PRACTICAL TECHNIQUES FOR IMPROVING PERFORMANCE AND EVALUATING SECURITY ON CIRCUIT DESIGNS

A Dissertation

by

WENBIN XU

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Jiang Hu
Committee Members,	Anxiao Jiang
	Peng Li
	Tie Liu
Head of Department,	Miroslav M. Begovic

August 2019

Major Subject: Computer Engineering

ABSTRACT

As the modern semiconductor technology approaches to nanometer era, integrated circuits (ICs) are facing more and more challenges in meeting performance demand and security. With the expansion of markets in mobile and consumer electronics, the increasing demands require much faster delivery of reliable and secure IC products. In order to improve the performance and evaluate the security of emerging circuits, we present three practical techniques on approximate computing, split manufacturing and analog layout automation.

Approximate computing is a promising approach for low-power IC design. Although a few accuracy-configurable adder (ACA) designs have been developed in the past, these designs tend to incur large area overheads as they rely on either redundant computing or complicated carry prediction. We investigate a simple ACA design that contains no redundancy or error detection/correction circuitry and uses very simple carry prediction. The simulation results show that our design dominates the latest previous work on accuracy-delay-power tradeoff while using 39% less area. One variant of this design provides finer-grained and larger tunability than that of the previous works. Moreover, we propose a delay-adaptive self-configuration technique to further improve the accuracy-delay-power tradeoff.

Split manufacturing prevents attacks from an untrusted foundry. The untrusted foundry has front-end-of-line (FEOL) layout and the original circuit netlist and attempts to identify critical components on the layout for Trojan insertion. Although defense methods for this scenario have been developed, the corresponding attack technique is not well explored. Hence, the defense methods are mostly evaluated with the k-security metric without actual attacks. We develop a new attack technique based on structural pattern matching. Experimental comparison with existing attack shows that the new attack technique achieves about the same success rate with much faster speed for cases without the k-security defense, and has a much better success rate at the same runtime for cases with the k-security defense. The results offer an alternative and practical interpretation for k-security in split manufacturing.

Analog layout automation is still far behind its digital counterpart. We develop the layout automation framework for analog/mixed-signal ICs. A hierarchical layout synthesis flow which works in bottom-up manner is presented. To ensure the qualified layouts for better circuit performance, we use the constraint-driven placement and routing methodology which employs the expert knowledge via design constraints. The constraint-driven placement uses simulated annealing process to find the optimal solution. The packing represented by sequence pairs and constraint graphs can simultaneously handle different kinds of placement constraints. The constraint-driven routing consists of two stages, integer linear programming (ILP) based global routing and sequential detailed routing. The experiment results demonstrate that our flow can handle complicated hierarchical designs with multiple design constraints. Furthermore, the placement performance can be further improved by using mixed-size block placement which works on large blocks in priority.

DEDICATION

To my mother, my father, my grandmother, and my grandfather.

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Jiang Hu, for his advice on my PhD program. During the past four years, I've learned a lot from Professor Hu, not only the academic knowledge but also the attitude to work and research. Without his guidance and encouragement, it is almost impossible for me to complete the program.

I would like to thank my committee members, Professor Anxiao Jiang, Professor Peng Li and Professor Tie Liu of Texas A&M University, for their helpful suggestions and comments on my research and dissertation.

I would like to thank Professor Sachin S. Sapatnekar of University of Minnesota for his help and advice on approximate computing project and analog layout automation project. I would like to thank Professor Jeyavijayan (JV) Rajendran of Texas A&M University for his help and advice on split manufacturing project. I would like to thank Professor Duncan M. (Hank) Walker of Texas A&M University for his helpful discussion on approximate computing project.

I would like to thank Lang Feng of Texas A&M University for his help on preparing test cases in split manufacturing project. I would like to thank Yaguang Li of Texas A&M University for his collaboration in analog layout automation project. I would like to thank Arvind Sharma, Kishor Kunal, and Meghna Madhusudan of University of Minnesota for preparing the benchmarks and helpful discussion on analog circuits.

Thanks to my mates in Texas A&M University, Hao He, Lin Huang, Hongxin Kong, Rongjian Liang, and Jiafan Wang. Special thanks to my former managers in Synopsys, Liwen Xu, Yan Zhai and Zhibo Ai who introduced me to the semiconductor industry. Special thanks to Daniel Xu for his support and encouragement during the past four years.

At last, many thanks to my parents for their strong support and endless love to me.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor Jiang Hu, advisor, and Professor Peng Li, Professor Tie Liu of the Department of Electrical and Computer Engineering, and Professor Anxiao Jiang of the Department of Computer Science and Engineering.

Some test cases for Chapter 3 were prepared by Lang Feng of Texas A&M University. Some codes of constraint-driven routing for Chapter 4 were implemented by Yaguang Li of Texas A&M University. The benchmarks for Chapter 4 were provided by Arvind Sharma, Kishor Kunal, and Meghna Madhusudan of University of Minnesota.

All other work conducted for the dissertation was completed by the student independently.

Funding Sources

Graduate study was supported by National Science Foundation (NSF) under Grant CCF-1255193, Grant CCF-1525749, Grant CCF-1525925, Grant CNS-1618824, and Grant CNS-1618797, and Semiconductor Research Corporation (SRC) under Grant 2016-TS-2688, and Grant 2016-TS-2689.

NOMENCLATURE

IC	Integrated Circuits
VLSI	Very Large Scale Integration
DVS	Dynamic Voltage Scaling
CVS	Clustered Voltage Scaling
FEOL	Front-End-Of-Line
BEOL	Back-End-Of-Line
SoC	System-on-Chip
EDA	Electronic Design Automation
DVFS	Dynamic Voltage and Frequency Scaling
DCT	Discrete Cosine Transform
CRA	Carry Ripple Adder
CLA	Carry Look-ahead Adder
VLCSA	Variable Latency Carry Select Adder
ETA	Error Tolerant Adder
CASA	Correlation-Aware Speculative Adder
ACA	Accuracy Configurable Adder
GDA	Accuracy Gracefully-Degrading Adder
RAP-CLA	Reconfigurable Approximate Carry Look-ahead Adder
SARA	Simple Accuracy Reconfigurable Adder
PDP	Power Delay Product
EDP	Energy Delay Product
PSNR	Peak Signal-to-Noise Ratio

LSB	Least Significant Bit
MSB	Most Significant Bit
DAR	Delay Adaptive Reconfiguration
RAC	ROM and Accumulator Component
SAT	Boolean Satisfiability
DRC	Design Rule Check
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Programming
HPWL	Half-parameter Wirelength

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGMENTS v

CONTRIBUTORS AND FUNDING SOURCES vi

NOMENCLATURE vii

TABLE OF CONTENTS ix

LIST OF FIGURES xii

LIST OF TABLES xv

1. INTRODUCTION AND MOTIVATION 1

    1.1 Approximate Computing 1
    1.2 Split Manufacturing for Hardware Security 2
    1.3 Analog Layout Automation 3

2. SIMPLE YET EFFICIENT ACCURACY CONFIGURABLE ADDER 4

    2.1 Introduction 4
    2.2 Prior Works and Rationale of Our Design 6
    2.3 Simple Accuracy Reconfigurable Adder 9
        2.3.1 Preliminaries 9
        2.3.2 SARA: Simple Accuracy Reconfigurable Adder Design 10
        2.3.3 Usage of SARA 12
    2.4 SARA Error Analysis 13
    2.5 Delay-Adaptive Reconfiguration of SARA 19
    2.6 Experimental Results 21
        2.6.1 Experiment Setup and Evaluation 21
        2.6.2 Results of Tradeoff for Different Configurations 22
        2.6.3 Results of Tradeoff for Delay-Adaptive Reconfiguration 26
        2.6.4 Impact of Detection Window in Delay-Adaptive Reconfiguration 28
        2.6.5 Results of Iso-delay Power and Area 31
    2.7 Applications 33
        2.7.1 Extension to Multiplier 33

2.7.2 DCT Computation in Image Processing 35
2.8 Conclusion 40

3. LAYOUT RECOGNITION ATTACKS ON SPLIT MANUFACTURING 41

3.1 Introduction 41
3.2 Preliminary 43
3.2.1 Attack Scenario 43
3.2.2 Related Work 44
3.2.3 SAT-based Bijective Mapping 45
3.3 Layout Recognition Attack by Structural Pattern Matching 48
3.3.1 Pattern Table 48
3.3.2 Matching a Subcircuit with FEOL Layout 50
3.3.3 Pruning by Hints from Design Conventions 52
3.3.4 Propagating Candidates along Subcircuit 53
3.3.5 Matching of the Entire Circuit 55
3.4 Experiment Results 57
3.4.1 Experiment Setup 57
3.4.2 Experiments on Cases without k-security Defense 57
3.4.3 Experiments on Cases with k-security Defense 57
3.5 Discussions 61
3.5.1 Comparison with Other Attack Methods 61
3.5.2 Extension to Attacks with Incomplete Netlist 61
3.6 Conclusion 61

4. HIERARCHICAL CONSTRAINT-DRIVEN ANALOG LAYOUT AUTOMATION 63

4.1 Introduction 63
4.2 Related Work 66
4.2.1 Analog Placement 66
4.2.2 Analog Routing 68
4.3 Constraint-driven Placement 69
4.3.1 Topological Representation 69
4.3.1.1 Sequence Pair 69
4.3.1.2 Constraint Graph 69
4.3.2 Handling of Placement Constraints 71
4.3.2.1 Symmetry Block Constraint 71
4.3.2.2 Alignment Constraint 73
4.3.2.3 Abutment Constraint 74
4.3.2.4 Boundary Constraint 75
4.3.2.5 Proximity Constraint 76
4.3.3 Simulated Annealing Algorithm 76
4.3.3.1 Overall Flow 76
4.3.3.2 Perturbations 77
4.3.3.3 Initial Solution 79
4.3.3.4 Cost Function 79

4.4 Constraint-driven Routing 80
4.4.1 Global Routing 81
4.4.1.1 Basic Flow 81
4.4.1.2 Global Grids 82
4.4.1.3 Multi-pin Net Decomposition 83
4.4.1.4 Route Candidate Generation 84
4.4.1.5 Constraint Annotation 84
4.4.1.6 ILP Optimization 85
4.4.2 Detailed Routing 86
4.4.2.1 Basic Flow 86
4.4.2.2 Handling of Symmetry Net Constraints 87
4.5 Bottom-up Hierarchical Flow 89
4.6 Mixed-size Block Placement 91
4.6.1 Overall Flow 92
4.6.2 Macro Placement 92
4.6.3 Full Cell Placement 94
4.7 Experiment Results 97
4.7.1 Experiment Setup 97
4.7.2 Experiment Results of Hierarchical Flow 98
4.7.3 Experiment Results of Mixed-size Block Placement 101
4.8 Conclusion 101

5. SUMMARY AND CONCLUSIONS 104

REFERENCES 105

LIST OF FIGURES

FIGURE Page

2.1 Error-correction-based configurable adder. 7

2.2 Carry-prediction-based configurable adder. 7

2.3 (a) Conventional full adder; (b) Our carry-out selectable full adder; (c) Our carry-in configurable full adder. 9

