for the physical design of asynchronous circuits. Our proposed algorithms optimize asynchronous circuits using placement, gate sizing, repeater insertion and pipeline buffer insertion techniques. An incremental maximum cycle ratio algorithm is also proposed to speed up the timing analysis of asynchronous circuits.

المنارة للاستشارات

www.manaraa.com

CHAPTER 1. OVERVIEW

With the continual diminishing of feature size, integrated circuit (IC) design is progressively more difficult. Synchronous design is facing particularly severe challenges due to the increasing variations in process parameters and demand in low power consumption. Asynchronous design provides a very promising solution to the emerging challenges in advanced technology. An asynchronous circuit, or self-timed circuit, is a sequential digital logic circuit which is not governed by a clock circuit or global clock signal. Instead they often use signals that indicate completion of instructions and operations, specified by simple data transfer protocols. In theory, the potential advantages of asynchronous design over synchronous design include robustness towards process-voltage-temperature (PVT) variations, lower power consumption, avoidance of the difficult problem of clock distribution, higher operating speed, less emission of electromagnetic noise, lower stress on power distribution network, improved security, and better composability and modularity [2] [9] [52]. However, even asynchronous design has all the potential advantages, synchronous circuits still predominate. The main reason is that asynchronous circuit is generally harder to design. It requires designers to pay a lot of attention to issues like hazards, dynamic states and ordering of operations. It also requires designers to develop a very different mindset and to learn very different techniques. Another reason is that computer-aided design (CAD) tool support for asynchronous design is grossly inadequate. As a result, asynchronous design is not popular in practice. Therefore, in this thesis, we force on developing the physical design algorithms for asynchronous circuits.

المنارة للاستشارات
www.manaraa.com

1.1 Physical Design for Asynchronous Circuits

In integrated circuit design, physical design is a step in the standard design cycle which follows after the circuit design. At this step, circuit representations of the components (devices and interconnects) of the design are converted into geometric representations of shapes which, when manufactured in the corresponding layers of materials, will ensure the required functioning of the components. This geometric representation is called integrated circuit layout. This step is usually split into several sub-steps, which include both design and verification and validation of the layout.

Physical design algorithms for synchronous circuits, including placement, gate sizing, repeater insertion, etc, have been studied for decades and are quit mature. However, very few works have been done for asynchronous circuits, and most existing works for asynchronous physical design directly leverage synchronous CAD tools [6] [90] [78]. For synchronous circuit, the circuit performance is bounded by the most critical path and synchronous CAD tools will minimize the maximum path delay between flip-flops. However, in the case of asynchronous circuit, the performance is bounded by the most critical cycle [9]. This creates issues when applying synchronous tools for asynchronous optimization, since the optimization objective is different and timing-loops are not supported. Another issue to leverage synchronous placement tool is to enforce the timing constraints necessary for the functional correctness of asynchronous circuits. Instead of setup and hold time constraints for typical synchronous design, certain asynchronous design style requires relative timing constraints which constrains the relative delay between two paths [74]. Some other design styles require minimum and maximum bounded delay values on gates and wires [76]. This difference of timing assumptions between asynchronous design and synchronous design creates extra difficulties when applying synchronous placement tools to asynchronous design. Therefore, it is important for us to propose physical design algorithms which specifically target at optimizing asynchronous circuits.

www.manaraa.com

1.1.1 Placement

Placement is a critical step in VLSI design flow. It decides cell locations, which to a large extent determines the length of interconnects and hence the interconnect parasitics. Placement has been shown to have a great impact on the final circuit quality implemented in advanced process [85]. In advanced technology, the importance of placement continues to grow, since it determines the interconnect delay, which has been a dominating factor of the circuit delay.

The placement of circuit is usually performed in two steps: global placement and detailed placement. Global placement aims at generating a rough placement solution that may violate some placement constraints (e.g., there may be overlaps among modules) while maintaining a global view of the whole netlist. Detailed placement further improves the legalized placement solution in an iterative manner by rearranging a small group of modules in a local region while keeping all other modules fixed.

The placement problem for asynchronous circuits is different from the traditional synchronous performance-driven placement formulation. Roughly speaking, as the performance is determined by the longest delay loop, the placement algorithm should minimize the length of interlocked loops. A similar problem has been considered in [34]. Another difference is that there is no need to consider the clock network construction during placement. In synchronous circuit placement, it is beneficial to bring sequencing elements closer together so as to minimize the wirelength of the clock network. We can simply ignore this issue here.

1.1.2 Gate Sizing and Repeater Insertion

Gate sizing and buffer insertion are techniques to allocate enough drive strength to drive the load of a circuit. They have been demonstrated to be very effective in optimizing circuit timing and in fulfilling signal transition time constraints. They are indispensable in synchronous design. It is very clear that they should also be applied to asynchronous design.

In advanced process, the resistance and capacitance of interconnects are dominating. Hence, one cannot reasonably estimate the interconnect parasitics until at least after placement. The dilemma is that without parasitics, logic and circuit optimizations (including gate sizing and

المنارة للاستشارات

www.manaraa.com

buffer insertion) cannot be reliably performed. But without performing optimizations, the netlist is not finalized and hence placement cannot be done. For synchronous design, this chicken-or-egg problem results in the timing closure issue. This is solved by the physical synthesis methodology, which integrates logic synthesis and optimization with placement, and incrementally refines the netlist and layout through multiple iterations. This iterative process sometimes takes a long time to converge and it may not converge to a good solution. For asynchronous circuit, the situation is better as it can adapt to any timing uncertainty. Even if logic synthesis and optimization are done completely before placement (i.e., without any information on interconnect parasitics), the resulting circuit will still be functional although the performance will be very sub-optimal.

Gate sizing and repeater insertion for synchronous circuits has been studied for decades and there are many works tackling these problems [14] [26] [80]. However, for asynchronous circuits, there are only very few works on it. Most of the automatic synthesis flows for asynchronous circuits try to directly leverage synchronous EDA tools [6] [78]. As the circuit structure, performance metric and timing constraints for asynchronous circuits are quite different from those for synchronous circuits, these approaches require to break the timing-loops and add explicit timing constraints the number of which is exponential to the circuit size. For large scale designs, the complicated timing constraints are beyond the ability of synchronous EDA tools to handle thus inferior results are generated. In [30], a genetic algorithm based simultaneous gate sizing and V_{th} assignment technique specific for asynchronous circuits has been proposed to minimize the leakage power while maintaining the performance requirements. However, genetic algorithms usually have long runtime and are not scalable, which makes it unsuitable for large scale circuits.

1.1.3 Slack Matching

Slack matching resolves the stalls of the circuit by inserting pipeline buffers, and this type of optimization is specifically used for the optimization of asynchronous circuits. In particular, stalls are major obstacles limiting the performance of pipelined asynchronous circuits [72]. Due to the slack elasticity for most asynchronous designs, adding pipeline buffers to the design

المنارة للاستشارات
www.manaraa.com

will not change its input/output functionality, but can help remedy the stalls [7]. Thus, slack matching, which inserts minimum number of pipeline buffers to guarantee the most critical cycle meets the desired cycle time, is widely used for asynchronous circuits [9]. Most previous works related to slack matching formulate the problem as a mixed integer linear program (MILP) [7] [67] [54], which is NP-Complete and the integral constraints need to be relaxed in order to solve the problem efficiently. In [81], a heuristic algorithm is proposed to solve the problem by leveraging the asynchronous communication protocol.

1.1.4 Timing Analysis

A fast and accurate static timing analysis (STA) method is essential to guide the physical design algorithms to achieve a good solution within a short amount of runtime. For synchronous circuits, this can be done by a simple graph traversal as the corresponding combinational logic network can be represented as a directed acyclic graph (DAG). However, for asynchronous circuits, the way to perform STA is not straightforward due to its more general circuit structure which might contain internal combinational loops. In [66], a STA flow on pre-charged half buffer (PCHB) and Multi-Level Domino (MLD) templates has been proposed, which leverages a commercialized synchronous timing analyzer. However, this approach is limited to template based designs as automatically finding the cut points requires a regular circuit structure. Also, the achieved timing value is not accurate as time borrowing across the broken segments is not allowed.

STA for asynchronous circuits can be reduced to the maximum cycle ratio problem. Given a directed cyclic graph and each edge in the graph is associated with two numbers: *cost* and *transition time*. Let the cost (respectively, transition time) of a cycle in the graph be the sum of the costs (respectively, transition times) of all the edges within this cycle. Assuming the transition time of a cycle is non-zero, the ratio of this cycle is defined as its cost divided by its transition time. The maximum cycle ratio problem finds the cycle whose ratio is the maximum in a given graph [18].

المنارة للاستشارات
www.manaraa.com

1.2 Research Contributions

The main contributions of this thesis are as follows:

- An effective timing-driven global placement approach for asynchronous circuits [88]. Based on the Lagrangian relaxation framework, we are able to transfer the timing-driven placement problem into the wirelength minimization problem. The transformed problem can then be efficiently solved using any standard wirelength-driven placement engine that can handle net weights.

- A timing-driven global placement approach which is applicable to synchronous circuits, synchronous circuits with sequential optimization techniques and asynchronous circuits. In particular, we proposed two different approaches on applying the Lagrangian relaxation framework to the timing driven placement problem and compared their effectiveness.

- A detailed placement approach which can handle designs with any number of double-row height standard cells [86]. In particular, we transformed single-row height cells to double-row height by either expanding the cell or pairing up two single-row height cells together based on our maximum weighted matching algorithm.

- A gate sizing and V_{th} assignment approach for asynchronous circuits [89]. We also proposed a fast and effective slew updating strategy which is able to address the timing-loops of asynchronous circuits during static timing analysis.

- A unified optimization flow for asynchronous circuits incorporating gate sizing, repeater insertion and pipeline buffer insertion together [87]. In particular, we proposed a methodology for handling pipeline buffer insertion under the LR framework, and a fast look-up table based repeater insertion approach to speed up the evaluation approach.

- An algorithm which is able to quickly find the maximum cycle ratio on an incrementally changing directed cyclic graph. We proposed a distance bucket approach to speed up the process of finding the longest paths on graph without positive cycle. A timing-driven detailed placement approach is also proposed.

المنارة للاستشارات

www.manaraa.com

1.3 Dissertation Organization

The rest of this thesis is organized as follows. Chapter 2 introduce our global placement approach for asynchronous circuits. Chapter 3 extends our global placement approach to three different circuit design styles: synchronous circuits, synchronous circuits with sequential optimization techniques and asynchronous circuits. Chapter 4 introduce our detailed placement approach which is able to handle the double-row height cells in our asynchronous standard cell library. Chapter 5 introduce our gate sizing and V_{th} assignment approach for asynchronous circuits. Chapter 6 extends our gate sizing approach to perform slack matching, gate sizing and repeater insertion simultaneously for asynchronous circuits. Chapter 7 presents our incremental maximum cycle ratio algorithm which is essential for the timing analysis of asynchronous circuits.

المنارة للاستشارات
www.manaraa.com

CHAPTER 2. ASYNCHRONOUS CIRCUIT PLACEMENT USING LAGRANGIAN RELAXATION

Recent asynchronous VLSI circuit placement approach tries to leverage synchronous placement tools as much as possible by manual loop-breaking and creation of virtual clocks. However, this approach produces an exponential number of explicit timing constraints which is beyond the ability of synchronous placement tools to handle. Thus, synchronous placer can only produce suboptimal results. Also, it can be very costly in terms of runtime. This paper proposed a new placement approach for asynchronous VLSI circuits. We formulated the asynchronous timing-driven placement problem and transform this problem into a weighted wirelength minimization problem based on a Lagrangian relaxation framework. The problem can then be efficiently solved using any standard wirelength-driven placement engine that can handle net weights. We demonstrate our approach on QDI PCHB asynchronous circuit with a state-of-art quadratic placer. The experimental results show that our algorithm can effectively improve the asynchronous circuits performance at placement stage. In addition, the runtime of our algorithm is shown to be more scalable to large-scale circuits compared with the loop-breaking approach.

2.1 Introduction

With the continual diminishing of feature size, integrated circuit (IC) design is progressively more difficult. Synchronous design is facing particularly severe challenges due to the increasing variations in process parameters and demand in low power consumption. Asynchronous design provides a promising solution to the emerging challenges in advanced technology. Its potential advantages over synchronous design include robustness towards process-voltage-temperature

المنارة للاستشارات
www.manaraa.com

(PVT) variations, lower power consumption, avoidance of the difficult problem of clock distribution, higher operating speed, less emission of electromagnetic noise, lower stress on power distribution network, improved security, and better composability and modularity [2] [9] [52]. However, even asynchronous design has all the potential advantages, synchronous circuits still predominate. A main reason is that EDA tool support for asynchronous design is grossly inadequate.

In current aggressive technologies, placement for asynchronous circuit becomes a more important issue, as wire delays are becoming more critical than gate delays. Most works on the timing-driven placement of asynchronous circuits directly leverage synchronous placement tools [6] [90] [78]. For synchronous circuit, the circuit performance is bounded by the most critical path and synchronous timing-driven placer will minimize the maximum path delay between flip-flops. However, in the case of asynchronous circuit, the performance is bounded by the most critical cycle [9]. This creates issues when applying synchronous placement tools for asynchronous timing optimization, since the optimization objective is different and timing-loops are not supported. In [6], a minimal set of cut-points is identified to break these timing loops. Then the handshaking cycles of the design are explicitly expressed as a set of set_max_delay timing constraints for each segment. The resulting number of constraints turns out to be far greater than in a typical synchronous placement flow and exponential to the circuit size. For large scale designs, it becomes impossible for placement engine to satisfy all these constraints within a reasonable amount of runtime. Also, these timing constraints are too conservative to achieve a good optimization result, as time borrowing is not allowed for segments along the same cycle.

Another issue to leverage synchronous placement tool is to enforce the timing constraints necessary for the functional correctness of asynchronous circuits. Instead of setup and hold time constraints for typical synchronous design, certain asynchronous design style requires relative timing constraints which constrains the relative delay between two paths [74]. Some other design styles require minimum and maximum bounded delay values on gates and wires [76]. This difference of timing assumptions between asynchronous design and synchronous design creates extra difficulties when applying synchronous placement tools to asynchronous design.

المنارة للاستشارات
www.manaraa.com

Few works have been done for timing aware placement algorithms targeting at optimizing critical cycles. In [34], sequential timing analysis based placement approach has been proposed to optimize the cycle delay for synchronous design under the assumption that retiming and clock skew scheduling can be applied. Unfortunately, this technique is practical only for the last stages of physical design and only a small amount of useful skew is allowed. In [42], performance and relative timing constraints for QDI circuits have been incorporated into a constructive placer to handle asynchronous designs. However, given their framework of construction based placement approach, this algorithm can easily be trapped into local minimum and produce suboptimal results. Also, this approach will lead to high density placement hot spots which can cause routability problems. In [35], a floorplan method for asynchronous circuits based on simulated annealing and sequence-pair has been proposed. However, they still need to leverage synchronous place-and-route tools at the placement stage.

In this paper, we proposed a new placement flow based on a Lagrangian relaxation framework. We formulated the asynchronous timing-driven placement problem considering both performance and timing constraints. Instead of adding explicit cycle constraints whose numbers can grow exponentially with circuit size, we incorporated the cycle metric calculation linear program into our problem formulation and the number of constraints we have is polynomial in circuit size. In addition, the special structure of the formulated timing-driven placement problem allows us to simplify the Lagrangian dual problem using Karush-Kuhn-Tucker (KKT) conditions and the original problem is transformed into a weighted wirelength minimization problem which can be solved effectively with existing placement approaches. The general modeling of performance and timing constraints also makes our approach applicable to a wide variety of asynchronous design styles, including Micropipeline [76], QDI [47], GasP [75] and Mousetraps [71] pipeline templates.

We explored our approach on quasi-delay-insensitive (QDI) Pre-Charged Half Buffer (PCHB) asynchronous designs synthesized using the Proteus RTL flow [6]. The Lagrangian relaxation frame work is incorporated into state-of-art quadratic placer POLAR [46]. We compared our results after detailed placement and routing with both an industrial placement tool and the Proteus placement flow.

المنارة للاستشارات
www.manaraa.com

The rest of this paper organized as follows. Section II introduces timing issues faced by asynchronous design. Section III elaborates our general Lagrangian relaxation framework. Section IV proposed our asynchronous placement flow on QDI PCHB asynchronous design. Section V shows the experimental results compared with other approaches. Finally, Section VI concludes the paper.

2.2 Timing for Asynchronous Circuits

First we define some notations that we use in this paper. An asynchronous circuit can be represented by a hypergraph $G = (V, E)$. Let $V = \{v_1, v_2, \ldots, v_{|V|}\}$ be the set of cells. Let $E = \{e_1, e_2, \ldots, e_{|E|}\}$ be the set of hyperedges. Let $AT = \{a_1, a_2, \ldots, a_{|V|}\}$ be the arrival time associated with each cell.

2.2.1 Performance for Asynchronous Circuits

Here we introduce a Petri net [64], which is a commonly used tool for modeling concurrent systems. A Petri net consists of places, transitions, and arcs. Places in a Petri net can have one or more tokens, or no token at all. The distribution of tokens over the places will represent an initial marking of the system. Transitions in a Petri net can fire if all its input places contain at least one token. When a transition fires, it consumes one token per input place and generates one token per output place. Specifically, a Petri net is called marked graph if all places have only one input and one output transition.

The performance of unconditional asynchronous circuits can be modeled using timed marked graphs. For conditional asynchronous circuits, we treat them as unconditional and the circuit performance can be guaranteed conservatively as proved in [53].

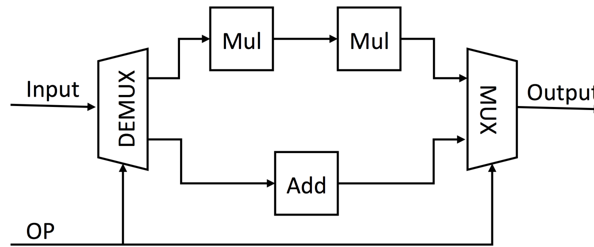

Figure 2.1: Asynchronous ALU.

المنارة للاستشارات
www.manaraa.com

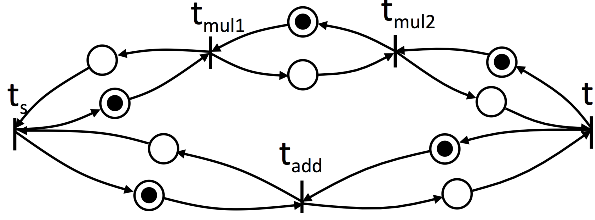

Figure 2.2: Marked graph representation for Asynchronous ALU.

As an example, an asynchronous ALU design with two operation modes, addition and multiplication, is shown in Fig. 2.1. We use the Full Buffer Channel Net model proposed in [8] to obtain timed marked graph. The extracted marked graph is shown in Fig. 2.2. Each cell is modeled using a transition (t) and asynchronous channels between cells are modeled with a pair of places (shown as circles), a forward place and a backward place, which are labeled with forward and backward delays of the corresponding channel. Black dots inside the circle denotes the initial marking of the marked graph.

For any marked graph, let C_p be the set of neighboring transition pairs which have a place between them. The cycle time τ can be obtained by solving the following linear program [49].

$$\begin{aligned} &\text{Minimize} \quad \tau \\ &\text{Subject to} \quad a_i + D_{ij} - m_{ij}\tau \leq a_j \quad \forall (i,j) \in C_p \end{aligned}$$

where a_i and a_j are the arrival time associated with transitions t_i and t_j, which correspond to nodes v_i and v_j in the graph. D_{ij} is the delay associated with place p between transition t_i and t_j, which corresponds to forward or backward path delay of an asynchronous channel. m_{ij} is the number of tokens in the place p. $m_{ij} = 0$ if the corresponding place p does not contain token, which is quite often.

2.2.2 Timing Constraints

Timing assumptions made for different logic implementation style have a direct influence in the timing constraints necessary to ensure hazard-free operation of asynchronous circuits. Except for delay-insensitive (DI) design [79] which are premised on the fact that they will function correctly regardless of the delays on the gates and the wires, timing constraints for other asynchronous designs fall into two categories [9].

المنارة للاستشارات
www.manaraa.com

First is explicit timing constraints in the form of minimum and maximum bounded delay values for gates and wires in the circuit. An example is the bounded-delay asynchronous circuits [76].

Let U_{ij} be the maximum bounded delay and L_{ij} be the minimum bounded delay between nodes v_i and v_j. Let C_e be the set of node pairs which we need to enforce explicit delay bounds. The explicit timing constraints can be written as:

$$L_{ij} \leq a_j - a_i \leq U_{ij} \quad \forall (i,j) \in C_e \tag{2.1}$$

Second is relative timing constraints, referred to as *relative timing* [74], which dictate the relative delay of two paths that stem from a common point of divergence. Examples design styles that have relative timing constraints include the quasi-delay-insensitive (QDI) design style, such as WCHB, PCHB and the Multi-Level Domino (MLD) template [9].

For a relative timing constraint from a node v_k and forking into two nodes v_i and v_j constraints can be written as:

$$|(a_i - a_k) - (a_j - a_k)| \leq I_{ij} \quad \forall (i,j) \in C_r \tag{2.2}$$

which bound the maximum difference in time that the signal arrives at the two end-points of the fork. This type of constraint captures the notion of an *isochronic fork* [9], a common type of constraint in quasi-delay-insensitive designs. Here I_{ij} is the delay bound for *isochronic fork*. C_r is the set of node pairs which have relative timing constraints.

2.3 Asynchronous Placement with Lagrangian Relaxation

Given an asynchronous circuit, we are interested in the minimum total wirelength and cycletime achievable with respect to the timing constraints necessary to guarantee functional correctness. In Section III-A, we first show how to formulate this problem as a constrained optimization problem. Then, we apply Lagrangian relaxation in Section III-B which is a general technique for converting constrained optimization problem into unconstrained problem. In Section III-C, we explore the special structure of the primal problem which allows us to extensively simplify the Lagrangian subproblem. In Section III-D, we show how to solve the

المنارة للاستشارات
www.manaraa.com

simplified Lagrangian subproblem as weighted wirelength minimization problem. In Section III-E, we describe how to solve Lagrangian dual problem using a direction finding approach.

2.3.1 Problem Formulation

We denote the x-coordinates of cells by a vector $\mathbf{x} = \left(x_1, x_2, \cdots, x_{|V|}\right)$, and y-coordinates by a vector $\mathbf{y} = \left(y_1, y_2, \cdots, y_{|V|}\right)$. For pure wirelength-driven placement, the objective is to minimize the sum of the half-perimeter bounding box (HPWL) for all hyperedge e:

$$\text{HPWL}(\mathbf{x}, \mathbf{y}) = \sum_{e \in E} [\max_{i \in e} x_i - \min_{i \in e} x_i + \max_{i \in e} y_i - \min_{i \in e} y_i]$$

Then the problem of minimizing both total HPWL and cycletime subject to timing constraints (1), (2) in section II can be formulated directly as:

$$\begin{aligned} &\text{Minimize} \quad \text{HPWL}(\mathbf{x}, \mathbf{y}) + \alpha\tau \\ &\text{Subject to} \quad a_i + D_{ij} - m_{ij}\tau \le a_j \quad \forall (i,j) \in C_p && (2.3) \\ &\qquad L_{ij} \le a_j - a_i \le U_{ij} \quad \forall (i,j) \in C_e && (2.4) \\ &\qquad |(a_i - a_k) - (a_j - a_k)| \le I_{ij} \quad \forall (i,j) \in C_r && (2.5) \end{aligned}$$

where the constant α in the objective function can be chosen to adjust the tradeoff between minimizing wirelength and cycletime.

www.manaraa.com

Note that Equations (4), (5) can be rewritten into the same form as:

$$(a_i + L_{ij} \leq a_j) \wedge (a_j - U_{ij} \leq a_i) \tag{2.6}$$

$$(a_j - I_{ij} \leq a_i) \wedge (a_i - I_{ij} \leq a_j) \tag{2.7}$$

To make equations in Section III-B, Section III-C more concise, we combine Equations (3), (6), (7) and the primal problem can be rewritten as:

$$\begin{aligned} &\text{Minimize} \quad \text{HPWL}(\mathbf{x}, \mathbf{y}) + \alpha\tau \\ &\text{Subject to} \quad a_i + W_{ij} - \widetilde{m}_{ij}\tau \leq a_j \quad \forall(i,j) \end{aligned}$$

where $W_{ij} = L_{ij}$ or $-U_{ij}$ $\forall(i,j) \in C_e$, $W_{ij} = -I_{ij}$ $\forall(i,j) \in C_r$ and $W_{ij} = D_{ij}$ $\forall(i,j) \in C_p$. Similarly, $\widetilde{m}_{ij} = m_{ij}$ $\forall(i,j) \in C_p$ and $\widetilde{m}_{ij} = 0$ $\forall(i,j) \in C_e$ or C_r.

2.3.2 Lagrangian Relaxation

We relax all the constraints following the Lagrangian relaxation procedure. Nonnegative Lagrangian multiplier λ_{ij} is introduced for each constraint. Let $\boldsymbol{\lambda}$ be a vector of all the Lagrange multipliers.

$$\begin{aligned} \text{Let} \quad \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{a}, \tau) = &\ \text{HPWL}(\mathbf{x}, \mathbf{y}) + \alpha\tau \\ &+ \sum_{\forall(i,j)} \lambda_{ij}(a_i + W_{ij} - \widetilde{m}_{ij}\tau - a_j) \end{aligned}$$

Then the Lagrangian subproblem, which gives a lower bound for the primal problem for any $\boldsymbol{\lambda} \geq 0$ [69], can be formulated as:

$$\mathcal{LRS}: \quad \text{Mimimize} \quad \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{a}, \tau)$$

Let the function $q(\boldsymbol{\lambda})$ be the optimal value of the problem $\mathcal{LRS}$. We are interested in finding the values for the Lagrangian multipliers $\boldsymbol{\lambda}$ to give the maximum lower bound, which is labeled as the Lagrangian dual problem and defined as follows:

$$\begin{aligned} \mathcal{LDP}: \quad &\text{Maximize} \quad q(\boldsymbol{\lambda}) \\ &\text{Subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0} \end{aligned}$$

Solving $\mathcal{LDP}$ will provide a solution to the primal problem.

www.manaraa.com

2.3.3 Simplification of $\mathcal{LRS}$

Inspired by [14], we rearrange terms here and the Lagrangian function $\mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{a}, \tau)$ can be rewritten as:

$$\begin{aligned}\mathcal{L} = \mathrm{HPWL}(\mathbf{x}, \mathbf{y}) + (\alpha - \sum_{\forall(i,j)} \lambda_{ij}\widetilde{m}_{ij})\tau \\ + \sum_{k \in V}(\sum_{\forall(k,j)} \lambda_{kj} - \sum_{\forall(i,k)} \lambda_{ik})a_k \\ + \sum_{\forall(i,j)} \lambda_{ij} W_{ij}\end{aligned}$$

The KKT conditions imply $\partial\mathcal{L}/\partial a_i = 0$ for $1 \leq i \leq |v|$ and $\partial\mathcal{L}/\partial\tau = 0$ at the optimal solution of the primal problem. Then the optimality conditions $\mathcal{K}$ on $\boldsymbol{\lambda}$ can be obtained as:

$$\alpha = \sum_{\forall(i,j)} \lambda_{ij}\widetilde{m}_{ij}$$

$$\sum_{\forall(k,j)} \lambda_{kj} = \sum_{\forall(i,k)} \lambda_{ik} \quad \forall k \in V$$

Apply the optimality conditions into $\mathcal{LRS}$, we can obtain a simplified Lagrangian subproblem $\mathcal{LRS}^*$:

$$\text{Minimize} \quad \mathcal{L}^*(\mathbf{x}, \mathbf{y}) = \mathrm{HPWL}(\mathbf{x}, \mathbf{y}) + \sum_{\forall(i,j)} \lambda_{ij} W_{ij}$$

It can easily be seen that solving $\mathcal{LRS}$ is equivalent of solving $\mathcal{LRS}^*$.

2.3.4 Solving $\mathcal{LRS}^*$

Since it is impossible to get accurate timing without detailed placement and routing, as an approximation, we take the wire delay as being proportional to the HPWL of the hyperedge e associated with node i and j, which can be written as:

$$W_{ij} = d_i + HPWL_e \cdot \gamma_e$$

where d_i is the intrinsic gate delay and $HPWL_e \cdot \gamma_e$ is the total wire load delay. γ_e is a constant value associated with each edge and depends on the driver cell, load cells and electrical characterization for the wires.

www.manaraa.com

Note that for constraints (4) and (5), the corresponding W_{ij} is a constant value. For simplicity, we don't write them here explicitly. Then $\mathcal{LRS}^*$ can be written as:

$$\begin{aligned}
&\text{Minimize} \quad \mathcal{L}^*(\mathbf{x},\mathbf{y}) \\
&= \text{HPWL}(\mathbf{x},\mathbf{y}) + \sum_{\forall(i,j)} \lambda_{ij}(d_i + HPWL_e \cdot \gamma_e) \\
&\quad + \text{terms independent of } \mathbf{x},\ \mathbf{y} \\
&= \text{HPWL}(\mathbf{x},\mathbf{y}) + \sum_{\forall(i,j)} HPWL_e \cdot \lambda_{ij}\gamma_e \\
&\quad + \text{terms independent of } \mathbf{x},\ \mathbf{y}
\end{aligned}$$

Here the objective function only contains $\mathbf{x}$, $\mathbf{y}$ as variables. $\mathcal{LRS}^*$ becomes a weighted wirelength minimization problem for a set of hyperedges, which can be solved well by existing standard synchronous placement engine with the ability to weight nets.

2.3.5 Solving $\mathcal{LDP}$

Traditional approach of solving Lagrangian dual problem is to apply the subgradient optimization method [69]. However, this approach requires projection after updating $\boldsymbol{\lambda}$ in order to maintain $\boldsymbol{\lambda}$ within the dual feasible region. In addition, practical convergence of the subgradient optimization is difficult and usually requires a good choice of initial solution and step size. Here we apply a direction finding approach inspired by [83] to solve $\mathcal{LDP}$, which is shown to have better convergence compared with the traditional approach. Combining $\mathcal{LDP}$ defined in Section III-B with optimality condition $\mathcal{K}$ derived in Section III-C, we rewrite $\mathcal{LDP}$ here as:

$$\begin{aligned}
\mathcal{LDP}: \quad &\text{Maximize} \quad q(\boldsymbol{\lambda}) \\
&\text{Subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\lambda} \in \mathcal{K}
\end{aligned}$$

For any feasible $\boldsymbol{\lambda}$, we want to find an improving feasible direction $\Delta\boldsymbol{\lambda}$ and a step size β such that:

$$q(\boldsymbol{\lambda} + \beta\Delta\boldsymbol{\lambda}) > q(\boldsymbol{\lambda})$$

www.manaraa.com

Note that $\nabla \mathcal{L}^*{}_{\lambda_{ij}}(\mathbf{x}, \mathbf{y}) = W_{ij}\ \forall (i,j)$. Then an increasing feasible direction $\Delta\boldsymbol{\lambda}$ can be found by solving the following linear program $\mathcal{D}$:

$$\begin{aligned} \text{Maximize} \quad & \sum_{\forall(i,j)} \Delta\lambda_{ij} W_{ij} \\ \text{Subject to} \quad & \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\lambda} \in \mathcal{K} \\ & \max(-u, -\lambda_{ij}) \leq \Delta\lambda_{ij} \leq u \end{aligned}$$

where u is a constraint we introduced to bound the objective function, similar to [83].

Our algorithm to solve $\mathcal{LDP}$ is shown in Algorithm 1. It starts from an initial dual feasible $\boldsymbol{\lambda}$, then the method iteratively improves $q(\boldsymbol{\lambda})$ by finding an improving direction and performing a line search. The algorithm terminates when change of $q(\boldsymbol{\lambda})$ is small enough or the duality gap $\text{HPWL}(\mathbf{x}, \mathbf{y}) + \alpha\tau - q(\boldsymbol{\lambda})$ is less than *Error bound*.

Algorithm 1 Solve Lagrangian Dual Problem

Ensure: $\boldsymbol{\lambda}$ which maximizes $\mathcal{LRS}^*$

1: $n =: 1$; /* step counter */
2: $\boldsymbol{\lambda} =:$ initial positive value satisfy optimality condition $\mathcal{K}$;
3: Solve linear program $\mathcal{D}$ to obtain optimal increasing direction $\Delta\boldsymbol{\lambda}$;
4: Perform line search on $q(\boldsymbol{\lambda})$. Then a step size β which improves function value $q(\boldsymbol{\lambda} + \beta\Delta\boldsymbol{\lambda}) > q(\boldsymbol{\lambda})$ can be found. Terminate the algorithm if the change of $q(\boldsymbol{\lambda})$ is small enough;
5: Moving one step further by updating $\boldsymbol{\lambda} = \boldsymbol{\lambda} + \beta\Delta\boldsymbol{\lambda}$;
6: $n =: n + 1$;
7: Repeat Step 3-6 until $(\text{HPWL}(\mathbf{x}, \mathbf{y}) + \alpha\tau - Q(\boldsymbol{\lambda})) \leq$ *Error bound*;

2.4 Asynchronous Placement Flow for QDI Pipeline Templates

2.4.1 Asynchronous Design with Pre-Charged Half Buffer (PCHB) Templates

PCHB is a QDI template developed at Caltech [47], which designed with dual-rail asynchronous channels and 1-of-N handshaking protocol [9]. Fig. 2.3 shows a three stage PCHB pipeline structure with control circuit (CTRL), C-elements (C) and domino logic (FU) for computation.

Marked lines in Fig. 2.3 show an example of timing assumptions made by PCHB. In particular, it requires the input to the domino block go low before a rising transition on the control signal 'en' occurs. This timing assumption is a relaxed interpretation of the *isochronic*

المنارة للاستشارات
www.manaraa.com

fork assumption [66] and can easily be met without special care. We ignore this timing constraint at global placement stage and leave it to be checked after detailed placement and routing, similar to [6] and [78].

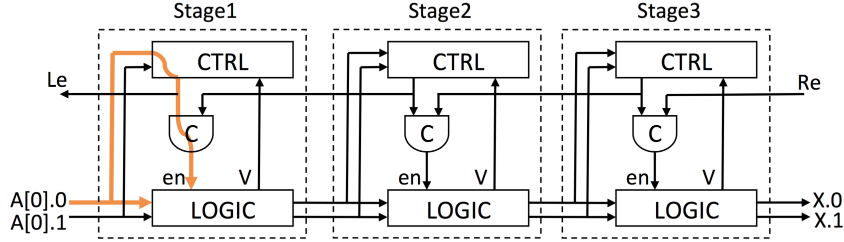

Figure 2.3: PCHB pipeline template.

2.4.2 Asynchronous Placement Flow

Fig. 2.4 illustrates our placement flow for QDI PCHB pipeline templates. We use a state-of-art quadratic placer POLAR [46] as the placement engine. Initially, pure wirelength-driven placement is performed on the asynchronous circuit, as a good starting placement with minimized wirelength is necessary in order to achieve better cycletime in later stage.