2.4 Design of SARA. 11

2.5 Implementation of 12-bit adder in (a) CRA and (b) SARA. 12

2.6 Average error of 9-bit SARA in different configuration. 18

2.7 Design of DAR for SARA operating in a) approximate mode and b) accurate mode.. 20

2.8 SARA: PSNR versus power-delay product. 23

2.9 SARA: Average error versus power-delay product. 24

2.10 SARA: The worst case error versus power-delay product. 24

2.11 SARA: PSNR versus energy-delay product. 25

2.12 SARA: Average error versus energy-delay product. 26

2.13 SARA: The worst case error versus energy-delay product. 26

2.14 SARA-DAR: Error rate versus power-delay product. 27

2.15 SARA-DAR: PSNR versus power-delay product. 28

2.16 SARA-DAR: Error rate versus energy-delay product. 29

2.17 SARA-DAR: PSNR versus energy-delay product. 29

2.18 Error rate of SARA4-DAR with different detection window. 30

2.19 PSNR of SARA4-DAR with different detection window. 30

2.20 Iso-delay power comparison. The numbers are PSNR. 31

2.21 Area comparison. 32

2.22 Basic structure of multiplier. 34

2.23 Multiplier: PSNR versus power-delay product. 35

2.24 Multiplier: The worst case error versus power-delay product. 35

2.25 Multiplier: Average error versus energy-delay product. 36

2.26 Multiplier: Error rate versus energy-delay product. 36

2.27 Comparison of image lenna: (a) accurate adder; (b) SARA4; (c) SARA8; (d) SARA4-DAR2; (e) GDA; (f) RAP-CLA. 37

2.28 Comparison of image cameraman: (a) accurate adder; (b) SARA4; (c) SARA8; (d) SARA4-DAR2; (e) GDA; (f) RAP-CLA. 37

2.29 Comparison of image kiel: (a) accurate adder; (b) SARA4; (c) SARA8; (d) SARA4-DAR2; (e) GDA; (f) RAP-CLA. 37

2.30 Comparison of image house: (a) accurate adder; (b) SARA4; (c) SARA8; (d) SARA4-DAR2; (e) GDA; (f) RAP-CLA. 38

2.31 2 dimensional descrete cosine transform. 38

3.1 Graph representations of (a) circuit netlist and (b) its FEOL layout. Each vertex indicates a logic gate, whose logic type is represented by grayscale. 45

3.2 Example of the pattern table: (a) circuit netlist and (b) its pattern tables associated with input pins of logic gates and indices. 49

3.3 Matching cells with FEOL layout by using pattern tables (solid line: FEOL connection, dashed line: BEOL connection). 52

3.4 Example of propagating candidates along subcircuit: (a) the subcircuit from layout and (b) the pattern table from netlist. 55

3.5 Experiment results of cases without k-security defense. 59

3.6 Experiment results of cases with 2-security defense. 59

3.7 Experiment results of cases with 3-security defense. 60

3.8 Experiment results of cases with 4-security defense. 60

4.1 Example of constraint graph with symmetry constraint. 73

4.2 Overview of simulated annealing algorithm. 77

4.3 Flow of global route. 81
4.4 Model of metal templates. 83
4.5 Scheme of global grids in (a) full view and (b) detailed view. 84
4.6 Flow of detailed routing. 86
4.7 Example of detailed routing on a three-pin net. 88
4.8 Example of symmetry nets (a) before and (b) after detailed routing. 88
4.9 Infrastructure in hierarchical process and the data flow. 89
4.10 Scheme of the bottom-up hierarchical flow. 90
4.11 Scheme of the mixed-size block placement. 93
4.12 Example of design reduction: (a) original design v.s. (b) reduced design. 94
4.13 Layouts of design opamp. 100
4.14 Layouts of design SCF. 100
4.15 Mixed-size placement result after macro placement. 102
4.16 Mixed-size placement result after analytical placement. 102
4.17 Mixed-size placement result after legalization. 103
4.18 Simulated annealing based placement result. 103

LIST OF TABLES

TABLE Page

2.1 Comparison of characteristics for different techniques. 8

2.2 Definition of parameters for error analysis 13

2.3 Error rate of sub-adder with different width 15

2.4 Image Quality Comparison in PSNR 39

3.1 Comparison of different attack methods. 41

3.2 Comparison of different defense methods. 42

3.3 Experiment results on cases without k-security defense (“MR" indicates the ratio of correctly matched cell). 58

3.4 Experiment results on cases with k-security defense ("MR" indicates the ratio of correctly matched cell). Number in bold indicates higher ratio than the expected success rate that k-security can guarantee. 62

4.1 Benchmark circuits for experiments. 98

4.2 Results of hierarchical flow on benchmarks. 99

4.3 Comparison of performance between different placement on design trackhold. 101

1. INTRODUCTION AND MOTIVATION

As the modern semiconductor technology approaches to nanometer era, integrated circuits (ICs) are facing more and more challenges in meeting performance demand and security. With the expansion of markets in mobile and consumer electronics, the increasing demands require much faster delivery of reliable and secure IC products. The conflict between circuit performance, design effort/period and intellectual property security becomes more prominent than ever before. Our research aims to examine the performance and security of circuit designs in three aspects, approximate computing in circuit level, split manufacturing in design level and analog layout automation in flow level. In order to improve the performance and evaluate the security of emerging circuits, we are proposing three practical techniques covering three sub-topics, simple accuracy configurable adders for approximate computing circuits, layout recognition attacks for split-manufactured hardware and hierarchical constraint-driven method for analog layout automation.

1.1 Approximate Computing

As the VLSI technology advances to nanometer process, power consumption has become a well-known challenge to further improve the circuit performance. Low power techniques for the conventional exact computing paradigm, such as dynamic voltage scaling (DVS) and clustered voltage scaling (CVS), have been already extensively studied. A comparatively new direction is approximate computing, where errors are intentionally allowed in exchange for power reduction. In many applications, such as audio, video, haptic processing and machine learning, occasional small errors are indeed acceptable. Such error-tolerant applications are found in abundance in emerging applications and technologies.

Approximate computing covers a wide range of research activities across different layers of computing system, from programming languages [1] to transistor-level circuits [2]. Unlike stochastic computing, approximate computing introduces deterministic designs/systems which produce imprecise results [3]. The statistical properties of errors in data or algorithms can be utilized to

tradeoff computation accuracy with performance and energy. As one of the essential basic modules of arithmetic and logic systems, multiple-bit adders have been recently studied in the context of approximate computing[4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. The main idea is based on the fact that critical delay of adder is determined by the length of carry propagation. The reduction in the path of carry propagation provides the opportunity for power reduction by voltage scaling or performance improvement with higher operating frequency.

1.2 Split Manufacturing for Hardware Security

Split manufacturing is a security technique against untrusted foundries. By having only front-end-of-line (FEOL) layers manufactured at an untrusted foundry while the back-end-of-line (BEOL) is fabricated at a trusted foundry, attackers in the untrusted foundry do not have complete information to perform attacks such as Trojan insertion, piracy, and overproduction [14, 15, 16]. The same principle can be applied to 3D ICs, where different dies are manufactured by different foundries, since each 3D IC contains two or more independently manufactured ICs which are vertically stacked on top of each other [17]. Despite the security enhancement, split manufacturing still has a significant risk of being successfully attacked [18, 19, 20].

There are two attack scenarios in split manufacturing. (i) The attacker at FEOL foundry does not have circuit netlist and attempts to reverse engineering the entire design for stealing intellectual property, and conducting piracy and overproduction; (ii) The attacker has circuit netlist and tries to recognize critical components on the layout for inserting Trojans. The investigation on scenario (ii) is mostly restricted to only defense techniques. For example, a previous work [17] proposed a method using a metric called k-security to protect the layout by obfuscation. The metric k-security means that for every group of components of which the connections are visible in FEOL, it will have at least another $k-1$ groups (k is an integer larger than 1) which are identical to it logically. They also proposed a SAT-based algorithm [17] for selecting wires to be manufactured in BEOL to achieve the k-security. Moreover, another work [21] improves the algorithm in [17] and realizes a shorter runtime for achieving k-security. However, the lack of actual attack still limits the ability of evaluating the security of split-manufactured circuits.

1.3 Analog Layout Automation

Analog circuits are playing a more and more important role in modern system-on-chip (SoC) applications. At the same time, the demands of analog electronic design automation (EDA) have dramatically increased. However, the physical implementation of analog designs has not been automated to the same degree as digital designs. Analog layout synthesis still remains a manual and time-consuming task due to a large amount of expert knowledge involved, such as complex constraints that are specified manually and satisfied through manual layout.

In the past decade, constraint-driven design approach has been recognized as one potential evolution in analog design automation. The performance specifications can be mapped onto geometric analog-specific constraints relevant to the physical parasitics in layouts. Then, each constraint is enforced during the physical implementation, which guarantees the satisfaction of the original specifications. The quality of an analog design is mostly determined by the degree to which complicated constraints can be satisfied and pre-defined design objectives achieved.

Analog placement is one of the most important step in analog layout synthesis. It determines the physical location of each device or device group, which also limits the degree of design freedom in routing. The problem of placing analog devices with different symmetry and matching constraints has been extensively studied [22, 23, 24, 25, 26]. Moreover, the symmetry constraint, common centroid constraint and other general placement constraints can be simultaneously handled in [27]. Although these geometric constraints can help reduce the errors induced by parasitic mismatch, there is still no guarantee for the performance of the resulting placements. Another problem is that it is very difficult to estimate the circuit performance without actual simulation in placement stage. Besides placement part, analog routing also has great impact on the circuit performance. Although a plenty of work has tried to improve the routing method [28, 29, 30, 31, 32], the challenges in handling multiple constraints and complicated designs are still left unresolved.

2. SIMPLE YET EFFICIENT ACCURACY CONFIGURABLE ADDER*

2.1 Introduction

Power constraints are a well-known challenge in advanced VLSI technologies. Low power techniques for the conventional exact computing paradigm have already been extensively studied. A comparatively new direction is approximate computing, where errors are intentionally allowed in exchange for power reduction. In many applications, such as audio, video, haptic processing and machine learning, occasional small errors are indeed acceptable. Such error-tolerant applications are found in abundance in emerging applications and technologies.

A great deal of approximate computing research has been concentrated on arithmetic circuits, which are essential building blocks for most of computing hardware. In particular, several approximate adder designs have been developed [2, 3, 5, 4, 6, 7, 8, 9, 10, 11, 33, 34, 35, 36]. One such design [2] achieves 60% power reduction for DCT (Discrete Cosine Transform) computation without making any discernible difference to the images being processed. In realistic practice, accuracy requirements may vary for different applications. In mobile computing devices, different power modes may entail different accuracy constraints even for the same application. Specifically, arithmetic accuracy can be adjusted at runtime using methods such as dynamic voltage and frequency scaling (DVFS) to obtain the best accuracy-power tradeoff. The benefit of runtime accuracy adjustment is demonstrated in [33], but their approximation is realized by voltage over-scaling, where errors mostly occur at the timing-critical path associated with the most significant bits, i.e., errors are often large.

To reduce the overall error, a few approximate designs have been developed by intentionally allowing errors in lower bits with shorter carry chain in addition operation. In [34], a design that considers only the previous k inputs instead of all input bits can approximate the result with the benefit in half of the logarithmic delay. Reliable variable latency carry select adder (VLCSA)

*

shows a speculation technique which introduces carry chain truncation and carry select addition as a basis [4]. A series of Error Tolerant Adders (ETAI, ETAII, ETAIIM), which truncate the carry propagation chain by dividing the adder into several segments, have been proposed [6, 7, 8]. Correlation-aware speculative adder (CASA) in [9] relies on the correlation between MSBs of input data and carry-in values. Another approximate adder that exploits the generate signals for carry speculation is presented [10]. These designs focus on static approximation which pursues almost correct results at the required accuracy. However, in some applications such as image processing or audio/video compression, the required accuracy might vary during runtime. To meet the need of runtime accuracy adjustment, a series of designs are developed to implement accuracy configurable approximation which could be reconfigured online to save more power.

A few accuracy configurable adder designs that use approximation schemes other than voltage over-scaling have been proposed. An early work [12], called ACA, starts with an approximate adder and augments it with an error detection and correction circuit, which can be configured to deliver varying approximation levels or accurate computing. Its baseline approximate adder contains significant redundancy and the error detection/correction circuit further increases area overhead. The ACA design [12] is generalized to a flexible framework GeAr in [37]. In both ACA and GeAr, the error correction must start from the least significant bits and hence accuracy improves slowly in the progression of configurations. The work of Accurus [38] modifies ACA/GeAr to overcome this drawback and achieves graceful degradation. However, in ACA, GeAr as well as Accurus, the error correction circuit is pipelined, implying that the computation in accurate mode takes multiple clock cycles and causes data stalls.

An alternative direction of accuracy configurable adder design is represented by GDA [13] and RAP-CLA [39]. These methods start with an accurate adder and use carry prediction for optional approximation. As such, they no longer need error detection/correction and do not incur any data stall. In addition, they intrinsically support graceful degradation. The GDA design [13] is composed by accurate CRA (Carry Ripple Adder) and extra configurable carry prediction circuitry, similar as the carry look-ahead part of CLA (Carry Look-ahead Adder). Thus, its area is generally

quite large. RAP-CLA [39] is based on accurate CLA design and reuses a portion of the carry look-ahead circuit as carry prediction. This leads to an overall area that is less than GDA but greater than CLA. In [39], the carry-prediction-based approach is shown to be superior to error-correction-based design [37].

We propose a new carry-prediction-based accuracy configurable adder design: **SARA** (Simple Accuracy Reconfigurable Adder). It is a simple design with significantly less area than CLA, which, to the best of our knowledge, has not been achieved in the past in accuracy configurable adders. SARA inherits the advantages of all previous carry-prediction-based approaches: no error correction overhead, no data stall and allowing graceful degradation. Compared to GDA [13], SARA incurs 50% less PDP (Power Delay Product) and can reach the same PSNR (Peak Signal-to-Noise Ratio). Moreover, SARA demonstrates remarkably better accuracy-power-delay tradeoff than the latest, and arguably the best, previous work RAP-CLA [39]. A delay-adaptive reconfiguration technique is developed to further improve the accuracy-power-delay tradeoff. The proposed designs are also validated by multiplication and DCT computation in image processing.

2.2 Prior Works and Rationale of Our Design

We review a few representative works on accuracy configurable adder design and show the relation with our method. These designs can be generally categorized into two groups: error-correction-based configurations [12, 37, 38] and carry-prediction-based configurations [13, 39].

The main idea of an error-correction-based approach [12, 37, 38] is shown in Figure 2.1. The scheme starts with an approximate adder (the dashed box), where the carry chain is shortened by using separated sub-adders with truncated carry-in. In order to reduce the truncation error, the bit-width in some sub-adders contains redundancy. For example, *sub-adder2* calculates the sum for only bit 8 and 9, but it is an 8-bit adder using bit $[9:2]$ of the addends, 6 bits of which are redundant. Even with the redundancy, there is still residual error which is detected and corrected by additional circuits. In Figure 2.1, the errors of *sub-adder2* must be corrected by *error-correction2* before the errors of *sub-adder3* are rectified by *error-correction3*. As such, the configuration progression always starts with small accuracy improvements. The redundancy and error detection/correction

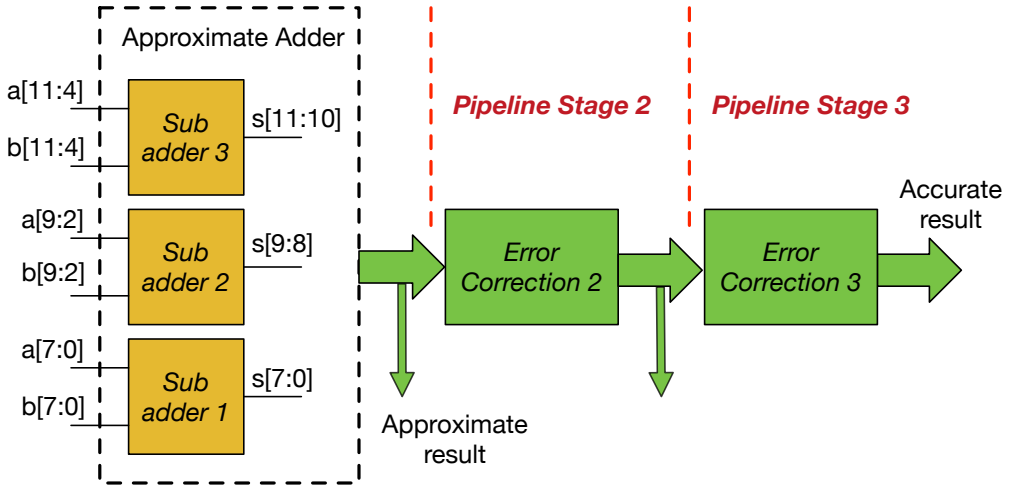


Figure 2.1: Error-correction-based configurable adder.

incur large area overhead. Since the error correction circuits are usually pipelined, an accurate computation may take multiple clock cycles and could stall entire datapath, depending on the addend values.

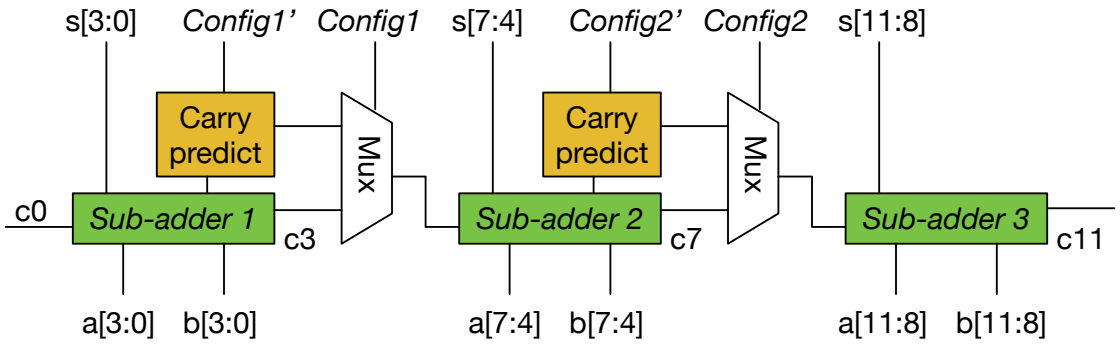


Figure 2.2: Carry-prediction-based configurable adder.

The framework of carry-prediction-based methods [13, 39] is shown in Figure 2.2. These schemes start with an accurate adder design, which is formed by chaining a set of sub-adders.

Each sub-adder comes with a fast but approximated carry prediction circuit. By selecting between the carry-out from sub-adder or carry prediction, the overall accuracy can be configured to different levels. Such an approach does not need error detection/correction circuitry. Moreover, the configuration of higher bits is independent of lower bits. This leads to fast convergence or graceful degradation in the progression of configurations. In GDA [13], the sub-adders are CRA designs while the carry-prediction circuit is similar to the carry look-ahead part of CLA. Further, its carry prediction can be configured to different accuracy levels. However, the complicated carry prediction induces large area overhead. The RAP-CLA scheme [39] uses CLA for its baseline where the carry-ahead of each bit is computed directly from the addends of all of its lower bits. Its carry prediction reuses a part of the look-ahead circuit rather than building extra dedicated prediction circuitry, and hence is more area-efficient than GDA. But its baseline is much more expensive than GDA.

Table 2.1: Comparison of characteristics for different techniques.

Method	Baseline sub-adder	Error correction	Graceful degradation	Carry prediction
ACA [12]	Redundant CRA	Yes	No	No
GeAr [37]	Redundant CRA	Yes	No	No
Accurus [38]	Redundant CRA	Yes	Yes	No
GDA [13]	CRA	No	Yes	Stand-alone
RAP-CLA [39]	CLA	No	Yes	Reuse
SARA (ours)	CRA	No	Yes	Reuse

Our design is a carry-prediction-based approach. Its sub-adders are CRA instead of expensive CLA as in RAP-CLA. Its carry prediction also reuses part of the sub-adders rather than having dedicated prediction circuitry. As such, it avoids the disadvantages of both GDA and RAP-CLA. A comparison among the characteristics of these different techniques is provided in Table 2.1.

2.3 Simple Accuracy Reconfigurable Adder

2.3.1 Preliminaries

An N-bit adder operates on two addends $A = (a_N, ..., a_i, ..., a_1)$ and $B = (b_N, ..., b_i, ..., b_1)$. For bit i, its carry-in is c_{i-1} and its carry-out is c_i. Defining the carry generate bit $g_i = a_i \cdot b_i$, propagate bit $p_i = a_i \oplus b_i$ and kill bit $k_i = \bar{a}_i \cdot \bar{b}_i$, the conventional full adder computes the sum s_i and carry c_i according to

$$s_i = p_i \oplus c_{i-1}, \tag{2.1}$$

$$c_i = g_i + p_i \cdot c_{i-1}. \tag{2.2}$$

A gate level schematic of conventional full adder is provided in Figure 2.3(a). A CRA is used to chain N bits of conventional full adders together.