After initial placement, we calculate the total wirelength and initial cycletime to set parameter $\alpha = k \cdot \text{HPWL}(\mathbf{x}, \mathbf{y})/\tau$. k is normally set to 1 in order to achieve a good balance between total wirelength and cycletime. We first extract the performance constraints based on our marked graph modeling of the original circuit. Then, a linear program similar to $\mathcal{D}$ is solved using Gurobi [1] to obtain an initial non-negative $\boldsymbol{\lambda}$ satisfy $\mathcal{K}$. Next, net weights as derived in Section II-C are added into the hypergraph which is used to guide POLAR solving $\mathcal{LRS}^*$. At this point, we enter the timing optimization stage of our flow. We start the loop of solving $\mathcal{LDP}$ by iteratively solving the direction finding problem with Gurobi and weighted wirelength minimization problem with POLAR to find a direction and step size which increase the objective value $q(\boldsymbol{\lambda})$, until we achieve a small duality gap or the improvement on the objective value is tiny. The solution to $\mathcal{LDP}$ is a placement with optimized timing which will be the output of our flow. Note that the constraints we have for PCHB will not introduce negative edge weights. Thus, it can be handled well by quadratic placer POLAR. For other asynchronous design style which has different timing constraints and negative weight edge exists, we need to choose a

www.manaraa.com

non-linear type placement engine to solve $\mathcal{LRS}^*$. Finally, we export our design into Encounter to perform legalization and routing.

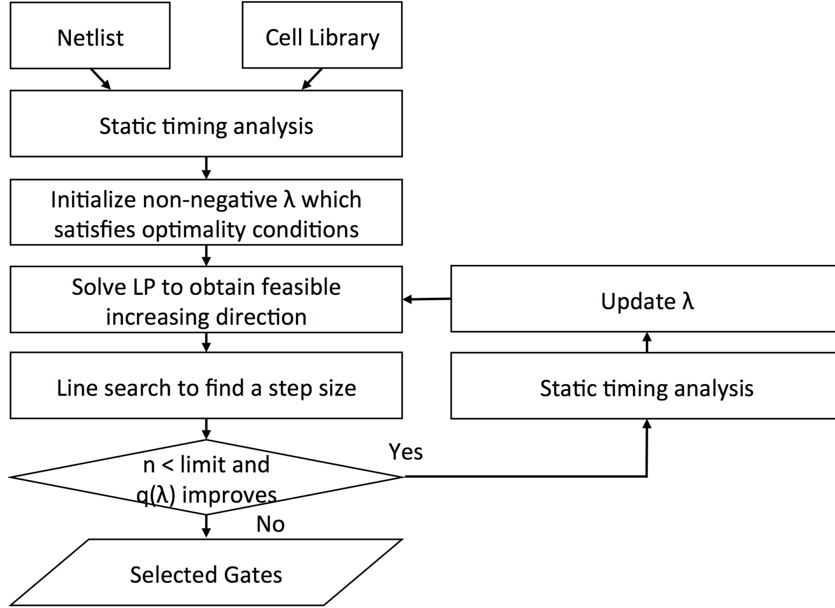

Figure 2.4: QDI PCHB placement flow.

2.5 Experimental Results

The proposed approach is implemented using C++. All experiments were run on a Liunx PC with 47GB of memory and Intel Core-i3 3.3GHz CPU.

We run our flow on two sets of benchmarks. First is a set of ISCAS89 benchmark circuits which are converted to unconditional asynchronous circuits using Proteus's legacy RTL design flow [6]. In particular, flip-flops from ISCAS89 are mapped as token buffers and combinational gates are mapped as logic cells in PCHB cell library. In addition, we developed several benchmarks in RTL level and synthesized it using Proteus. For ALU and Accumulator design, we choose different bit width for the datapath to create a set of benchmarks with different sizes.

We compared our flow against both Encounter and the Proteus placement flow. The die size and I/O pin locations are set to be the same for all three flows. In our case, Encounter is used as a pure wirelength-driven placer without any input timing constraints. For Proteus flow, we disabled the gate resizing at its placement stage to avoid the changing of input netlists.

المنارة للاستشارات
www.manaraa.com

Table 2.1: Comparison on asynchronous benchmarks

Design	Size	Routed Wirelength x 10^6 (nm)			Cycletime (ns)			Runtime (s)		
		POLAR	Proteus	Encounter	POLAR	Proteus	Encounter	POLAR	Proteus	Encounter
s444	256	10.58	11.32	10.46	4.67	5.72	6.06	36	245	8
s510	519	33.10	35.38	32.84	6.87	7.01	7.25	37	330	12
s526	307	13.30	14.85	14.24	3.96	4.83	4.20	34	257	8
s526a	297	13.34	13.79	12.22	4.14	3.75	4.89	33	249	8
s641	636	22.83	26.96	22.02	4.41	4.88	5.00	36	313	10
s713	584	21.42	24.38	21.06	5.72	5.71	6.15	40	228	9
s820	681	44.30	48.53	43.39	7.10	8.34	10.40	41	431	13
s832	706	46.15	53.04	45.84	6.19	7.32	6.82	40	465	15
s838	707	38.65	37.49	33.32	4.88	5.58	5.91	37	337	13
s953	931	64.01	71.85	61.84	6.90	7.10	6.12	41	576	18
s1488	1314	124.88	137.06	130.13	11.80	11.35	12.79	49	771	27
s1423	1119	63.21	71.25	64.48	8.47	8.37	14.85	39	692	20
s9234	2108	120.19	134.48	119.40	6.76	8.19	9.83	51	517	31
s13207	5658	381.02	386.13	338.18	11.17	11.86	13.72	82	1202	67
s38417	15447	1310.20	1208.16	1253.42	42.44	68.30	80.43	298	1050	283
ALU4	413	16.92	18.78	18.01	5.22	5.99	5.68	44	261	10
ALU8	916	50.31	53.89	55.03	4.44	5.11	8.43	41	470	16
acc32	1187	65.50	59.87	49.09	5.46	5.57	6.45	41	528	17
acc64	3355	161.67	145.88	138.06	5.15	7.37	10.01	64	757	39
GCD	1505	23.55	24.17	23.32	18.63	20.68	20.05	45	604	21
FetchingUnit	5304	435.78	453.41	396.16	9.31	7.81	16.06	104	958	62
Average		1.00	1.06	0.97	1.00	1.12	1.32	1.00	10.34	0.43

المنارة للاستشارات
www.manaraa.com

Our experimental results are shown in Table 2.1. The size column shows the number of cells in each design. The wirelength column shows the detailed routed wirelength reported by Encounter. All the designs are shown to be routable. The cycletime column shows the cycletime calculated using the linear program introduced in Section II-A for the final routed design using our delay model.

We show significant improvement in cycletime compared with the non-timing driven Encounter placement and Proteus placement flow. For all the benchmarks, we achieved an average improvement in cycletime of 12% over the Proteus placement flow and 32% over the results of non-timing driven placement by Encounter. The wirelength of our approach is 3% worse compared with Encounter, which is expected considering the extensive timing optimization that has been performed and 6% better than Proteus placement flow. In addition, our runtime is also shown to be much more scalable in comparison with the Proteus approach.

2.6 Conclusion

In this paper, we have proposed a timing-driven placement approach targeting asynchronous circuits. Our problem formulation only introduce polynomial number of performance constraints which is more efficient and effective than the approaches using loop-breaking techniques or enforcing explicit cycle constraints. The flexibility of our Lagrangian relaxation framework also makes our framework applicable to a wide range of asynchronous design styles. In addition, we simplified the timing-driven placement problem into a weighted wirelength minimization problem which can be solved by standard placement algorithms with the ability to handle net weights. We implemented a placement flow with a quadratic placer to demonstrate our idea. The experimental results shows our approach can greatly improve the performance for a given asynchronous circuits at the placement stage. The runtime and placement quality is also shown to be much better than the previous state-of-the-art.

المنارة للاستشارات
www.manaraa.com

CHAPTER 3. TIMING-DRIVEN PLACEMENT BY LAGRANGIAN RELAXATION

In this work, we propose Lagrangian relaxation based algorithms to optimize both circuit performance and total wirelength at the global placement stage. We introduce a general timing-driven global placement problem formulation that is applicable to three different circuit design styles: synchronous circuits, synchronous circuits with sequential optimization techniques and asynchronous circuits. Lagrangian relaxation is applied to handle the timing constraints of the formulated problem. Based on how the cell spreading constraints are handled, two different approaches are proposed: One approach handles the spreading constraints inside the Lagrangian relaxation framework and transforms the timing-driven placement problem into a series of weighted wirelength minimization problems, which can be solved by directly leveraging existing wirelength-driven placers. The other approach handles the spreading constraints outside the Lagrangian relaxation framework. Thus, only timing constrains need to be taken care of in the Lagrangian relaxation framework and better solutions can be expected. In both approaches, we simplified the Lagrangian relaxation subproblem using Karush-Kuhn-Tucker conditions. Our algorithms are implemented based on a state-of-the-art wirelength-dirven quadratic placer. The experiments demonstrate that the proposed algorithms are able to achieve significant improvements on circuit performance compared with a commercial wirelength-driven placement flow and a commercial asynchronous timing-driven placement flow.

3.1 Introduction

Placement is a critical step in VLSI design flow, as the placement quality and optimization metrics can greatly affect the performance, routability, heat distribution and power consump-

www.manaraa.com

tion of a design [85]. In advanced technology, the importance of placement continues to grow, since it determines the interconnect delay, which has been a dominating factor of the circuit delay.

Traditional wirelength-driven placement algorithms only consider minimizing the total chip wirelength and do not take into account circuit timing during the optimization process. As a result, the wirelength-driven placement algorithms are no longer sufficient to close timing at modern technology nodes, which often have stricter timing constraints. Therefore, the quests for timing-driven placement (TDP) algorithms start to receive closer attention.

Timing-driven placement problem has been extensively studied for decades. One category of TDP algorithms optimizes the circuit timing by capturing the timing criticality of each net through net weighting (e.g., [12] [24]) or net constraints (e.g.,[28]). These algorithms are often referred to as net based algorithms. However, net based approaches only estimate the circuit timing locally and do not have a global view on the entire timing paths. Another category of TDP algorithms directly work on a set of critical timing paths and ensure all considered timing paths meet the constraints. This category is often referred to as path based algorithms (e.g., [36]). To avoid explicitly considering all the timing paths, the number of which can be exponential to the circuit size, timing graph based approaches embed the timing graph into the formulation of timing-driven placement problem, and all topological timing paths can then be implicitly considered [62].

Depending on the specific circuit design style, the corresponding TDP problem can target at optimizing either the most critical path between registers (as in synchronous circuits) or the most critical cycle (as in asynchronous circuits [88][37] or synchronous circuits with sequential optimization techniques [34]). In addition, timing-driven placement can be performed at the global placement stage, detailed placement stage, or both. Normally, at the global placement stage, cells are placed to improve circuit performance based on a rough timing estimation, with a small amount of cell overlapping allowed. At the detailed placement stage, cells are moved to legal locations either respecting the global placement results [34] or applying certain techniques to further improve the timing of the circuit [68].

المنارة للاستشارات
www.manaraa.com

Lagrangian relaxation (LR) is a popular approach people used for timing-driven placement due to several important reasons: First, it relaxes the complex timing constraints in the formulated problem and results in a LR subproblem which is much easier to solve [69]. Second, the special circuit structure allows Karush-Kuhn-Tucker (KKT) conditions to be applied to further simplify the LR subproblem to get rid of its arrival time and cycle time variables [14]. Finally, Lagrangian relaxation has great flexibility and is capable of handling various objectives and complex design constraints. LR-based algorithms have shown to be successful in handing timing constraints in many previous works. In [73][31], Lagrangian relaxation is combined with the wirelength-driven placer GORDIAN [41] to solve the timing-driven placement problem for synchronous circuits. The relaxed LR-subproblem is either solved as a quadratic program [73] or through the resistance network approach [31]. In [34], Lagrangian relaxation is used as a refinement step after global placement, in order to improve the circuit performance for synchronous circuits with sequential optimization techniques.

Apart from the complex timing constraints of the timing-driven placement problem, we also need to consider its cell spreading constraints, which can be even more difficult to handle. One reason is that these constraints are often non-convex and not differentiable. Thus, it is not straightforward to solve them mathematically. Another reason is that modern placement techniques often develop some heuristic algorithms to spread out the cells and do not have an exact formulation of the cell spreading constraints [39][46]. This makes it unclear how to incorporate these techniques for the timing-driven placement problem.

In this paper, we apply Lagrangian relaxation to handle the timing constraints, while also explore different approaches to incorporate the cell spreading techniques with the LR framework. In particular, we formulate a general timing-driven global placement problem which is applicable to synchronous circuits, synchronous circuits with sequential optimization techniques, and asynchronous circuits. We propose two different approaches for handling the cell spreading constraints: One approach handles the spreading constraints inside the LR framework and the LR subproblem becomes a weighted wirelength minimization problem, which can be solved effectively using existing wirelength-driven placers with the ability to handle net weights. The other approach spreads the cells outside the LR framework and the resulting LR

المنارة للاستشارات
www.manaraa.com

subproblem becomes an unconstrained optimization problem which can be solved optimally using standard mathematical techniques.

The proposed approaches are implemented based on the state-of-art quadratic placer POLAR [46]. We evaluated both of our approaches on quasi-delay-insensitive (QDI) Pre-Charged Half Buffer (PCHB) asynchronous designs synthesized using the Proteus asynchronous synthesis flow [6]. The experimental results of both approaches are compared with a commercial wirelength-driven placement flow and a commercial timing-driven placement flow.

The main contributions of this paper are as follows:

- A general problem formulation for the timing-driven placement problem is proposed which can be applied to a large variety of design styles.
- Two different approaches of applying Lagrangian relaxation to the formulated problem are presented.
- For both approaches, better computational efficiency is achieved by simplifying the LR subproblem using the KKT conditions.
- An effective approach to handle timing-driven placement at the detailed placement stage is proposed.
- Promising experimental results are presented.

The rest of this paper is organized as follows. Section II presents the problem formulations for various design styles. Section III elaborates two different approaches we used to apply Lagrangian relaxation framework to the formulated timing-driven placement problem. Section IV presents the detailed implementation of the proposed approaches. Section V shows our experimental results compared with other placement approaches. Finally, Section VI concludes the paper.

3.2 Problem Formulations

In Section II-A, we will first discuss the problem formulation for the pure wirelength-driven placement problem. Next, in Section II-B, we introduce three different design styles:

www.manaraa.com

synchronous circuits, synchronous circuits with sequential optimization techniques, and asynchronous circuits. Their corresponding formulations for the timing-driven placement problem will also be presented. Finally, in Section II-C, we summarize all the design styles and propose a general problem formulation covering all of them.

A circuit can be represented by a hypergraph $G = (V, E)$ where $V = \left\{v_1, v_2, \ldots, v_{|V|}\right\}$ corresponds to the set of cells, and $E = \left\{e_1, e_2, \ldots, e_{|E|}\right\}$ corresponds to the set of nets. In addition, we use vector $\mathbf{x} = \left(x_1, x_2, \cdots, x_{|V|}\right)$ to denote the x-coordinates of the cells and vector $\mathbf{y} = \left(y_1, y_2, \cdots, y_{|V|}\right)$ to denote the y-coordinates of the cells.

The wirelength of a hyperedge e depends on the locations of the cells associated to it. Thus, we use $\mathrm{WL}_e(\mathbf{x}, \mathbf{y})$ to denote the wirelength of a hyperedge e. The definition of this wirelength function is ignored at this point to make our problem formulation general. In practice, the wirelength function can be modeled as HPWL, quadratic, Log-Sum-Exp function, etc. [85].

Some notations used in this paper are shown in Table I.

Table I. The key notations used in this paper.

$\mathcal{WDP}$	wirelength-driven placement
$\mathcal{TDP}$	timing-driven placement
$\mathcal{STDP}$	synchronous timing-driven placement
$\mathcal{STDPS}$	synchronous timing-driven placement with sequential optimization techniques
$\mathcal{ATDP}$	asynchronous timing-driven placement
$\mathcal{GTDP}$	general timing-driven placement
$\mathcal{LRS}$	Lagrangian relaxation subproblem
$\mathcal{LRS}$-$\mathcal{S}$	simplified Lagrangian relaxation subproblem
$\mathcal{LDP}$	Lagrangian dual problem
a_i	arrival time variable at node i
D_{ij}	delay value associated with edge (i, j)
λ_{ij}	Lagrange multiplier associated with edge (i, j)
τ	cycletime variable
$q(\boldsymbol{\lambda})$	optimal value of $\mathcal{LRS}$ for a given vector $\boldsymbol{\lambda}$
$\mathcal{L}_{\boldsymbol{\lambda}}(\mathbf{x}, \mathbf{y}, \mathbf{a}, \tau)$	objective function of $\mathcal{LRS}$
$\mathcal{L}_{\boldsymbol{\lambda}}(\mathbf{x}, \mathbf{y})$	objective function of $\mathcal{LRS}$-$\mathcal{S}$

المنارة للاستشارات
www.manaraa.com

3.2.1 Wirelength-driven Placement Problem ($\mathcal{WDP}$)

The wirelength-driven global placement tries to minimize the total chip wirelength by assigning cells to locations on the chip, while keeping the cells spread out. Therefore, the wirelength-driven placement problem can be formulated as:

$$\mathcal{WDP}: \quad \text{Minimize} \quad \sum_{e \in E} \mathrm{WL}_e(\mathbf{x}, \mathbf{y})$$
$$\text{Subject to} \quad \text{cell spreading constraints}$$

Here, we also skip the details about the cell spreading constraints to make our problem formulation general. Various techniques can be practically used to implement the cell spreading constraints, such as the center-of-gravity (COG) constraints [41], spreading forces [24], density penalty functions [56], etc.

3.2.2 Timing-driven Placement Problems

To further capture the timing information of the circuit, we introduce a timing graph $G' = (V, E')$, where E' denotes the set of timing edges. In particular, we use V_I to denote the set of vertices representing the starting points of timing paths, i.e., the output pins of registers or the primary inputs. Similarly, we use V_O to denote the set of vertices representing the timing path end points, which are the input pins of registers or the primary outputs. In addition, let $AT = \{a_1, a_2, \ldots, a_{|V|}\}$ be the set of arrival time variables associated with each node. Let τ denotes the minimum cycle time to ensure hazard-free operation of the circuit. Also, we denote the delay associated with the edge between node v_i and v_j in the timing graph as D_{ij}, whose value depends on the interconnection between v_i and v_j and hence depends on the placement.

3.2.2.1 Synchronous timing-driven placement ($\mathcal{STDP}$)

The minimum cycle time for synchronous circuits is bounded by the delay of its longest timing path. However, the total number of timing paths is exponential to the circuit size. Therefore, instead of explicitly considering all timing paths, we capture the circuit timing

المنارة للاستشارات
www.manaraa.com

using the set of worst case arrival times, which can be calculated by propagating the largest arrival time at each node:

$$a_i + D_{ij} \leq a_j \quad \forall (i,j) \in E'$$

Then, the synchronous timing-driven placement problem, which simultaneously minimizes the total wirelength and cycle time can be formulated as:

$$\mathcal{STDP}: \quad \text{Minimize} \quad \sum_{e \in E} \mathrm{WL}_e(\mathbf{x}, \mathbf{y}) + \alpha\tau$$

$$\text{Subject to} \quad a_i + D_{ij} \leq a_j \quad \forall (i,j) \in E' \tag{3.1}$$

$$a_k \leq \tau \qquad \forall k \in V_O \tag{3.2}$$

$$a_k \geq W_k \quad \forall k \in V_I \tag{3.3}$$

$$\text{cell spreading constraints}$$

Here, α is a constant value which we can use to adjust the effort between optimizing wirelength and cycle time. W_k denotes the constant delay value that signal arrives at the primary inputs or the output of registers. Please note that we ignore the hold time violations in the formulation of $\mathcal{STDP}$, as designers normally only consider the longest timing paths at the global placement stage. The shortest paths causing hold time violations are fixed at a later stage, i.e., after detailed placement and routing.

3.2.2.2 Synchronous timing-driven placement with sequential optimization techniques ($\mathcal{STDPS}$)

Retiming [44] [45] and clock skew scheduling [25] are two commonly used sequential optimization methods. Retiming improves the circuit performance through changing the structural location of registers. Instead, clock skew scheduling preserves the circuit structure, while intentionally introduces skews to registers to improve the performance of a circuit.

Let c denote a timing loop composed by a set of timing path segments. The basic idea for both of the above two sequential optimization methods is to perform a coarse balancing on the timing budgets of the path segments along the timing loop.

المنارة للاستشارات
www.manaraa.com

The optimization potential of these methods is bounded by the maximum mean delay over all timing loops:

$$\tau \geq \max_{c \subset G'} \left\{ \frac{\sum_{(i,j)\in c} D_{ij}}{\text{\# of registers in } c} \right\}$$

Instead of enumerating all the timing loops whose number is exponential to circuit size, we can simply obtain the cycle time by solving the following linear program [49]:

$$\begin{aligned} &\text{Minimize} \quad \tau \\ &\text{Subject to} \quad a_i + D_{ij} - m_{ij}\tau \leq a_j \quad \forall (i,j) \in E' \end{aligned}$$

where $m_{ij} = 1$ if the corresponding edge is a fanout edge of node $v \in V_I$, and $m_{ij} = 0$ otherwise.

Accordingly, to increase the optimization potential of such sequential methods, we should target improving the maximum mean cycle delay during the placement stage. Thus, the synchronous timing-driven placement problem with sequential optimization techniques is formulated as:

$$\begin{aligned} \mathcal{STDPS} : \text{Minimize} \quad & \sum_{e \in E} \text{WL}_e(\mathbf{x}, \mathbf{y}) + \alpha\tau \\ \text{Subject to} \quad & a_i + D_{ij} - m_{ij}\tau \leq a_j \ \forall (i,j) \in E' \\ & \text{cell spreading constraints} \end{aligned}$$

3.2.2.3 Asynchronous timing-driven placement ($\mathcal{ATDP}$)

Instead of governing the circuit using global clock signals, an asynchronous circuit only synchronizes neighboring stages through the handshaking signals [9]. Similar to synchronous circuits with sequential optimization techniques, the performance of asynchronous circuits is also bounded by the maximum mean cycle delay, while the difference is that the average-case performance for asynchronous circuits is achieved naturally without needing extra optimization techniques.

Depending on the timing assumptions made by the specific logic implementation style, different types of timing constraints need to be satisfied for asynchronous circuits. Except for delay-insensitive (DI) designs [79] which are premised on the fact that they will function

المنارة للاستشارات
www.manaraa.com

correctly regardless of the delays of the gates and the wires, timing constraints for other asynchronous designs fall into two categories [9].

First are explicit timing constraints in the form of minimum and maximum bounded delay values for gates and wires in the circuit. An example is the bounded-delay asynchronous circuits in [76].

Let U_{ij} be the maximum bounded delay and L_{ij} be the minimum bounded delay between nodes v_i and v_j. Let E_e be the set of node pairs which we need to enforce explicit delay bounds. The explicit timing constraints can be written as:

$$L_{ij} \leq a_j - a_i \leq U_{ij} \quad \forall (i,j) \in E_e \tag{3.4}$$

Second are relative timing constraints, referred to as *relative timing* [74], which dictate the relative delay of two paths that stem from a common point of divergence. Example design styles that have relative timing constraints include the quasi-delay-insensitive (QDI) design style, such as WCHB, PCHB and the Multi-Level Domino (MLD) template [9].

For a relative timing constraint from a node v_k and forking into two nodes v_i and v_j, constraints can be written as:

$$|(a_i - a_k) - (a_j - a_k)| \leq I_{ij} \quad \forall (i,j) \in E_r \tag{3.5}$$

This bounds the maximum difference in time that the signals arrive at the two end-points of the fork. This type of constraint captures the notion of an *isochronic fork* [9], a common type of constraint in quasi-delay-insensitive designs. Here I_{ij} is the delay bound for *isochronic fork*. E_r is the set of node pairs which have relative timing constraints.

Combining everything together, the asynchronous timing-driven placement problem, which minimizes both total wirelength and cycle time subject to timing constraints (4), (5) can be formulated directly as:

المنارة للاستشارات

www.manaraa.com

$\mathcal{ATDP}$:

$$\text{Minimize} \quad \sum_{e \in E} \text{WL}_e(\mathbf{x}, \mathbf{y}) + \alpha\tau$$

$$\text{Subject to} \quad a_i + D_{ij} - m_{ij}\tau \leq a_j \quad \forall (i,j) \in E' \tag{3.6}$$

$$L_{ij} \leq a_j - a_i \leq U_{ij} \qquad \forall (i,j) \in E_e \tag{3.7}$$

$$|(a_i - a_k) - (a_j - a_k)| \leq I_{ij} \qquad \forall (i,j) \in E_r \tag{3.8}$$

cell spreading constraints

where $m_{ij} = 1$ if the corresponding edge is a fanout edge of a token buffer, and $m_{ij} = 0$ otherwise.

3.2.3 A General Timing-driven Placement Problem ($\mathcal{GTDP}$)

In this section, we show that all the three different types of problems we presented in Sec. II-B can be generalized to the timing-driven placement problem shown as follows:

$$\mathcal{GTDP} : \text{Minimize} \quad \sum_{e \in E} \text{WL}_e(\mathbf{x}, \mathbf{y}) + \alpha\tau$$

$$\text{Subject to} \quad a_i + \hat{D}_{ij} - \hat{m}_{ij}\tau \leq a_j \quad \forall (i,j) \tag{3.9}$$

cell spreading constraints

Here, $\hat{D}_{ij}$ captures the wire delay after we combine everything together. For $\mathcal{STDP}$, $\hat{m}_{ij}$ is always equal to 0. For $\mathcal{STDPS}$ or $\mathcal{ATDP}$, $\hat{m}_{ij} = 1$ when the corresponding edge is a fanout edge of a register or a token buffer, and $\hat{m}_{ij} = 0$ otherwise.

It is obvious that $\mathcal{STDPS}$ is directly equivalent to the formulation of $\mathcal{GTDP}$. Next, we show how $\mathcal{STDP}$ and $\mathcal{ATDP}$ can also be reduced to this form.

المنارة للاستشارات
www.manaraa.com

3.2.3.1 Transform $\mathcal{STDP}$ to $\mathcal{GTDP}$

We add two new nodes into the timing graph: v_s and v_t and let their corresponding arrival times to be a_s and a_t. In addition, we add the set of edges $(s, v_i)\ \forall v_i \in V_I$ and $(v_j, t)\ \forall v_j \in V_O$ into the timing graph, with the edge delay $D_{sv_i} = W_i$ and $D_{v_jt} = 0$. Finally, we add edge (v_t, v_s) into the timing graph with $D_{v_tv_s} = -\tau$. The new timing graph after this modification is shown in Fig. 6.1.

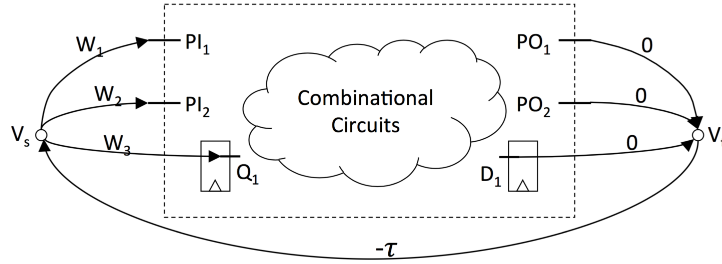

Figure 3.1: Modified timing graph for $\mathcal{STDP}$.

After this graph transformation, timing constraints (2) and (3) can be rewritten as follows:

$$a_s + W_i \leq a_i \qquad \forall i \in V_O \tag{3.10}$$

$$a_j + 0 \leq a_t \qquad \forall j \in V_I \tag{3.11}$$

$$a_t - \tau \leq a_s \tag{3.12}$$

The new constraints (10) (11) (12) can easily fit into the constraints (9) of $\mathcal{GTDP}$. Then, $\mathcal{STDP}$ can be transformed to $\mathcal{GTDP}$ by combining constraints (1) (10) (11) (12) into constraints (9).

3.2.3.2 Transform $\mathcal{ATDP}$ to $\mathcal{GTDP}$

We can rewrite the timing constraints in Equations (7) and (8) into the same form as Equation (9) as follows:

$$(a_i + L_{ij} \leq a_j) \wedge (a_j - U_{ij} \leq a_i) \tag{3.13}$$

$$(a_j - I_{ij} \leq a_i) \wedge (a_i - I_{ij} \leq a_j) \tag{3.14}$$

Then, combining Equation (6) with the reformulated Equations (13) and (14), we can easily transform $\mathcal{ATDP}$ to $\mathcal{GTDP}$.

www.manaraa.com

3.3 Our Proposed Approaches on Solving $\mathcal{GTDP}$

It is difficult to directly solve $\mathcal{GTDP}$, due to its complex timing constraints (9) and cell spreading constraints. In this section, we discuss how we handle the $\mathcal{GTDP}$ problem and propose two different approaches which can solve $\mathcal{GTDP}$ effectively. For both approaches, we use Lagrangian relaxation to relax the timing constraints of $\mathcal{GTDP}$, while the difference is how we handle the spreading constraints. In particular, one approach, as we will present in Section III-A, handles the spreading constraints inside the LR framework during the LR subproblem. We refer this approach as the spreading-inside approach. The other approach, which we will present in Section III-B, handles the cell spreading outside the Lagrangian relaxation framework, while only takes care of the timing constraints within the LR framework. We refer this approach as the spreading-outside approach.

3.3.1 Spreading-inside Approach

The spreading-inside approach extends one of our previous works in [88]. An overview of the spreading-inside approach is shown in Fig. 6.2(a). To make things clear, we highlighted the cell spreading step, which is inside the LR framework denoted as the red dotted box. In the beginning, we relax all the timing constraints of $\mathcal{GTDP}$ and initialize a vector of $\boldsymbol{\lambda}$ satisfying the KKT conditions. The relaxed LR subproblem is denoted as $\mathcal{LRS}$, which still contains the spreading constraints. Next, at each iteration, instead of directly solving $\mathcal{LRS}$, we explore the special structure of $\mathcal{GTDP}$ and solve an equvalent yet simpler version $\mathcal{LRS}$-$\mathcal{S}$ of the subproblem. In particular, $\mathcal{LRS}$-$\mathcal{S}$ is a weighted wirelength minimization problem that can be solved by a standard wirelength-driven placer. Typically, the wirelength-driven placer incorporates the spreading constraints into the objective function of $\mathcal{LRS}$-$\mathcal{S}$ and solve it as an unconstrained optimization problem. After the $\mathcal{LRS}$-$\mathcal{S}$ is solved, we update the vector of $\boldsymbol{\lambda}$ using any standard method. The Lagrangian relaxation loop terminates when there is no improvement in the objective function or the runtime limit is exceeded.

المنارة للاستشارات
www.manaraa.com

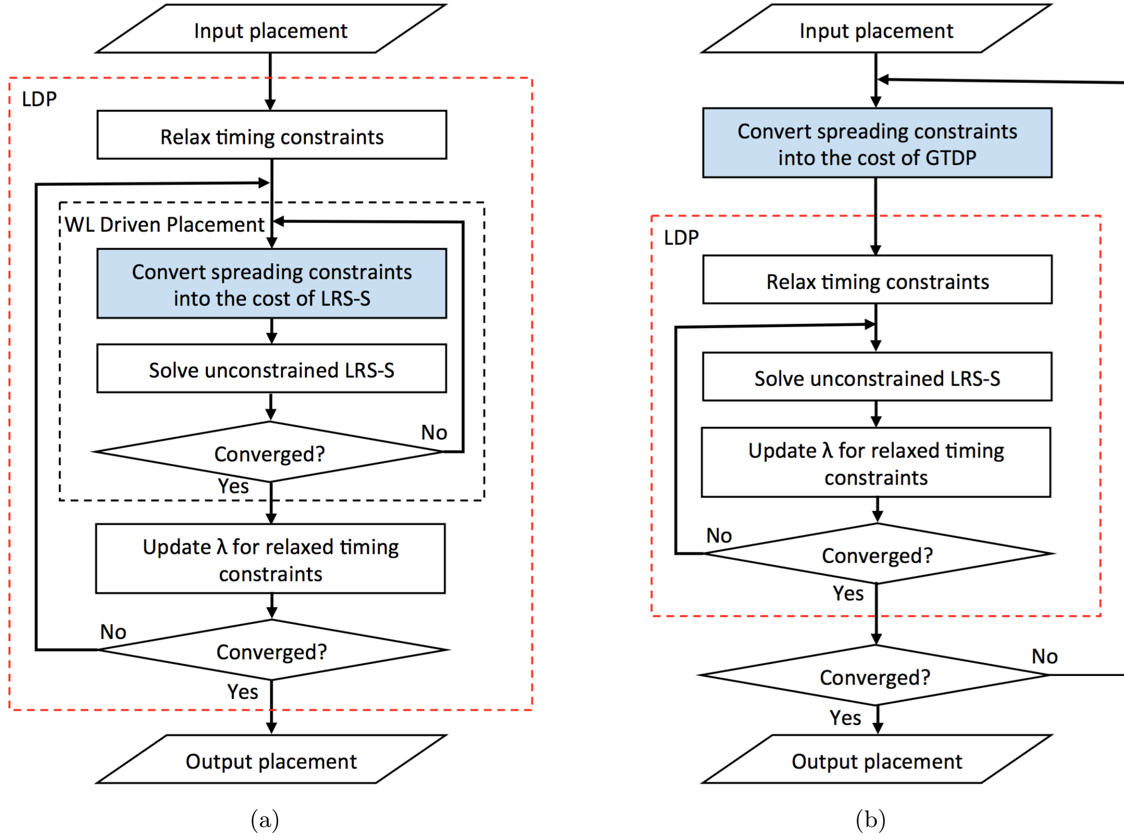

Figure 3.2: (a) The spreading-inside approach. (b) The spreading-outside approach.