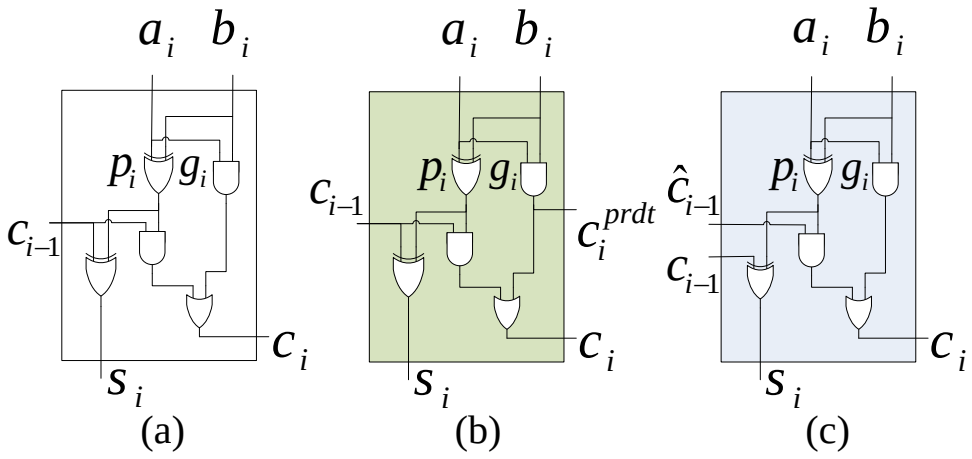


Figure 2.3: (a) Conventional full adder; (b) Our carry-out selectable full adder; (c) Our carry-in configurable full adder.

By applying Equation (2.2) recursively, one can get

$$c_i = g_i + p_i g_{i-1} + \ldots + g_1 \prod_{k=2}^{i} p_k + c_0 \prod_{k=1}^{i} p_k. \tag{2.3}$$

This equation implies that c_i can be computed directly from g and p of all bits, without waiting for the c of its lower bits to be computed. This observation is the basis for CLA adder.

2.3.2 SARA: Simple Accuracy Reconfigurable Adder Design

In SARA, an N-bit adder is composed by K segments of L-bit sub-adders, where $K = \lceil N/L \rceil$ (see Figure 2.2). Each sub-adder is almost the same as CRA except that the MSB (Most Significant Bit) of a sub-adder, which is bit i, provides a carry prediction as

$$c_i^{prdt} = g_i \tag{2.4}$$

For the LSB (Least Significant Bit) of the higher-bit sub-adder, which is bit $i+1$, its carry-out c_{i+1} can be computed using one of two options: either by the conventional $c_{i+1} = g_{i+1} + p_{i+1} \cdot c_i$, or by using the carry prediction as

$$c_{i+1} = g_{i+1} + p_{i+1} \cdot c_i^{prdt} = g_{i+1} + p_{i+1} \cdot g_i \tag{2.5}$$

The selection between the two options is realized using MUXes as in Figure 2.4 and the MUX selection result is denoted as $\hat{c}_i$. Comparing Equation (2.5) with (2.3), we can see that the carry prediction is a truncation-based approximation to carry computation*. Therefore, $\hat{c}_i$ can be configured to either accurate mode or approximation mode, i.e.,

$$\hat{c}_i \leftarrow \begin{cases} c_i^{prdt}, & \text{if approximation mode} \\ c_i, & \text{if accurate mode.} \end{cases} \tag{2.6}$$

It should be noted that the carry prediction c_i^{prdt} reuses g_i in an existing full adder instead of introducing an additional dedicated circuit as in [13] or Figure 2.2. This prediction scheme makes a very simple modification to the conventional full adder, as shown in Figure 2.3(b).

One can connect $\hat{c}_i$ to its higher bit $i+1$ to compute both carry c_{i+1} and sum s_{i+1} , as in

*A similar approximation is used in static approximate adder design [10].

GDA [13] and RAP-CLA [39]. We suggest an improvement over this approach by another simple change as in Figure 2.3(c), where s_{i+1} is based on c_i instead of $\hat{c}_i$. Such approach can help reduce the error rate in outputs when an incorrect carry is propagated. Because the sum keeps accurate and the carry will not be propagated when addends are exactly the same. Moreover, out of all four configurations of sum/carry calculation by approximate/accurate carry-in, the most meaningful way is to have sum bit calculated by accurate carry and make carry bit configurable. So sum s_{i+1} is calculated directly by accurate carry c_i without the option of c_i^{prdt}. Applying this in SARA as in Figure 2.4, in the approximation mode, computing s_{j+1} from c_j can still limit the critical path to be between c_{i-1}^{prdt} and s_{j+1}, but has higher accuracy than computing s_{j+1} from $\hat{c}_j$. Compared to sum computation in GDA and RAP-CLA, this technique improves accuracy with almost no additional overhead. Compared to CRA, the overhead of SARA is merely the MUXes, which is almost the minimum possible for configurable adders.

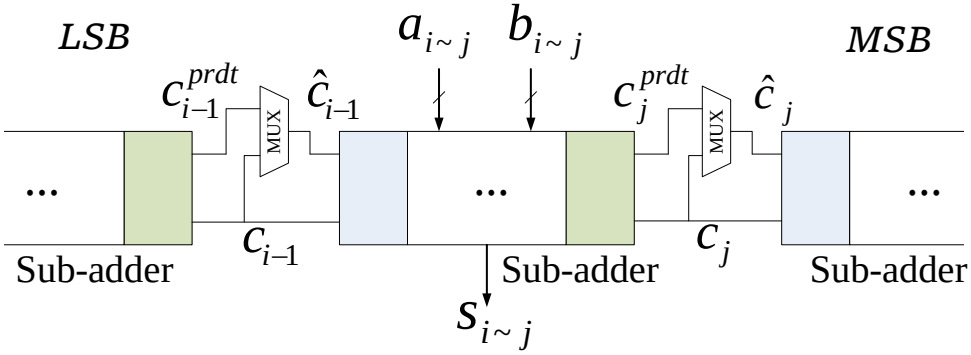


Figure 2.4: Design of SARA.

Although s_{j+1} is calculated by accurate carry c_j, its delay can still be reduced by approximate carry in lower sub-adder. In a multi-bit adder, the delay of sum bit depends on the carry chain propagated from its lower bits. In our SARA structure, even when accurate carry c_j is propagated at bit j, the carry chain might be truncated by approximate carry in other lower bits. In Figure 2.4, when c_{i-1}^{prdt} is propagated, the delay of s_{j+1} is reduced as its path is shorten to be between bit $i-1$

and $j+1$. We can take the 12-bit adder in Figure 2.5 as an example. For 12-bit SARA working in approximate mode, the sum s_9 uses the accurate carry c_8 from a lower sub-adder (bit 5 to 8). But c_8 is propagated from approximate carry c_4^{prdt} of another sub-adder (bit 1 to 4). As shown in the figure, the delay of s_9 in SARA is about 6 stages. Compared with the same bit in CRA, the delay of sum bit s_9 in SARA is reduced by 3 stages. Similar delay reduction can be observed in other sum bits (bit 6 to 12). For sums at bit 1 to 5, their delay is the same as CRA because they are using an accurate carry c_0 from LSB. As a result, the maximum delay in 12-bit SARA is reduced, since for a multi-bit adder its maximum delay depends on the longest critical path.

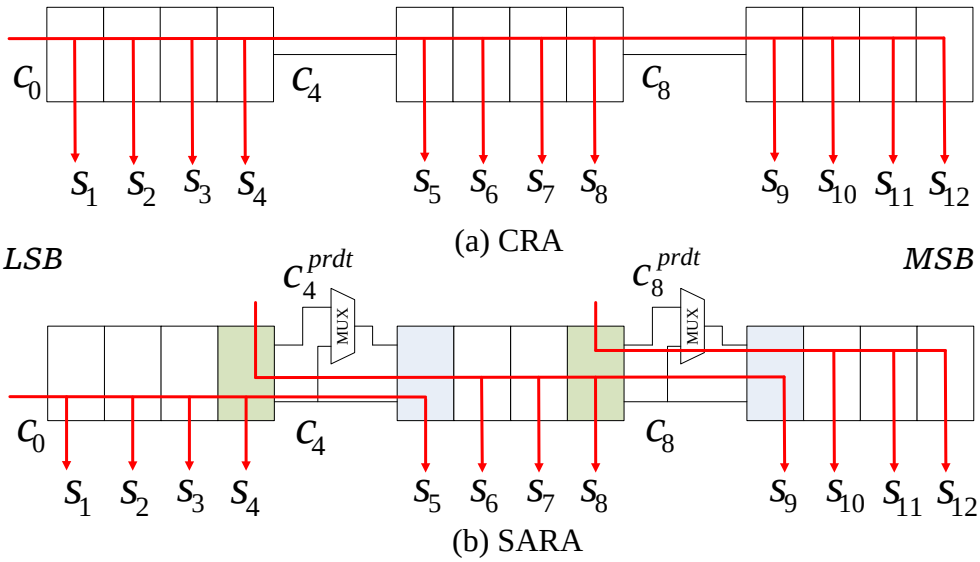


Figure 2.5: Implementation of 12-bit adder in (a) CRA and (b) SARA.

2.3.3 Usage of SARA

When $\hat{c}_i$ is configured to be c_i for all K sub-adders, SARA operates very much like the CRA, where the critical path is along N-bit full adders. If all $\hat{c}_i$ are selected to be c_i^{prdt}, the critical path is shortened to roughly L-bit full adders. This large delay reduction can be translated to power reduction by supply voltage scaling. Voltage scaling (reducing supply voltage) on digital circuits will lead to increase in delay. So we can reduce the supply voltage on SARA to make its critical

delay same as that of CRA under normal voltage. As the supply voltage decreases, the power consumption could be reduced. There can be 2^{K-1} different configurations. For two configurations with the same critical path length, obviously we only need the one with higher accuracy. Therefore, there are K effective configurations, with critical path length of L-bit, $2L$-bit, ..., $K \cdot L \simeq N$-bit full adders. The delay of such configurable design varies according to configured accuracy, which results in different power reduction by voltage scaling.

2.4 SARA Error Analysis

In this section, we give a theoretical analysis on the expected error of our SARA design and validate the results by numerical experiments. To make it easier for readers to follow the analysis, we list the parameters used in this section as Table 2.2.