3.3.1.1 Lagrangian Relaxation Subproblem ($\mathcal{LRS}$)

We relax the timing constraints of $\mathcal{GTDP}$ following the Lagrangian relaxation procedure and introduce a nonnegative Lagrange multiplier λ_{ij} for each timing constraint. Let $\boldsymbol{\lambda}$ be a vector of all the Lagrange multipliers.

$$\text{Let} \quad \mathcal{L}_{\boldsymbol{\lambda}}(\mathbf{x}, \mathbf{y}, \mathbf{a}, \tau) = \sum_{e \in E} \mathrm{WL}_e(\mathbf{x}, \mathbf{y}) + \alpha\tau$$
$$+ \sum_{\forall (i,j)} \lambda_{ij}(a_i + \hat{D}_{ij} - \hat{m}_{ij}\tau - a_j)$$

Then the LR subproblem, which gives a lower bound for $\mathcal{GTDP}$ for any $\boldsymbol{\lambda} \geq \mathbf{0}$ [69], can be formulated as:

$$\mathcal{LRS}: \quad \text{Mimimize} \quad \mathcal{L}_{\boldsymbol{\lambda}}(\mathbf{x}, \mathbf{y}, \mathbf{a}, \tau)$$
$$\text{Subject to} \quad \text{cell spreading constraints}$$

المنارة للاستشارات
www.manaraa.com

3.3.1.2 Simplified LR Subproblem ($\mathcal{LRS}$-$\mathcal{S}$)

Inspired by [14], we rearrange the terms here and the Lagrangian function $\mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{a}, \tau)$ can be rewritten as:

$$\begin{aligned}\mathcal{L} = \sum_{e\in E} \mathrm{WL}_e(\mathbf{x}, \mathbf{y}) + (\alpha - \sum_{\forall(i,j)} \lambda_{ij}\hat{m}_{ij})\tau \\ + \sum_{k\in V}(\sum_{\forall(k,j)} \lambda_{kj} - \sum_{\forall(i,k)} \lambda_{ik})a_k \\ + \sum_{\forall(i,j)} \lambda_{ij}\hat{D}_{ij}\end{aligned}$$

The KKT conditions imply $\partial\mathcal{L}/\partial a_i = 0$ for $1 \leq i \leq |V|$ and $\partial\mathcal{L}/\partial\tau = 0$ at the optimal solution of the primal problem. Then the optimality conditions $\mathcal{K}$ on $\boldsymbol{\lambda}$ can be obtained as:

$$\alpha = \sum_{\forall(i,j)} \lambda_{ij}\hat{m}_{ij}$$

$$\sum_{\forall(k,j)} \lambda_{kj} = \sum_{\forall(i,k)} \lambda_{ik} \quad \forall k \in V$$

Apply the optimality conditions into $\mathcal{LRS}$, we can obtain a simplified Lagrangian relaxation subproblem $\mathcal{LRS}$-$\mathcal{S}$:

$\mathcal{LRS}$-$\mathcal{S}$:

$$\text{Minimize} \quad \mathcal{L}_{\boldsymbol{\lambda}}(\mathbf{x}, \mathbf{y}) = \sum_{e\in E} \mathrm{WL}_e(\mathbf{x}, \mathbf{y}) + \sum_{\forall(i,j)} \lambda_{ij}\hat{D}_{ij}$$

Subject to cell spreading constraints

It can be shown that solving $\mathcal{LRS}$ is equivalent to solving $\mathcal{LRS}$-$\mathcal{S}$.

3.3.1.3 Lagrangian Dual Problem ($\mathcal{LDP}$)

Let the function $q(\boldsymbol{\lambda})$ be the optimal value of the problem $\mathcal{LRS}$. We are interested in finding the values for the Lagrange multipliers $\boldsymbol{\lambda}$ to give the maximum lower bound of $\mathcal{GTDP}$. This problem is called the Lagrangian dual problem and is defined as follows. Solving $\mathcal{LDP}$ will provide a solution to the primal problem.

$\mathcal{LDP}$: Maximize $q(\boldsymbol{\lambda})$

Subject to the optimality conditions $\mathcal{K}$ on $\boldsymbol{\lambda}$

www.manaraa.com

3.3.1.4 Solving $\mathcal{LRS}$-$\mathcal{S}$

The detailed timing model is irrelevant to our problem formulation and transformation presented in the previous sections, but it is required when we start to discuss how to solve these problems. Therefore, in this subsection, we first present the timing model used in this paper.

Since detailed placement and routing are not performed yet, it will be wasteful and time consuming to use an accurate delay model during the global placement stage. Thus, as an approximation, we use a linear delay model which sets the wire delay $\hat{D}_{ij}$ to be proportional to the wirelength of the hyperedge e associated with nodes i and j:

$$\hat{D}_{ij} = d_i + WL_e(\mathbf{x}, \mathbf{y}) \cdot \gamma_e$$

where d_i is the intrinsic gate delay and $WL_e(\mathbf{x}, \mathbf{y}) \cdot \gamma_e$ is the total wire load delay. γ_e is a constant value associated with each edge and depends on the driver cell, load cells and electrical characterization for the wires.

Based on the linear delay model proposed above, $\mathcal{LRS}$-$\mathcal{S}$ can be written as:

$$\begin{aligned}
\text{Minimize} \quad & \mathcal{L}_{\boldsymbol{\lambda}}(\mathbf{x}, \mathbf{y}) \\
& = \sum_{e \in E} \mathrm{WL}_e(\mathbf{x}, \mathbf{y}) + \sum_{\forall (i,j)} \lambda_{ij}(d_i + WL_e(\mathbf{x}, \mathbf{y}) \cdot \gamma_e) \\
& \quad + \text{terms independent of } \mathbf{x}, \mathbf{y} \\
& = \sum_{e \in E} \mathrm{WL}_e(\mathbf{x}, \mathbf{y}) + \sum_{\forall (i,j)} WL_e(\mathbf{x}, \mathbf{y}) \cdot \lambda_{ij}\gamma_e \\
& \quad + \text{terms independent of } \mathbf{x}, \mathbf{y} \\
\text{Subject to} \quad & \text{cell spreading constraints}
\end{aligned}$$

The new objective function only contains $\mathbf{x}$, $\mathbf{y}$ as variables. Therefore, $\mathcal{LRS}$-$\mathcal{S}$ becomes a weighted wirelength minimization problem for a set of hyperedges, which can be solved well by existing wirelength-driven placement engine with the ability to handle net weights.

المنارة للاستشارات
www.manaraa.com

3.3.1.5 Solving $\mathcal{LDP}$

In general, $\mathcal{LDP}$ can be solved by solving a sequence of $\mathcal{LRS}$-$\mathcal{S}$. Many previous works use or modify the subgradient optimization method to solve $\mathcal{LDP}$ (e.g., [14][73]). The basic idea of the subgradient optimization method is straightforward. At each iteration, we first update $\boldsymbol{\lambda}$ based on the criticality of all timing edges. Then, based on the updated $\boldsymbol{\lambda}$, we solve $\mathcal{LRS}$-$\mathcal{S}$ again to generate a new placement. Besides the subgradient optimization method, we can also use the direction finding approach [83], which has been shown to have better convergence in practice.

3.3.2 Spreading-outside Approach

The benefit of the spreading-inside approach is that one can leverage an existing wirelength-driven placer as a black box to solve $\mathcal{GTDP}$ without any modification. However, cell spreading constraints are non-convex by nature. Besides, they are usually modeled by non-continuous and non-differentiable functions in modern placers [39][46]. Thus, this approach cannot guarantee that an optimal solution of the $\mathcal{LDP}$ is also optimal for the primal problem. To avoid this issue, we propose another approach to solve $\mathcal{GTDP}$, which we referred to as the spreading-outside approach.

An overview of the spreading-outside approach is shown in Fig. 6.2(b). As implied by the name, we handle the cell spreading constraints outside the LR framework. In the beginning, similar to typical wirelength-driven placement algorithms, we convert the cell spreading constraints into a cost which is incorporated into the objective function of the placement problem. Different from wirelength-driven placement, the resulting problem still has the timing constraints instead of being unconstrained. Here, we leverage the LR framework to solve this problem, since LR has shown to be very effective in handling the timing constraints. After the LR loop converges, we will update and convert the cell spreading constraints again if needed.

We neglect the derivation of the LR subproblem and Lagrangian dual problem for the spreading-outside approach, as it is similar to what we have presented in Section III-A, except we do not have the cell spreading constraints this time. In particular, for this approach, the

www.manaraa.com

$\mathcal{LRS}$-$\mathcal{S}$ will just be an unconstrained optimization problem and can be easily solved using any standard methods.

By tackling the complex and non-differentiable cell spreading constraints outside the LR framework, LR only needs to handle a problem with timing constraints, which are converted into terms linear to wirelength in the objective function of $\mathcal{LRS}$-$\mathcal{S}$. Thus, better solutions can be expected at the LR step. In particular, if wirelength is modeled as convex function and the spreading constraints are converted into convex functions, the problem to be solved by LR is a convex optimization problem. Then, the strong duality will hold and this convex problem can be solved optimally using the LR framework. The disadvantage of the spreading-outside approach is that existing weighted wirelength minimization placers will no longer be directly applicable. Instead, one needs to implement ones own cell spreading step and detach the cell spreading part from the wirelength-driven placer in order to use it.

3.3.3 Comparing our approaches with previous Lagrangian relaxation based $\mathcal{TDP}$ algorithms

In this subsection, we discuss in details about the differences between our approaches and several previous works.

In [73], the proposed LR framework relaxes the cell spreading constraints together with the timing constraints. However, this framework only works for placers with explicit modeling of spreading constraints, i.e., GORDIAN with center-of-gravity constraints [41]. For some state-of-the-art placers, this framework might not work, since the spreading constraints are often handled implicitly using heuristic algorithms [39][46]. In [31], the spreading constrains are not relaxed, while the path based approach makes the proposed framework not applicable for large-scale circuits. In [34], Lagrangian relaxation is only used as a refinement step after global placement, and the COG based cell spreading constraints are not updated. Therefore, the effectiveness of the proposed approach is greatly limited. In addition, all previous works did not simplify the LR subproblems through exploring the special structure of the circuit graph. Thus, extra effort is required to calculate the cycle time and arrival time variables.

المنارة للاستشارات
www.manaraa.com

In addition, more iterations are required to search for the optimal $\boldsymbol{\lambda}$ in $\mathcal{LDP}$ as $\boldsymbol{\lambda}$ is not limited by the optimality conditions $\mathcal{K}$.

Different from previous works, both the spreading-inside approach and spreading-outside approach proposed by us simplify the $\mathcal{LRS}$ using KKT conditions based on the special structure of the circuit. In addition, our approaches do not relax the cell spreading constraints by LR. Thus they are more compatible with various type of placement techniques, especially for modern placers with implicit modeling of cell spreading constraints. Finally, our approaches can be used to optimize either the critical paths or the critical cycles of the circuit, and hence are suitable for different circuit design styles.

3.4 Detailed Implementation

In this section, we talk about the detailed implementation of the timing-driven placement approaches proposed in Section III. In particular, our TDP tool, which is referred to as TD-POLAR here, incorporates the proposed timing-driven placement approaches with the state-of-the-art quadratic placer POLAR [46]. In Section IV-A, we first discuss the quadratic placement and rough legalization techniques which are the core techniques used in the POLAR algorithm. Next, in Section IV-B, we will discuss how we leverage POLAR to solve the $\mathcal{GTDP}$ problem using the proposed approaches.

3.4.1 POLAR: a wirelength-driven placer based on quadratic and rough legalization techniques

3.4.1.1 Quadratic Placement

Assuming all the nets $e \in E$ are two pin nets. The wirelength for a particular net e can be modeled using the HPWL, which is given by the Manhattan distance between the two cells connected by e. Then, the total wirelength can be calculated by the total sum of HPWL for all the nets:

$$\mathrm{HPWL}(\mathbf{x}, \mathbf{y}) = \sum_{e \in E} [\max_{i \in e} x_i - \min_{i \in e} x_i + \max_{i \in e} y_i - \min_{i \in e} y_i]$$

www.manaraa.com

The function HPWL($\mathbf{x}, \mathbf{y}$) is convex, but it is not differentiable. To make the optimization easier, the quadratic technique approximates the Manhattan distance of the two pin net by the squared Euclidean distance, also known as quadratic wirelength. Let Q_x and Q_y be the connection matrices. The objective of the wirelength-driven placement can be defined as:

$$\text{Minimize} \quad \phi = \frac{1}{2}\mathbf{x}^T Q_x \mathbf{x} + \mathbf{c}_x^T \mathbf{x} + \frac{1}{2}\mathbf{y}^T Q_y \mathbf{y} + \mathbf{c}_y^T \mathbf{y} + const$$

It can be proved that both Q_x and Q_y are symmetric positive definite matrices. Thus, ϕ is convex and differentiable, and the minimum solution of ϕ can be found by setting its derivatives to 0 and solving the resulting system of linear equations:

$$Q_x \mathbf{x} + \mathbf{c}_x + Q_y \mathbf{y} + \mathbf{c}_y = 0 \tag{3.15}$$

3.4.1.2 Rough Legalization

If we consider minimizing ϕ alone, the cells will not be spread out and the placement solution will not be legalizable. Therefore, extra techniques are required to avoid excessive cell overlapping.

POLAR adopts the rough legalization (RL) [39] approach to reduce the cell overlapping. At each placement iteration, RL quickly spreads out the cells and generates an almost legal placement, as shown in Fig. 6.3(b). The roughly legalized placement is used to generate the spreading forces, which are incorporated into the objective function of the wirelength-driven placement problem and guide the quadratic placement on the next iteration, as shown in Fig. 6.3(c).

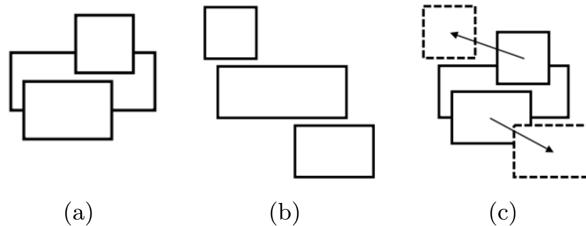

Figure 3.3: (a) Cell overlaps after quadratic placement. (b) An almost legal placement obtained by rough legalization. (c) Use of the roughly legal placement to guide the spreading force generation.

www.manaraa.com

3.4.2 TD-POLAR: a general timing-driven placement tool

For both the spreading-inside and the spreading-outside approaches, we use the direction finding approach inspired by [83] to solve $\mathcal{LDP}$. We are not using the subgradient optimization method as it requires a projection of $\boldsymbol{\lambda}$ unto the optimality conditions $\mathcal{K}$ after each iteration to maintain $\boldsymbol{\lambda}$ within the feasible region of $\mathcal{LDP}$. For $\mathcal{STDP}$ problem, projection can be done by simply traversing the circuit in topological order since the corresponding graph of the circuit is a directed acyclic graph. For $\mathcal{STDPS}$ or $\mathcal{ATDP}$, it will not be easy to redistribute $\boldsymbol{\lambda}$ since the corresponding circuit structure contains loops.

In particular, the direction finding approach wants to find an improving feasible direction $\Delta\boldsymbol{\lambda}$ and a step size β such that at each step we have:

$$q(\boldsymbol{\lambda} + \beta\Delta\boldsymbol{\lambda}) > q(\boldsymbol{\lambda})$$

The improving feasible direction $\Delta\boldsymbol{\lambda}$ can be found by solving the following linear program:

$$\begin{aligned} \mathcal{DF}: \quad & \text{Maximize} \quad \sum_{\forall(i,j)} \Delta\lambda_{ij}\hat{D}_{ij} \\ & \text{Subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0},\ \boldsymbol{\lambda} \in \mathcal{K} \\ & \qquad\qquad\quad \max(-u, -\lambda_{ij}) \leq \Delta\lambda_{ij} \leq u \end{aligned}$$

where u is used to bound the objective function from going to infinity, similar to [83].

3.4.2.1 Implementation of the spreading-inside approach

It is straightforward to incorporate POLAR with our timing-driven placement flow using the spreading-inside approach. Since the $\mathcal{LRS}$-$\mathcal{S}$ is a weighted wirelength optimization problem with spreading constraints, we can directly call POLAR to solve it. To capture the timing of the circuit, we add an extra pseudo two-pin net for each timing edge to the circuit. After the $\boldsymbol{\lambda}$ update step, the weights of the added two-pin nets should be updated accordingly, while the weights of original nets are kept the same.

المنارة للاستشارات
www.manaraa.com

3.4.2.2 Implementation of the spreading-outside approach

The implementation of the spreading-outside approach requires a tighter integration with the placement engine. We split POLAR into the quadratic placement step, which we have presented in Section IV-A 1), and the rough legalization step, which is done by a heuristic algorithm. Then, for the spreading-outside approach as shown in Fig. 6.2(b), the step of solving the unconstrained $\mathcal{LRS}$-$\mathcal{S}$ will be similar to the quadratic placement step of POLAR, except now there are weights associated with nets given by the Lagrange multipliers. In addition, the step of converting spreading constraints into the cost function will be replaced by the rough legalization step of POLAR. Therefore, for the spreading-outside approach, we first perform the rough legalization to generate the spreading forces for the current placement. The spreading forces are then incorporated into the objective function of $\mathcal{GTDP}$ to guide the placement process in the quadratic placement step. Next, we apply Lagrangian relaxation framework on the quadratic placement step to handle the timing constraints of $\mathcal{GTDP}$. The iteration continues until there is no improvement in the objective function or we exceed the runtime limit.

3.4.3 Timing-driven Detailed Placement

Since traditional detailed placement algorithms only target reducing the total chip wirelength, timing degradation might happen if we directly use them to optimize the global placement results generated by TD-POLAR. In order to minimize the disturbance on circuit timing, we developed a timing-driven detailed placement step to further optimize the global placement results and also help generating a legalized placement. In particular, we leveraged the existing wirelength-driven detailed placement engine FastDP [63] and applied net weights into its cost function. The pseudo nets to capture timing at global placement are kept. The net weights we used for the pseudo nets are the same as those at the final round of the global placement stage. Thus, it reflects the timing criticality for each net. By doing this, FastDP is able to respect the timing criticality at the global placement stage during its optimization process and the disturbance on timing is greatly reduced.

www.manaraa.com

Table 3.1: Comparison on non-timing-driven placement flow and commercial timing-driven placement flow

Design	Routed Wirelength x 10^6 (nm)				Cycletime (ns)				Runtime (s)			
	Encounter	Proteus	SI	SO	Encounter	Proteus	SI	SO	Encounter	Proteus	SI	SO
s444	10.46	11.32	10.35	10.60	6.06	5.72	4.22	4.22	8	245	8	16
s510	32.84	35.38	34.06	32.21	7.25	7.01	5.98	6.31	12	330	15	29
s526	14.24	14.85	13.14	12.92	4.20	4.83	3.37	3.60	8	257	8	18
s526a	12.22	13.79	12.87	13.04	4.89	3.75	3.69	3.30	8	249	8	18
s641	22.02	26.96	23.70	23.86	5.00	4.88	4.39	4.33	10	313	12	25
s713	21.06	24.38	21.78	20.87	6.15	5.71	5.87	5.18	9	228	11	23
s820	43.39	48.53	45.83	44.91	10.40	8.34	6.33	6.22	13	431	17	34
s832	45.84	53.04	45.89	46.48	6.82	7.32	5.79	5.81	15	465	18	37
s838	33.32	37.49	34.73	33.84	5.91	5.58	5.46	4.82	13	337	17	36
s953	61.84	71.85	64.11	63.77	6.12	7.10	7.51	5.63	18	576	24	48
s1488	130.13	137.06	129.38	127.53	12.79	11.35	10.58	10.64	27	771	42	78
s1423	64.48	71.25	61.68	63.59	14.85	8.37	6.91	7.13	20	692	34	64
s9234	119.40	134.48	117.46	120.20	9.83	8.19	7.22	6.58	31	517	56	106
s13207	338.18	386.13	341.98	328.83	13.72	11.86	12.01	10.66	67	1202	156	301
s38417	1253.42	1208.16	1267.24	1245.69	80.43	68.30	45.24	42.66	283	1050	615	997
ALU4	18.01	18.78	23.07	20.99	5.68	5.99	4.38	4.07	10	261	6	22
ALU8	55.03	53.89	76.26	71.44	8.43	5.11	4.65	4.13	16	470	14	52
acc32	49.09	59.87	59.46	58.13	6.45	5.57	4.39	5.10	17	528	15	54
acc64	138.06	145.88	144.53	146.51	10.01	7.37	5.55	5.88	39	757	48	154
GCD	23.32	24.17	27.37	26.50	20.05	20.68	16.35	17.32	21	604	7	16
FU	396.16	453.41	447.56	445.41	16.06	7.81	9.94	8.60	62	958	98	315
Average	1.000	1.051	1.042	1.026	1.000	0.846	0.689	0.660	1.000	15.900	1.739	3.456

المنارة للاستشارات
www.manaraa.com

3.5 Experiments

The proposed approach is implemented using C++. All experiments were run on a Liunx PC with 47GB of memory and Intel Core-i3 3.3GHz CPU.

We demonstrate our approaches using asynchronous circuits since it is the most general circuit design style among the three design styles which we have introduced in Section II. In particular, the asynchronous circuits we used are based on the PCHB template [47], which is a QDI template designed with dual-rail asynchronous channels and 1-of-N handshaking protocol [9]. Fig. 6.4 shows a three-stage PCHB pipeline structure with control circuit (CTRL), C-element (C) and domino logic (LOGIC) for computation.

Marked lines in Fig. 6.4 show an example of timing assumptions made by PCHB. It requires the input to the domino block go low before a rising transition on the control signal 'en' occurs. This timing assumption is a relaxed interpretation of the *isochronic fork* assumption [66] and can easily be met without special care. We ignore this timing constraint at global placement stage and leave it to be checked after detailed placement and routing, similar to [6] and [78].

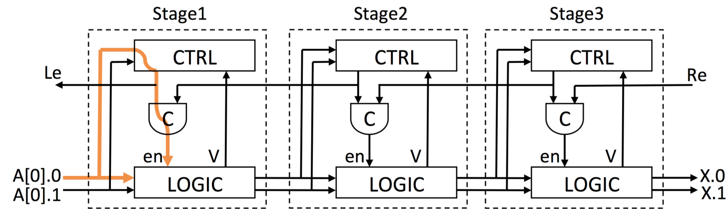

Figure 3.4: PCHB pipeline template.

We run TD-POLAR on two sets of benchmarks. First is a set of ISCAS89 benchmark circuits which are converted to unconditional asynchronous circuits using the front-end synthesis flow of Proteus [6]. In particular, flip-flops from ISCAS89 are mapped to token buffers and combinational gates are mapped to logic cells in PCHB cell library. The second set consists of several benchmarks synthesized from RTL to netlist using Proteus.

www.manaraa.com

For ALU and Accumulator (ACC) design, we choose different bit width for the datapath to create a set of benchmarks with different number of cells.

The statistics of our benchmark circuits are shown in Table 6.2. The column "# of vertices" shows the total number of cells in each design. The column "# of edges" shows the total number of edges, which includes the original nets of the circuit and the added two-pin timing edges. An estimation of the total number of variables for the corresponding $\mathcal{GTDP}$ problem is reported in column "# of vars".

Table 3.2: Statistics of the Circuits

Design	# of vertices	# of edges	# of vars
s444	256	2719	8730
s510	519	5535	17676
s526	307	3366	10792
s526a	297	3284	10520
s641	636	5346	17328
s713	584	4904	15866
s820	681	6952	22268
s832	706	7327	23448
s838	707	7300	23466
s953	931	9805	31422
s1488	1314	15000	47950
s1423	1119	13010	41592
s9234	2108	22118	71058
s13207	5658	56164	181288
s38417	15447	182865	584402
ALU4	413	4239	13666
ALU8	916	10140	32550
ACC32	1187	11605	37252
ACC64	3355	32741	105706
GCD	1505	4901	15664
FU	5304	52023	167212

First, we compare the two approaches implemented in TD-POLAR with a wirelength-driven placement flow and the timing-driven commercial asynchronous optimization flow Proteus.

For the wirelength-driven placement flow, we use the industrial placer Encounter to place the design without setting any input timing constraint. This means Encounter will act as a pure wirelength-driven placer and only targeting at optimizing the total wirelength of the circuit.

www.manaraa.com

The Proteus flow performs both the global and detailed placement on the input circuits through leveraging synchronous placement tools. In particular, the Proteus flow breaks the timing loops according to the PCHB template and add explicit timing constraints on each path segments to improve the timing. To avoid the changing of input netlist, we also disable the gate resizing step during the placement stage of Proteus flow.

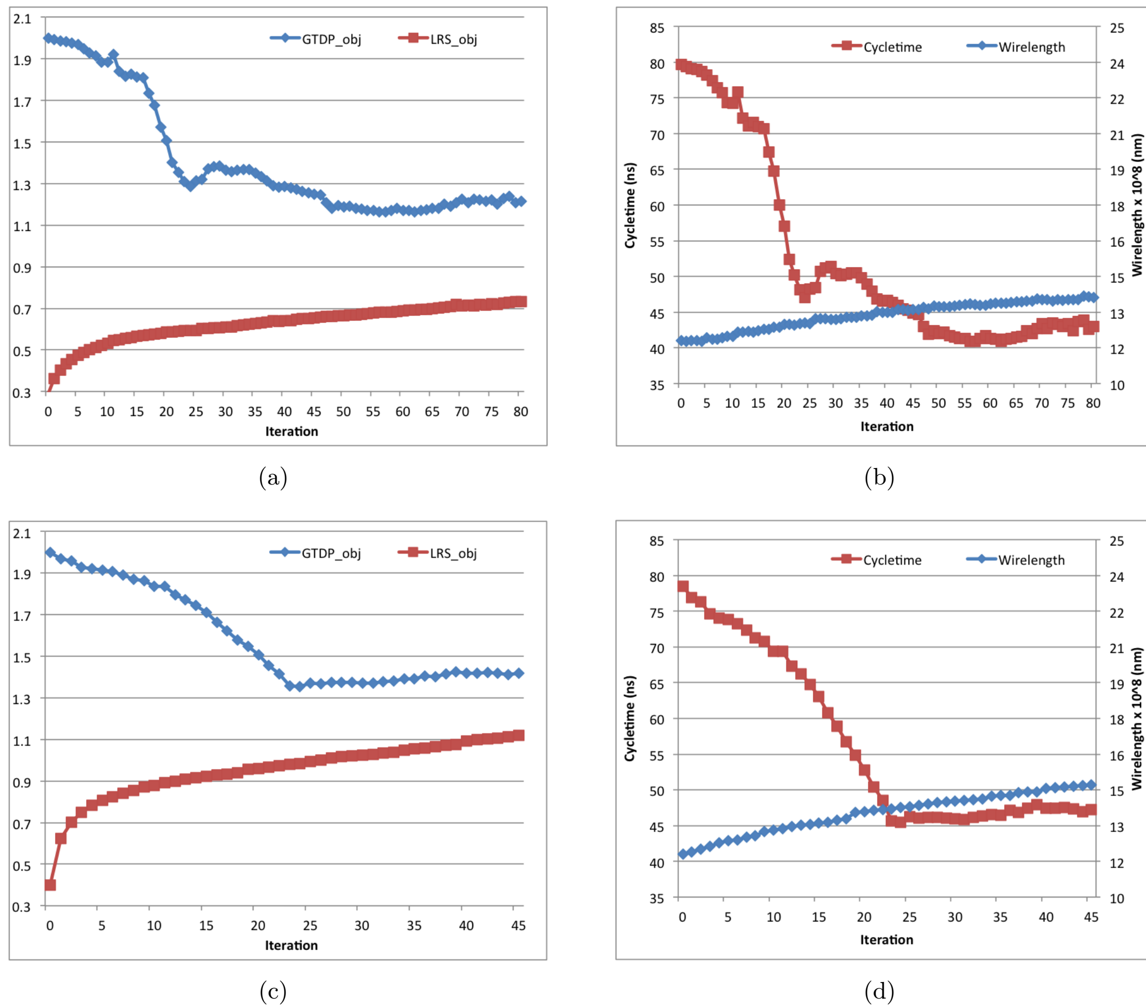

Figure 3.5: (a) The convergence of s38417 by SI. (b) The wirelength and cycltime trend of s38417 by SI. (c) The convergence of s38417 by SO. (d) The wirelength and cycltime trend of s38417 by SO.

www.manaraa.com

For our approaches, we perform the timing-driven placement using TD-POLAR at the global placement stage. At the detailed placement stage, we use the modified FastDP [63] presented in Section IV-C as our detailed placement engine. Finally, the placement results are exported to Encounter to perform routing.

The comparison results are shown in Table 6.1. The “Routed wirelength” column shows the final detailed routed wirelength reported by Encounter for all flows. The “Cycletime” column shows the cycletime calculated based on our delay model. The “Encounter” column denotes the pure wirelength-driven placement performed by Encounter. The “SI” column denotes the spreading-inside approach. The “SO” column denotes the spreading-outside approach. Regarding the total routed wirelength, as expected, all the flows which perform timing optimization of the circuit have a higher total wirelength than the non-timing-driven flow Encounter. Among the timing-driven placement flows, both the spreading-inside and spreading-outside approach can achieve a shorter total wirelength compared with the Proteus flow, while the spreading-outside approach achieves the smallest wirelength. Regarding the cycletime, the timing-driven placement flows can achieve much better cycletime than the non-timing-driven Encounter flow. In particular, the Proteus flow is 15.4% better than the Encounter flow, while the spreading-inside approach and spreading-outside approach is 31.1% better and 34% better than the Encounter flow respectively. This shows the importance of timing-driven placement on optimizing the timing of the circuits. It also shows our proposed approaches are more effective in improving the timing of asynchronous circuits than the Proteus flow, as our approaches consider all timing loops globally and there is no need for extra explicit timing constraints. In addition, on average, the spreading-outside approach achieves a shorter wirelength and a smaller cycletime compared with the spreading-inside approach. This is because the spreading-outside approach uses LR to handle a problem with timing constraints only, while the cell spreading constraints are handled separately outside the LR loop, and hence better solutions can be expected. Regarding the runtime, the Encounter flow has the shortest runtime, since it does not perform any optimization on circuit timing. The Proteus flow has the longest runtime, due to its added explicit timing constraints which can be exponential to the circuit size. Both our approaches are shown to be much faster and scalable in comparison with the Proteus flow. In particular,

المنارة للاستشارات
www.manaraa.com

the spreading-inside approach is about 2X faster than the spreading-outside approach. This is because the spreading-inside approach converges faster and can be stopped earlier.

The convergence sequences of our largest circuit s38417 using the spreading-inside approach and the spreading-outside approach are shown in Fig. 6.5(a) and (c) respectively, where the blue line denotes the objective value of the $\mathcal{GTDP}$ and the red line denotes the objective value of the $\mathcal{LRS}$-$\mathcal{S}$. The corresponding changes of cycle time and total chip wirelength at each iteration is shown in Fig. 6.5(b) and (d) respectively, where the red line denotes the cycletime and the blue line denotes the wirelength. It can be seen that both approaches are very effective in reducing the cycletime of the circuit. In addition, each iteration of the spreading-outside approach includes five iterations of LR plus one step of rough legalization, while each iteration of the spreading-inside approach only includes one step of LR and one step of rough legalization. Thus, even though the total number of iterations for the spreading-inside approach is larger than that of the spreading-outside approach in Fig. 6.5, it is actually stopped earlier. However, as shown in the figure, the spreading-outside approach converges smoother than the spreading-inside approach, due to its more fine-grained optimization at each step. Therefore, after the detailed placement is performed, the spreading-outside approach is able to achieve better results.

Next, we compare different detailed placement techniques in Table 6.3 and 6.4. For Table 6.3, the detailed placement are performed on the global placement results generated by the spreading-inside approach. For Table 6.4, the detailed placement are performed on the global placement results generated by the spreading-outside approach. For both tables, the column "FastDP" denotes the flow where we directly use FastDP to perform wirelength-driven detailed placement without adding weights. The column "TD-FastDP" denotes the timing-driven detailed placement, where we add the weight from the global placement stage into FastDP. The column "Legalize" denotes the detailed placement flow which only performs the legalization of the global placement results using Encounter.

From the experimental results in Table 6.3 and 6.4, we can see that directly applying FastDP to perform detailed placement can achieve a smaller total wirelength, but the cycletime improvement we achieved at the global placement stage will be degraded a lot. In comparison,

المنارة للاستشارات
www.manaraa.com

Table 3.3: Comparison on DP algorithms using spreading-inside approach

Design	Routed Wirelength x 10^6 (nm)			Cycletime (ns)			Runtime (s)		
	FastDP	TD-FastDP	Legalize	FastDP	TD-FastDP	Legalize	FastDP	TD-FastDP	Legalize
s444	10.27	10.35	10.58	4.46	4.22	4.67	0.07	0.12	2.86
s510	31.99	34.06	33.10	6.67	5.98	6.87	0.13	0.17	3.13
s526	13.00	13.14	13.30	5.96	3.37	3.96	0.11	0.14	2.94
s526a	12.83	12.87	13.34	4.19	3.69	4.14	0.10	0.12	3.00
s641	22.79	23.70	22.83	4.33	4.39	4.41	0.15	0.20	3.39
s713	20.65	21.78	21.42	7.25	5.87	5.72	0.11	0.19	3.20
s820	43.60	45.83	44.30	6.73	6.33	7.10	0.14	0.23	3.11
s832	45.25	45.89	46.15	5.77	5.79	6.19	0.14	0.23	3.36
s838	34.24	34.73	38.65	5.25	5.46	4.88	0.13	0.23	3.26
s953	61.88	64.11	64.01	5.94	7.51	6.90	0.17	0.34	3.43
s1488	126.06	129.38	124.88	11.18	10.58	11.80	0.26	0.41	3.52
s1423	60.02	61.68	63.21	7.84	6.91	8.47	0.43	0.36	3.29
s9234	113.93	117.46	120.19	6.65	7.22	6.76	0.35	0.67	3.65
s13207	332.25	341.98	381.02	12.39	12.01	11.17	0.92	1.68	7.03
s38417	1228.40	1267.24	1310.20	59.35	45.24	42.44	3.59	7.90	7.37
ALU4	22.04	23.07	16.92	8.15	4.38	5.22	0.12	0.14	2.94
ALU8	74.17	76.26	50.31	3.64	4.65	4.44	0.19	0.27	3.17
ACC32	58.53	59.46	65.50	6.28	4.39	5.46	0.23	0.37	3.22
ACC64	141.41	144.53	161.67	6.04	5.55	5.15	0.55	0.83	4.52
GCD	26.31	27.37	23.55	17.06	16.35	18.63	0.11	0.15	3.14
FU	438.13	447.56	435.78	10.49	9.94	9.31	0.85	1.63	4.82
Average	1.000	1.029	1.049	1.000	0.875	0.893	1.000	1.850	8.850

www.manaraa.com

Table 3.4: Comparison on DP algorithms using spreading-outside approach

Design	Routed Wirelength x 10^6 (nm)			Cycletime (ns)			Runtime (s)		
	FastDP	TD-FastDP	Legalize	FastDP	TD-FastDP	Legalize	FastDP	TD-FastDP	Legalize
s444	10.04	10.60	12.54	4.17	4.22	4.45	0.09	0.14	2.80
s510	30.96	32.21	35.78	5.92	6.31	7.07	0.12	0.19	3.05
s526	12.99	12.92	15.24	3.65	3.60	3.86	0.09	0.12	3.33
s526a	12.61	13.04	14.86	3.38	3.30	3.17	0.09	0.11	3.27
s641	22.45	23.86	25.55	4.60	4.33	4.60	0.13	0.16	3.19
s713	20.53	20.87	23.01	7.35	5.18	5.47	0.12	0.18	3.20
s820	43.70	44.91	51.25	6.52	6.22	7.15	0.14	0.22	3.12
s832	44.52	46.48	53.97	8.03	5.81	6.19	0.18	0.50	2.96
s838	33.81	33.84	40.62	5.31	4.82	4.99	0.17	0.74	3.01
s953	61.21	63.77	74.16	5.52	5.63	6.41	0.23	0.29	3.31
s1488	125.67	127.53	147.69	11.71	10.64	12.08	0.24	0.43	3.26
s1423	60.68	63.59	77.57	7.05	7.13	8.88	0.22	0.40	3.24
s9234	116.52	120.20	135.68	6.83	6.58	6.99	0.37	0.54	3.53
s13207	320.50	328.83	387.80	11.72	10.66	12.85	1.03	1.64	5.55
s38417	1201.03	1245.69	1509.41	63.77	42.66	45.38	3.70	9.25	7.14
ALU4	20.37	20.99	22.70	6.02	4.07	3.48	0.10	0.14	3.17
ALU8	70.43	71.44	76.23	3.46	4.13	6.63	0.39	0.31	3.44
ACC32	56.41	58.13	65.11	5.67	5.10	5.50	0.21	0.37	3.38
ACC64	144.50	146.51	162.78	6.61	5.88	5.90	0.53	1.04	4.11
GCD	24.87	26.50	29.88	10.36	17.32	22.95	0.38	0.19	3.04
FU	435.34	445.41	492.34	18.12	8.60	11.19	0.95	1.62	4.96
Average	1.000	1.023	1.131	1.000	0.475	0.618	1.000	1.957	8.013

المنارة للاستشارات
www.manaraa.com

the weighted FastDP results in a small increase in the total wirelength, but the final cycletime of the circuit will be much better. Also, the TD-FastDP approach is much better than the legalization approach, which does not perform any optimization on wirelength and timing. In terms of the runtime, the TD-FastDP is slightly slower than FastDP, as extra computation is required to calculate the cost based on net weights.

3.6 Conclusion

In this paper, we have formulated a general timing-driven placement problem which is applicable to various design styles. The proposed problem is solved through Lagrangian relaxation technique. We simplified the relaxed problem using KKT conditions and proposed two different approaches on incorporating the LR framework to solve the formulated general timing-driven placement problem. One approach provides a quick way to leverage existing wirelength-driven placers on solving the timing-driven placement problem. The other approach provides an option to tightly combine the LR framework with the existing wirelength-driven placer, and hence better results can be achieved. To demonstrate the proposed approaches, we implemented a placement tool based on a state-of-the-art wirelength driven quadratic placer. The experimental results shows our approaches can greatly improve the performance of the given circuits at the placement stage.

3.7 Acknowledgments

We would like to give special thanks to Ismail Bustany of Mentor Graphics for providing invaluable insights and helping us to refine this paper.

المنارة للاستشارات

www.manaraa.com

CHAPTER 4. DETAILED PLACEMENT ALGORITHM FOR VLSI DESIGN WITH DOUBLE-ROW HEIGHT STANDARD CELLS

Conventional detailed placement algorithms typically assume all standard cells in the design have the same height. However, as the complexity and design requirement increase in modern VLSI design, designs with mixed single-row height and double-row height standard cells come into existence in order to address the emerging standard cell design challenges. A detailed placement algorithm without considering these double-row height cells will either have to deal with a lot of movable macros or waste a significant amount of placement area, depending on what type of techniques people use to accommodate such design. This paper proposes a new placement approach which can handle designs with any number of double-row height standard cells. We transform design with mixed-height standard cells into one which only contains same height standard cells by pairing up single-row height cells into double-row height. Then conventional detailed placement algorithms can be applied. In particular, we generate cell pair candidates by formulating a maximum weighted matching problem. A subset of the cell pair candidates are then carefully selected to form double-row height cells based on the local bin density. A refinement procedure is performed at the end to further improve our placement quality. We compare our approach with two alternative detailed placement methods on mixed-height asynchronous and synchronous designs. The experimental results show that our approach can achieve much better quality and robustness.

4.1 Introduction

Standard cell methodology has been widely adopted as a quick and efficient method to overcome the continuously increasing complexity of integrated circuit design. In standard cell

المنارة للاستشارات
www.manaraa.com

library design, cell height is fixed to an integer multiples of a unit row height, but cell width can be variable. Conventionally, a standard cell library only contains cells with single-row height [84], as smaller cell height can achieve a higher density for simple standard cells (e.g., inverter, nand, nor) and hence lower the cost. However, for complex standard cells (e.g., flip-flop, latch), the limitation on cell height will create heavy routing congestion. In this case, the persistence in single-row height standard cells will greatly decrease layout efficiency and consume more layout design time from engineers [4]. Also, for high performance applications, single-row height cells might not be able to deliver sufficient current because the transistor size is small [11]. Thus, multi-row height standard cells, in particular double-row height cells, are commonly intermixed with smaller single-row height standard cells in order to increase the area, design efficiency and help meet the design requirements [34][20].

Another case which motivates us to pay attention to double-row height cells is their common existences in some other VLSI design styles. For example, for asynchronous circuit design, most of its standard cell can be double-row height. This is because asynchronous logics require extra circuits to generate handshaking signals [47][9], which means a larger and more complex cell is required compared with their synchronous counterparts.

Placement has become a very critical step in today's VLSI physical design flow. While double-row height placement is available in commercial tools, it is still new in academic field and most placement techniques typically assume all single-row height cells to be standard cells and other cells to be fixed or movable macros [40]. The detailed placement algorithm will focus more on improving the placement of standard cells while relying on floorplanning techniques to place macros into a good location [63][65]. If the design only has a very small amount of double-row height cells, treating them as macros works well. However, there are designs which have more double-row height cells. In our case, the set of asynchronous benchmarks we used for our experiment has an average of 77% double-row height standard cells in each design. If we treat all double-row height cells here as movable macros, even the initial legalization step can take a lot of runtime and we might still get overlaps in the end. This is because the floorplanning techniques will not be scalable to hundreds and thousands of movable macros [3]. Also, during later detailed placement steps, no optimization (such as the commonly used

المنارة للاستشارات
www.manaraa.com

cell swapping methods [63][16]) is applied to these cells, as their locations are fixed in the very beginning. Therefore, the placement quality of the design will be greatly affected.

An alternative way to place these design is to expand all single-row height cells to double-row height. Then all standard cells will have the same height which is compatible to conventional detailed placement algorithms. However, the cell expansion can increase chip utilization quite a lot, depending on how many single-row height cells we have in our design. If the chip utilization becomes too high, there will not be enough free space for the detailed placement algorithm to explore a good placement solution. If the utilization becomes more than 100%, we will even not be able to obtain a legalized placement. Also, cell expansion can lead to some cells not be able to be placed closer together which means the wirelength will be increased.

In this paper, we are focusing on the problem of detailed placement for designs with any number of double-row height cells. The input to our placement problem is a placement region, a set of modules, and a set of nets. Also, we are given a set of rough locations for each modules which is obtained from the global placement result. Our algorithm finds a position for each module within the placement region so that there is no overlap among the modules and the total wire length is minimized. Our idea is to transform mixed-height cells in a design into the same height. Then conventional detailed placement techniques can be applied on them. The equalization of cell heights is realized using a combination of two techniques: cell expansion and cell pairing. In particular, in low density areas, we apply cell expansion by doubling the height of single-row height cells to have minimum restriction on the cell movement. While in high density areas, appropriate pairs of single-row height cells are identified and combined into double-row height cells to achieve more available free space compared with simply doing cell expansion.

Our detailed placement approach is compared with two alternative placement methods: for the first method, all the height of single-row height cells will be doubled before doing detailed placement. Then conventional detailed placement algorithm is applied on this equal cell height but expanded design. For the second method, we directly applied conventional detailed placement algorithm, which means all double-row height cells will be treated as movable macros during the detailed placement process. The experimental results show that our placement

www.manaraa.com

approach can always achieve a better wirelength compared with the other two methods and is much more robust in handling designs with different amount of double-row height cells.

The rest of the paper is organized as follows: Section II provides an overview of our approach. Section III explains each technique we used in details. Section IV shows the experimental results compared with other methods. Finally, Section V concludes the paper.

4.2 Overview

Our goal here is to develop an algorithm which can handle designs with any number of double-row height cells, while minimize the total wirelength of the design. Although we only consider wirelength in this paper, other objectives (e.g., timing, routability, manufacturability) can easily be considered by using a detailed placer optimizing those objectives.

A single-row height cell can be changed to double-row height by either simply expanding the cell or pairing up two single-row height cells together and form a double-row height cell. Pairing up cells can help to place them more tightly, which reduces the local bin density and provide more free space for detailed placement algorithm to explore a good solution. However, at the same time, forcing two cells to be placed together will restrict their movement and some potential placement solution cannot be explored. In order to make sure that the formed cell pairs do not impose too much restriction during the detailed placement, we only pick those pairs which can provide us the most "benefit", which will be defined in Sec. III. Also, we go back to the simple cell expansion strategy in low density areas where there is sufficient free space even after expansion, in order to give each single-row height cell the maximum freedom to move.

A high-level view of the flow developed in this paper is shown in Fig. 5.1.

Our flow works on a global placement result. The flow starts with a cell pairing procedure, which is formulated as a maximum weighted matching problem and composed of three stages: matching graph construction, maximum weighted matching and matching pair selection. Each pair of single-row height cells will be merged into a double-row height cell. In cell expansion step, single-row height cells which have not been paired up will be expanded to double-row height. Next, conventional detailed placement algorithm is applied on the transformed design

المنارة للاستشارات
www.manaraa.com

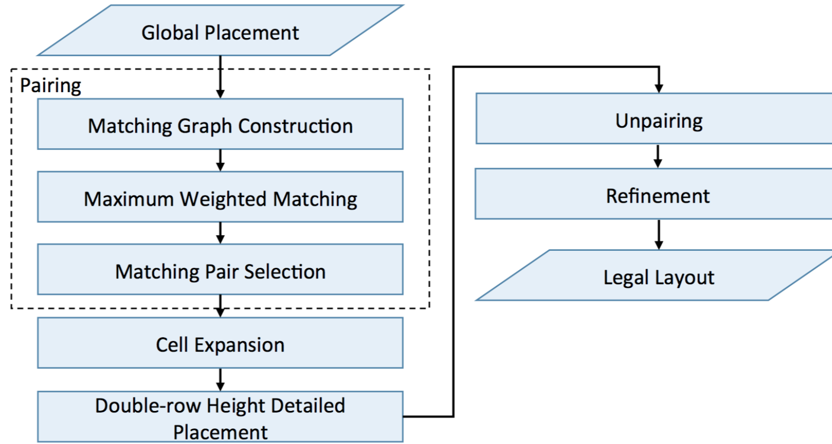

Figure 4.1: Detailed Placement Flow

which only contains double-row height cells. At the end, a refinement procedure is performed based on the previous detailed placement results, in which we fix the location of double-row height cells and run detailed placement algorithm on unpaired single-row height cells in order to further improve the placement quality.

4.3 Detailed Placement Approach

4.3.1 Matching Graph Construction

The benefit of pairing up single-row height cells is modeled using a matching graph here. The first thing we need to consider while constructing the matching graph is a good trade off between solution space and algorithm running time. If we simply construct a matching graph which an edge exists between any two single-row height cells in the design, we are able to explore all possible cell pairing solutions. However, we will end up having a complete graph with the number of edges quadratic to the number of single-row height cells and the runtime of our matching algorithm will not be acceptable. On the other hand, if we construct a matching graph which each cell is only connected with very few other cells, the total number of pairing candidates for a cell will be too small. It is quite possible that some good cell pairs are missed.

Our matching graph is constructed like this: We divide the placement region into $m \times n$ equal sized bins. Only cells within neighboring bins are considered to help limit the choice when we search for candidates. In particular, for each single-row height cell u, we first locate

المنارة للاستشارات
www.manaraa.com

the bin containing this cell. Then we look for all other single-row height cells $V = \{v_1, ..., v_n\}$ within this bin and the nearest neighboring bins. Next, we want to create an edge between u and any $v_i \in V$, if the Manhattan distance between u and v_i is less than a target distance r. Here the target distance r is carefully selected such that each cell will have enough candidates to be chosen from and the overall running time of the matching algorithm will not be too much. Ignoring cells out of range r to be a candidate does not make a big impact on the quality of matching algorithm, as pairing up cells far away can dramatically change the global placement result and make the overall wirelength worse.

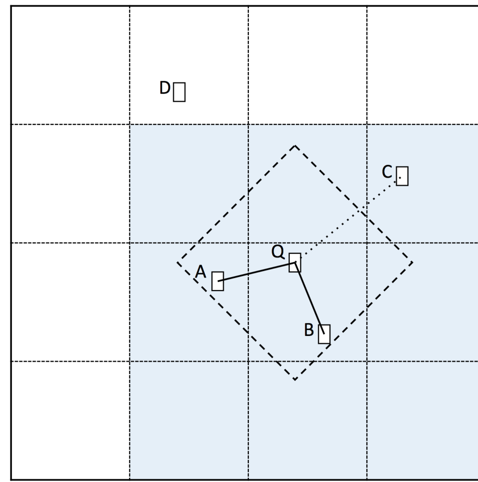

Figure 4.2: Construct Matching Graph

An example of matching graph construction is shown in Fig. 5.2. To search for matching candidates of cell Q, we first look at all single-row height cells within the shaded region, which is cell A, B and C. Cell D will be ignored in this case. Then we check if the distance between (Q, A), (Q, B) and (Q, C) is within a feasible range shown as a doted diamond in this figure. In this example, A and B is selected as matching candidates and edges (Q, A) and (Q, B) are added to the matching graph. We do not consider C as a matching candidate, as C is out of the range.

المنارة للاستشارات
www.manaraa.com

4.3.2 Edge Weight Calculation

We want to calculate edge weight based on the associated benefit if two cells are paired up. Three different factors are considered during our weight calculation: cell connectivity, area increase and cell displacement.

4.3.2.1 Cell connectivity

Here we want to consider the connectivity between cells and give more chance to those with strong connectivity in the netlist to form a pair.

Given two single-row height cells u and v. Let C be the connectivity factor for edge (u, v) in our matching graph. Let e be a hyperedge connecting cells u and v in the netlist. Let $|e|$ be the number of cells that are incident to this hyperedge. Clique model is applied here to decompose hyperedges.

We define C as:

$$C = \sum_{e \in E | u, v \in e} \frac{1}{|e|}$$

where E is the set of hyperedges in the netlist.

4.3.2.2 Penalty on area increase

The width difference between two cells can play an important role while we form pairs. Consider a simple example which four cells A, B, C and D want to form into two pairs. If a wide cell is paired up with a thin cell as shown in Fig. 5.3 (a), a lot of placement area will be wasted after grouping this two cells together. Instead, if we pair up cells with similar width as shown in Fig. 5.3 (b), the total area after pairing will be much smaller than the previous method.

Here we define PA as the penalty on area increase. Consider two standard cells u and v which have single-row height h. We can set PA as follows, where W_u and W_v are the widths of cells u and v.

$$PA = h * |W_u - W_v|$$

المنارة للاستشارات
www.manaraa.com

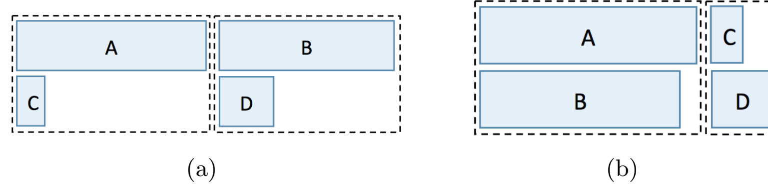

Figure 4.3: (a) Pair up cells with large width difference (b) Pair up cells with small width difference

4.3.2.3 Penalty on cell displacement

Another factor we considered is cell displacement. We want to encourage pairing up two cells which are closer together while adding penalty on forming pairs which the two cells are relatively far away. The idea is to minimize the perturbation of global solution, such that the total wirelength will not be affected too much after pairing up cells.

Let PD be the penalty of cell displacement. Let $d(u, v)$ be the Manhattan distance between two cells u and v. The cell displacement penalty can be defined as:

$$PD = d(u, v)$$

Putting everything together and let B be the benefit of forming a pair, we can get:

$$B = C - \alpha_1 * PA - \alpha_2 * PD$$

where constant α_1 and α_2 can be chosen to adjust the effect among cell connectivity, area penalty and displacement penalty.

4.3.3 Maximum Weighted Matching

We use maximum weighted matching to find a set of candidate cell pairs such that pairing them up will result in maximum benefit. Matching problem is one of the most well-studied problems in computer science. There are many existing algorithms in the literature which can either find a perfect matching or do some approximation [22]. In order to find the one which is most suitable to our problem, we implemented two different matching algorithms in our experiment to perform weighted matching. One is based on the Edmond's maximum weighted matching algorithm [23] which provides $O(nmlog(n))$ runtime complexity and another is a

المنارة للاستشارات
www.manaraa.com

simple greedy approach [21] which can find an approximate solution within a linear runtime. The experiment results show that these two approaches provide similar wirelength results, while the second approach has a much better runtime. Thus, the second one is chosen in our flow.

4.3.4 Matching Pair Selection

It is not the best to pair up all single-row height cells from the matching result. Forming a pair will force the two cell to be placed together and put restrictions on detailed placement. For the matched pairs with small edge weight, the benefit they provide cannot justify to tie them up. This is especially true for matched pairs in low density area. It would be better to simply expand those cells and give them freedom to move, since in low density area, there will always be enough free space even after cell expansion.

In our flow, we only pick the top portion of matching results based on edge weight and local bin density to perform actual cell pairing, while the remaining single-row height cells are simply expanded to become double-row height.

We should notice that, after the cell pairing and expansion, the local bin density might change. The new density will depend on not only the initial bin density of the given placement, but also the total area of single-row height cells which do not form a pair. This is because the unpaired single-row height cells will double their area when they are expanded to double-row height.

Here we divide the chip into $p \times q$ equal sized bins. The size of the bins we used here can be different from the one we used during matching graph construction in Sec. III A. Let D_b be the density for bin b after global placement. Let k be the area percentage of single-row height cells which do not form a pair in this bin. Then the new density D'_b after pairing and expansion will be: $D'_b = 2 * kD_b + (1 - k)D_b = (1 + k)D_b$. Here D'_b is just an approximate value. Pairing process might move single-row height cells from one bin to another bin which can affect the bin density. There will also be some area wastage after we pack two single-row height cells together. However, considering both the cell movement and area wastage is small, we simply ignore these facts to make our algorithm simple. As we are only locally estimating D'_b, for designs with very high utilization, there might be an utilization overflow issue after cell

www.manaraa.com

expansion. But because of what we did, the chance of having such an issue is very unlikely and we have never encountered any issue in practice.

After cell candidates are generated, we first globally filter out matched pairs which have a small edge weight. Then for each bin, we select part of the matching results to form cell pairs based on D'_b. In particular, we want to keep D'_b within a good range to make sure that each bin will have enough free space and we also do not form too many pairs to limit the cell movement. The selected single-row height cell pairs are then paired up into double-row height cells with the new location in the middle of the corresponding single-row height cells.

4.3.5 Unpair and Refinement

After we run conventional detailed placement algorithm on the transformed design and get a legalized solution, there might still be room for wirelength improvement. First is because the pairs we formed in the design restrict two cells not be able to be placed in separate places. Second, the expansion on some single-row height cells also limits them to be put closer to other cells. Thus, we run detailed placement for a second time without these restrictions by unpairing cells and deflating the expanded cells. The locations of double-row height cells are fixed at this run, as we assume they have already been placed into a good location during the previous detailed placement process.

4.4 Experimental Results

The proposed approach is implemented using C++ and run on a Linux PC with 8 GB of memory and Intel 2.4 GHz CPU.

Two sets of benchmarks are used in our experiment. First is a set of asynchronous VLSI designs synthesized using an asynchronous frond-end flow [6]. Another set of benchmarks are created based on the ISPD05 placement benchmark suite. We randomly selected about 30% single-row height standard cells in the design and doubled their height. The placement region area is keep to be the same. POLAR [46] is used to generate global placement results as an input to our flow.

www.manaraa.com

We choose FastDP [63] as our detailed placement engine with small modification to make sure double-row height cells are placed only on even rows. Since FastDP is designed for single-row height benchmarks, if all standard cells are placed on even rows while some macros are aligned with odd rows, FastDP might create some overlaps. We solve this problem by adding placement blockages on the row above and / or below each macros. Two alternative methods are developed to be compared with our approach. In method 1, we apply cell expansion technique in which we double the height of single-row hight cells. Then, FastDP is run on this expanded design with all cells having equal height. In method 2, we treat the double-row height cells as movable macros. The design is first legalized using techniques described in [82], then we use FastDP to perform the detailed placement. The values of α_1 and α_2 are experimentally determined and we set $\alpha_1 = 2 \times 10^{-3}$ and $\alpha_2 = 2 \times 10^{-4}$ for all the benchmarks.

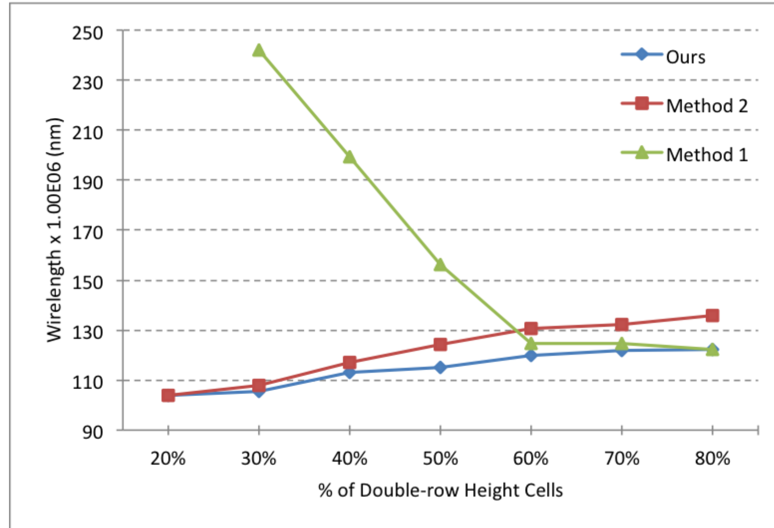

Figure 4.4: Experiment on adaptec2_dr benchmark

Comparison results on asynchronous benchmarks are shown in Table I. Second column shows the total number of cells. The percentage of double-row height cells and the chip utilization are shown in the third and fourth column. The “Init” column shows the wirelength of the global placement results after legalization. For the wirelength, our approach is 3% better than the first method and 14.8% better than the second method. The runtime of our approach is a little bit worse than the first method, as our approach run FastDP twice, but we are much better compared with the second method. The “Overlaps” column shows the number of

www.manaraa.com

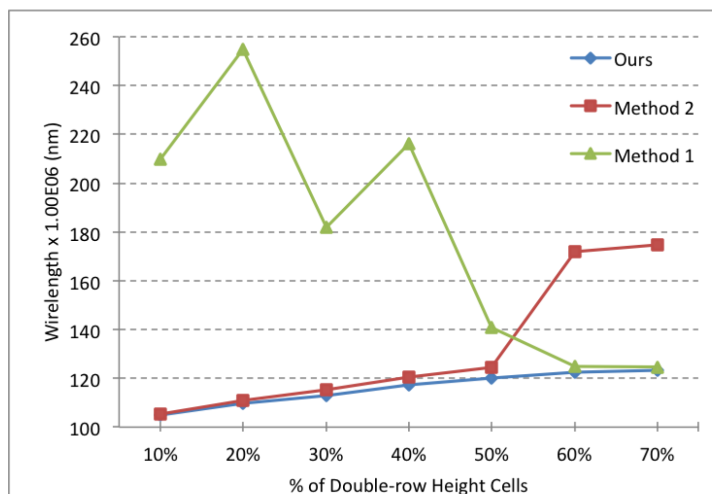

Figure 4.5: Experiment on bigblue1_dr benchmark

overlaps reported by FastDP after the detailed placement. Both our approach and method 1 can generate a legalized placement with zero overlaps. However, for the second method, almost all the designs cannot be legalized, as there are too many macros which is beyond the ability of FastDP to handle.

Table II shows the comparison results on synchronous benchmarks. On average, our wirelength is 72.2% better than the first method and 3.5% better than the second one. Also, our algorithm runtime on average is better compared with the other two methods. For method 1, the reason is because the cell expansion increases chip utilization too much and ends up making FastDP take a much longer time to perform optimization. For method 2, its runtime is better than our approach only when the design size is small. As the design size increases, legalization techniques used inside FastDP will not be scalable to the increasing number of movable macros. Thus, the runtime becomes much slower. We can also see that method 1 failed to place some designs, as their chip utilization become greater than 1 after we expanded single-row height cells in this two designs.

To further illustrate the robustness of our approach, we did another set of experiments using adaptec2 and bigblue1. We generate a new set of benchmarks with different percentage of double-row height cells in a design by randomly picking certain number of single-row height cells and doubling their height. Then, our approach and the other two methods are run on this new set of benchmarks. The original adaptec2 and bigblue1 design have the utilization of 81%

www.manaraa.com

Table 4.1: Comparison on asynchronous benchmarks

Design	Size	DH Cells	Util	Wirelength x 10^6 (nm)				Runtime (s)			Overlaps		
				Init	Ours	Method1	Method2	Ours	Method1	Method2	Ours	Method1	Method2
s9234	2108	85.10%	75.02%	167.74	134.10	133.55	159.04	1.14	1.21	244.36	0	0	294
s15850	6778	81.45%	75.35%	671.51	561.80	575.36	650.43	3.09	2.55	2112.03	0	0	1372
s13207	5658	84.36%	77.73%	521.02	432.58	435.20	509.90	2.72	2.97	1729.70	0	0	1104
s38417	15447	61.68%	68.73%	1457.15	1176.30	1223.99	1331.89	5.99	4.77	6211.27	0	0	200
ALUMAN16	2442	61.38%	63.26%	230.69	174.16	181.71	196.78	3.99	2.72	171.86	0	0	6
ALUMAN32	6866	63.17%	66.22%	819.71	643.14	692.35	733.20	3.84	5.49	1470.53	0	0	36
ALUMAN64	28974	70.69%	70.59%	3863.54	3204.05	3310.77	3659.70	15.18	9.05	26576.30	0	0	2018
acc64	3355	92.28%	75.68%	275.48	225.39	224.90	269.17	3.67	2.04	605.15	0	0	800
acc128	8401	90.44%	79.05%	692.96	601.34	604.00	682.36	8.90	6.32	3239.38	0	0	2760
GCD	445	67.64%	63.30%	36.35	24.14	25.09	30.00	1.00	0.45	6.39	0	0	0
FetchingUnit	5304	92.18%	75.05%	674.84	563.58	566.45	667.37	3.63	1.98	1411.80	0	0	1350
				1.216	1	1.030	1.148	1	0.744	823.612			

المنارة للاستشارات
www.manaraa.com

Table 4.2: Comparison on synchronous benchmarks

Design	Size	DH Cells	Util	Wirelength x 10^6 (nm)				Runtime (s)			Overlaps		
				Init	Ours	Method1	Method2	Ours	Method1	Method2	Ours	Method1	Method2
adaptec1_dr	211447	30.18%	90.84%	97.24	91.35	Fail	94.28	38.33	Fail	28.49	0	Fail	0
adaptec2_dr	255023	30.16%	89.12%	109.07	105.66	242.12	107.94	44.36	300.86	42.62	0	0	0
adaptec3_dr	451650	30.11%	78.44%	249.69	242.13	433.40	245.89	82.44	417.53	75.98	0	0	0
adaptec4_dr	496045	30.19%	67.70%	214.30	208.92	295.04	211.46	84.03	296.20	65.83	0	0	0
bigblue1_dr	278164	30.14%	73.44%	117.73	113.09	181.99	115.42	41.60	244.19	38.47	0	0	0
bigblue2_dr	557866	32.90%	68.99%	169.32	160.86	278.03	164.62	85.88	262.05	87.33	0	0	0
bigblue3_dr	1096812	30.31%	91.10%	490.91	418.97	Fail	466.60	233.67	Fail	440.98	0	Fail	423
bigblue4_dr	2177353	30.26%	73.88%	913.23	882.51	Fail	895.39	469.78	Fail	753.53	0	Fail	171
				1.062	1	1.722	1.035	1	4.495	1.420			

www.manaraa.com

and 61%. Since most of the cell area is occupied by macros, there will not be an utilization overflow.

The results are shown in Fig. 5.4 and Fig. 5.5. X axis is the ratio of double-row height cells in the design and Y axis is the total wirelength. It can be seen that method 1 which apply cell expansion techniques do a very bad job when single-row height cells are dominating in the design. The total wirelength gradually get improved with the increasing number of double-row height cells. In contrast, method 2 which treats all double-row height cells as macros works well when double-row height cells are not so many, but the wirelength gets worse when total number of double-row height gets increased. For adaptec2_dr, method 2 cannot even finish within a reasonable amount of runtime when the number of double-row height cells below 30%. In comparison, no matter how many double-row height cells exists in the design, our approach always produces placement results with better wirelength.

4.5 Conclusions and Future Work

In this paper, we have proposed a detailed placement approach targeting at designs with mixed single-row height and double-row height standard cells. We incorporated cell paring and cell expansion techniques and transformed the mixed-height design into design containing only standard cells of the same height. Then any conventional detailed placement algorithm can be applied. Our approach is compared with other two alternative methods to place design with mixed-height standard cells and achieves both better quality and robustness.

Our future work is to incorporate bin utilization constraints into our algorithm. In particular, we want to consider bin utilization during the edge weight calculation in Sec III-B and use a detailed placement engine which supports bin utilization constraints.

المنارة للاستشارات
www.manaraa.com

CHAPTER 5. GATE SIZING AND VTH ASSIGNMENT FOR ASYNCHRONOUS CIRCUITS USING LAGRANGIAN RELAXATION

Gate sizing and threshold voltage selection is an important step in the VLSI physical design process to help reduce power consumption and improve circuit performance. Recent asynchronous design flows try to directly leverage synchronous EDA tools to select gates, which have a lot of limitations due to the intrinsic difference between asynchronous and synchronous circuits. This paper presents a new simultaneous gate sizing and Vth assignment approach for asynchronous designs. We formulate the asynchronous gate version selection problem considering both leakage power consumption and cycle time. Then, the optimization is performed based on a Lagrangian relaxation framework. A fast and effective slew updating strategy is also proposed to address the timing-loops of asynchronous circuits during static timing analysis. Our approach is evaluated using a set of asynchronous designs based on the pre-charged half buffer (PCHB) template and compared with the Proteus asynchronous design flow which is leveraging synchronous EDA tools. The experiments show our approach can achieve much better quality results in terms of both leakage power and cycle time compared with the other approach.

5.1 Introduction

As the feature size of advanced fabrication process is down to nanometer scale, the design of synchronous circuit is facing more and more issues such as process variation and power consumption. Asynchronous design provides a very attractive alternative to synchronous design due to its robustness, lower power consumption and higher operating speed. Its advantages have been demonstrated by many fabricated chips [50] [51] [19]. However, asynchronous design

المنارة للاستشارات
www.manaraa.com

is still not widely adopted in the industry because of its long learning curve and the lack of asynchronous EDA tools.

Gate sizing and V_{th} assignment have been shown to be very effective techniques to optimize the power and performance of synchronous circuits. We expect these techniques to have similar impact on asynchronous circuits. In particular, we want to minimize the leakage power, as in advanced process it contributes to a large part of total power consumption which has become an important design objective nowadays due to the limited battery life of the widely used portable devices. For asynchronous design, leakage power minimization can be even more critical because of its higher gate count than synchronous design. Therefore, in this paper, we are focusing on the problem of leakage power and cycle time minimization for asynchronous design by selecting gates of different sizes and V_{th} from a standard cell library.

Both gate sizing and V_{th} assignment techniques for synchronous circuits have been extensively studied for decades [26] [10] [14] [57]. However, for asynchronous circuits, there are only very few works on it. Most of the automatic synthesis flows for asynchronous circuits try to directly leverage synchronous EDA tools [6] [78]. As the circuit structure, performance metric and timing constraints for asynchronous circuits are quite different from those for synchronous circuits, these approaches require to break the timing-loops and add explicit timing constraints the number of which is exponential to the circuit size. For large scale designs, the complicated timing constraints are beyond the ability of synchronous EDA tools to handle thus inferior results are generated. In [30], a genetic algorithm based simultaneous gate sizing and V_{th} assignment technique specific for asynchronous circuits has been proposed to minimize the leakage power while maintaining the performance requirements. However, genetic algorithms usually have long runtime and are not scalable, which makes it unsuitable for large scale circuits.

A fast and accurate static timing analysis (STA) method is essential to guide the gate selection algorithm to achieve a good solution within a short amount of runtime. For synchronous circuits, this can be done by a simple graph traversal as the corresponding combinational logic network can be represented as a directed acyclic graph (DAG). However, for asynchronous circuits, the way to perform STA is not straightforward due to its more general circuit structure which might contain internal combinational loops. In [66], a STA flow on pre-charged half

المنارة للاستشارات
www.manaraa.com

buffer (PCHB) and Multi-Level Domino (MLD) templates has been proposed, which leverages a commercialized synchronous timing analyzer. However, this approach is limited to template based designs as automatically finding the cut points requires a regular circuit structure. Also, the achieved timing value is not accurate as time borrowing across the broken segments is not allowed.

This paper presents a new simultaneous gate sizing and V_{th} assignment approach for asynchronous circuits. We formulate the gate selection problem to minimize both the leakage power and cycle time while satisfying various type of asynchronous timing constraints. A Lagrangian relaxation framework is applied on the formulated problem to transform it into a sequence of Lagrangian relaxation subproblems (LRS). In particular, the arrival time based linear constraints allow us to simplify LRS using Karush-Kuhn-Tucker (KKT) conditions [5]. This simplified LRS can then be easily solved using an effective greedy algorithm. The proposed gate selection approach considers discrete cell sizes and threshold voltages from the standard cell library, and is implemented based on the accurate non-linear delay model (NLDM). In addition, to overcome the obstacle of STA for asynchronous circuits, we propose an iterative slew update algorithm which is accurate and guarantees fast convergence with library-based timing models.

We evaluate our flow using a set of asynchronous designs based on the PCHB templates and compare it with a latest asynchronous design flow Proteus [6]. Our flow is shown to be consistently better and we have achieved significant improvements in both the cycle time and leakage power. Our approach is more effective than those which twist and trick synchronous EDA tools to generate a functional circuit as it directly handles timing loops, which means time borrowing along the loop is allowed and the number of constraints is polynomial in circuit size.

The rest of this paper is organized as follows. In Section II, we give an overview about the synchronous gate selection techniques. Several timing issues related to asynchronous circuits are discussed and the gate selection problem is then formulated. In Section III, a Lagrangian relaxation based approach is presented to solve the gate selection problem. In Section IV, we discuss the proposed STA approach for asynchronous circuit. In Section V, we summarize the implemented asynchronous gate selection flow.