Table 2.2: Definition of parameters for error analysis

Parameter	Definition
p_i	propagate bit at bit i
g_i	generate bit at bit i
k_i	kill bit at bit i
c_i	accurate carry-out bit at bit i
c_i^{prdt}	approximate carry-out bit at bit i
$\hat{c}_i$	carry-in bit at bit $i+1$
ER_i^{prdt}	error rate of c_i^{prdt}
$\widehat{ER_i}$	error rate of $\hat{c}_i$

For any bit i in carry-out selectable full adder as in Figure 2.3(b), an error in approximate carry-out occurs when $c_i^{prdt} \neq c_i$. There is only one situation where this error may happen: when $c_{i-1} = 1, p_i = 1, c_i^{prdt} = 0$ and $c_i = 1$. Then the error rate, or probability of such error, is given by

$$\begin{aligned} ER_i^{prdt} &= P(c_i^{prdt} \neq c_i) = P(c_i^{prdt} = 0, c_i = 1) \\ &= P(c_{i-1} = 1, p_i = 1) \\ &= P(c_{i-1} = 1)P(p_i = 1) \end{aligned} \tag{2.7}$$

where P indicates probability and the last part assumes that c_{i-1} and p_i are independent of each other. Then, if the approximate/accurate carry-out can be selected by a MUX gate, the error rate of MUX output $\hat{c}_i$ is

$$\widehat{ER}_i = P(\hat{c}_i \neq c_i) = \begin{cases} ER_i^{prdt}, & \text{if } \hat{c}_i \leftarrow c_i^{prdt} \\ 0, & \text{if } \hat{c}_i \leftarrow c_i. \end{cases} \tag{2.8}$$

Let's consider a configuration of SARA in Figure 2.4, which has both bit j and bit $i-1$ in approximate mode. For the sub-adder which calculates addends from bit i to bit j, its LSB (bit i) is using carry-in configurable full adder, while its MSB (bit j) is in carry-out selectable full adder. According to Equation (2.7) and (2.8), the error rate of $\hat{c}_j$ is determined by the probabilities of $c_{j-1} = 1$ and $p_j = 1$.

$$\widehat{ER}_j = P(c_{j-1} = 1)P(p_j = 1) \tag{2.9}$$

According to the logic of addition, the carry-out bit is calculated by the carry-in and addends. There are two cases which can result in $c_{j-1} = 1$: generate bit g_{j-1} should be 1 in case of carry-in $c_{j-2} = 0$; or kill bit k_{j-1} must be 0 when carry-in comes with $c_{j-2} = 1$. Then, the probability of $c_{j-1} = 1$ can be computed by the probability of $c_{j-2} = 1$ as

$$\begin{aligned} P(c_{j-1} = 1) &= P(c_{j-2} = 0, g_{j-1} = 1) + P(c_{j-2} = 1, k_{j-1} = 0) \\ &= P(c_{j-2} = 0)P(g_{j-1} = 1) + P(c_{j-2} = 1)P(k_{j-1} = 0) \\ &= [1 - P(c_{j-2} = 1)]P(g_{j-1} = 1) + P(c_{j-2} = 1)P(k_{j-1} = 0). \end{aligned} \tag{2.10}$$

Similarly, the probability of $c_{j-2} = 1$, $c_{j-3} = 1$, ..., $c_{i+1} = 1$ can be calculated using the same formula. For the probability of $c_i = 1$, it's a little different because the carry-out c_i in our carry-in configurable full adder is based on predicted carry-in $\hat{c}_{i-1}$ instead of c_{i-1}. Considering that bit $i-1$ is configured in approximate mode, we have

$$P(\hat{c}_{i-1} = 1) = P(c_{i-1}^{prdt} = 1) = P(g_{i-1} = 1). \tag{2.11}$$

Table 2.3: Error rate of sub-adder with different width

Sub-adder length L	Calculated error rate	Simulated error rate
1	1/8 = 0.125	0.1257
2	3/16 = 0.1875	0.1879
3	7/32 = 0.21875	0.2187
4	15/64 = 0.234375	0.2347
5	31/128 = 0.2421875	0.2424
6	63/256 = 0.24609375	0.2464

Then, the probability of $c_i = 1$ can be expressed as

$$P(c_i = 1) =[1 - P(g_{i-1} = 1)]P(g_i = 1) + P(g_{i-1} = 1)P(k_i = 0). \tag{2.12}$$

By expanding Equation (2.10) recursively till bit i, the probability of $c_{j-1} = 1$ can be calculated by a function of generate bit and kill bit from bit $i - 1$ to bit $j - 1$.

$$P(c_{j-1} = 1) = f\{P(g_{i-1} = 1), \ldots, P(g_{j-1} = 1), P(k_i = 0), \ldots, P(k_{j-1} = 0)\}. \tag{2.13}$$

Assuming that the inputs for adder are uniformly distributed random numbers, we have $P(g = 1) = 1/4, P(k = 0) = 3/4$. As the length of sub-adders varies from 1 to 6, the error rates of $\hat{c}_j$ calculated by Equation (2.9) are listed in the second column of Table 2.3. Corresponding data from numerical simulation in Matlab are also presented in the last column. The error rates calculated by our method match well with experiment results, which demonstrates the correctness of our mathematical analysis. We can also observe that as the length of sub-adder increases the error rate is bounded by 0.25. That is because when the length of sub-adder comes to infinite the probability of $c = 1$ will become 0.5 as the normal carry in accurate adder.

Theorem 1. *If $\mathcal{I}$ is the set of bits with MUX at output, the expected error of SARA for unsigned integers is*

$$\sum_{\forall i \in \mathcal{I}} \widehat{ER}_i \cdot P(p_{i+1} = 1) \cdot 2^{i+1}.$$

Proof. The overall expected error of SARA can be calculated by summing respective error intro-

duced by every approximate bit from LSBs to MSBs. But the propagation of inaccurate carry bit may cause error in higher bit which also be counted in the calculation of lower bit. So we need to exclude those errors to avoid over-calculation in the total error.

Let's consider the SARA design in Figure 4 which have approximate configuration at both bit $i-1$ and bit j. Assuming that bit $i-1$ is the lowest bit configured in approximate mode, we know that all sum bits s_k ($k \in [1, i-1]$) as well as carry bit c_{i-1} are accurate.

$$c_{i-1} = c_{i-1}^{acc} \tag{2.14}$$

Then the probability that carry prediction at MUX output $\hat{c}_{i-1}$ mismatches with accurate carry c_{i-1}^{acc} should be the same as the error rate of MUX ouput $\hat{c}_{i-1}$.

$$P(\hat{c}_{i-1} \neq c_{i-1}^{acc}) = P(\hat{c}_{i-1} \neq c_{i-1}) = \widehat{ER}_{i-1} \tag{2.15}$$

According to the structure of carry-in configurable full adder (Figure 3(c)), sum bit s_i calculated from c_{i-1} is always accurate; however, the carry-out bit c_i becomes conditionally accurate which depends on both carry-in bit and propagate bit. As shown in Equation (2.16), the scenario of accurate carry-out can be attributed to two conditions: when the carry-in is not accurate, the carry-out bit becomes accurate as the propagate bit is false; otherwise, it must be accurate no matter what kind of addends are given.

$$P(c_i = c_i^{acc}) = P(\hat{c}_{i-1} = c_{i-1}^{acc}) + P(\hat{c}_{i-1} \neq c_{i-1}^{acc})P(p_i = 0) \tag{2.16}$$

Its complementary part, the probability of inaccurate carry c_i, can be expressed as

$$\begin{aligned} P(c_i \neq c_i^{acc}) &= P(\hat{c}_{i-1} \neq c_{i-1}^{acc})P(p_i = 1) \\ &= P(\hat{c}_{i-1} \neq c_{i-1})P(p_i = 1) \\ &= \widehat{ER}_{i-1} \cdot P(p_i = 1). \end{aligned} \tag{2.17}$$

As a result, the approximation at bit $i-1$ would cause an inaccurate carry-in c_i at bit $i+1$, which introduces the magnitude of 2^i to the overall error in final result. Then the expected error introduced by approximation at bit $i-1$ can be estimated by

$$E[e_{i-1}] = P(c_i \neq c_i^{acc}) \cdot 2^i = \widehat{ER}_{i-1} \cdot P(p_i = 1) \cdot 2^i. \tag{2.18}$$

Next, we consider the expected error introduced by approximation at bit j. As bit j is not the lowest bit in approximate mode, there is a chance that the propagation of inaccurate carry from bit $i-1$ induces error at bit j while it has be taken into account in the error calculation of bit $i-1$. Then the problem is whether the carry c_j is accurate when there is a mismatch between $\hat{c}_j$ and c_j. If not, we need to exclude the impact from lower bit when estimating the error at bit j. Let's answer this question in the following cases.

- Case 1: If any propagate bit in sub-adder (bit i to j) equals 0, the error propagation by inaccurate carry will be paused. In another word, the error carried by inaccurate carry bit cannot be propagated to higher bit any more, because the carry-out is independent of carry-in when propagate bit is false. In this case, the carry c_j should be always accurate regardless of the configuration at bit j.

- Case 2: If all propagate bits of sub-adder equal 1, the value of inaccurate carry $\hat{c}_{i-1}$ (0 instead of 1) will be propagated to c_j. In this situation, the actual value of c_j propagated from bit $i-1$ must be 0, while the accurate value should be 1. Assuming that $\hat{c}_j$ mismatches with c_j, we can state that the value of $\hat{c}_j$ must be 1. However, it conflicts with the generation of c_j, because carry c_j is the logical conjunction of $\hat{c}_j$ and $p_j \cdot c_{j-1}$. So there should be no mismatch between $\hat{c}_j$ and c_j in this case.

In conclusion, when there is a mismatch between $\hat{c}_j$ and c_j, the value of carry c_j must be accurate. We can further conclude that the contributions of every approximate bit to the total error

are independent to each other. Similar to bit $i-1$, the expected error at bit j can be estimated by

$$E[e_j] = \widehat{ER}_j \cdot P(p_{j+1} = 1) \cdot 2^{j+1}. \tag{2.19}$$

Thus, the total error can be obtained by summing up the errors respectively introduced by every approximate bit.

$$E = \sum_{\forall i \in \mathcal{I}} E[e_i] = \sum_{\forall i \in \mathcal{I}} \widehat{ER}_i \cdot P(p_{i+1} = 1) \cdot 2^{i+1} \tag{2.20}$$

□

If input addends are random variables following uniform distribution, the expected error of SARA is given by

$$E = \sum_{\forall i \in \mathcal{I}} \widehat{ER}_i \cdot 2^i. \tag{2.21}$$

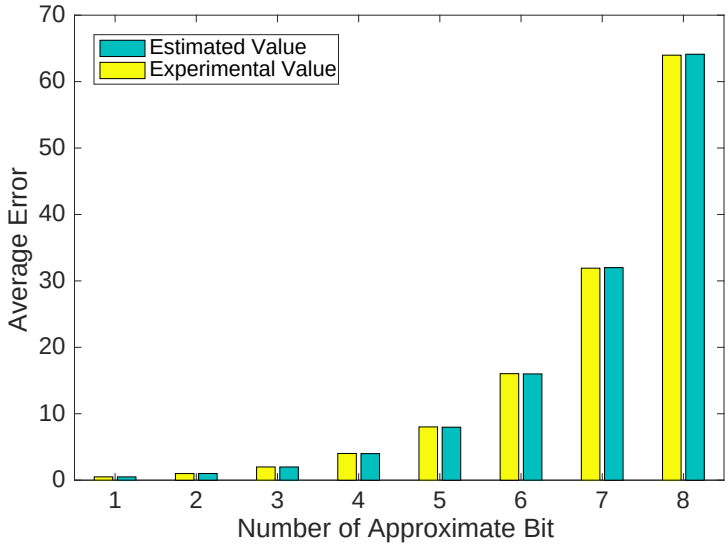


Figure 2.6: Average error of 9-bit SARA in different configuration.

We can verify Equation (2.21) by numerical simulation of a 9-bit SARA design. In our experiment, SARA consists of 9 sub-adders whose width is 1 bit. The results are from 200K run

of Monte Carlo simulation with uniform distributed numbers as input. As shown in Figure 2.6, there are 2 sets of data for comparison, experimental data are obtained directly in experiments and estimated data are calculated by Equation (2.21). The average errors from experiment are almost the same as the estimated values. According to the analysis above, we can estimate the average error of SARA in any configuration, given the distribution of input numbers.