المنارة للاستشارات
www.manaraa.com

5.2 Preliminaries

5.2.1 Gate Selection Techniques for Synchronous Circuits

The discrete gate sizing problem is proved to be NP-hard [58]. Therefore, various heuristic algorithms like convex programming [70], sensitivity based algorithms [33] and so on have been proposed by researchers to assign proper sizes and threshold voltages for gates in synchronous circuits. There are even organized gate sizing contests [59] [60] to help expose the challenges faced in modern industrial designs to the academic field. One powerful heuristic approach adopted by leading synchronous gate selection algorithms [27] [48] is to apply the Lagrangian relaxation (LR) technique. In [14], foundations for LR-based gate sizing approach is first established, which considers continuous sizing and simple delay models. The LR-based approach is then continuously got improved as people are combining it with library-based timing model, discrete gate sizing, V_{th} assignment, dynamic programming and network flow algorithms [27] [48] [61] [83]. The advantage of LR-based approach is that it can be easily modified to handle different objectives and various complex design constraints. Even though convexity cannot be claimed for the discrete gate sizing problem, the LR-based approach is shown to have fast convergence in practice and is practical to large scale problems. Although extensive research has been done for the LR-based gate selection algorithms of synchronous circuits, whether it is applicable and effective to asynchronous circuits is still not being explored.

5.2.2 Full Buffer Channel Net Model

Here we use the Full Buffer Channel Net (FBCN) [8] to model our asynchronous circuits. A FBCN is a specific form of timed marked graph. The idea is to model each leaf cell as a *transition* and asynchronous channels between cell ports are modeled with a pair of *places* which are annotated with delay information. Please note that here we treat the asynchronous circuits as unconditional. For conditional asynchronous circuits modeled using FBCN, the circuit performance can be guaranteed conservatively as proved in [53].

As an example, a simple ALU design is shown in Fig. 3.1 and the corresponding marked graph based on the FBCN model is shown in Fig. 3.2. Here, t_{mul1} and t_{mul2} represent the

المنارة للاستشارات
www.manaraa.com

two-stage multiplication cells and t_{add} represents the addition cell. All the places are denoted as circles. In particular, places containing tokens are represented by circles marked with a black dot. Two channels on the left are assumed to be in the full state and have tokens assigned on the forward places. The rest of the channels are assumed to be empty and have tokens assigned on the backward places.

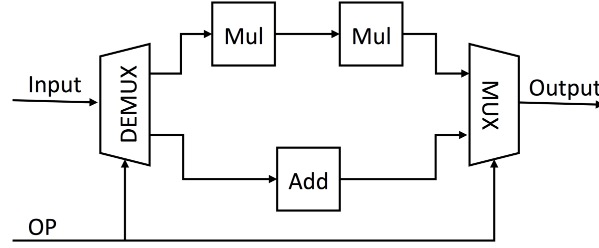

Figure 5.1: Asynchronous ALU.

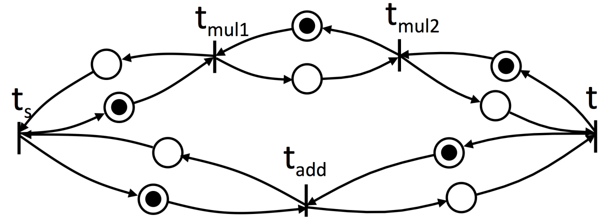

Figure 5.2: Marked graph representation for Asynchronous ALU.

5.2.3 Asynchronous Performance Analysis

There are many existing algorithms which are able to capture the cycle metric of asynchronous circuits. In this paper, we adopt a linear programming based approach [49], as it can be easily incorporated in to our Lagrangian relaxation framework.

For any asynchronous circuit modeled with FBCN, the cycle time τ can be obtained by solving the following linear program, where a_i and a_j are the arrival time associated with transitions t_i and t_j.

$$\text{Minimize} \quad \tau$$

$$\text{Subject to} \quad a_i + D_{ij} - m_{ij}\tau \leq a_j \quad \forall (i,j)$$

المنارة للاستشارات
www.manaraa.com

D_{ij} is the delay associated with a place p between two neighboring transitions t_i and t_j. $m_{ij} = 1$ if the corresponding place p contains a token and 0 otherwise.

Please note that in practice, there can be multiple delay values such as the rising and falling delay associate with a place p, The different delay values can be considered by simply extending the existing constraints. Here we ignore these details in order to make our presentation more concise.

5.2.4 Asynchronous Timing Constraints

Compared with synchronous circuits which only have setup and hold time constraints, asynchronous circuits can have some totally different timing constraints depending on the timing assumptions made by the specific asynchronous logic implementation style. For the bounded-delay asynchronous designs [76], two-sided timing constraints need to be enforced which require both a minimum and maximum allowable delay of a specified gate or wire. Instead, for the quasi-delay-insensitive (QDI) design style such as WCHB, PCHB, MLD template, relative delay constraints referred to as *relative timing* [74] need to be enforced, which dictate the relative delay of two paths that stem from a common point of divergence. These two design categories cover most of the existing timing constraints specific to asynchronous circuits. We will show how to incorporate these constraints into our problem formulation in Sec. II E.

5.2.5 Asynchronous Gate Sizing and Vth Assignment

In this subsection, we formulate the asynchronous gate selection problem considering the performance and timing constraints. For an asynchronous circuit modeled with FBCN, let T be the set of transitions and P be the set of places in the timed marked graph. In particular, we use $p(i, j)$ to denote the place between neighboring transitions t_i and t_j. Let $\boldsymbol{a} = \{a_1, a_2, ..., a_{|T|}\}$ be the set of arrival times corresponding to T. Let $\boldsymbol{g} = \{g_1, g_2, ..., g_{|T|}\}$ be the set of gates corresponding to T and we use v_i^j to represent a specific selected version j for gate g_i. Let $\boldsymbol{g_0}$ be the initial set of selected gates before optimization and τ_0 be its corresponding cycle time. In addition, we use P_b to denote the set of places annotated with two-sided delay bounds. We use P_{rt} to denote the set of places annotated with relative timing constraints. Then the problem of

المنارة للاستشارات
www.manaraa.com

minimizing both total leakage power consumption and cycle time subject to timing constraints can be formulated as:

$$\text{Minimize} \quad \text{leakage}(\boldsymbol{g})/\text{leakage}(\boldsymbol{g_0}) + \alpha\tau/\tau_0$$

$$\text{Subject to} \quad a_i + D_{ij} - m_{ij}\tau \leq a_j \qquad \forall\, p(i,j) \in P \tag{5.1}$$

$$L_{ij} \leq a_j - a_i \leq U_{ij} \qquad \forall\, p(i,j) \in P_b \tag{5.2}$$

$$|(a_i - a_k) - (a_j - a_k)| \leq I_{ij}\ \forall\, p(i,j) \in P_{rt} \tag{5.3}$$

where the constant α can be chosen to adjust the tradeoff between minimizing the normalized leakage power and the normalized cycle time. L_{ij} and U_{ij} denote the minimum and maximum bounded delay. I_{ij} denotes the relative delay stemming from transition t_k and forking into two transitions t_i and t_j. leakage($\boldsymbol{g}$) captures the summation of leakage power for the set of gates $\boldsymbol{g}$ with selected versions.

We can rewrite the timing constraints in Equations (2) and (3) into the same form with the performance constraints in Equation (1) as follows:

$$(a_i + L_{ij} \leq a_j) \wedge (a_j - U_{ij} \leq a_i) \tag{5.4}$$

$$(a_j - I_{ij} \leq a_i) \wedge (a_i - I_{ij} \leq a_j) \tag{5.5}$$

Then, combining Equation (1) with the reformulated Equations (4) and (5), we can get a more concise representation of our primal problem:

$$\mathcal{PP}: \quad \text{Minimize} \quad \text{leakage}(\boldsymbol{g})/\text{leakage}(\boldsymbol{g_0}) + \alpha\tau/\tau_0$$

$$\text{Subject to} \quad a_i + \hat{D}_{ij} - \hat{m}_{ij}\tau \leq a_j \quad \forall(i,j)$$

where $\hat{D}_{ij}$ represents D_{ij}, L_{ij}, $-U_{ij}$ or $-I_{ij}$ depending on the corresponding places annotated with performance or timing constraints. Also, we have $\hat{m}_{ij} = m_{ij}$ for all performance constraints and $\hat{m}_{ij} = 0$ for all timing constraints. $\forall(i,j)$ represents all the i, j pairs corresponding to $p(i,j) \in P \cup P_b \cup P_{rt}$.

www.manaraa.com

5.3 Simultaneous Gate Sizing and Vth Assignment by Lagrangian Relaxation

In this section, we propose our LR-based approach to solve the formulated asynchronous gate selection problem $\mathcal{PP}$. In Sec. III A, the Lagrangian relaxation subproblem ($\mathcal{LRS}$) which provides a lower bound to the solution of $\mathcal{PP}$ is obtained by applying LR technique to $\mathcal{PP}$. In Sec. III B, we first simplify $\mathcal{LRS}$ using applying KKT conditions. Then, an effective greedy algorithm is proposed Sec. III C to solve this simplified $\mathcal{LRS}$. In Sec. III D, we solve the Lagrangian dual problem ($\mathcal{LDP}$) to achieve a solution of $\mathcal{PP}$ by iteratively solving a sequence of simplified $\mathcal{LRS}$.

5.3.1 Lagrangian Relaxation Subproblem ($\mathcal{LRS}$)

First, we apply Lagrangian relaxation to the primal problem. We attach a set of nonnegative Lagrangian multipliers $\boldsymbol{\lambda} = \{\lambda_{ij} \mid \forall(i,j)\}$ to all the constraints in $\mathcal{PP}$ and relax these constraints into the objective function. Then the Lagrangian relaxation subproblem we get is:

$$\begin{aligned}\mathcal{LRS}: \quad \text{Mimimize} \quad & \text{leakage}(\boldsymbol{g})/\text{leakage}(\boldsymbol{g_0}) + \alpha\tau/\tau_0 \\ & + \sum_{\forall(i,j)} \lambda_{ij}(a_i + \hat{D}_{ij} - \hat{m}_{ij}\tau - a_j)\end{aligned}$$

The relaxed problem becomes an unconstrained optimization problem. Please note that the variables we have for $\mathcal{LRS}$ are $\boldsymbol{g}$, $\boldsymbol{a}$ and τ while $\boldsymbol{\lambda}$ is a given parameter. For any given set of $\boldsymbol{\lambda} \geq \mathbf{0}$, solving $\mathcal{LRS}$ will provide a lower bound to the optimal solution of $\mathcal{PP}$ [5].

5.3.2 Simplified Lagrangian Relaxation Subproblem ($\mathcal{LRS}^*$)

Similar to [14], we rearrange terms here and the $\mathcal{LRS}$ can be rewritten as:

$$\begin{aligned}\text{Mimimize} \quad & \text{leakage}(\boldsymbol{g})/\text{leakage}(\boldsymbol{g_0}) + (\alpha - \sum_{\forall(i,j)} \lambda_{ij}\hat{m}_{ij})\tau/\tau_0 \\ & + \sum_{k\in T}(\sum_{\forall(k,j)} \lambda_{kj} - \sum_{\forall(i,k)} \lambda_{ik})a_k \\ & + \sum_{\forall(i,j)} \lambda_{ij}\hat{D}_{ij}\end{aligned}$$

المنارة للاستشارات
www.manaraa.com

The idea behind this rearrangement is to group all the coefficients associated with cycle time variable τ and arrival time variables $\boldsymbol{a}$, which make them easier to be removed as we will show in the next step.

Let $\mathcal{L}(\boldsymbol{g}, \boldsymbol{a}, \tau)$ be the objective function of $\mathcal{LRS}$. The KKT stationarity conditions imply $\partial\mathcal{L}/\partial a_i = 0$ for $1 \leq i \leq |T|$ and $\partial\mathcal{L}/\partial\tau = 0$ at the optimal solution of the primal problem. Then we can get the following optimality conditions:

$$\mathcal{KKT}: \qquad \alpha = \sum_{\forall(i,j)} \lambda_{ij}\hat{m}_{ij}$$

$$\sum_{\forall(k,j)} \lambda_{kj} = \sum_{\forall(i,k)} \lambda_{ik} \quad \forall\, k \in T$$

Apply the optimality conditions into $\mathcal{LRS}$, we can obtain a simplified Lagrangian relaxation subproblem as follows:

$$\mathcal{LRS}^*: \quad \text{Minimize} \quad \text{leakage}(\boldsymbol{g})/\text{leakage}(\boldsymbol{g_0}) + \sum_{\forall(i,j)} \lambda_{ij}\hat{D}_{ij}$$

After the simplification, variables τ and $\boldsymbol{a}$ are removed. The only variables left are the set of selected gates $\boldsymbol{g}$ associated with transitions. It can be seen that $\mathcal{LRS}^*$ is equivalent to $\mathcal{LRS}$ and it is much easier to solve.

5.3.3 Solving $\mathcal{LRS}^*$

Algorithm 1 shows our algorithm to solve $\mathcal{LRS}^*$. First, we assign an initial version to each gate and insert all the gates into a set $\mathcal{G}$. Then, we pick any gate g_i from $\mathcal{G}$ and use Algorithm 2 to find a better version for it, i.e., picking a different size or threshold voltage for that gate. If the selected new version v_i^k of g_i is different from its old version v_i^j, we assign v_i^k to the gate and insert all its fanout gates not in $\mathcal{G}$ into $\mathcal{G}$. Otherwise, we do not reevaluate its downstream cells if they are not in $\mathcal{G}$. In particular, if the gate has been visited more than a certain number (n) of times, we also do not reevaluate its downstream cells in order to save runtime. The algorithm terminates when $\mathcal{G}$ is empty.

The algorithm to select a new gate version is shown in Algorithm 2. For each possible version v_i^j of a specific gate g_i, we first update the gate to this new version.

المنارة للاستشارات
www.manaraa.com

Algorithm 2 Solve $\mathcal{LRS}^*$

Ensure: a proper version for each gate which minimize $\mathcal{LRS}^*$
1: Initially assign all the gates with a version;
2: Insert all the gates into a set $\mathcal{G}$;
3: **while** $\mathcal{G} \neq \emptyset$ **do**
4: Pick one gate g_i from $\mathcal{G}$. Let its current version be v_i^j;
5: Select a new version v_i^k for gate g_i; /* Algorithm 2 */
6: **if** $v_i^j \neq v_i^k$ **then**
7: Assign g_i with this new version v_i^k;
8: **if** g_i is visited less than or equal to n times **then**
9: Insert all gates $\notin \mathcal{G}$ and directly driven by g_i into $\mathcal{G}$;
10: **end if**
11: **end if**
12: Remove g_i from set $\mathcal{G}$;
13: **end while**

Next, we need to estimate the timing impact made by this gate version change. Instead of doing STA for the entire circuit which can be very time consuming, we perform a local timing update here. In particular, we update the output load of all the fanin gates ($fanin(g_i)$) which is driving g_i and the input slew of all the fanout gates ($fanout(g_i)$) which is driven by g_i. We also update the input slew of all the side gates ($side(g_i)$) which are defined as all the gates driven by $fanin(g_i)$ except g_i. Then we recompute the delay for all the timing arcs associated with g_i and its fanin, fanout and side gates.

Let $Arc_i = \text{timingArcs}(g_i \cup fanin(g_i) \cup fanout(g_i) \cup side(g_i))$ be the set of updated timing arcs. After the local timing update, we can evaluate the cost to objective value of $\mathcal{LRS}^*$ as follows:

$$\text{Cost}(g_i) = \text{leakage}(g_i) + \sum_{(u,v) \in Arc_i} \lambda_{uv} \hat{D}_{uv}$$

After all the possible options for the current gate have been evaluated, the one providing the minimum cost will be returned as the best choice.

5.3.4 Lagrangian Dual Problem ($\mathcal{LDP}$)

In Sec. III A, we mention that a lower bound to the optimal solution of $\mathcal{PP}$ can be obtained by solving $\mathcal{LRS}$ for any given set of $\boldsymbol{\lambda} \geq \mathbf{0}$.

المنارة للاستشارات
www.manaraa.com

Algorithm 3 Gate Version Selection

Ensure: Best version for the gate g_i which minimize $\mathcal{LRS}^*$
1: **for each** available option v_i^j for the gate g_i **do**
2: Assign v_i^j to g_i;
3: Local timing update;
4: **if** Cost(g_i) < bestCost **then**
5: bestVersion = v_i^j;
6: bestCost = Cost(g_i);
7: **end if**
8: **end for**
9: **return** bestVersion;

Now we discuss how to find the specific $\boldsymbol{\lambda}$ that gives us the maximum (i.e., tightest) lower bound. It is formulated as the Lagrangian dual problem as follows:

$$\mathcal{LDP}: \quad \text{Maximize} \quad \mathcal{LRS}$$
$$\text{Subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0}$$

Please note that $\boldsymbol{g}$, $\boldsymbol{a}$, τ along with $\boldsymbol{\lambda}$ are all variables for the Lagrangian dual problem. Solving $\mathcal{LDP}$ will provide the best solution to the primal problem.

Instead of maximizing $\mathcal{LRS}$, we want to incorporate the optimality conditions and maximize the equivalent yet simpler problem $\mathcal{LRS}^*$. Thus, the Lagrangian dual problem can be rewritten as:

$$\text{Maximize} \quad \mathcal{LRS}^*$$
$$\text{Subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0},\ \boldsymbol{\lambda} \in \mathcal{KKT}$$

5.3.5 Solving $\mathcal{LDP}$

$\mathcal{LDP}$ can be solved by iteratively solving a sequence of $\mathcal{LRS}^*$. A commonly used strategy is the subgradient optimization method [5]. However, this method requires a projection for $\boldsymbol{\lambda}$ after each iteration in order to maintain $\boldsymbol{\lambda}$ within the feasible region of $\mathcal{LDP}$. For synchronous circuits, this can be done by simply traversing the circuit in topological order. For asynchronous circuits, it will not be easy to redistribute $\boldsymbol{\lambda}$ as the corresponding circuit structure contains loops. In addition, achieving a good convergence using this subgradient optimization method is difficult and usually requires a careful choice of initial solution and step size.

www.manaraa.com

To resolve these issues, we apply a direction finding approach inspired by [83] to solve $\mathcal{LDP}$, which is shown to have better convergence and no projection is needed.

Let $q(\boldsymbol{\lambda})$ denotes the optimal objective value of $\mathcal{LRS}^*$ for a given set of $\boldsymbol{\lambda}$. The direction finding approach wants to find an improving feasible direction $\Delta\boldsymbol{\lambda}$ and a step size β such that at each step we have:

$$q(\boldsymbol{\lambda} + \beta\Delta\boldsymbol{\lambda}) > q(\boldsymbol{\lambda})$$

In particular, the improving feasible direction $\Delta\boldsymbol{\lambda}$ can be found by solving the following linear program:

$$\begin{aligned} \mathcal{DF}: \quad & \text{Maximize} \quad \sum_{\forall(i,j)} \Delta\lambda_{ij}\hat{D}_{ij} \\ & \text{Subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0},\ \boldsymbol{\lambda} \in \mathcal{KKT} \\ & \qquad\qquad\quad \max(-u, -\lambda_{ij}) \leq \Delta\lambda_{ij} \leq u \end{aligned}$$

where u is used to bound the objective function and avoid it goes to infinity, similar to [83].

After we find the improving feasible direction, the step size β can be obtained by optimizing along this direction using any line search technique.

The detailed algorithm to solve $\mathcal{LDP}$ is presented in Algorithm 3. It starts from an initial dual feasible $\boldsymbol{\lambda}$, which is non-negative and satisfies the optimality conditions. Then the method iteratively improves $q(\boldsymbol{\lambda})$ by finding an improving direction and performing a line search to find the best step size. The algorithm terminates when $q(\boldsymbol{\lambda})$ is not improving or the total number of iterations exceeds the limit.

5.4 Static Timing Analysis for Asynchronous Circuits

An efficient asynchronous STA method is necessary for us to compute the delay values and cycle time τ using library-based timing model. In order to do the STA, we first find the output slew values of each gate using an iterative slew rate update approach described in Sec. IV A. Then, $\hat{D}_{ij}$ can be achieved by lookup table interpolation in the same manner as synchronous STA. Finally, the cycle time can be computed using the linear program described in Sec. II C.

المنارة للاستشارات
www.manaraa.com

Algorithm 4 Solve $\mathcal{LDP}$

Ensure: $\boldsymbol{\lambda}$ which maximizes $\mathcal{LRS}^*$

1: $n = 1$; /* loop counter */
2: $\boldsymbol{\lambda}$ = initial non-negative value satisfy optimality conditions;
3: **while** $n < limit$ **do**
4: Solve $\mathcal{DF}$ to obtain improving direction $\Delta\boldsymbol{\lambda}$;
5: **while** line search not terminate **do**
6: Compute β based on specific line search technique;
7: $\boldsymbol{\lambda}' = \boldsymbol{\lambda} + \beta\Delta\boldsymbol{\lambda}$;
8: Solve $\mathcal{LRS}^*$ to obtain $q(\boldsymbol{\lambda}')$;
9: **if** $q(\boldsymbol{\lambda}') > bestObj$ **then**
10: $bestStep = \beta$;
11: $bestObj = q(\boldsymbol{\lambda}')$;
12: **end if**
13: **end while**
14: **if** $bestObj \leq q(\boldsymbol{\lambda})$ **then** /* $q(\boldsymbol{\lambda})$ is not improving */
15: **exit loop**;
16: **end if**
17: $\boldsymbol{\lambda} = \boldsymbol{\lambda} + bestStep * \Delta\boldsymbol{\lambda}$; /* move one step further */
18: Solve $\mathcal{LRS}^*$;
19: $n = n + 1$;
20: **end while**

5.4.1 Iterative Slew Update Approach

The algorithm to implement the iterative slew update approach is presented as Algorithm 4. It is similar to Algorithm 1 which solves $\mathcal{LRS}^*$. Here, we also keep a set $\mathcal{G}$ of gates. A gate is in $\mathcal{G}$ if its current output slew is potentially *inconsistent* with its current input slews. In particular, we define an output slew to be *inconsistent*, if the resulting output slew might be larger than the current one when we evaluate it based on the current input slews. We define it to be "larger than" as we are trying to find an upper bound of all the slews.

In the beginning, we initialize the output slew of all gates to 0. Thus, all gates should be in the set $\mathcal{G}$ because all of them are potentially *inconsistent*. In each step, we pick any gate g_i from the set and update its output slew. If the new output slew s_{new} is larger than the current one s_{old}, we update the output slew of g_i to s_{new} and put all gates driven by this gate while not in $\mathcal{G}$ into the set $\mathcal{G}$. When $\mathcal{G}$ is empty, i.e., the output slews of all gates are *consistent*, the algorithm terminates.

www.manaraa.com

Algorithm 5 Iterative Slew Update

Ensure: A tight upper bound of the output slew for all the gates;
1: Initialize the output slew to 0 for all the gates;
2: Insert all the gates into a set $\mathcal{G}$;
3: **while** $\mathcal{G} \neq \emptyset$ **do**
4: Pick one gate g_i from $\mathcal{G}$. Let its current output slew be s_{old};
5: Compute new output slew s_{new} of g_i based on its input slew;
6: **if** $s_{new} > s_{old}$ **then**
7: Update the output slew of g_i to s_{new};
8: Insert all gates $\notin \mathcal{G}$ and directly driven by g_i into $\mathcal{G}$;
9: **end if**
10: Remove g_i from set $\mathcal{G}$;
11: **end while**

5.4.2 Convergence of the proposed approach

We can easily use induction technique to prove that the output slew of every gate will be monotonically increasing throughout the execution of Algorithm 4. Since all the slew values are upper-bounded, we know the proposed algorithm always converges. Let $\mathcal{S} = \{s_1, ..., s_{|T|}\}$ be the set of slew values computed by the proposed algorithm. Then, we can have the following theorem:

Theorem 1. For each gate, its corresponding output slew value in $\mathcal{S}$ is a tight (i.e., smallest) upper bound of all its output slew values during the operation of the circuit.

Proof: We prove this using contradiction. Assume the slew values computed by Algorithm 4 are not tight, which means there exists gates whose corresponding slew value in $\mathcal{S}$ is larger than its maximum achievable slew value during the circuit operation. Let g_i be the first gate during the iterative slew evaluation process such that its output slew is set to s_i which is larger than its maximum achievable slew value. Based on the slew evaluation process, we know the output slew of g_i is computed based on the output slew of one of its fanin gates g_j. Since s_i is unachievable, the current output slew s_j of gate g_j must also be unachievable. This makes g_i not the first gate the output slew of which is set to an unachievable value, which contradicts our assumption. Therefore, we can conclude that the computed slew in $\mathcal{S}$ is a tight upper bound of all the slew values. $\square$

المنارة للاستشارات
www.manaraa.com

5.4.3 Extension to a tight lower bound

A tight upper bound for all the slew values will guarantee the design satisfies performance constraints as shown in Equation (1). However, for other type of constraints such as the two-sided timing constraints in Equation (2), we might need a tight low bound in order to conservatively satisfy them. This can be achieved by a simple modification of Algorithm 4. Instead of setting all the slew values to 0 in the beginning, here we initialize all of them to the maximum possible slew value in the cell timing library. Then at line 6, we change the condition to $s_{new} < s_{old}$, which makes the output slew of every gate monotonically decrease throughout the execution. Similar to the proof of Theorem 1, the modified algorithm will give us a tight lower bound of all the slew values.

5.5 Asynchronous Gate Selection Flow

We summarize our flow for asynchronous gate sizing and V_{th} assignment in Fig. 3.3.

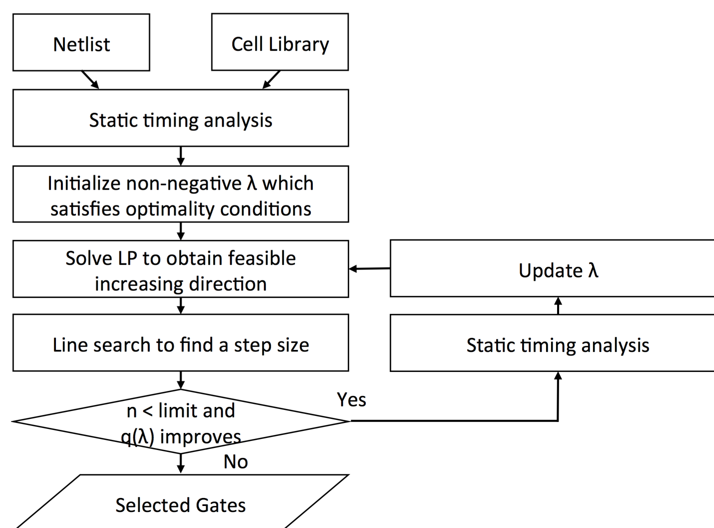

Figure 5.3: Asynchronous Gate Selection Flow.

The input to our flow is the characterized cell timing library and the netlist. Initially, STA is run for the whole circuit to update the timing and an initial set of $\boldsymbol{\lambda} \geq \mathbf{0}$ satisfying the optimality conditions is found. Next, we enter the loop to solve the $\mathcal{LDP}$ similar to Algorithm 3. The final output of our flow is a gate sizing and V_{th} assignment solution with both cycle time and leakage power being minimized.

www.manaraa.com

5.6 Experiments

The proposed gate sizing approach is implemented in C++ and runs on a Linux PC with 8 GB of memory and 2.4 GHz Intel Core i7 CPU.

We are using the Proteus standard cell library [6] which is based on an implementation of the PCHB template. Cell delay and slew values are calculated using the static timing analysis method described in Sec. IV with the accurate non-linear delay model lookup tables from the cell timing library. Cell interconnections are modeled as simple lumped capacitance in our experiment. The lumped capacitance value is obtained from the SPEF file generated by Proteus flow after placement and routing. As the leakage power is not available in Proteus standard cell library, we assign it to be proportional to cell area, which is the same strategy used in ISPD 2013 discrete gate sizing contest benchmarks [60].

In order to show the effectiveness on the power saving after applying the V_{th} assignment techniques, we also extends the Proteus standard cell library into a multi-V_{th} library, as the original library only considers single V_{th}. We scale the leakage power value and look up tables according the ratio between different V_{th} cells in the cell library provided by the ISPD 2013 gate sizing contest. In particular, based on the original cells, we generate a set of low threshold voltage cells with 4X more leakage power but 0.9X smaller delay. Similarly, we generate a set of high threshold voltage cells with 0.25X leakage power but 1.15X larger delay.

We evaluate our approach using two sets of benchmarks. First is a set of asynchronous designs transformed from ISCAS89 benchmarks, which have flip-flops mapped as token buffers and combinational gates mapped as logic cells using Proteus. We also have a set of specific asynchronous designs developed using Verilog and synthesized into gate level netlist using Proteus. Different bit widths are applied to some of the benchmarks to create designs with different sizes.

We need to set the parameter α, which is the tradeoff between leakage power consumption and circuit performance. As our flow starts optimization with all cells assigned with the minimum cell size which have the minimum possible leakage power, we set $\alpha = 2$ to put a similar effort on optimizing power and cycle time of the given circuit. The *limit* for the total number

www.manaraa.com

of iterations is set to be 50, which is shown to be able to provide enough improvement within a short amount of runtime. We do not perform any benchmark specific parameter tuning during our experiment. After running the gate selection flow, we run our STA algorithm until convergence to measure the cycle time and the leakage power consumption of our gate selection result.

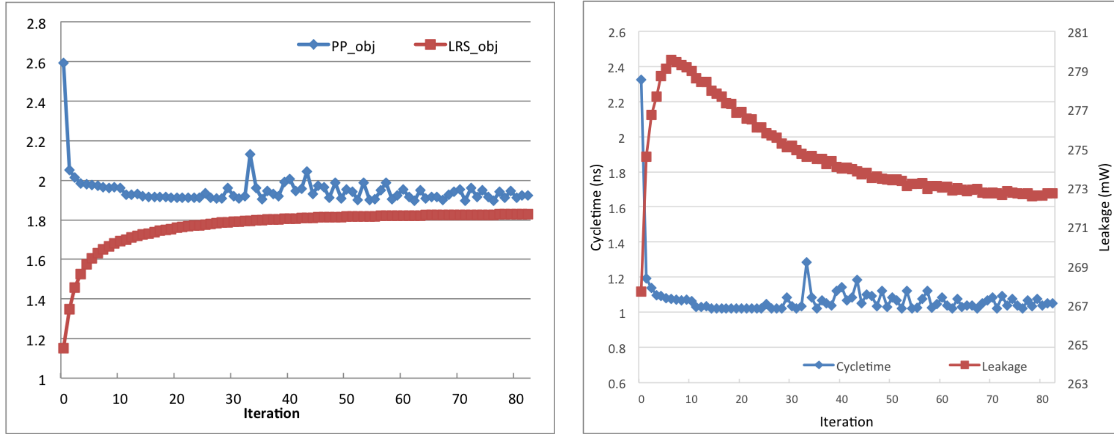

Figure 5.4: (a) The convergence sequence of s38417. (b) Cycle time and leakage power trends of s38417.

First, we do the experiment by running our flow using the original single-V_{th} library in order to compare with Proteus. Both flows use the same input netlist starting with cells assigned with minimum size. Comparison results on the transformed ISCAS89 benchmarks are shown in Table 3.1. "Total itr." column shows the total number iterations performed for each benchmarks until termination. PP_{obj} denotes the achieved objective value for the primal problem. LRS_{obj} denotes the achieved objective value for the $\mathcal{LRS}$. "Gap" column shows the duality gap which helps us to know how close our solution is to the optimal solution. The duality gap here is calculated using $(PP_{obj} - LRS_{obj})/PP_{obj}$. "Init" columns show the initial cycle time and leakage power, which all the cells are assigned with the minimum size. "Proteus" and "Ours" columns show the cycle time and leakage power for gate sizes generated by the Proteus flow and our flow respectively. The results show our approach is consistently better in both cycltime and power consumption for all benchmarks. For the cycle time, on average, our approach is 2.19X better than the initial minimum sized circuit and 21.3% better than the gate sizing results from Proteus. For the leakage power consumption, our approach is 9.5% better

المنارة للاستشارات
www.manaraa.com

than Proteus and only consumes 3.7% more power than the initial minimum sized circuits, which have the minimum possible leakage value. Table 3.2 shows the comparison results on the specific asynchronous benchmarks and have achieved similar improvements. Proteus does not have a separate gate sizing step and makes us not able to measure the time it spends only for gate sizing. On average, the runtime of our algorithm is around 5 minutes, which is less than 10% of the total runtime of the entire Proteus flow. This indicates our flow runs fast enough and will not be a runtime bottleneck during the entire design process.

Next, we run our flow using the expanded multi-V_{th} library. The results are shown in the "Multi-Vt" columns in Table 3.1 and Table 3.2. In Table 3.1, compared with running our flow on the single-V_{th} library, the V_{th} assignment technique using multi-V_{th} library can save 70.5% of total leakage power consumption with only 2.4% increase on the cycle time. Similar improvement is shown in Table 3.2.

The convergence sequence of our largest circuit s38417 is shown in Fig. 3.4(a), which is generated by running the flow on single-V_{th} library without limiting the total number of iterations. The corresponding changes of cycle time and leakage power at each iteration is shown in Fig. 3.4(b). It can be seen that our algorithm converges smoothly and the final results are very close to the optimal solution. It also shows that the improvement after 50th iteration is small and suggests that terminating the algorithm earlier can save a large amount of runtime without sacrificing the quality too much.

5.7 Conclusions

This paper proposes a gate selection flow for asynchronous circuits. To solve the gate selection problem, we incorporate the linear-programming-based performance evaluation method with the Lagrangian relaxation framework. The relaxed problem is simplified and the $\mathcal{LRS}$ can then be easily solved. We also proposed an iterative slew updating approach for the static timing analysis of asynchronous circuits. Our STA approach is simple yet effective which can directly handle timing loops. We compared our approach with an asynchronous design flow which is leveraging synchronous EDA tools and significant improvement is achieved.

المنارة للاستشارات
www.manaraa.com

Table 5.1: Comparison on transformed ISCAS89 benchmarks

Design	Total Itr.	PP_{obj}	LRS_{obj}	Gap	Cycle Time (ns)				Leakage (μW)			
					Init	Proteus	Ours	Multi-Vt	Init	Proteus	Ours	Multi-Vt
as27	12	2.204	2.183	0.96%	0.62	0.41	0.35	0.38	454.12	547.43	483.15	139.45
as298	20	2.104	2.040	3.04%	0.85	0.48	0.44	0.46	3111.09	3837.54	3322.60	1025.05
as344	24	2.101	2.051	2.39%	1.14	0.65	0.59	0.61	4044.90	5448.73	4266.78	1331.77
as386	21	2.027	1.999	1.42%	1.14	0.67	0.54	0.58	3574.20	4435.43	3875.56	1127.87
as349	43	2.032	2.005	1.32%	1.20	0.65	0.59	0.60	4052.51	5153.59	4261.94	1321.23
as382	19	2.085	1.996	4.30%	0.94	0.58	0.49	0.50	3950.90	4713.29	4159.64	1247.96
as400	25	2.032	1.992	1.99%	1.00	0.59	0.49	0.51	4228.76	5163.96	4447.18	1398.12
as420	15	2.114	2.047	3.14%	0.74	0.43	0.39	0.41	4354.56	5563.47	4673.89	1415.92
as444	19	2.015	1.963	2.56%	0.98	0.58	0.47	0.49	4008.27	4716.75	4209.41	1270.43
as510	28	2.028	1.980	2.39%	1.41	0.74	0.66	0.69	7565.18	9401.01	8269.52	2435.62
as526	16	2.029	1.946	4.08%	0.99	0.58	0.48	0.50	4877.80	5951.92	5162.57	1558.83
as641	50	1.916	1.891	1.31%	1.36	0.65	0.61	0.61	7957.79	11109.70	8178.28	2488.67
as713	31	1.915	1.863	2.72%	1.21	0.64	0.53	0.54	7173.27	8931.69	7427.64	2228.43
as820	19	1.902	1.845	2.99%	1.63	0.80	0.68	0.69	9646.39	12654.50	10347.30	3174.51
as832	27	1.918	1.860	3.02%	1.69	0.76	0.73	0.73	10178.60	13679.50	10795.90	3365.11
as838	16	2.161	2.015	6.76%	0.86	0.55	0.47	0.47	10450.30	12322.00	11290.10	3541.02
as953	26	1.918	1.866	2.69%	1.57	0.73	0.67	0.69	14065.20	18748.80	14962.40	4586.63
as1196	20	2.028	1.938	4.43%	0.78	0.45	0.37	0.39	21147.30	26729.40	22739.80	6596.47
as1488	31	1.867	1.824	2.33%	1.96	0.92	0.77	0.81	21038.70	27106.10	22678.30	6611.33
as1238	18	2.013	1.935	3.89%	0.77	0.44	0.36	0.39	21815.70	26416.30	23438.60	6483.97
as1423	44	2.021	1.944	3.81%	1.94	1.04	0.95	0.95	18684.50	22470.90	19514.60	5901.81
as5378	42	1.964	1.866	4.95%	1.19	0.57	0.54	0.55	40032.20	52317.60	42443.10	12548.60
as9234	39	1.880	1.780	5.35%	1.69	0.82	0.71	0.70	32586.60	40344.70	33821.10	9963.48
as13207	44	1.856	1.697	8.58%	1.68	0.83	0.68	0.70	86949.50	99875.60	90722.80	26414.00
as15850	50	1.831	1.691	7.69%	2.44	1.29	0.99	0.96	105889.00	123374.00	107923.00	31622.60
as38417	50	1.901	1.813	4.66%	2.32	2.04	1.02	1.01	267671.00	267062.00	273556.00	80887.90
			Normalized	3.57%	2.192	1.213	1.000	1.024	0.963	1.095	1.000	0.295

المنارة للاستشارات
www.manaraa.com

Table 5.2: Comparison on asynchronous benchmarks

Design	Total Itr.	PP_{obj}	LRS_{obj}	Gap	Cycle Time (ns)				Leakage (μW)			
					Init	Proteus	Ours	Multi-Vt	Init	Proteus	Ours	Multi-Vt
ALU8	38	2.100	2.010	4.32%	0.68	0.40	0.35	0.38	14849.70	20252.20	15668.10	4223.92
ALU16	16	1.988	1.848	7.08%	0.84	0.78	0.39	0.40	41103.60	41017.20	43402.50	12194.50
ALU32	17	1.762	1.605	8.87%	1.26	1.18	0.45	0.50	118062.00	118125.00	123546.00	34077.00
ALU64	19	1.848	1.567	15.18%	1.38	1.25	0.55	0.59	493119.00	494252.00	520688.00	144837.00
ACC16	27	2.114	2.073	1.91%	0.99	0.65	0.50	0.55	7198.16	8565.35	7945.34	2216.16
ACC32	50	2.092	1.987	5.01%	1.30	0.73	0.67	0.67	16291.60	22786.80	17217.10	5046.62
ACC64	37	2.049	1.975	3.63%	1.13	0.59	0.56	0.57	48672.90	63712.10	51894.60	15904.20
ACC128	50	2.081	1.895	8.94%	1.35	0.89	0.69	0.69	125107.00	144956.00	131821.00	39325.50
GCD	35	1.971	1.925	2.30%	1.77	1.59	0.82	0.84	7017.06	6967.30	7359.21	2139.78
FU	40	1.903	1.855	2.52%	1.77	0.80	0.74	0.76	75562.70	106374.00	80706.60	24465.00
			Normalized	5.98%	2.180	1.550	1.000	1.041	0.947	1.027	1.000	0.284

المنارة للاستشارات

www.manaraa.com

CHAPTER 6. SIMULTANEOUS SLACK MATCHING, GATE SIZING AND REPEATER INSERTION FOR ASYNCHRONOUS CIRCUITS

Slack matching, gate sizing and repeater insertion are well known techniques applied to asynchronous circuits to improve their power and performance. Existing asynchronous optimization flows typically perform these optimizations sequentially, which may result in sub-optimal solutions as all these techniques are interdependent and affect one another. In this paper, we present a unified leakage power optimization framework by performing simultaneous slack matching, gate sizing and repeater insertion. In particular, we apply Lagrangian relaxation to integrate all these techniques into a single optimization step. A methodology to handle slack matching under the Lagrangian relaxation framework is proposed. Also, an effective look-up table based repeater insertion technique is developed to speed up the algorithm. Our approach is evaluated using a set of asynchronous designs and compared with both a sequential approach and a commercial asynchronous optimization flow. The experimental results have achieved significant savings in leakage power and demonstrated the effectiveness of our approach.

6.1 Introduction

Asynchronous designs have been demonstrated to be able to achieve both higher performance and lower power compared with their synchronous counterparts [50] [51] [19]. However, due to the lack of proper EDA tool support, the design cycle for asynchronous circuits is much longer compared with the one for synchronous circuits. Thus, even with many advantages, asynchronous circuits are still not the mainstream in the industry, and it is very important to develop EDA tools for asynchronous circuits design.

المنارة للاستشارات
www.manaraa.com

Stalls are major obstacles limiting the performance of pipelined asynchronous circuits [72]. Due to the slack elasticity for most asynchronous designs, adding pipeline buffers to the design will not change its input/output functionality, but can help remedy the stalls [7]. Thus, slack matching, which inserts minimum number of pipeline buffers to guarantee the most critical cycle meets the desired cycle time, is widely used for asynchronous circuits [9]. Most previous works related to slack matching formulate the problem as a mixed integer linear program (MILP) [7] [67] [54], which is NP-Complete and the integral constraints need to be relaxed in order to solve the problem efficiently. In [81], a heuristic algorithm is proposed to solve the problem by leveraging the asynchronous communication protocol.

Other than slack matching, gate sizing and repeater insertion are also very effective techniques to reduce the delay and power consumption for asynchronous circuits. Gate sizing and repeater insertion for synchronous circuits has been studied for decades and there are many works tackling these problems [14] [26] [80]. However, those works cannot be directly applied to asynchronous circuits, due to the intrinsic differences between asynchronous and synchronous circuits in terms of performance analysis and optimization. In [89], a gate sizing and V_{th} assignment approach for asynchronous circuits has been proposed. It achieved significant improvements compared with previous asynchronous gate sizing approaches. To the best of our knowledge, there is no work on repeater insertion for asynchronous circuits.

Most automated asynchronous design flows apply slack matching, gate sizing and repeater insertion separately either in a sequential manner [90] [78] or in an iterative manner [6]. In Proteus [6], a MILP based slack matching optimization is performed first, followed by gate sizing and repeater insertion which are done by leveraging synchronous EDA tools. Since all these three optimization techniques are closely related to each other, doing them separately may not explore the solution space sufficiently thus yields sub-optimal results. During our experiments, in term of the number of gates, the circuits optimized by Proteus contain 27.1% pipeline buffers on average. This huge amount of pipeline buffers inserted at the slack matching step create a serious area and power overhead, which can be even more critical for asynchronous circuits, since asynchronous designs intrinsically have higher gate count than its corresponding synchronous design. Another disadvantage is that earlier steps of the separated optimization

المنارة للاستشارات
www.manaraa.com

approach have to perform optimizations based on inaccurate delay values. In Proteus, the slack matching is performed based on a rough unit delay model, which simply counts the number of gates along the timing path. Considering the dominating interconnect delays in advanced technologies and the gate sizing and repeater insertion operations performed in later steps, this unit delay model can be very inaccurate and even misleading to the optimization algorithms.

In this paper, we address the problem of minimizing the total leakage power consumption while guaranteeing a target cycle time for unconditional asynchronous pipelined circuits. Three different optimization techniques: slack matching, gate sizing and repeater insertion are effectively joined together under the Lagrangian relaxation (LR) framework. As far as we know, this is the first work that formulates and solves this simultaneous optimization problem combining all these three techniques together.

Our approach is distinctive from previous ones by offering the following benefits:

- Much fewer pipeline buffers can be used for the slack matching purpose, since some stalls can simply be fixed by either gate sizing or repeater insertion which have much less area overhead and consume less power.
- Our approach can prevent excessive sizing when a gate is driving a large load, as we consider repeater insertion together with the gate sizing.
- When design contains extremely long wire delays, i.e., cross chip interconnections, our approach can explore the solution of adding pipeline buffers to break the channel into multiple pipeline stages, which is more beneficial than just doing gate sizing or repeater insertion.
- More accurate delay estimation at each optimization step can be achieved by calculating the delay using the non-linear delay model (NLDM).

The main contributions of this paper are as follows:

- A unified LR framework incorporating gate sizing, repeater insertion and pipeline buffer insertion together.

المنارة للاستشارات
www.manaraa.com

- Methodology for handling pipeline buffer insertion under the LR framework, especially how to update the corresponding Lagrangian multipliers.
- A fast look-up table based repeater insertion approach.
- Results which show significant improvements compared with both the sequential approach and a commercial asynchronous optimization flow.

The rest of this paper is organized as follows. In Section II, a motivating example is presented. Section III shows an overview of the framework. Section IV introduces the backgrounds of LR. Section V presents our optimization algorithm. Finally, the experiments are presented in Section VI.

6.2 A Motivating Example

In this paper, we use the Full Buffer Channel Net (FBCN) [7] to model our asynchronous circuits. A FBCN is a specific form of Petri net [64]. The idea is to model each leaf cell as a *transition*. Channels between cell ports are modeled with a pair of *places* which are annotated with delay information. The handshaking signals are modeled as *tokens*. A simple asynchronous three-stage pipeline and its corresponding FBCN model is shown in Fig. 4.1 (a) and (b).

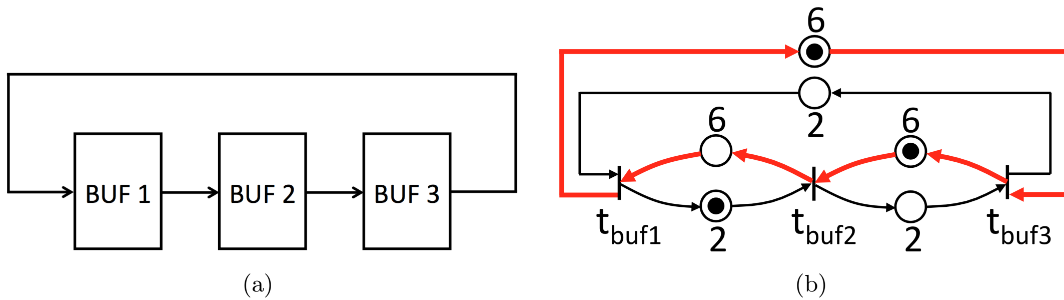

Figure 6.1: (a) Three-stage pipeline. (b) FBCN model.

Here, *transition* t_{buf1}, t_{buf2} and t_{buf3} represent the buffer cells. Circles are *places* which represent the channels between neighboring buffer cells. In particular, *places* containing *tokens* are represented by circles marked with a black dot. In Fig. 4.1, the propagation delay is $2+2+2=6$, which is the time for tokens propagate from t_{buf1} to t_{buf3} and back to t_{buf1}. The

المنارة للاستشارات
www.manaraa.com

local cycle time is $2 + 6 = 8$, which is the shortest time for a buffer to complete a handshake with its neighbors. Since the propagation delay is less than local cycle time, stall happens. The performance of this design is bounded by the highlighted most critical cycle. The corresponding cycle time is $(6+6+6)/2 = 9$, which is calculated by the cycle delay divided by the total number of tokens along this cycle.

The stall can be resolved by slack matching, which adds an extra pipeline stage in the design as shown in Fig. 4.2 (a). The slack matched design operates at desired local cycle time. The most critical cycle is highlighted, which is same as the local handshaking cycle.

Another way to resolve the stall is to improve the acknowledgment (*ack*) time, as shown in Fig. 4.2 (b). This can be done by simply sizing up BUF3 or inserting repeaters at its output *ack* pin. Sizing gates or inserting repeaters are much more economical than inserting pipeline buffers, since pipeline buffers, which contain extra handshaking circuits, are much bigger. For the cell library we have, the smallest size pipeline buffer is 4.8X bigger than the smallest size repeater.

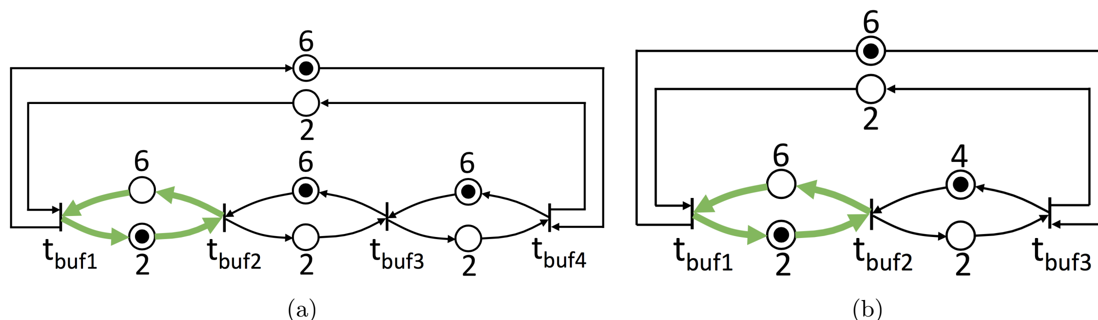

Figure 6.2: (a) Stall fixed by inserting pipeline buffers. (b) Stall fixed by gate sizing or repeater insertion.

Considering typical asynchronous flows which perform the optimization sequentially, if a slack matching solution as shown in Fig. 4.2 (a) is applied first, it is very difficult for the flow to go back to the better solution as shown in in Fig. 4.2 (b). Therefore, we develop the unified optimization approach which is able to achieve much better results.

المنارة للاستشارات
www.manaraa.com

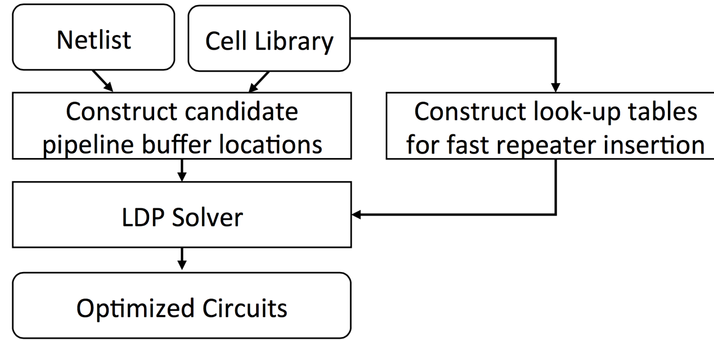

Figure 6.3: High-level View of Our Framework.

6.3 Optimization Framework Overview

A high-level view of our optimization framework is shown in Fig. 4.3. The fundamental idea is to size the gates or insert proper repeaters / pipeline buffers to minimize the leakage power while satisfying the timing constraints. However, if we simply enumerate all the gate size, repeater or pipeline buffer choices, the runtime will not be affordable due to the enormous number of possible combinations. Thus, before solving the problem, the first thing we need to consider is how to limit the solution space and speed up the evaluation process, while still keeping a good solution quality. We do this by constructing candidate pipeline buffer locations and performing table look-up for repeater insertion.

The generated candidate buffer locations and look-up tables are then fed into our Lagrangian dual problem (LDP) solver, where the gate sizing, buffer insertion and repeater insertion problems are joined by Lagrangian multipliers, acting as "weights" associated with each timing arc. The weights help us to find a proper sizing and buffer / repeater insertion solution for the circuit, and the LR framework provide us a systemic way to adjust the weights at each iteration.

6.4 Lagrangian Relaxation Framework

LR is a very useful mathematical approach which transforms the constrained primal problem ($\mathcal{PP}$) into an unconstrained and easier LR subproblem ($\mathcal{LRS}$). Inspired by [14], the special circuit structure allows us to further transform $\mathcal{LRS}$ into an equivalent but even simpler prob-

www.manaraa.com

lem $\mathcal{LRS}^*$. For a given set of non-negative LR multipliers $\boldsymbol{\lambda}$, solving $\mathcal{LRS}^*$ provides us a lower bound of $\mathcal{PP}$. Then, the LR dual problem ($\mathcal{LDP}$) which provides us a solution to $\mathcal{PP}$ can be solved by iteratively solving a sequence of $\mathcal{LRS}^*$.

For an asynchronous circuit modeled with FBCN, our primal problem which minimizes total leakage power subject to performance constraints can be formulated similar to [89] as shown below:

$$\mathcal{PP}: \quad \textbf{minimize} \quad \text{leakage}(\mathbf{g}, \mathbf{b}, \mathbf{r})$$
$$\text{Subject to} \quad a_i + D_{ij} - m_{ij}\tau \leq a_j \quad \forall\, p(i,j) \in P$$

where $\boldsymbol{g}$ is the set of select gates, $\boldsymbol{b}$ is the buffer solution and $\boldsymbol{r}$ is the repeater solution. τ is the given target cycle time. Let T be the set of transitions and P be the set of places in the FBCN model. a_i and a_j denote the arrival times associated with transitions t_i and t_j. $p(i,j)$ denotes the place between t_i and t_j. D_{ij} is the delay associated with $p(i,j)$. $m_{ij} = 1$ if $p(i,j)$ contains a token and 0 otherwise.

By relaxing all constraints into the objective function, we can obtain the $\mathcal{LRS}$ as:

$$\mathcal{LRS}: \quad \textbf{minimize} \quad \text{leakage}(\mathbf{g}, \mathbf{b}, \mathbf{r})$$
$$+ \sum_{\forall(i,j)} \lambda_{ij}(a_i + D_{ij} - m_{ij}\tau - a_j)$$

Similar to [14], $\mathcal{LRS}$ can be further simplified into $\mathcal{LRS}^*$ by applying $\mathcal{KKT}$ optimality conditions:

$$\mathcal{KKT}: \quad \sum_{\forall(k,j)} \lambda_{kj} = \sum_{\forall(i,k)} \lambda_{ik} \quad \forall\, k \in T$$

$$\mathcal{LRS}^*: \textbf{minimize} \;\; \text{leakage}(\mathbf{g}, \mathbf{b}, \mathbf{r})$$
$$- \sum_{\forall(i,j)} \lambda_{ij} m_{ij}\tau + \sum_{\forall(i,j)} \lambda_{ij} D_{ij}$$

Finally, we obtain $\mathcal{LDP}$ by maximizing $\mathcal{LRS}^*$:

$$\mathcal{LDP}: \quad \textbf{maximize} \quad \mathcal{LRS}^*$$
$$\text{Subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0},\ \boldsymbol{\lambda} \in \mathcal{KKT}$$

www.manaraa.com

6.5 Simultaneous Gate Sizing, Repeater Insertion and Pipeline Buffer Insertion

6.5.1 Constructing Candidate Pipeline Buffer Location

Fig. 4.4 shows a three-stage asynchronous pipeline implemented using the PCHB template [47]. Stage 1 and stage 3 are computation stages. In particular, domino logic cells (LOGIC) are used for computation and control circuit (CTRL), C-elements (C) are used to perform handshaking. The dual rail channel contains two data wires ($A[0].0$, $A[0].1$), and one wire (L_e) for the *ack* signal. Thus, we can easily identify all the channels in the circuit and pre-insert a candidate pipeline buffer inside each channel, similar to stage 2 in Fig. 4.4.

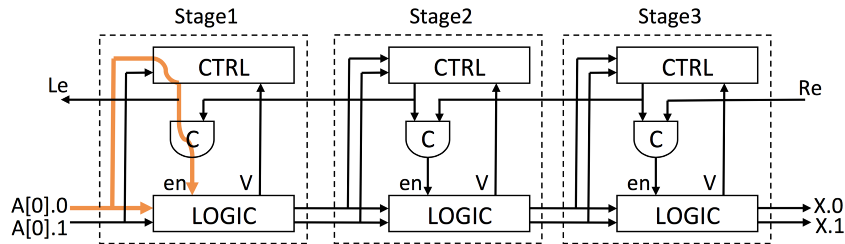

Figure 6.4: A three-stage PCHB pipeline.

Different from regular pipeline buffers, we assign the pre-inserted pipeline buffer with two modes: transparent and opaque, as shown in Fig. 4.5 (a) and (b) respectively. The transparent mode is used to model the situation in which no buffer is inserted and the opaque mode is used to model the opposite situation. In transparent mode, the pipeline buffer has three timing arcs denoted as $t1$ to $t3$, acting as wires connecting the corresponding *ack* or data pins. It contributes zero leakage power to the circuit. Also, during static timing analysis, the slew values seen at its input pins will be propagated to the corresponding output pins for all its fanout cells. Similarly, the load capacitance seen at its output pins will be forwarded to the input pins for the fanin cells. In opaque mode, the pre-inserted buffer acts as a normal pipeline buffer and there are 9 timing arcs denoted as $t1$ to $t9$ from each input pin to each output pin. Then, instead of actually modifying the netlist, the algorithm can simply switch the buffer between transparent and opaque mode to achieve the same effects as removing / inserting the buffer.

www.manaraa.com

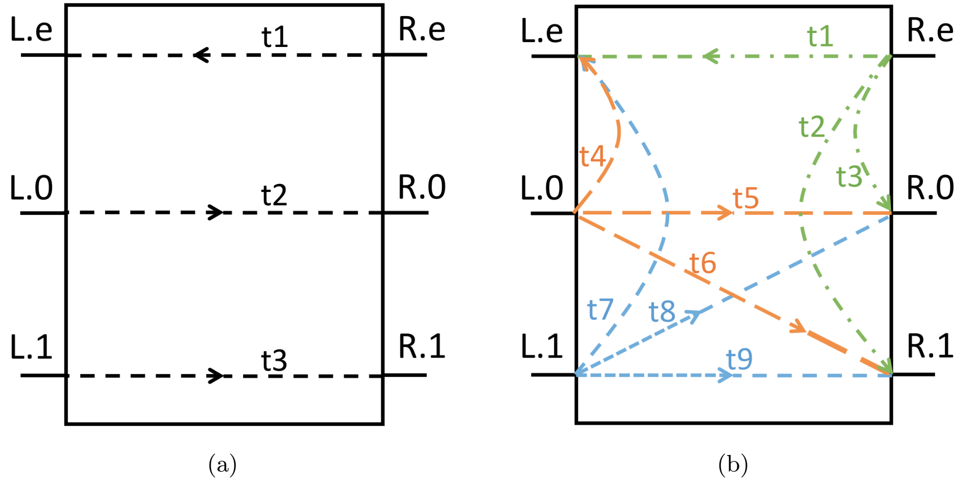

Figure 6.5: Pre-inserted pipeline buffer: (a) Transparent mode (b) Opaque mode

As described in Sec. IV, the set of Lagrangian multiplier $\boldsymbol{\lambda}$ needs to satisfy $\mathcal{KKT}$ conditions during the update process. Let λ_{t1} denote the λ associated with timing arc $t1$ and similarly for all other λs. Let us consider the λ sum at pin $L.e$ and $R.e$. Since these two pins are connected by a single timing arc in transparent mode, the λ sum at pin $L.e$ and the λ sum at pin $R.e$ should be equal, and they should keep to be equal when the buffer transforms between transparent mode and opaque mode. This requires us to have: $\lambda_{t1} + \lambda_{t2} + \lambda_{t3} = \lambda_{t1} + \lambda_{t4} + \lambda_{t7}$ for the λs in opaque mode. However, the λ update in opaque mode might not follow the above rule, which can make the λs violating the $\mathcal{KKT}$ conditions when the buffer is transformed back to transparent mode. Similarly, for other pins, the same issue will also happen.

A simple solution is to only update λ_{t1}, λ_{t5} and λ_{t9} while keeping all other λs to be 0 in opaque mode. However, this might put too much restrictions on λ values and make them unable to accurately reflect the criticality of each timing arc. Thus, we propose a better solution where we enforce the following equality constraints during λ update:

$$(\lambda_{t2} = \lambda_{t7}) \wedge (\lambda_{t3} = \lambda_{t4}) \wedge (\lambda_{t6} = \lambda_{t8})$$

These constraints guarantee the updated λs always satisfy the $\mathcal{KKT}$ conditions in both modes, while it avoids putting too much restrictions on the original λ values.

المنارة للاستشارات
www.manaraa.com

6.5.2 Constructing Look-up Tables for Fast Repeater Insertion

If we do not consider any blockages, repeaters can be inserted at any location of the wires. In order to limit the solution space and simplify our algorithm, here we only consider inserting repeaters at the input / output pins of each gate. However, even under such an assumption, the possible choices for repeater insertion are still too many even for one gate, because we can have different size or number of repeaters at each pin. Therefore, we propose a look-up table technique to speed up our evaluation process. In particular, we construct a 2D look-up table for each pin of each gate in our standard cell library. The X-axis of the look-up table is the load capacitance driven by the repeaters. The Y-axis is the sum of the values of λs at this pin, as shown in Fig. 4.6.

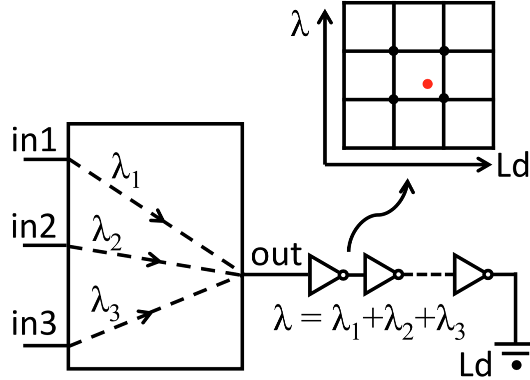

Figure 6.6: Look-up table at each pin.

Given the load and λ value of each look-up table entry, we evaluate all the possible repeater insertion choices at this pin based on a typical input slew. The best choice, i.e., the one providing the smallest cost, will be stored in the table. The cost is evaluated based on the following cost function:

$$\text{Cost}(g_i) = \text{leakage}(g_i) + \sum_{(u,v)\in Arc_i} \lambda_{uv} D_{uv} \tag{6.1}$$

Here, Arc_i is defined to be the set of timing arcs of repeater g_i and all the fanin and fanout gates of g_i.

It might happen that the actual λ and load value calculated by the LDP solver does not match any of the index values in the look-up table. In this situation, we simply evaluate all four choices surrounding this point and pick the best one.

المنارة للاستشارات
www.manaraa.com

6.5.3 Solving $\mathcal{LDP}$

We apply a direction finding approach inspired by [83] to solve $\mathcal{LDP}$, as shown in Fig. 4.7.

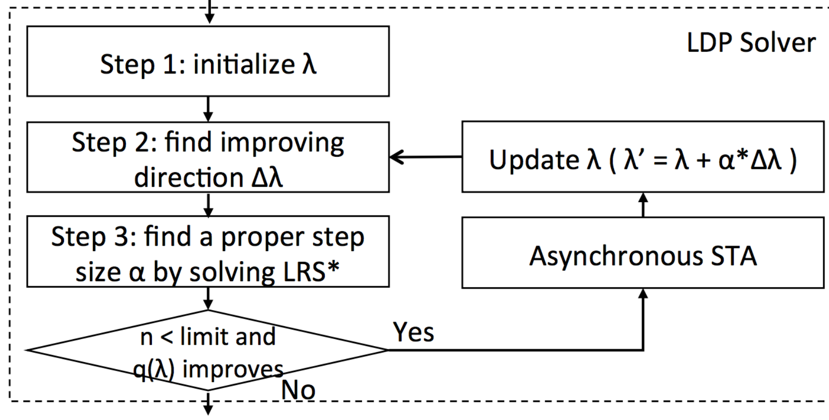

Figure 6.7: Lagrangian dual problem solver.

In step 1, we find an initial set of non-negative λs satisfying the $\mathcal{KKT}$ conditions.

In step 2, an improving feasible direction $\Delta\boldsymbol{\lambda}$ can be found by solving the following linear program, which maximize the first order approximation of $\mathcal{LRS}^*$:

$$\mathcal{DF}: \quad \textbf{maximize} \quad \sum_{\forall(i,j)} \Delta\lambda_{ij} D_{ij} - \sum_{\forall(i,j)} \Delta\lambda_{ij} m_{ij} \tau$$

$$\text{Subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0},\ \boldsymbol{\lambda} \in \mathcal{KKT}$$

$$\max(-u, -\lambda_{ij}) \leq \Delta\lambda_{ij} \leq u$$

here u is used to bound the objective function and avoid it goes to infinity, similar to [83].

In step 3, we find a proper step size α by optimizing along the feasible direction $\Delta\boldsymbol{\lambda}$ using line search techniques. In particular, we solve the $\mathcal{LRS}^*$ for a given set of $\boldsymbol{\lambda}$ at each potential step. The step size which achieves $q(\boldsymbol{\lambda}+\alpha\Delta\boldsymbol{\lambda}) > q(\boldsymbol{\lambda})$ will be selected as α. Here, $q(\boldsymbol{\lambda})$ denotes the optimal objective value of $\mathcal{LRS}^*$.

We keep iterating between steps 2 and 3 until $q(\boldsymbol{\lambda})$ does not improve or the number of iterations (n) exceeds the limit.

www.manaraa.com

6.5.4 Solving $\mathcal{LRS}^*$

Since asynchronous circuits contain loops, it does not has a topological order which is commonly used in synchronous optimization algorithms. Thus, here we use a sequential update technique as shown in Algorithm 1. The idea is to traverse all the gates in a sequential order and locally pick a best solution which minimize $\mathcal{LRS}^*$. If the newly picked solution is different from the old one, we will reevaluate all its fanout gates. Please note that in Algorithm 1, the *gate* refers to all the gates in the original circuit and the pre-inserted candidate pipeline buffers, but it does not represent the repeaters inserted by our algorithm. For a regular gate, a *solution* at a gate means a proper size and repeater insertion choices of this gate. For a candidate pipeline buffer, the *solution* also denotes whether the buffer is in opaque or transparent mode.

Algorithm 6 Solve $\mathcal{LRS}^*$

Ensure: a proper solution for each gate which minimize $\mathcal{LRS}^*$
1: Assign all the gates with an initial solution;
2: Insert all the gates into a set $\mathcal{G}$;
3: **while** $\mathcal{G} \neq \emptyset$ **do**
4: Pick one gate g_i from $\mathcal{G}$. Let its current solution be s_i^j;
5: Select a better solution s_i^k for gate g_i; /* Algorithm 2 */
6: **if** $s_i^j \neq s_i^k$ **then**
7: Assign g_i with this new solution s_i^k;
8: **if** g_i is visited less than or equal to n times **then**
9: Insert all gates $\notin \mathcal{G}$ and directly driven by g_i into $\mathcal{G}$;
10: **end if**
11: **end if**
12: Remove g_i from set $\mathcal{G}$;
13: **end while**

Algorithm 2 shows our local evaluation algorithm which find the best local solution at each gate based on the following cost function:

$$\text{Cost}(g_i) = \text{leakage}(g_i) - \sum_{(u,v)\in Arc_i} \lambda_{ij} m_{ij} \tau + \sum_{(u,v)\in Arc_i} \lambda_{uv} D_{uv} \tag{6.2}$$

Similar to equation (1), Arc_i is the set of timing arcs of gate g_i and all the timing arcs of g_i's fanin and fanout gates.

In step 2 of Algorithm 2, the method we used to pick proper size and repeaters of this gate is simply evaluating all its possible sizing and repeater insertion options. In particular, we first

المنارة للاستشارات
www.manaraa.com

Algorithm 7 Local Evaluation

Ensure: Best solution for g_i which locally minimize $\mathcal{LRS}^*$

```
 1: if g_i is a regular gate then
 2:     Select a proper sizing, repeater insertion option for g_i;
 3: else        /* g_i is a candidate pipeline buffer */
 4:     if g_i is in opaque mode then
 5:         Change g_i to transparent mode;
 6:         Local timing update;
 7:         If the cost is not reduced, recover to opaque mode;
 8:     else        /* g_i is in transparent mode */
 9:         Change g_i to opaque mode;
10:         Update λ for all the timing arcs of g_i;
11:         Select best sizing and repeater insertion solution for g_i;
12:         If the cost is not reduced, recover to transparent mode;
13:     end if
14: end if
15: return bestSolution;
```

select a certain size for this gate, then the repeater insertion options at each of its pin can be found using the look-up tables. The cost of each sizing and repeater insertion combination will be calculated based on equation (1).

6.6 Experiments

The proposed optimization approach is implemented in C++ and runs on a Linux PC with 8 GB of memory and 2.4 GHz Intel Core i7 CPU.

Our unified optimization approach is tested using two sets of asynchronous benchmarks. First is a set of asynchronous benchmarks transformed from ISCAS89 benchmarks. Second is a set of specific asynchronous designs. In particular, we use different bit widths on the datapath of ALU and Accumulator designs to generate the set of benchmarks with different number of gates. All the designs are transformed or synthesized using the Proteus front-end flow [6].

Accurate non-linear delay model is used to calculate delay and slew value based on the look-up tables from Proteus standard cell library. Cell interconnections are modeled as lumped capacitance, which is obtained by extraction after placement and routing. Since the original cell library does not contain leakage power, we assign a leakage power for each cell which is proportional to its area. The cycle time achieved by Proteus is used as a timing constraint for our unified flow and the sequential approach described below.