Since $|\mathcal{I}| = K-1$, the error of the worst case approximation mode increases with the number of sub-adders, K. In addition, area overhead increases with K. On the other hand, a large K implies smaller L, and thus often facilitates shorter critical path and more power reductions. Therefore, K significantly affects the tradeoff among accuracy, power, delay and area.

2.5 Delay-Adaptive Reconfiguration of SARA

Almost all previous works on accuracy configurable adder [12, 37, 38, 13, 39] reasonably assume that accuracy configuration is decided by architecture/system level applications. We propose a self-configuration technique for the scenarios where architecture/system level choice is either unclear or difficult. Simulation results show that SARA with the self-configuration outperforms several previous static approximate adder designs.

The main idea of self-configuration is based on the observation that the actual worst case path delay depends on addend values. Specifically, the actual path delay is large only when a carry is propagated through several consecutive bits. Any false propagate bit from the addends results in a shorter carry propagation chain. When the actual carry propagation chain is short, there is no need to use approximation configuration, which is intended to cut carry chain shorter. We propose a Delay Adaptive Reconfiguration (**DAR**) technique: the output of a MUX in SARA is set to approximation mode only when a potentially long carry chain is detected. Compared to the constantly-approximate configuration, some errors for actual short carry chains are avoided, the actual long carry chain is cut shorter, and delay/power reduction can be still obtained.

The long carry chain detection and SARA-DAR design are shown in Figure 2.7(a). When MUX is switched to accurate mode by any false propagate bit in detection window, the actual carry chain is retained by the position of false propagate bit. To obtain a shorter carry chain in

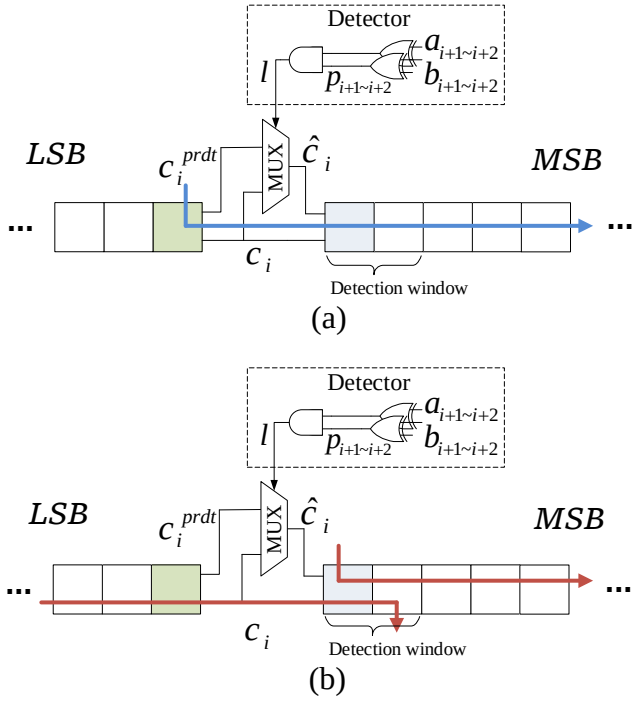


Figure 2.7: Design of DAR for SARA operating in a) approximate mode and b) accurate mode.

accurate mode, the detection window for MUX at bit i in MSB should start from bit $i+1$. In the example of Figure 2.7, we use a detection window of 2 bits (p_{i+1} and p_{i+2}) to tell if there is a carry propagation across two sub-adders, and configure the MUX according to

$$\hat{c}_i \leftarrow \begin{cases} c_i^{prdt}, & \text{if } p_{i+1} \cdot p_{i+2} \text{ is true} \\ c_i, & \text{otherwise.} \end{cases} \tag{2.22}$$

In approximation mode, the effective carry chain is represented by the blue line in Figure 2.7(a) and its length is no greater than $L+1$ bits. When the MUX is set to accurate mode, the carry chain is indicated by the red lines in Figure 2.7(b) and their lengths can be restrained to within $L+2$ bits. Since the propagate bits only depend on local primary inputs, we can reuse propagate bits in higher bits to save cost. Note that in this case the detection overhead here is almost the minimum possible, i.e., only one NAND gate for configuring each MUX.

In Figure 2.7, we use 2-bit detection window, which can be generalized to W-bit. Then, the error rate for MUX at bit i becomes

$$\widehat{ER}_i^{dar} = ER_i^{prdt} \cdot \prod_{j=1}^{W} P(p_{i+j} = 1) \tag{2.23}$$

The detection window size W decides the tradeoff between accuracy and the effective carry chain length in accurate mode, which is $L + W$. When W increases, the error rate decreases while the critical path length in accurate mode increases.

2.6 Experimental Results

2.6.1 Experiment Setup and Evaluation

Our SARA, SARA-DAR and several previous designs are synthesized to 32-bit adders by Synopsys Design Compiler using the Nangate 45nm Open Cell Library. The synthesized circuits are placed and routed by Cadence Encounter. The default supply voltage level is 1.25V. To make fair comparisons across architectures, we describe all designs by structural modeling in Verilog to reduce the impact of synthesis and optimization. For comparison, we synthesize the accurate adder in behavioral modeling which is described by expressional operator in Verilog. The netlist of such accurate adder should be automatically optimized by synthesizer in Design Compiler, which is different from any man-craft gate-level design. In addition, we set the same supply voltage and no delay constraint on all designs for the same reason.

The evaluation of accuracy configurable adder designs can be subtle and therefore is worth some discussion.

1. **Area**: In the literature, the area sometimes refers to the part of the circuit working in a certain mode, e.g., the circuit for the accurate part is not included in area estimation when evaluating approximation mode. We report the routed layout area of each entire design.

2. **Delay**: Some configurable adders, such as ACA [12] and GeAr [37], implement error correction with pipelining, which sometimes takes multiple clock cycles to determine the complete

result. The delay or performance evaluation of such designs is much more complicated than unpipelined designs. Our work is focused on unpipelined implementation, although it can be pipelined. Thus, the reported delay is the maximum combinational logic path delay obtained from Synopsys PrimeTime with consideration of wire delay.

3. **Power**: The power dissipation is estimated by Synopsys PrimeTime considering both static and dynamic power.

4. **Accuracy**: We use PSNR (Peak Signal-to-Noise Ratio), where errors are treated as noise, as a composite accuracy metric for considering both error magnitude and error rate. In addition, the worst case error, which is equivalent to the maximum error magnitude [11], and error rate are also reported. Each error result is from 100K-run Matlab-based Monte Carlo simulation assuming uniform distribution of addends.

5. **Tunability**: This means the range and granularity of runtime accuracy configurations. Sometimes, this can be confused with design-time flexibility.

6. **Tradeoff**: The tradeoff among the above factors is complex and is difficult to capture in a simple picture. To this end, we use composite metrics including power-delay product (PDP), energy-delay product (EDP) and iso-delay power.

2.6.2 Results of Tradeoff for Different Configurations

In this part, we mainly compare the following accuracy configurable adder designs:

- GDA [13]: We use the same design as in [13], where each sub-adder has 4 bits. This design can be configured by choosing accurate or predicted carry-out for each sub-adder. The carry prediction at each segment can also be configured to different accuracy levels by using different number of lower-bit addends.

- RAP-CLA [39]: We implement four different designs with carry prediction bit-width from 1 bit to 4 bits, which is reflected in the name. For example, RAP-CLA2 means each of its

carry prediction is from its 2 lower bits. As in [39], each design can be configured to either only one approximation mode or accurate mode.

- SARA: This is our proposed design and we evaluate sub-adder bit-width of 1 bit, 4 bits and 8 bits, referred to as SARA1, SARA4 and SARA8, respectively.

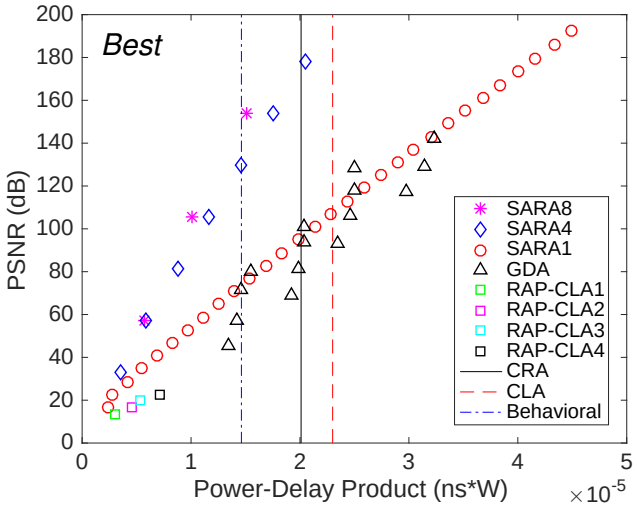


Figure 2.8: SARA: PSNR versus power-delay product.

The main result is shown in Figure 2.8, where each point is from one configuration of one design. The computation accuracy is evaluated by PSNR while the conventional design objectives are characterized by PDP. A design and configuration is ideal if it has large PSNR but low PDP, i.e., northwest in the figure. PDPs of two classic accurate designs, CRA and CLA, are indicated by the two vertical lines as their PSNR is near infinity. The result of SARA working in completely accurate mode is unable to be presented in the figure, because its infinite PSNR cannot be displayed as a single dot in the plot. Evidently, the best solutions are from SARA4 and SARA8. At $100dB$ PSNR, the PDP of SARA4 and SARA8 is about a half of GDA or CRA. The solutions from RAP-CLA, the latest previous work, are also largely dominated by SARA in PSNR-PDP tradeoff. An interesting case is SARA1. Its tradeoff is similar as GDA and not as good as SARA4 or SARA8.

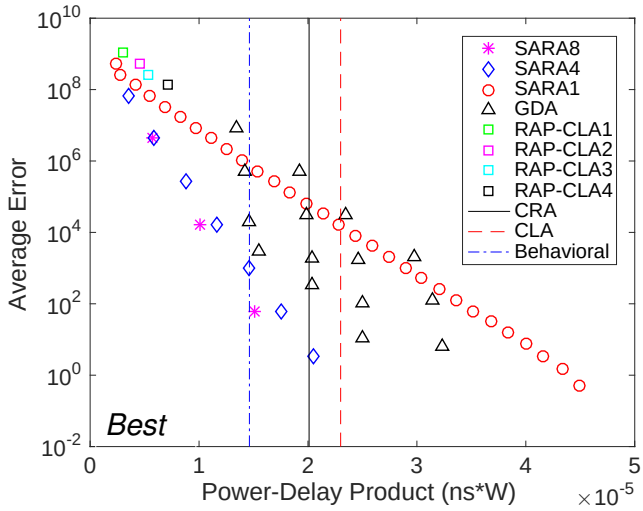


Figure 2.9: SARA: Average error versus power-delay product.