المنارة للاستشارات
www.manaraa.com

Table 6.1: Comparison on transformed ISCAS89 benchmarks

Design	# of gates	Cand.	Cycle Time (ns)			Leakage (μW)			Buffers			Repeaters		
			Target	Seq.	Ours	Proteus	Seq.	Ours	Proteus	Seq.	Ours	Proteus	Seq.	Ours
as27	47	13	0.40	0.39	0.39	601.34	488.056	395.16	7	6	3	6	3	1
as298	223	98	0.52	0.45	0.49	3528.58	2940.16	2351.60	63	36	7	14	42	13
as386	268	115	0.73	0.68	0.63	4227.10	2953.01	2834.33	70	21	21	22	59	21
as349	324	115	0.73	0.65	0.73	4972.49	3676.08	2718.56	85	43	3	30	49	14
as382	268	119	0.62	0.61	0.58	4437.92	3705.66	2757.96	69	40	0	15	51	14
as400	281	126	0.59	0.57	0.58	4774.81	3176.96	3175.65	83	18	19	14	62	20
as420	324	122	0.44	0.39	0.43	5217.87	4456.1	3394.69	88	25	12	20	45	20
as444	270	119	0.56	0.51	0.54	4571.60	3252.37	2845.32	73	19	5	14	52	13
as510	572	294	0.92	0.91	0.92	8935.97	8862.22	6486.70	118	111	29	53	144	44
as526	323	147	0.60	0.60	0.53	5448.18	4659.59	3846.67	98	61	25	16	65	25
as641	723	190	0.79	0.78	0.78	9860.11	6157.28	4799.62	213	59	2	87	103	30
as713	666	175	0.70	0.64	0.69	9031.29	5896.07	4677.90	177	29	8	82	97	32
as832	803	361	0.99	0.99	0.97	12194.40	9511.26	8190.31	190	80	38	97	171	55
as838	747	300	0.58	0.54	0.58	12049.80	8014.05	7371.23	226	34	31	40	168	53
as953	1015	469	1.01	0.98	0.99	16281.70	12698	12528.00	271	107	140	84	239	94
as1488	1430	772	1.14	1.03	0.98	24655.10	20808.1	19559.20	410	275	196	116	135	134
as5378	2742	1242	0.71	0.67	0.71	46083.00	31520.9	27859.40	870	233	67	187	591	204
as9234	2236	1021	1.05	1.01	1.05	36528.00	24193.2	22925.20	686	83	58	130	152	155
as13207	6088	2583	1.02	0.93	0.99	96456.20	68898.5	59234.00	1945	567	154	451	353	336
				Normalized		1.565	1.141	1.000	7.020	2.258	1.000	1.156	2.020	1.000

www.manaraa.com

Table 6.2: Comparison on asynchronous benchmarks

Design	# of gates	Cand.	Cycle Time (ns)			Leakage (μW)			Buffers			Repeaters		
			Target	Seq.	Ours	Proteus	Seq.	Ours	Proteus	Seq.	Ours	Proteus	Seq.	Ours
ALU8	957	364	0.46	0.46	0.41	17107.00	15006.80	18086.70	146	120	92	41	219	73
ALU16	2511	1002	0.66	0.48	0.56	41069.20	40319.60	37103.50	594	382	256	69	622	188
ACC16	609	215	0.62	0.60	0.59	8974.96	7585.64	6386.76	100	72	18	69	105	31
ACC32	1323	469	0.88	0.83	0.80	20363.20	16475.50	15534.40	268	175	192	136	218	97
ACC64	3619	1188	0.71	0.71	0.69	57698.60	44570.90	33062.80	1291	294	111	264	587	200
FU	5805	2107	1.42	1.21	1.32	75910.00	67900.00	63584.70	1692	590	609	501	1047	385
GCD	475	223	1.58	1.20	1.30	6988.86	7544.86	5710.49	89	73	7	30	106	35
				Normalized		1.271	1.111	1.000	3.253	1.328	1.000	1.100	2.878	1.000

المنارة للاستشارات

www.manaraa.com

For comparison purpose, we implemented a sequential approach similar to our unified optimization flow, but it only performs one type of optimization at each iteration. In particular, the sequential approach starts with 10 iterations of pipeline buffer insertion, followed by 10 iterations of repeater insertion and 30 iterations of gate sizing. We use this order because gate sizing is the more fine-grained optimization and it is better to be applied at last.

Comparison results on the transformed ISCAS89 benchmarks are shown in Table 4.1. "# of gates" column shows the total number of gates. "Cand." column shows the number of pre-inserted candidate pipeline buffers. "Target" column shows the target cycletime obtained from Proteus flow. The "Proteus", "Seq." and "Ours" columns show the results of the Proteus, the sequential approach and our approach respectively. Leakage power, the number of inserted buffers and the number of inserted repeaters are compared among different flows. All results satisfy the target cycletime constraints. The results show our approach is much better in power consumption and insert fewer buffers and repeaters. Comparing the leakage power, on average, our approach is 56.5% better than the Proteus flow and 14.1% better than the sequential flow. Table 4.2 shows the comparison results on the specific asynchronous benchmarks. Similar improvements are achieved. For the leakage power, our approach is 27.1% better than the Proteus flow and 11.1% better than the sequential flow.

The significant improvements in both sets of benchmarks suggest that all these techniques are very closely related to each other and the proposed joint optimization approach can provide significant benefits compared with the non-simultaneous ones. Proteus does not have a separate circuit optimization step and so we are not able to measure its runtime. The average runtime of our algorithm is around 6.5 minutes, which indicates our flow runs fast enough and will not be a runtime bottleneck of the design process.

6.7 Conclusions

In this paper, we have proposed a simultaneous slack matching, gate sizing and repeater insertion approach for asynchronous circuits. We apply Lagrangian relaxation to integrate all these techniques into a single optimization step. The relaxed problem is further simplified using KKT conditions. Effective techniques to handle pipeline buffer insertion and repeater

المنارة للاستشارات
www.manaraa.com

insertion under the Lagrangian relaxation framework are proposed. A local evaluation algorithm is also developed to solve the relaxed problem efficiently. The experimental results show significant improvements on power consumption and demonstrate the benefits of performing these optimizations simultaneously rather than sequentially.

المنارة للاستشارات

www.manaraa.com

CHAPTER 7. A FAST INCREMENTAL MAXIMUM CYCLE RATIO ALGORITHM

In this paper, we propose an algorithm to quickly find the maximum cycle ratio (MCR) on an incrementally changing directed cyclic graph. Compared with traditional MCR algorithms which have to recalculate everything from scratch at each incremental change, our algorithm efficiently finds the MCR by just leveraging the previous MCR and the corresponding largest cycle before the change. In particular, the previous MCR allows us to safely break the graph at the changed node. Then, we can detect the changing direction of the MCR by solving a single source longest path problem on a graph without positive cycle. A distance bucket approach is proposed to speed up the process of finding the longest paths. Our algorithm continues to search upward or downward based on whether the MCR is detected as increased or decreased. The downward search is quickly performed by a modified Karp-Orlin algorithm reusing the longest paths found during the cycle detection. In addition, a cost shifting idea is proposed to avoid calculating MCR on certain type of incremental changes. We evaluated our algorithm on both random graphs and circuit benchmarks. A timing-driven detailed placement approach which applies our algorithm is also proposed. Compared with Howard's and Karp-Orlin MCR algorithm, our algorithm shows much more efficiency on finding the MCR in both random graphs and circuit benchmarks.

7.1 Introduction

Given a directed cyclic graph and each edge in the graph is associated with two numbers: *cost* and *transition time*. Let the cost (respectively, transition time) of a cycle in the graph be the sum of the costs (respectively, transition times) of all the edges within this cycle. Assuming

www.manaraa.com

the transition time of a cycle is non-zero, the ratio of this cycle is defined as its cost divided by its transition time. The maximum cycle ratio (MCR) problem finds the cycle whose ratio is the maximum in a given graph [18]. Applications of the MCR problem exist in various areas, e.g., performance analysis of synchronous or asynchronous circuits, time separation analysis of concurrent systems, graph theory [17].

In practice, most of the optimization processes which apply MCR algorithms are actually performed incrementally. For example, during the detailed placement stage of VLSI circuits, one step of the algorithm adjusts the coordinates of only a few cells. Then, evaluation is performed for this modified circuit before the next move [63] [86]. Similarly, in the gate sizing process of circuits, the algorithm might adjust the size of one gate at a time, instead of changing the sizes of all the gates at once [89]. Considering the above type of applications which only few changes are made at each step, the MCR algorithm might also be able to do the calculation "incrementally" by leveraging the information calculated at the previous step, and therefore be able to find the MCR much faster. In this paper, we focus on the MCR problem considering such incremental changes, which we referred to as the incremental MCR problem. By leveraging the previously calculated information, we expect the incremental MCR algorithm to be faster than traditional MCR algorithms, which have to recalculate everything from scratch at each step.

The MCR problem without considering the incremental changes has been well studied [18][17][29]. One way to solve the MCR problem is by linear programming [49]. In addition, various MCR algorithms are proposed to solve the problem more efficiently. Experimental study of existing MCR algorithms shows the Karp and Orlin's algorithm (KO) [38] and an efficient implementation of KO [91] is the fastest among them [17]. When the graph size is small, the Howard's algorithm (HOW) [32] is also able to generate comparable results [18]. For the incremental MCR problem, only very few researches have been done. In [13], the authors developed an adaptive negative cycle detection algorithm and incorporated it into the Lawler's MCR algorithm [43]. However, the experiments in [13] are performed only on very small graphs, and thus the efficiency of the algorithm cannot be confirmed.

المنارة للاستشارات
www.manaraa.com

In addition, Lawler's algorithm finds MCR based on the binary search idea, which is much slower compared with KO and HOW [17].

In this paper, we propose an efficient incremental MCR algorithm. The only information we need to leverage is the previous MCR and the corresponding largest cycle in the graph before the incremental change is made. Our algorithm contains three parts: cycle detection, local upward search and global downward search. After an incremental change is made on the given graph, the cycle detection is performed first. During the cycle detection, we filter out the cases which the incremental change will not affect the MCR using our cost shifting idea. If we cannot guarantee the MCR will not be affected, the algorithm continues to detect whether the MCR is increased or decreased. If the MCR is detected to be increased, we perform the local upward search to identify the new MCR in the changed graph. Otherwise, we perform the global downward search to identify the new MCR. To speed up the cycle detection and the local upward search, we propose a bucket distance idea which can quickly build a longest path tree in a graph without positive cycle. Also, we reuses the longest paths found in cycle detection by a modified KO algorithm to speed up the global downward search. We evaluate our algorithm on both random graphs and the ISPD 2005 placement benchmarks [55]. To evaluate our algorithm on circuit benchmarks, we propose a timing-driven detailed placement approach which applies our incremental MCR algorithm. The experimental results show our algorithm to be very efficient on calculating MCR compared with the fastest traditional MCR algorithms.

The rest of this paper is organized as follows. In Section II, we briefly review the Howard's algorithm and the Karp-Orlin algorithm. In Section III, we present our incremental MCR algorithm. In Section IV, we present our timing-driven detailed placement approach. Finally, the experimental results are shown in Section V.

www.manaraa.com

7.2 Preliminaries

7.2.1 Maximum cycle ratio problem

We formally define the MCR problem in this section. Let $G = (V, E)$ be a directed cyclic graph. Each edge $e \in E$ is associated with a cost denoted as $w(e)$ and a transition time denoted as $t(e)$. Let c denotes a cycle in G. Let $\tau(c)$ denotes the cycle ratio of c. With the assumption that $\sum_{\forall e \in c} t(e) > 0$, the MCR problem finds the maximum $\tau(c)$ $\forall c \in G$ as:

$$\tau^*(G) = \max_{c \subset G} \left\{ \frac{\sum_{\forall e \in c} w(e)}{\sum_{\forall e \in c} t(e)} \right\}$$

Here, we use $\tau^*(G)$ to denote the MCR of G. As an example, Fig. 7.1 shows a graph with two cycles (a, b, c) and (a, b, d, c). The two numbers associated with each edge denote its cost and transition time respectively. By calculating the ratio of both cycles, we can identify the largest cycle shown as the dotted lines in Fig. 7.1. However, for larger graphs, it is difficult for us to enumerate all the cycles to find out which one is the largest, as the total number of cycles can be exponential to the graph size.

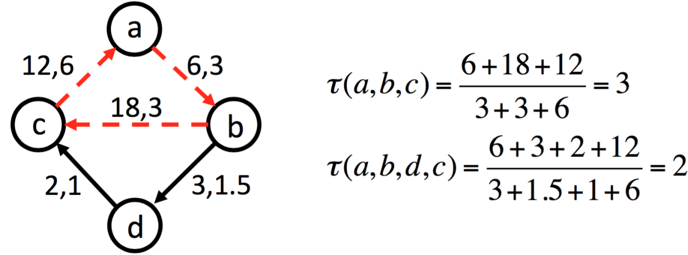

Figure 7.1: Finding the maximum cycle ratio in a graph.

One way to solve the maximum cycle ratio problem is to formulate it as a linear program [49]:

Minimize τ

Subject to $d(i) + w(i,j) - t(i,j) * \tau \leq d(j) \quad \forall (i,j) \in E$

Here, (i, j) denotes the edge connecting node i to node j. $d(i)$ and $d(j)$ are free variables for each node $v \in V$ denoting its *distance*.

In addition to the linear program solution, various algorithms have been proposed and are able to solve the problem more efficiently. We discuss these algorithms below.

المنارة للاستشارات
www.manaraa.com

7.2.2 Traditional maximum cycle ratio algorithms

Given a cycle ratio τ, we can construct another graph $G_\tau = (V, E_\tau)$ based on G. G_τ is identical to G, except now each edge in E_τ is associated with only one number *length*, instead of having two numbers (i.e. cost and transition time). Each $e \in E_\tau$ will have a corresponding edge $e' \in E$, and the *length* of e is defined as $l(e) = w(e') - \tau * t(e')$. Correspondingly, the *length* of a cycle $c \in G_\tau$ is defined as $l(c) = \sum_{\forall e \in c} l(e)$.

G_τ has many interesting features which can help us identify the largest cycle in G. In particular, if G_τ contains positive length cycles, it means the given cycle ratio τ is less than $\tau^*(G)$. If G_τ contains zero length cycles and does not contain positive length cycles, the given τ will be equal to $\tau^*(G)$, and the zero length cycle in G_τ will correspond to the largest cycle in G. If G_τ only contains negative cycles, it means the given cycle ratio τ is larger than $\tau^*(G)$. In this case, single source longest path trees rooted at any node $v \in V$ exist in G_τ. Detailed proofs of these facts can be found in [17].

Most of the MCR algorithms use the above facts to transfer the MCR problem into the problem of either detecting positive cycles in G_τ or maintaining a longest path tree in G_τ. Howard's algorithm and Karp-Orlin algorithm are two of the fastest algorithms among them, and they tackles the MCR problem in exactly opposite directions. In particular, Howard's algorithm starts with a very small τ and gradually increases τ until it cannot detect a positive length cycle in G_τ. Karp and Orlin's algorithm starts with a very large τ and gradually decreases τ while maintaining a longest path tree in G_τ.

7.2.2.1 Howard's algorithm (HOW)

HOW can be separated into two phases: the discovery phase and the verification phase. If the starting cycle ratio τ is small enough, all the cycles in G_τ will have a positive length. In the discovery phase, an arbitrary positive length cycle $c \in G_\tau$ is located, and we increase τ to τ' such that $l(c) = 0$ in $G_{\tau'}$. Next, in the verification phase, a positive cycle detection algorithm (e.g., the Bellman–Ford algorithm) can be used to check if there are still positive cycles in $G_{\tau'}$.

www.manaraa.com

If so, we repeat the discovery phase. If not, we are safe to exit the algorithm and output τ' as $\tau^*(G)$. More details and pseudo code for HOW can be found in [18][17].

7.2.2.2 Karp and Orlin's algorithm (KO)

KO starts with a large enough τ such that all cycles in G_τ is negative and thus longest paths are well defined in G_τ. Here, we use a simple example to illustrate the basic idea of KO. For more details, please refer to [18][17][91].

In the beginning, KO modifies G by adding a node s and a set of edges E_s connecting s to all nodes $v \in V$, with $w(e) = 0$ and $t(e) = 1$ for all $e \in E_s$. Let a path from s to node v in G be denoted by $p(s, v)$, which corresponds to the path from root s to v in the longest path tree T_s in G_τ. For each node $v \in V$, we have $w(v) = \sum_{\forall e \in p(s,v)} w(e)$ and $t(v) = \sum_{\forall e \in p(s,v)} t(e)$, shown as $(w(v), t(v))$ in Fig. 7.2. For each edge $(i, j) \in E$, let $\Delta w(i, j) = w(i) + w(i, j) - w(j)$ and $\Delta t(i, j) = t(i) + t(i, j) - t(j)$. Then, a max heap containing all the edges in G is maintained using the key value calculated as follows:

$$key(i,j) = \begin{cases} \Delta w(i,j)/\Delta t(i,j), & \text{if } \Delta t(i,j) > 0. \\ -\infty, & \text{otherwise.} \end{cases}$$

Fig. 7.2 shows the process of calculating $\tau^*(G)$ using KO for the graph G shown in Fig. 7.1. Fig. 7.2(a) shows the initial longest path tree T_s in G_{τ_0} and the corresponding max heap. Next, edge (b, c) which has the maximum key value is retrieved from the heap, and T_s is updated by replacing edge (s, c) with (b, c) as shown in Fig. 7.2(b). This tree update makes T_s to be the longest path tree in G_{τ_1}. We will continue the max heap update and tree update until a cycle is found which gives us $\tau^*(G)$, as shown in Fig. 7.2(d).

7.3 Our Incremental Cycle Ratio Algorithm

We define an incremental change on an edge as a cost change on this edge, and an incremental change on a node as the cost changes on all the input and output edges of this node. The transition times remain to be the same for both the edge change and the node change. In

www.manaraa.com

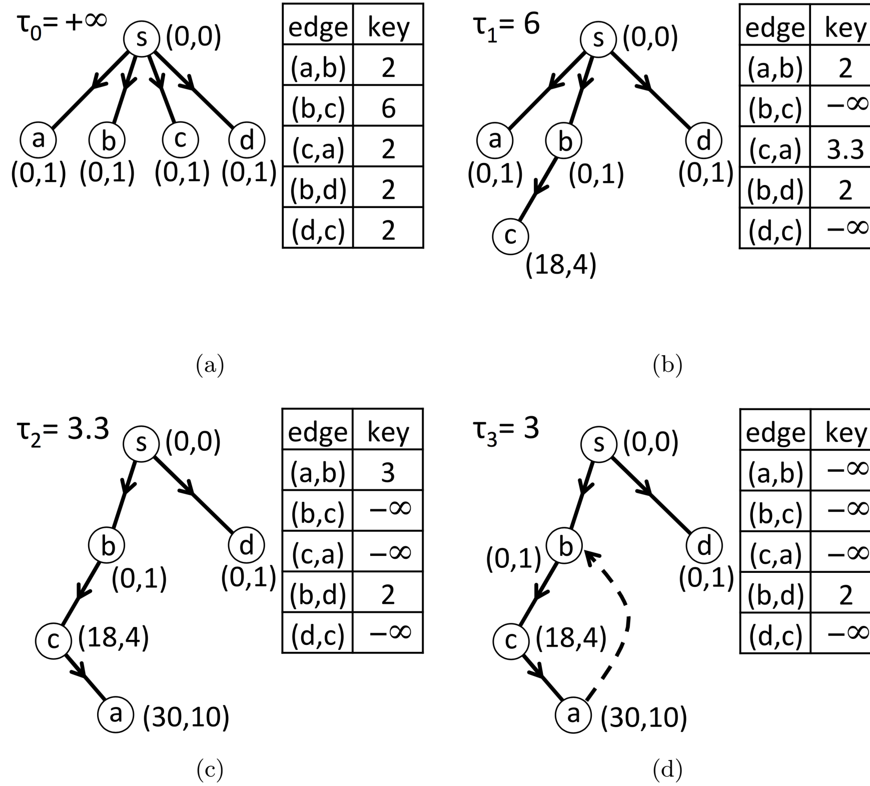

Figure 7.2: An example of Karp and Orlin's algorithm

Section III-A, we first look into details about how MCR is affected by an edge change. In Section III-B, we consider the incremental changes happened on a node, which is the assumption made by our algorithm.

7.3.1 Considering incremental changes on an edge

Let e denotes the changed edge. We use C_e to denote the set of cycles passing through e, and when $w(e)$ changes, only the cycles in C_e will be affected. In addition, we use G to denote the graph before the change and G' to denote the graph after the change, with corresponding largest cycle to be c^* and $c^{*\prime}$ respectively. Based on whether $w(e)$ is decreased or increased and whether $e \in c^*$ or not, we can separate the incremental changes into the following four cases:

www.manaraa.com

7.3.1.1 $e \notin c^*$ and $w(e)$ is decreased

This is the easiest case, as it can be guaranteed that $c^{*\prime} = c^*$ after the incremental change. Before the change happens, we know $\tau(c) \leq \tau(c^*)\ \forall c \in C_e$. Since decreasing the cost of e will only decrease $\tau(c)\ \forall c \in C_e$, none of these cycles will get a chance to become larger than c^*. Thus, c^* will remain to be the largest cycle in G'.

7.3.1.2 $e \notin c^*$ and $w(e)$ is increased

Increasing $w(e)$ will increase $\tau(c)\ \forall c \in C_e$. It is possible that $\tau(c)$ of a cycle $c \in C_e$ becomes larger than $\tau(c^*)$ and thus dominates all other cycles and becomes the largest cycle in G'. If this happens, we have $c^{*\prime} = c$, and thus it can be guaranteed that $c^{*\prime}$ is passing through e.

7.3.1.3 $e \in c^*$ and $w(e)$ is decreased

Decreasing $w(e)$ will decrease $\tau(c^*)$. Thus, another cycle in G can replace c^* and becomes dominating in G'. If this happens, there is no clue for us to know where this new largest cycle is located.

7.3.1.4 $e \in c^*$ and $w(e)$ is increased

Increasing $w(e)$ will increase $\tau(c^*)$ and also increase $\tau(c)\ \forall c \in C_e$. Thus, it is guaranteed that $c^{*\prime}$ is passing through e. However, there is no guaranteed that $c^{*\prime} = c^*$. As an example, let the graph in Fig. 7.1 to be G and the graph in Fig. 7.3 to be G'. It can be seen that after increase $w(c, a)$ from 12 to 400, the largest cycle also get changed.

7.3.2 Considering incremental changes on a node

Only considering changes on a single edge is certainly not enough, as applications typically involve multiple edge changes. We can transform the multiple edge changes into single edge

المنارة للاستشارات
www.manaraa.com

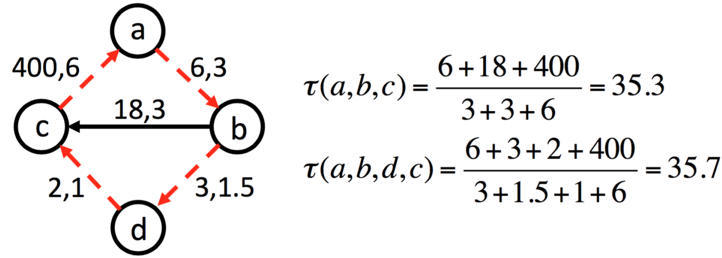

Figure 7.3: $e \in c^*$ and $w(e)$ is increased.

change by only processing one edge at a time, but this can slow down the incremental MCR algorithm. Therefore, instead of only considering a single edge change, our algorithm handles the case of a single node change, which makes it more suitable for real applications. A single node change can create more complicated situations compared with an edge change, as cycle ratio of some cycles can decrease while others can increase at the same time. However, similar to the edge change, only the cycles passing through the changed node will be affected. If the change is happened on more than one node, our algorithm will just transform it to the single node change by processing one node change at a time.

7.3.3 Considering HOW and KO incrementally

HOW and KO are not suitable to perform the MCR calculation incrementally. One reason is that most of the middle information (i.e. node distances, the longest path tree) is calculated based on G_τ whose edge lengths depend on the parameter τ. Once τ is changed, all edge lengths in G_τ will be updated and the middle information calculated in the previous iteration will become useless. It is also not realistic to keep these middle information for each possible τ value, as the possible τ values correspond to all cycles in the graph whose total number is exponential to the graph size. Another reason is that, the cycle ratio can change in both directions when an incremental change is made, while HOW or KO can only search from one direction. In particular, if MCR is decreased, it will be difficult for HOW to go backward and locate the new largest cycle. Similarly for KO when MCR is increased. This also suggests the incremental MCR algorithm needs to be able to search from both directions. When MCR is increased, the algorithm can search upward starting from the previous MCR, similar to HOW.

المنارة للاستشارات
www.manaraa.com

When MCR is decreased, the algorithm can search downward similar to KO. This is just the basic idea of our algorithm, which we will discuss below.

7.3.4 An overview of our incremental MCR algorithm

An overview of our incremental MCR algorithm is shown in Fig. 7.4. Give an initial graph G with its MCR to be $\tau^*(G)$ and the corresponding largest cycle to be c^*. Assuming a node v in G is updated and the set of cycles passing through v is C_v, our algorithm first detects the changing direction of MCR. If the MCR is detected as increased, we perform a local upward search to identify the new MCR. If the MCR is detected as decreased, we perform a global downward search to identify the new MCR. The output of our algorithm is $\tau^*(G')$ and $c^{*\prime}$ for G' which denotes the graph after the change.

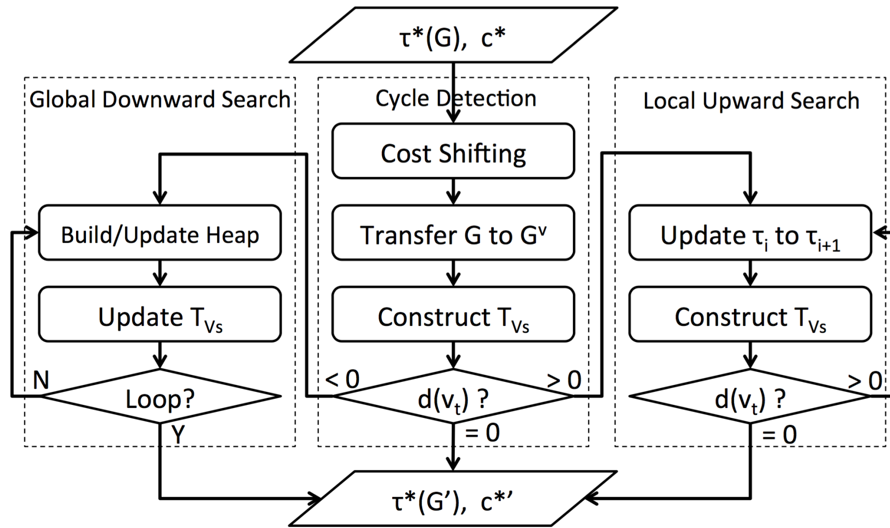

Figure 7.4: Overview of our incremental MCR algorithm.

7.3.5 Cycle detection

In the beginning, at our cost shifting step, we filter out the incremental change which will not affect MCR. If this is the case, we can directly exit the MCR algorithm and output $\tau^*(G)$ as $\tau^*(G')$. If the change has a potential to affect MCR, we continue to detect whether the MCR is increased or decreased. To do this, we first transform G into G^v by replacing node $v \in V$

المنارة للاستشارات
www.manaraa.com

with two new node v_s, v_t. Next, we build the longest path tree T_{v_s} rooted at v_s. Based on the longest distance from v_s to v_t, we will be able to detect the changing direction of MCR.

7.3.5.1 Cost shifting

As we discussed in Section III-A case 1), if the changed edge is not on c^* and the edge cost is decreased, we can guarantee that the MCR will not be affected. The idea of cost shifting is to transform all edge changes into the above case, by shifting edge costs from the input (or output) edges of v to the output (or input) edges of v. As an example, Fig. 7.5(a) shows the current edge cost changes of v with 4 decreased edges and 1 increased edge. By shifting 9 units of cost from the output edges of v to the input edges of v, we get the new cost changes shown in Fig. 7.5(b). Assuming $v \notin c^*$, since only decreased edges exist after cost shifting, this change of v can be identified as not affecting MCR.

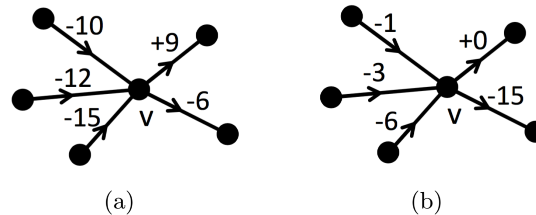

Figure 7.5: (a) Before the cost shifting. (b) After the cost shifting.

In general, by applying the cost shifting idea, we can exit the MCR algorithm if $v \notin c^*$ and the increment change belongs to one of the following two cases: (1) all edge costs are decreased. (2) all edge costs on one side (input or output) of v are decreased, and the smallest amount of decreasing at the decreased side is largest than largest amount of increasing on the other side.

7.3.5.2 Transform G into G^v

If the incremental change has a potential to affect MCR, we continue to this step and transform G into G^v as follows. We remove v from G and add two new nodes v_s, v_t to G, with

المنارة للاستشارات
www.manaraa.com

v_s connecting to all v's output edges and v_t connecting to all v's input edges as shown in Fig. 7.6.

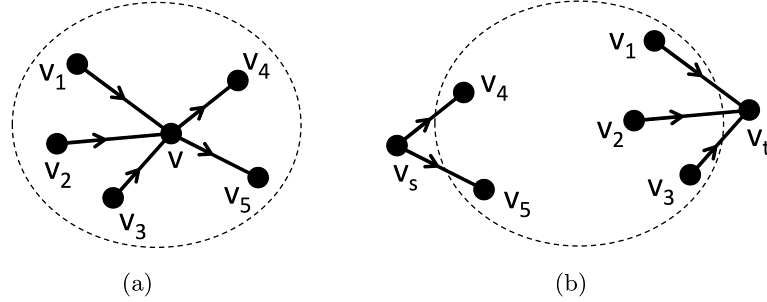

Figure 7.6: (a) Before breaking at v. (b) After breaking at v.

Let $\tau_0 = \tau^*(G)$, we can get the corresponding $G^v_{\tau_0}$ for G^v. Let T_{v_s} denotes the longest path tree rooted at v_s. Then we can have the following Theorem:

Theorem 1. T_{v_s} is well defined in $G^v_{\tau_0}$.

Proof: Before the incremental change, we have $l(c) \leq 0\ \forall c \in G_{\tau_0}$. After the incremental change, only the cycles passing through v can be positive in G_{τ_0}. Since all $c \in C_v$ is broken at v in $G^v_{\tau_0}$, they will not form a positive cycle in $G^v_{\tau_0}$. Therefore, we have $l(c) \leq 0\ \forall c \in G^v_{\tau_0}$, and thus the longest paths in $G^v_{\tau_0}$ are well defined. □

7.3.5.3 Constructing T_{v_s} on $G^v_{\tau_0}$

Constructing T_{v_s} on $G^v_{\tau_0}$ is equivalent to the problem of finding a single source longest path tree in a graph without positive cycle. Since $G^v_{\tau_0}$ contains both negative and positive length edges, Dijkstra's algorithm is not applicable here. One way to construct T_{v_s} is to use the Bellman–Ford algorithm and update the node distances in a breath first search manner, as suggested in [77]. However, the breath first search has a very limited control on the updating order of the nodes, and thus each node can be repeatedly updated for many times [15]. Therefore, the runtime of this approach is not good.

We propose a distance bucket approach to help us update the nodes in an appropriate order, which can effectively reduce the total number of updates on each node and therefore

المنارة للاستشارات
www.manaraa.com

speed up the process of constructing T_{v_s}. The basic idea of the distance bucket approach is similar to the Dijkstra's algorithm: we always pick the node which has the largest distance to update. Different from Dijkstra's algorithm, this cannot guarantee that the updated node will not be updated again, but the chance that this node get updated again will be much smaller compared with a random updating order. Instead of maintaining a priority queue to exactly find the node with largest distance, we only differentiate the nodes by putting them into certain buckets based on the range of their distance. One reason we do this is that it is not necessary to differentiate the node distances exactly, as repeated update of the nodes cannot be avoided anyway. Another reason is that maintaining a priority queue is expensive, especially considering the total number of edge update operations is huge.

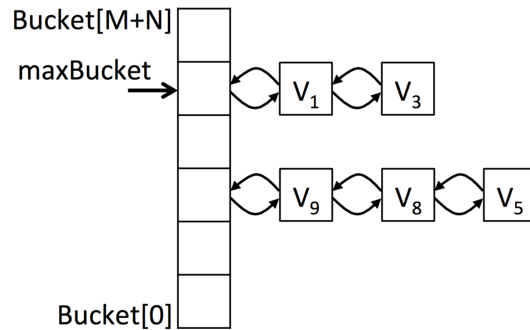

Figure 7.7: The distance bucket data structure.

Assuming we have $M + N$ buckets denoted from $bucket[0]$ to $bucket[M + N - 1]$ with M buckets for negative distances and N buckets for positive distances, as shown in Fig. 7.7. Let d_u be a unit range of distance covered by a bucket. Then, for a particular node v, its corresponding bucket index can be calculated as $M + d(v)/d_u$. Instead of storing a copy of all the contained nodes, the distance bucket only pointing to one of the contained node as shown in Fig. 7.7. The rest of the contained nodes will simply be connected to the this node in a doubly linked list manner. The details of our approach is shown in Algorithm 1.

المنارة للاستشارات
www.manaraa.com

Algorithm 8 The distance bucket approach

Ensure: Constructing T_{v_s} on $G^v_{\tau_0}$.
1: Insert s to $bucket[0]$ and set $max := 0$;
2: **while** $max > 0$ **do**
3: Pick and delete node u from the $bucket[max]$.
4: **for each** $(u, v) \in E_{\tau_0}$ **do**
5: **if** $d(v) < d(u) + l(u, v)$ **then**
6: $d(v) := d(u) + l(u, v)$;
7: Find the bucket index i based on $d(v)$;
8: **if** v is not in any buckets **then**
9: Insert v to $bucket[i]$;
10: **else**
11: Delete v from its current bucket;
12: Insert v to $bucket[i]$;
13: **end if**
14: **if** $i > max$ **then**
15: $max := i$;
16: **end if**
17: **end if**
18: **end for**
19: **while** $bucket[max]$ is empty **do**
20: $max := max - 1$
21: **end while**
22: **end while**

7.3.5.4 Detecting the changing direction of MCR

After constructing T_{v_s}, we can get $d(v_t)$ which is the longest distance from v_s to v_t in $G^v_{\tau_0}$. If $d(v_t) > 0$, it means a positive cycle passing through v exists in G_{τ_0}, and $\tau^*(G) < \tau^*(G')$. Therefore, we search upwards to find the new MCR. If $d(v_t) = 0$, it means $\tau^*(G) = \tau^*(G')$ and c^* remains to be the largest cycle in G'. So we can exit the MCR algorithm. If $d(v_t) < 0$, it means the largest cycle passing through v in G_{τ_0} is negative. If $v \notin c^*$, we can exit the MCR algorithm as the MCR will not be affected in this case. Otherwise, it means $\tau^*(G) > \tau^*(G')$ and we search downwards to find the new MCR.

7.3.6 Local upward search

In this step, we search upwards until $\tau^*(G')$ is identified. It is safe for us to only perform a local search among all the cycles in C_v based on the following theorem:

Theorem 2. If $\tau^*(G') > \tau^*(G)$, $c^{*\prime} \in C_v$.

المنارة للاستشارات
www.manaraa.com

Proof: In Section III-A, the only cases which the incremental change can increase the MCR are case 2) and case 4). As we have discussed, we can guarantee $c^{*\prime}$ is passing through the changed edge e in both these two cases. Since e is connected to v, this means $c^{*\prime}$ must also pass through v. □

The strategy we used to perform the local upward search is similar to HOW. Assuming the current cycle ratio is τ_i, we first increase τ_i to τ_{i+1}, which makes $d(v_t) = 0$ in current T_{v_s}. Next, we construct the new T_{v_s} in $G^v_{\tau_{i+1}}$ using Algorithm I and get the corresponding new $d(v_t)$. If $d(v_t) > 0$, it means there are still positive cycles existing in $G_{\tau_{i+1}}$ whose cycle ratio is larger than τ_{i+1}. So we repeat the first step and keep updating the cycle ratio. Otherwise, we can exit the MCR algorithm and output τ_{i+1} as $\tau^*(G')$.

7.3.7 Global downward search

Our algorithm enters this step only when the cycle ratio of the previous largest cycle is decreased, i.e., $\tau(c^*) < \tau^*(G)$ in G'. In one case, c^* might remain to be the largest cycle in G' and we need to perform a global search to verify that $\tau(c) \leq \tau(c^*)$ $\forall c \in G'$. In the other case, another cycle can replace c^* and becomes the new largest cycle in G'. Since we have no clue where this largest cycle is located, a global search for all cycles in G' is also required.

We leverage KO to perform this downward global search. In particular, the T_{v_s} we calculated during the cycle detection can be reused here. Thus, instead of running KO starting from a very large τ with the initial longest path tree as shown in Fig. 7.2(a), we can start KO from $\tau^*(G)$ with T_{v_s}. However, T_{v_s} is a longest path tree in $G^v_{\tau_0}$ rooted at node v, while the original KO requires the longest path tree rooted at an artificial node s, as shown in Fig. 7.2. Simply starting KO from $\tau^*(G)$ with T_{v_s} will make all cycles $c \in C_v$ cannot be examined, and the algorithm will be incorrect if $c^{*\prime} \in C_v$. Therefore, we modify KO like this: we add a pseudo edge (v_t, v_s) which is connecting node v_t to node v_s, and insert (v_t, v_s) into the max heap with $key(v_t, v_s) = d(v_t)/t(v_t)$. If (v_t, v_s) is picked during the execution of KO, it means $c^{*\prime} \in C_v$. Since $d(v_t)$ represents the largest cycle in C_v, it is safe for us to exit the MCR algorithm and output $\tau^*(G') = d(v_t)/t(v_t)$.

المنارة للاستشارات
www.manaraa.com

7.4 Timing-driven detailed placement

We propose a timing-driven detailed placement approach which applies our incremental MCR algorithm. Considering the type of circuits, i.e. asynchronous circuits [9] or synchronous circuits using retiming or clock scheduling techniques [34], whose performance is bounded by the MCR of the most critical cycle (c^*) in the circuit. For asynchronous circuits, c^* is defined as the timing loop which has the largest cycle delay divided by the number of tokens along the cycle. For synchronous circuits, c^* is defined as the timing loop which has the largest cycle delay divided by the number of flip-flops along the cycle. Here, we assume delay is proportional to the wirelength. Given an initial legalized placement, our goal is to reduce the MCR of the circuit by sequentially swapping a cell on c^* with one which is not on c^*.

The basic idea of our approach is illustrated in Fig. 7.7. First, we randomly pick a cell on c^*, i.e. cell v in Fig. 7.7. Next, we find the two neighboring cells of v on c^*, i.e. cell v_1 and v_2. The coordinates of v_1 and v_2 can define an optimal region $(x(v_1), x(v_2), y(v_1), y(v_2))$ for v, shown as the blue rectangle in Fig. 7.7. Assuming v' is a cell within this optimal region, by swapping v with v', we can minimize the total Manhattan distance of (v_1, v) and (v, v_2). Thus, $\tau(c^*)$ is reduced. However, it is possible that some cycle passing through v' becomes worse than c^* after the swap. Hence, we need to perform timing analysis using the incremental MCR algorithm to see whether the swap is beneficial before actually swapping the two cells. The details of our approach is shown in Algorithm 2.

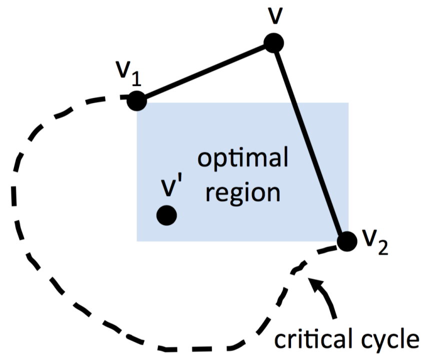

Figure 7.8: Timing-driven detailed placement.

www.manaraa.com

Table 7.1: Comparison on small size random graphs

Graph	# of nodes	# of edges	MCR			M1 Runtime (s)				M2 Runtime (s)			
			Init	M1	M2	LP	HOW	KO	Ours	LP	HOW	KO	Ours
r01	12752	36681	4.03	4.03	3.76	566.43	4.07	1.92	0.44	720.50	3.97	2.58	1.15
r02	19601	61829	4.39	4.39	4.06	2238.94	4.69	3.12	0.74	2885.36	9.92	4.79	2.06
r03	23136	66429	5.04	5.04	3.81	2380.95	3.20	2.72	0.92	3130.45	8.26	5.07	2.32
r04	27507	74138	4.50	4.50	3.61	2963.66	3.65	3.20	0.94	3866.92	12.48	5.16	2.77
r05	29347	98793	4.39	4.49	4.18	5090.83	18.03	8.07	1.42	4644.15	20.34	10.28	4.12
r06	32498	93493	3.87	3.87	3.56	6829.63	14.60	7.50	1.42	9060.00	24.86	9.35	4.02
r07	45926	127774	4.18	4.18	3.58	12377.20	9.95	6.93	1.80	15743.50	33.72	13.38	5.57
r08	51309	154644	4.84	4.84	4.11	–	10.01	7.52	2.11	–	27.24	11.40	6.31
r09	53395	161430	4.70	4.70	4.18	–	9.19	8.28	2.12	–	41.73	15.24	6.42
					Norm.	–	6.496	4.136	1.000	–	5.253	2.223	1.000
r10	69429	223090	4.45	4.50	4.36	–	70.51	24.41	4.09	–	66.74	23.96	9.78
r11	70558	199694	3.84	3.84	3.57	–	95.00	24.77	4.20	–	95.62	24.42	9.53
r12	71076	241135	4.83	4.83	4.47	–	109.14	24.35	4.77	–	91.43	24.91	10.57
r13	84199	257788	4.79	4.79	4.07	–	29.65	17.12	4.25	–	100.52	23.84	10.80
r14	154605	394497	5.08	5.08	3.48	–	32.52	18.99	5.43	–	103.43	30.80	17.90
r15	161570	529562	6.30	6.30	4.45	–	22.46	22.59	7.21	–	268.69	52.77	26.33
r16	183484	589253	4.76	4.76	4.36	–	187.87	42.79	8.44	–	315.99	66.09	28.64
r17	185495	671174	5.24	5.24	4.94	–	488.10	74.59	12.75	–	620.42	98.54	42.62
r18	210613	618020	4.34	4.34	4.17	–	390.57	64.55	11.27	–	442.07	82.08	36.02
					Norm.	–	22.848	5.034	1.000	–	10.952	2.224	1.000
r19	262144	851968	5.14	5.14	4.38	–	90.15	63.92	13.36	–	481.11	115.48	46.68
r20	311744	1013166	6.03	6.03	4.48	–	65.41	62.41	16.18	–	705.11	134.60	55.87
r21	370728	1204865	4.65	4.65	4.41	–	341.74	125.63	25.92	–	1023.05	169.74	65.69
r22	440879	1432834	5.06	5.06	4.65	–	399.31	154.68	36.86	–	1329.38	178.20	76.65
r23	524288	1703936	6.04	6.04	4.50	–	268.13	149.38	48.55	–	1112.68	240.61	97.21
r24	623487	2026333	4.59	4.59	4.44	–	1868.32	327.39	49.96	–	1679.11	305.82	116.13
r25	741455	2409729	29.75	29.75	43.13	–	71.00	47.10	38.21	–	75.93	53.94	91.21
r26	881744	2865667	5.52	5.52	4.66	–	294.65	189.08	44.39	–	1222.89	297.44	141.97
r27	1048576	3407872	4.78	4.78	4.57	–	1217.15	332.73	54.74	–	3355.29	450.52	180.88
					Norm.	–	14.066	4.426	1.000	–	12.593	2.231	1.000

المنارة للاستشارات
www.manaraa.com

Table 7.2: Analysis on medium and large size random graphs

Graph	Updates per node		M2 Runtime (s)		DB Runtime Breakdown (s)			
	BFS	DB	BFS	DB	CD	LUS	GDS	Others
r10	9.00	1.23	27.03	9.78	3.53	1.84	4.03	0.37
r11	9.58	1.32	22.17	9.53	3.65	1.00	4.53	0.35
r12	9.21	1.35	24.96	10.57	4.27	1.36	4.59	0.35
r13	9.27	1.21	28.89	10.80	3.84	1.30	5.31	0.35
r14	6.27	1.09	40.13	17.90	6.05	2.27	9.00	0.58
r15	9.75	1.19	72.70	26.33	8.39	3.78	13.48	0.66
r16	9.68	1.16	77.73	28.64	9.98	3.87	13.80	0.99
r17	10.14	1.16	89.75	42.62	13.09	5.65	22.49	1.39
r18	9.74	1.16	90.33	36.02	12.17	6.15	15.96	1.74
Norm.	7.614	1.000	2.465	1.000	0.338	0.142	0.485	0.035
r19	9.98	1.24	132.18	46.68	15.89	6.19	23.17	1.43
r20	10.71	1.21	175.30	55.87	19.02	8.25	27.04	1.56
r21	11.18	1.27	199.96	65.69	23.51	8.58	31.33	2.27
r22	11.22	1.15	232.59	76.65	24.71	8.71	40.64	2.58
r23	10.32	1.19	268.26	97.21	34.86	11.28	48.30	2.77
r24	10.99	1.29	381.76	116.13	42.58	15.95	52.21	5.40
r25	3.17	1.00	205.87	91.21	37.04	20.43	30.83	2.91
r26	8.85	1.05	439.19	141.97	44.63	21.62	70.85	4.87
r27	11.07	1.20	779.65	180.88	62.75	23.76	87.36	7.01
Norm.	8.257	1.000	3.227	1.000	0.350	0.143	0.472	0.035

المنارة للاستشارات
www.manaraa.com

Table 7.3: Comparison on ISPD 2005 benchmarks

Design	# of nodes	# of nets	c^* moves	Skip moves	Total moves	HPWL x 10^6 (nm)		MCR		Runtime (s)		
						Init	Final	Init	Final	HOW	KO	Ours
adaptec1	210861	644176	39	2	514	77.78	77.89	2582.00	1390.60	248.42	409.98	48.19
adaptec2	254425	731135	23	4	398	87.68	87.73	4301.60	3456.80	80.10	49.43	34.40
adaptec3	450642	1289483	18	7	280	203.33	203.35	8084.50	2523.00	139.44	130.75	35.13
adaptec4	494590	1246535	24	1	482	183.59	183.72	2729.00	2115.00	327.05	171.16	95.88
bigblue1	277022	794445	24	2	225	97.21	97.30	3125.67	2973.00	40.99	30.59	32.13
bigblue2	528704	1267929	35	2	422	147.80	147.81	3494.91	3491.91	305.14	104.46	98.49
bigblue3	1094904	2511616	26	6	413	324.91	325.05	6020.33	3692.25	305.37	261.76	178.60
bigblue4	2168351	5691264	29	4	448	790.65	790.85	6085.50	3885.25	1128.32	704.68	403.59
		Norm.	0.069	0.009	1.000	1.000	1.000	1.000	0.646	2.779	2.011	1.000

www.manaraa.com

Algorithm 9 A timing-driven detailed placement approach

Ensure: Reduce MCR of the circuit
1: $n = 1$; /* loop counter */
2: $best_MCR = +\infty$;
3: **while** $n < limit$ **do**
4: Randomly pick a cell v on c^* with neighboring cells v_1, v_2;
5: Set $optimal_region := (x(v_1), x(v_2), y(v_1), y(v_2))$;
6: Set $x_{opt} := 0.5 * (x(v_1) + x(v_2))$;
7: Set $y_{opt} := 0.5 * (y(v_1) + y(v_2))$;
8: Move v to (x_{opt}, y_{opt}).
9: Incrementally calculate MCR;
10: **for each** v' in $optimal_region$ **do**
11: Move v' to (x_v, y_v);
12: Incrementally calculate MCR as $current_MCR$;
13: **if** $current_MCR < best_MCR$ **then**
14: $best_MCR = current_MCR$;
15: $best_node = v'$;
16: **end if**
17: **end for**
18: Move v to $(x_{best_node}, y_{best_node})$;
19: Incrementally calculate MCR;
20: Move $best_node$ to (x_v, y_v);
21: Incrementally calculate MCR as $best_MCR$;
22: $n = n + 1$;
23: **end while**

7.5 Experiments

The proposed incremental mean cycle algorithm is implemented in C++ and runs on a Linux PC with 94 GB of memory and 2.67 GHz Intel Xeon CPU.

We generate a set of random graphs following the same graph size and method used in [17]. Given an input total number of nodes and total number of edges, we first generate the desired number of nodes in the graph. Next, we randomly pick two nodes in the graph and connect them. This step is repeated until the desired number of edges is reached. The self loops (an edge connecting a node to itself) and duplicated edges (two edges connecting the same pair of nodes) are disallowed. In addition, we connect all the nodes using a circle to make the graph strongly connected. Both the cost and transition of each edge are randomly generated between 1 and 300.

In the beginning (i.e., before any incremental changes are made), our algorithm uses KO to find the initial MCR and the corresponding largest cycle as a starting point. For all the random graphs and circuit benchmarks, our algorithm sets both the total number of negative

المنارة للاستشارات
www.manaraa.com

and positive buckets to be 10^6. In addition, we set d_u to be 10. Thus, a distance range $[-10^7, +10^7]$ is covered, which is more than enough. In case any node has a distance below or above this range, we will assign it to the first or last bucket. We implement three other MCR algorithms for comparison: the linear programming (LP), HOW and KO. LP is formulated as we discussed in Section II-A, and solved using the API of Gurobi optimizer [1]. Both HOW and KO are implemented following the description in [18][17]. In particular, we implement a binary heap as the max heap used in KO.

For each random graph, we sequentially perform 100 node changes and calculate the MCR after each change. For each changing node, we randomly change the costs of all its input and output edges. Two different methods are used to pick the changing node. In one method, which we referred as "M1", we randomly pick a node among all the nodes in the graph. Our algorithm is able to run faster in this case, as only the upward search might be performed if we are not changing c^*, which is quite often in M1. In another method, which we referred as "M2", we always pick a node on c^* to change. This is the most difficult case for our algorithm, as both downward and upward search might be performed.

Table 7.1 shows the experimental results for random graphs, which are divided into three sets to simulate the applications with different scale. Columns "Init", "M1" and "M2" reports the initial MCR, the final MCR after 100 node changes using M1, and the final MCR after 100 node changes using M2 respectively. In general, LP is much slower than other algorithms. We denote the runtime of LP as "–", if it exceeds our runtime limit. Compared with HOW, our algorithm is about 5X$\sim$23X faster among all the experiments. The performance of HOW is not good especially on large size graphs. Compared with KO, our algorithm is about 2X$\sim$5X faster among all the experiments. In addition, as expected, the runtime of our algorithm in M1 is better than the runtime in M2.

Table 7.2 shows analysis of our algorithm in M2 on medium and large size graphs. We compare the runtime of our MCR algorithm using the distance bucket approach (DB) with our MCR algorithm using the BFS approach in [77]. The column "Updates per node" shows the average number of distance updates per node, and is calculated as (total # of node distance updates)/(# of T_{v_s} $\times$ # of nodes in the graph). It shows the DB approach is effectively

المنارة للاستشارات
www.manaraa.com

reducing the updates per node and thus can achieve faster runtime. In addition, we show a runtime break down of our algorithm using DB. The columns “CD”, “LUS” and “GDS” denote the cycle detection, local upward search and global downward search process respectively.

We use ISPD 2005 benchmarks [55] as the circuit benchmarks. Assuming a hypernet is connecting one output pin and p input pins of some gates, we represent this hypernet with p two-pin nets, by connecting the output pin with each input pin. Since there is no cell library type information in [55], we cannot calculate the wire delay. Instead, we set the cost of each two-pin net to be its HPWL, and the transition time of each two-pin net to 1. In addition, we ignore the fixed cells (i.e. terminals, macro blocks) and the nets connecting to them.

We stop the detailed placement if the improvement of MCR is less than 0.1% when we do the swap. Since our algorithm needs to incrementally calculate MCR at each move, a swap operation will need two calculations, while it only needs one calculation for HOW and KO. Thus, line 9 and line 19 in Algorithm 2 is required for our algorithm, but it is not needed for HOW and KO. Therefore, the total # of MCR calculations for our algorithm is larger than HOW and KO in this application. Table 7.3 shows the experimental results on circuit benchmarks. Columns “c^* moves”, "Skip moves" and "Total moves" denote the # of moves on c^*, the # of moves skipped using our cost shifting idea and the total # of moves respectively. The runtime of the proposed detailed placement approach using three different MCR algorithms for timing analysis is compared. In particular, the placer based on our MCR algorithm is about 2X faster than the KO version and 2.8X faster than the HOW version.

7.6 Conclusions

In this paper, we have proposed an incremental MCR algorithm which is able to calculate the MCR more efficiently by considering the incremental changes. The previous MCR allows us to break the graph at the changed node, and therefore detecting the changing directions of the MCR by solving a longest path problem in a graph without positive cycle. Based on the detected direction, our algorithm will either search upward or downward until the new MCR is found. We preform experiments on both random graphs and circuit benchmarks. The experimental results show our algorithm is more efficient compared with HOW and KO.

www.manaraa.com

BIBLIOGRAPHY

[1] Gurobi Optimizer: http://www.gurobi.com.

[2] Semiconductor Industry Association. The International Technology Roadmap for Semiconductors, 2011.

[3] Saurabh N. Adya and Igor L. Markov. Combinatorial Techniques for Mixed-Size Placement. *TODAES*, 10(1):58–90, 2005.

[4] Sang Hoon Baek, Ha Young Kim, Young Keun Lee, Duck Yang Jin, Se Chang Park, and Jun Dong Cho. Ultra-High Density Standard Cell Library Using Multi-Height Cell Structure. In *Smart Materials, Nano-and Micro-Smart Systems*, 2008.

[5] Mokhtar S. Bazaraa, Hanif D. Sherali, and Chitharanjan Marakada Shetty. *Nonlinear Programming: Theory and Algorithms*. 2013.

[6] P. A. Beerel, G.D. Dimou, and A.M. Lines. Proteus: An ASIC Flow for GHz Asynchronous Designs. *Design Test of Computers, IEEE*, 28(5):36–51, Sept 2011.

[7] Peter A Beerel, Andrew Lines, Mike Davies, and Nam-Hoon Kim. Slack Matching Asynchronous Designs. In *ASYNC*, pages 11–pp, 2006.

[8] Peter A Beerel, Andrew Lines, Mike Davies, and Nam-Hoon Kim. Slack Matching Asynchronous Designs. In *Asynchronous Circuits and Systems (ASYNC), 2006*, pages 11–pp. IEEE, 2006.

[9] Peter A Beerel, Recep O Ozdag, and Marcos Ferretti. *A Designer's Guide to Asynchronous VLSI*. Cambridge University Press, 2010.

للاستشارات المنارة

www.manaraa.com

[10] Michel RCM Berkelaar and Jochen AG Jess. Gate Sizing in MOS Digital Circuits with Linear Programming. In *DATE*, pages 217–221, 1990.

[11] Duane G. Breid, Michael J. Colwell, Tushar R. Gheewala, and Henry H. Yang. Dual-Height Cell with Variable Width Power Rail Architecture. US Patent, 2005.

[12] Michael Burstein and Mary N Youssef. Timing Influenced Layout Design. In *Proceedings of the 22nd ACM/IEEE Design Automation Conference*, pages 124–130. IEEE Press, 1985.

[13] Nitin Chandrachoodan, Shuvra S Bhattacharyya, and KJ Liu. Adaptive Negative Cycle Detection in Dynamic Graphs. In *ISCAS 2001*.

[14] Chung-Ping Chen, C.C.N. Chu, and D. F. Wong. Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation. In *ICCAD 1998*, pages 617–624, Nov 1998.

[15] Boris V Cherkassky, Andrew V Goldberg, and Tomasz Radzik. Shortest Paths Algorithms: Theory and Experimental Evaluation. *Mathematical programming*, 73(2):129–174, 1996.

[16] Jason Cong and Min Xie. A Robust Detailed Placement for Mixed-Size Ic Designs. In *ASP-DAC*, page 7 pp., 2006.

[17] Ali Dasdan. Experimental Analysis of The Fastest Optimum Cycle Ratio and Mean Algorithms. *TODAES*, 9(4):385–418, 2004.

[18] Ali Dasdan, S Irani, and Rajesh K Gupta. An Experimental Study of Minimum Mean Cycle Algorithms. *Tech. rep. 98-32, UC Irvine*, 1998.

[19] Mike Davies, Andrew Lines, Jon Dama, Alain Gravel, Robert Southworth, Georgios Dimou, and Peter Beerel. A 72-Port 10G Ethernet Switch/Router Using Quasi-Delay-Insensitive Asynchronous Design. In *ASYNC*, pages 103–104, 2014.

[20] Sorin Dobre, Andrew B Kahng, and Jiajia Li. Mixed Cell-Height Implementation for Improved Design Quality in Advanced Nodes. In *ICCAD 2015*, pages 854–860. IEEE Press, 2015.

المنارة للاستشارات

www.manaraa.com

[21] Doratha E. Drake and Stefan Hougardy. A Simple Approximation Algorithm for the Weighted Matching Problem. *Information Processing Letters*, 2003.

[22] Ran Duan and Seth Pettie. Linear-Time Approximation for Maximum Weight Matching. *Journal of the ACM (JACM)*, 61(1):1, 2014.

[23] Jack Edmonds. Maximum Matching and a Polyhedron with 0, L-Vertices. *Journal of Research of the National Bureau of Standards*, 1965.

[24] Hans Eisenmann and Frank M Johannes. Generic Global Placement and Floorplanning. In *DAC*, pages 269–274. ACM, 1998.

[25] John P Fishburn. Clock Skew Optimization. *Computers, IEEE Transactions on*, 39(7):945–951, 1990.

[26] JP Fishburn. TILOS: A Posynomial Programming Approach to Transistor Sizing. In *ICCAD*, 1985.

[27] G. Flach, T. Reimann, G. Posser, M. Johann, and R. Reis. Effective Method for Simultaneous Gate Sizing and Vth Assignment Using Lagrangian Relaxation. *TCAD*, 33(4):546–557, 2014.

[28] Tong Gao, Pravin M Vaidya, and CL Liu. A Performance Driven Macro-cell Placement Algorithm. In *Design Automation Conference, 1992. Proceedings., 29th ACM/IEEE*, pages 147–152. IEEE, 1992.

[29] Loukas Georgiadis, Andrew V Goldberg, Robert E Tarjan, and Renato F Werneck. An Experimental Study of Minimum Mean Cycle Algorithms. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pages 1–13. Society for Industrial and Applied Mathematics, 2009.

[30] Behnam Ghavami and Hossein Pedram. Low Power Asynchronous Circuit Back-End Design Flow. *Microelectronics Journal*, 2011.

المنارة للاستشارات

www.manaraa.com

[31] Takeo Hamada, Chung-Kuan Cheng, and Paul M Chau. Prime: A Timing-driven Placement Tool using A Piecewise Linear Resistive Network Approach. In *DAC*, pages 531–536. ACM, 1993.

[32] Ronald A Howard. Dynamic Programming And Markov Process. *The M.I.T Press, Cambridge, Mass*, 1960.

[33] Jin Hu, Andrew B. Kahng, SeokHyeong Kang, Myung-Chul Kim, and Igor L. Markov. Sensitivity-Guided Metaheuristics for Accurate Discrete Gate Sizing. In *ICCAD*, pages 233–239, 2012.

[34] Aaron P. Hurst, P. Chong, and A. Kuehlmann. Physical Placement Driven by Sequential Timing Analysis. In *ICCAD 2004*, pages 379–386, Nov 2004.

[35] M. Iizuka and H. Saito. A Floorplan Method for ASIC Designs of Asynchronous Circuits with Bundled-data Implementation. In *New Circuits and Systems Conference (NEWCAS), 2013*, pages 1–4, June 2013.

[36] Michael AB Jackson and Ernest S Kuh. Performance-driven Placement of Cell based IC's. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 370–375. ACM, 1989.

[37] Robert Karmazin, Stephen Longfield, Carlos Tadeo Ortega Otero, and Rajit Manohar. Timing Driven Placement for Quasi Delay-Insensitive Circuits. In *Asynchronous Circuits and Systems (ASYNC), 2015 21st IEEE International Symposium on*, pages 45–52. IEEE, 2015.

[38] Richard M Karp and James B Orlin. Parametric Shortest Path Algorithms with An Application to Cyclic Staffing. *Discrete Applied Mathematics*, 3(1):37–45, 1981.

[39] Myung-Chul Kim, Dong-Jin Lee, and Igor L Markov. SimPL: An Effective Placement Algorithm. *TCAD*, 31(1):50–60, 2012.

[40] Myung Chul Kim, Natarajan Viswanathan, Zhuo Li, and Charles Alpert. ICCAD-2013 CAD Contest in Placement Finishing and Benchmark Suite. In *ICCAD 2013*.

المنارة للاستشارات
www.manaraa.com

[41] Jürgen M Kleinhans, Georg Sigl, Frank M Johannes, and Kurt J Antreich. GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization. *TCAD*, 10(3):356–365, 1991.

[42] E. Kounalakis and C.P. Sotiriou. CPlace: A Constructive Placer for Synchronous and Asynchronous Circuits. In *Asynchronous Circuits and Systems (ASYNC), 2011*, pages 22–29, April 2011.

[43] Eugene L Lawler. Combinatorial Optimization: Networks and Matroids . *Holt, Rinehart and Winston, New York*, 1976.

[44] Ce Leiserson and Jb Saxe. Optimizing Synchronous Systems. *Journal of VLSI and computer systems*, 1(1):41–67, 1983.

[45] Charles E Leiserson and James B Saxe. Retiming Synchronous Circuitry. *Algorithmica*, 6(1-6):5–35, 1991.

[46] Tao Lin, C. Chu, J.R. Shinnerl, I. Bustany, and I. Nedelchev. POLAR: Placement Based on Novel Rough Legalization and Refinement. In *ICCAD 2013*, pages 357–362, Nov 2013.

[47] Andrew Matthew Lines. *Pipelined Asynchronous Circuits*. Master's thesis, California Institute of Technology, 1998.

[48] Vinicius S. Livramento, Chrystian Guth, Jos Lus Gntzel, and Marcelo O. Johann. Fast and Efficient Lagrangian Relaxation-Based Discrete Gate Sizing. In *DATE*, pages 1855–1860, 2013.

[49] Jan Magott. Performance Evaluation of Concurrent Systems using Petri Nets. *Information Processing Letters*, 18(1):7–13, 1984.

[50] Alain J. Martin, Steven M. Burns, Tak-Kwan Lee, Drazen Borkovic, and Pieter J. Hazewindus. The First Aysnchronous Microprocessor: The Test Results. *SIGARCH*, pages 95–98, 1989.

المنارة للاستشارات

www.manaraa.com

[51] Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nystroem, Paul Penzes, Robert Southworth, and Uri Cummings. The Design of an Asynchronous MIPS R3000 Microprocessor. In *Advanced Research in VLSI*, pages 164–164, 1997.

[52] Chris J Myers. *Asynchronous Circuit Design*. John Wiley & Sons, 2004.

[53] M. Najibi and P. A. Beerel. Performance Bounds of Asynchronous Circuits with Mode-Based Conditional Behavior. In *Asynchronous Circuits and Systems (ASYNC)*, pages 9–16, May 2012.

[54] Mehrdad Najibi, Peter Beerel, et al. Integrated fanout optimization and slack matching of asynchronous circuits. In *ASYNC*, pages 69–76, 2014.

[55] Gi-Joon Nam, Charles J Alpert, Paul Villarrubia, Bruce Winter, and Mehmet Yildiz. The ISPD2005 Placement Contest and Benchmark Suite. In *ISPD 2005*.

[56] William C Naylor, Ross Donelly, and Lu Sha. Non-linear Optimization System and Method for Wirelength and Delay Optimization for An Automatic Electric Circuit Placer, October 9 2001. US Patent 6,301,693.

[57] David Nguyen, Abhijit Davare, Michael Orshansky, David Chinnery, Brandon Thompson, and Kurt Keutzer. Minimization of Dynamic and Static Power Through Joint Assignment of Threshold Voltages and Sizing Optimization. In *ISLPED*, pages 158–163, 2003.

[58] Wing Ning. Strongly NP-Hard Discrete Gate-Sizing Problems. *TCAD*, 13(8):1045–1051, 1994.

[59] Muhammet Mustafa Ozdal, Chirayu Amin, Andrey Ayupov, Steven Burns, Gustavo Wilke, and Cheng Zhuo. The ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite. In *ISPD*, pages 161–164, 2012.

[60] Muhammet Mustafa Ozdal, Chirayu Amin, Andrey Ayupov, Steven M. Burns, Gustavo R. Wilke, and Cheng Zhuo. An Improved Benchmark Suite for the ISPD-2013 Discrete Cell Sizing Contest. In *ISPD*, pages 168–170, 2013.

المنارة للاستشارات
www.manaraa.com

[61] Muhammet Mustafa Ozdal, Steven Burns, and Jiang Hu. Algorithms for Gate Sizing and Device Parameter Selection for High-performance Designs. *TCAD*, 31(10):1558–1571, 2012.

[62] David Z Pan, Bill Halpin, and Haoxing Ren. Timing-driven Placement. *Handbook of Algorithms for VLSI Physical Automation*, pages 223–233, 2007.

[63] Min Pan, N. Viswanathan, and C. Chu. An Efficient and Effective Detailed Placement Algorithm. In *ICCAD 2005*, pages 48–55, Nov 2005.

[64] James L Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981.

[65] Sergiy Popovych, Hung Hao Lai, Chieh Min Wang, Yih Lang Li, Wen Hao Liu, and Ting Chi Wang. Density-Aware Detailed Placement with Instant Legalization. In *DAC 2014*, pages 1–6, June 2014.

[66] Mallika Prakash. *Library Characterization and Static Timing Analysis of Asynchronous Circuits*. ProQuest, 2007.

[67] Piyush Prakash and Alain J Martin. Slack Matching Quasi Delay-insensitive Circuits. In *ASYNC*, pages 10–pp, 2006.

[68] Julia Casarin Puget, Guilherme Flach, Marcelo Johann, and Ricardo Reis. Jezz: An Effective Legalization Algorithm for Minimum Displacement. In *Proceedings of the 28th Symposium on Integrated Circuits and Systems Design*, page 19. ACM, 2015.

[69] Colin R Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Inc., 1993.

[70] Sachin S. Sapatnekar, Vasant B. Rao, Pravin M. Vaidya, and Sung-Mo Kang. An Exact Solution to the Transistor Sizing Problem for Cmos Circuits Using Convex Optimization. *TCAD*, 12(11):1621–1634, 1993.

المنارة للاستشارات
www.manaraa.com

[71] M. Singh and S.M. Nowick. MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines. *Very Large Scale Integration (VLSI) Systems*, 15(6):684–698, June 2007.

[72] Jens Spars and Steve Furber. *Principles of Asynchronous Circuit Design*. Springer, 2002.

[73] Arvind Srinivasan, Kamal Chaudhary, and Ernest S Kuh. RITUAL: A Performance Driven Placement Algorithm. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 39(11):825–840, 1992.

[74] Ken S Stevens, Ran Ginosar, and Shai Rotem. Relative Timing. *Very Large Scale Integration (VLSI) Systems*, 11(1):129–140, 2003.

[75] I. Sutherland and S. Fairbanks. GasP: A Minimal FIFO Control. In *Asynchronous Circuits and Systems (ASYNC), 2001*, pages 46–53, 2001.

[76] Ivan E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, 1989.

[77] R. E. Tarjan. Shortest Paths. *Tech reports*, 1981.

[78] Y. Thonnart, E. Beigne, and P. Vivet. A Pseudo-synchronous Implementation Flow for WCHB QDI Asynchronous Circuits. In *Asynchronous Circuits and Systems (ASYNC), 2012*, pages 73–80, May 2012.

[79] Jan Tijmen Udding. A Formal Model for Defining and Classifying Delay-insensitive Circuits and Systems. *Distributed Computing*, 1(4):197–204, 1986.

[80] Lukas PPP Van Ginneken. Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay. In *Circuits and Systems, 1990., IEEE International Symposium on*, pages 865–868. IEEE, 1990.

[81] Girish Venkataramani and Seth C Goldstein. Leveraging Protocol Knowledge in Slack Matching. In *ICCAD*, pages 724–729, 2006.

[82] Natarajan Viswanathan, Min Pan, and Chris Chong-Nuen Chu. FastPlace: an analytical placer for mixed-mode designs. In *ISPD*, pages 221–223, 2005.

المنارة للاستشارات
www.manaraa.com

[83] Jia Wang, D. Das, and Hai Zhou. Gate Sizing by Lagrangian Relaxation Revisited. *Computer-Aided Design of Integrated Circuits and Systems*, 28(7):1071–1084, July 2009.

[84] Jun Wang, Alfred K. Wong, and Edmund Y. Lam. Standard Cell Layout with Regular Contact Placement. *Semiconductor Manufacturing*, 2004.

[85] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Tim Cheng. In *Electronic Design Automation: Synthesis, Verification, and Test*, pages 635–685. Morgan Kaufmann, 2009.

[86] G. Wu and C. Chu. Detailed Placement Algorithm for VLSI Design with Double-Row Height Standard Cells. *TCAD*, (99):1–1, 2015.

[87] Gang Wu and Chris Chu. Simultaneous Slack Matching, Gate Sizing and Repeater Insertion for Asynchronous Circuits. In *DATE*, 2016.

[88] Gang Wu, Tao Lin, Hsin-Ho Huang, Chris Chu, and Peter A Beerel. Asynchronous Circuit Placement by Lagrangian Relaxation. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, pages 641–646. IEEE Press, 2014.

[89] Gang Wu, Ankur Sharma, and Chris Chu. Gate Sizing and Vth Assignment for Asynchronous Circuits Using Lagrangian Relaxation. In *ASYNC*, 2015.

[90] Alex Yakovlev, Pascal Vivet, and Marc Renaudin. Advances in Asynchronous Logic: From Principles to GALS & NoC, Recent Industry Applications, and Commercial CAD Tools. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1715–1724, March 2013.

[91] Neal E Young, Robert E Tarjant, and James B Orlin. Faster Parametric Shortest Path and Minimum-balance Algorithms. *Networks*, 21(2):205–221, 1991.

المنارة للاستشارات
www.manaraa.com