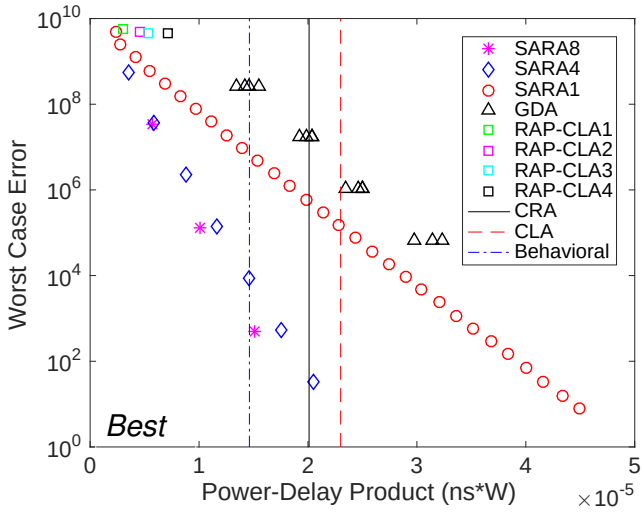


Figure 2.10: SARA: The worst case error versus power-delay product.

However, its runtime tunability is superior to all the other designs. It has the largest tuning range, the finest tuning granularity and very smooth tradeoff.

Figure 2.9 and 2.10 show the tradeoff between error magnitude and power-delay product. Ideally a better design or configuration has smaller average error or worst case error with lower PDP, which can be marked in the lower left corner of the figure. In Figure 2.9, SARA4 and SARA8 dom-

inate other designs in average error-PDP tradeoff. For each configuration, SARA4 and SARA8 have almost the lowest average error at a certain PDP level. Although SARA1 cannot achieve superior average error and PDP tradeoff to GDA, it shows fine-grant tunability in a large range same as PSNR-PDP tradeoff. Figures 2.10 depicts the worst case error versus PDP and confirms the trend observed in the PSNR-PDP tradeoff. All SARA designs even for SARA1 have lower worst case error than previous work at the same PDP level. In addition, the result of SARA working in accurate mode cannot be found in the plot. That's because the y-axis is in Logarithmic scale and zero error will be converted into infinite which cannot be displayed as a single dot.

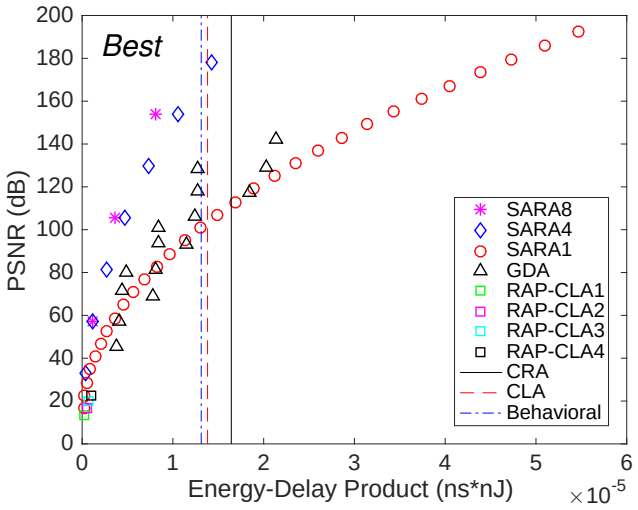


Figure 2.11: SARA: PSNR versus energy-delay product.

EDP is another metric to efficiently evaluate tradeoffs between circuit level power saving techniques for digital designs. Figure 2.11 to 2.13 illustrate accuracy versus EDP, which have similar trend in accuracy-PDP tradeoff. Most configurations of SARA4 and SARA8 have lower EDP than accurate adder CRA and CLA. At a certain EDP level, SARA4 and SARA8 still dominate GDA and RAP-CLA with larger PSNR, smaller average error or worst case error. SARA1 in different configurations cover the range from lowest to highest EDP, which provides finest tunability in accuracy-energy tradeoff among different architectures.

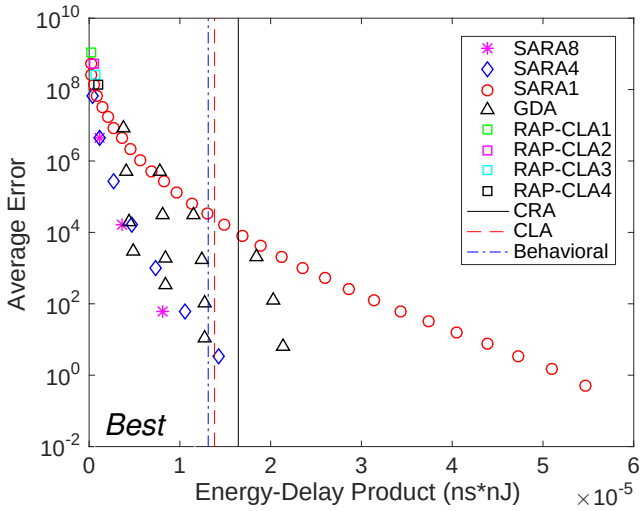


Figure 2.12: SARA: Average error versus energy-delay product.

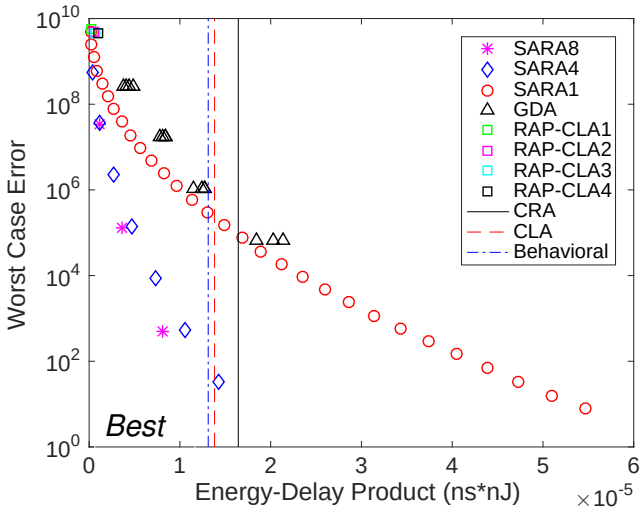


Figure 2.13: SARA: The worst case error versus energy-delay product.

2.6.3 Results of Tradeoff for Delay-Adaptive Reconfiguration

This part is to evaluate the SARA-DAR design, where the configuration decision has already been made. Hence, it makes sense to additionally compare with static approximate adders, where no configuration is needed. Static approximate adder designs including ETAII[6], FICTS[11] and AFICTS[11] are implemented in the experiment. In addition, CRA-based approximate designs

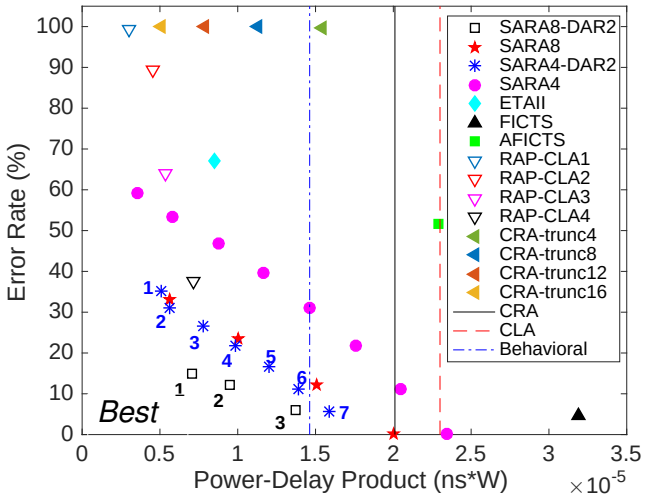


Figure 2.14: SARA-DAR: Error rate versus power-delay product.

CRA-trunc$i$ implemented by ignoring lowest i bits in addends are presented, which is a simple but good baseline for comparison. Seven SARA4-DAR designs are obtained based on seven configurations of SARA4 with detection window of 2 bits, while three SARA8-DAR are based on different configurations of SARA8. That is, if a MUX at bit i is configured to accurate carry in SARA4/SARA8, bit i of corresponding SARA-DAR is hard-wired to accurate carry without using MUX. When bit j in SARA4/SARA8 is in approximation mode, bit j of corresponding SARA-DAR uses the delay-adaptive reconfiguration.

Figure 2.14 shows the error rate versus PDP tradeoff. The dot of SARA4-DAR2 labeled with '1' represents the counterpart of SARA4 when all the MUXes are controlled by delay-adaptive reconfiguration. When we remove the MUX at the highest bit to propagate accurate carry and keep others in delay-adaptive reconfiguration, another SARA4-DAR2 design could be obtained (another dot labeled with '2' in the figure). If we go on to remove more MUXes in MSB, a series of SARA4-DAR2 designs shown as dots with label '3' to '7' can be obtained. Three SARA8-DAR2 designs are created in the same way. According to the figure, the error rate of SARA is mostly lower than RAP-CLA. By using delay-adaptive reconfiguration, SARA-DAR often has less error rate and PDP than SARA. SARA-DAR also greatly outperforms the static approximate adders in

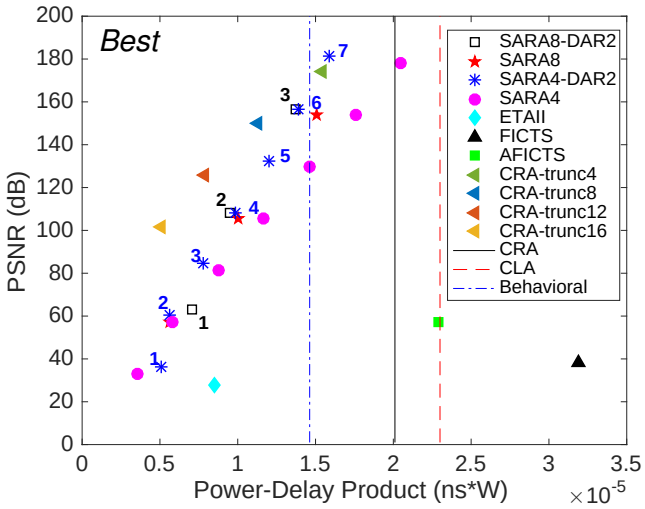


Figure 2.15: SARA-DAR: PSNR versus power-delay product.

both error rate and PDP. Moreover, in Figure 2.14 we can observe those dots right on the x-axis which represent SARA4 and SARA8 working in accurate mode. Both of them achieve zero error rate. PDP of accurate SARA8 is about $2 \times 10^{-5} ns \cdot W$, while PDP of accurate SARA4 is almost $2.3 \times 10^{-5} ns \cdot W$. In Figure 2.15, SARA-DAR also demonstrates better PSNR-PDP tradeoff than other designs, except for comparing with CRA-trunc at some low-PSNR levels. However, CRA-trunc has almost 100% error rate since it dismiss lower bits in addends, which is the worst among all static approximate adders. Figure 2.16 and 2.17 show tradeoff between accuracy and EDP. At a certain EDP level, SARA-DAR has almost the same PSNR as CRA-trunc, which is the best among all static approximate adders.

2.6.4 Impact of Detection Window in Delay-Adaptive Reconfiguration

This part shows the impact of detection window in the tradeoff for delay-adaptive reconfiguration. According to Equation (2.23), the error rate of MUX output can be reduced by delay-adaptive reconfiguration. As the length of detection window increases, the error rate would decrease because there are less probability that MUX is configured in approximate mode. As a result, the overall error rate varies with the size of detection window. Figure 2.18 and 2.19 show the changes of error rate and PSNR of SARA4-DAR with different detection window. As the size of detection

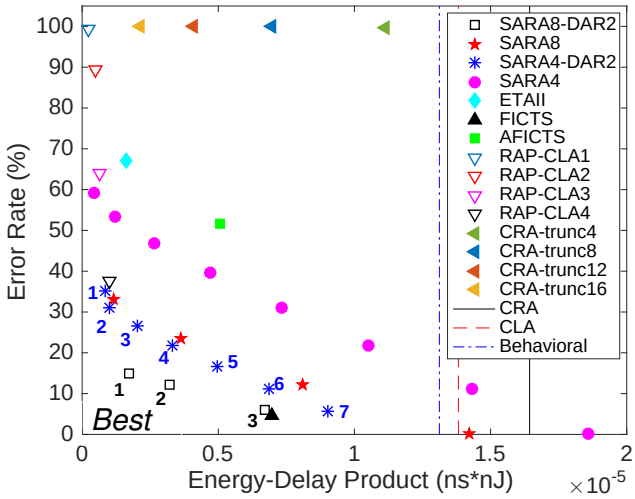


Figure 2.16: SARA-DAR: Error rate versus energy-delay product.

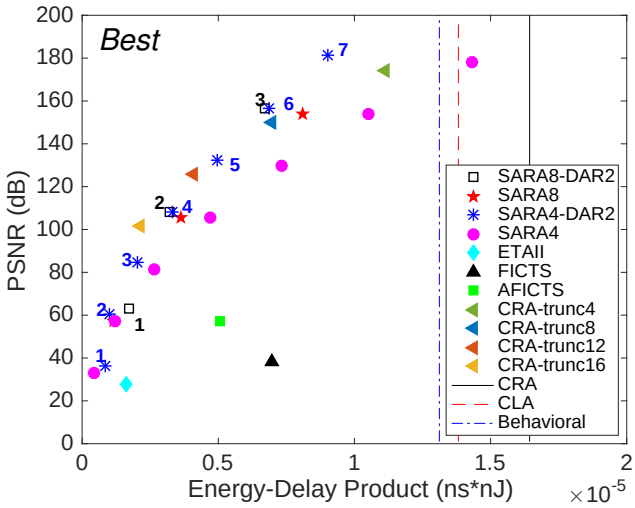


Figure 2.17: SARA-DAR: PSNR versus energy-delay product.

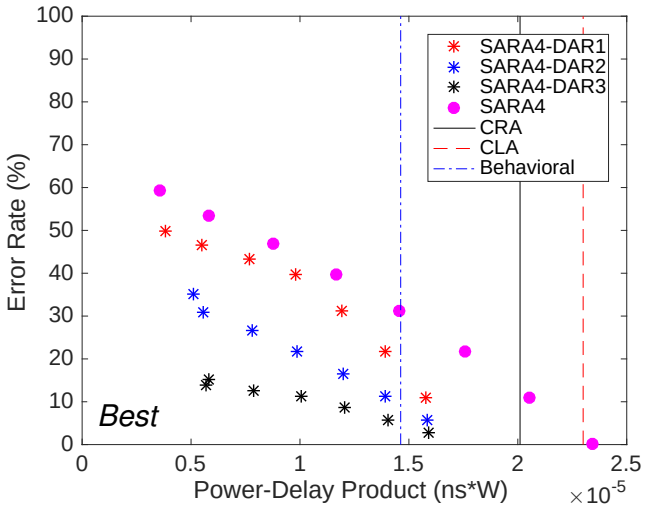


Figure 2.18: Error rate of SARA4-DAR with different detection window.

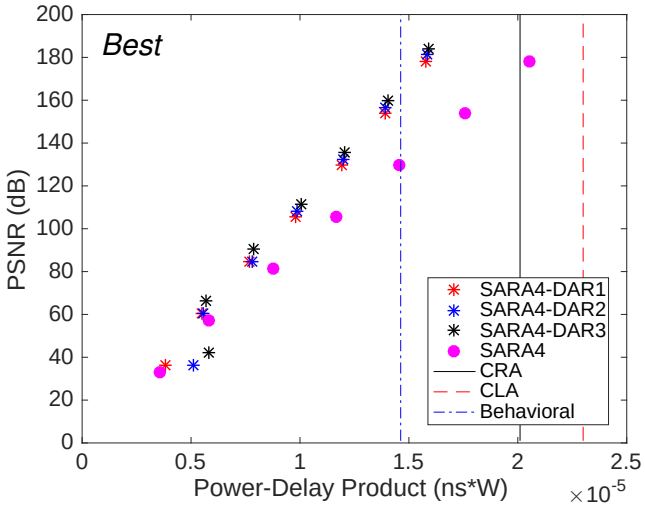


Figure 2.19: PSNR of SARA4-DAR with different detection window.

window increases from 1 to 3, the error rate decreases compared to its SARA counterpart. However, we can observe that the gap of error rate between SARA4-DAR2 and SARA4-DAR1 varies with different configuration. Although the change in error rate for individual MUX of SARA-DAR is proportional to the size of detection window (as shown in Equation (2.23)), the overall error rate in output results might not show linear change. When the size of detection window increases by 1,

PSNR of SARA4-DAR increases by about 3dB on average. We can also find that the PDP gap between SARA4-DAR2 and SARA4-DAR1 varies with different configurations in both figures. The change of PDP between SARA4-DAR2 and SARA4-DAR1 in most configurations is very small, while it's larger in the first configuration (which is presented as the first dot of SARA4-DAR in the left of the figures). It is mainly attributed to unproportioned change in delay between different configurations.

2.6.5 Results of Iso-delay Power and Area

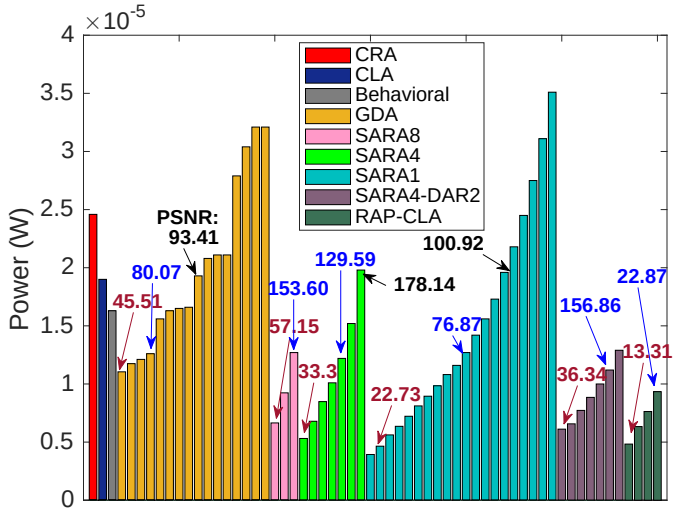


Figure 2.20: Iso-delay power comparison. The numbers are PSNR.

Although power-delay product results have been shown in Sections 2.6.2 and 2.6.3, the tradeoff between power and delay is still unclear. The power-delay tradeoff can be obtained by different accuracy configurations or varying supply voltages. Different combinations of configurations and voltages may lead to overwhelming volume of results, which are difficult to interpret, especially when implication to accuracy is involved at the same time. Thus, we indicate the tradeoff by investigating the iso-delay power, which is the power of each circuit tuned to the same critical path delay ($0.82ns$) by voltage scaling. The results are shown in Figure 2.20. In general, SARA4,

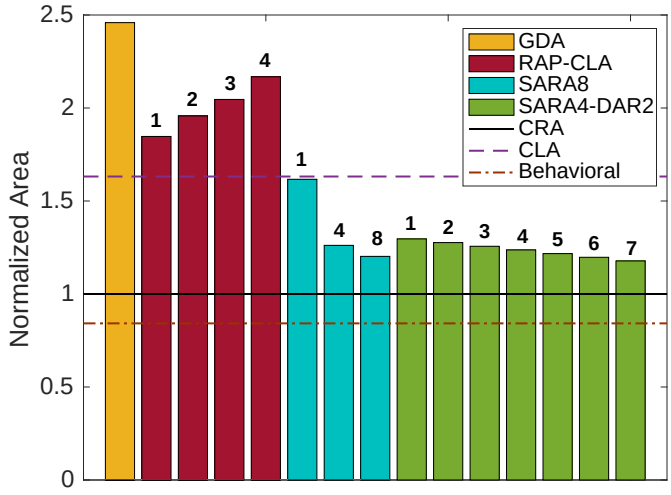


Figure 2.21: Area comparison.

SARA8 and SARA4-DAR can achieve much lower power than CRA. Although GDA and RAP-CLA seem to provide low power, their PSNR is much less than our designs. Compared at the same iso-delay power level, SARA has more than 20dB increase in PSNR than RAP-CLA, while GDA has more than 70dB decrease than SARA designs. SARA1 shows a large range of iso-power tuning which could reach the lowest and highest power among all adders. We do not have iso-delay power for approximate adders working in accurate mode, because the delay of such case is larger than CRA due to induction of MUXes which cannot provide sufficient room for reducing supply voltage.

Last but not the least, we compare area of these designs in Figure 2.21. Same as our expectation, GDA and RAP-CLA have greater area than CLA while area of SARA4 or SARA8 is significantly smaller than CLA. SARA1 has almost the same area as CLA due to MUXes in every bit which aid the accuracy configuration. On average, the area of SARA is 39% smaller than that of RAP-CLA and 50% smaller than that of GDA.

2.7 Applications

2.7.1 Extension to Multiplier

In complicated datapath system, multiplier is considered as a much bigger component in power consumption. Our carry-prediction-based approximation uses generate bit to predict the carry from lower segments. The critical delay can be restrained to a smaller value with shorter critical path in carry propagation. Further extension of our technique to multiplier depends on the multiplication structure used in hardware implementation. There is a variety of hardware designs for multiplication, according to the structures of reduction tree. In this section, we apply our technique on three kinds of multiplication structures including array multiplier, Wallace multiplier and Dadda multiplier.

As shown in Figure 2.22, the basic structure of multiplier employs a three-step process to multiply two integers.

- Step 1: Generate all partial products by using an AND gate array.
- Step 2: Combine the partial products in k stages by layers of half/full adder until the matrix height is reduced to two. Different types of structures depend on the reduction tree used to reduce the number of partial products in this step.
- Step 3: Sum the resulting numbers in the final stage by a conventional adder.

In array multiplier the carry bits in one stage are propagated diagonally downwards, which follows the basic shift-and-add multiplication algorithm. Wallace multiplier based on Wallace tree combines the partial products as early as possible, which makes it faster than array multiplier[40]. Dadda's strategy is to make the combination take place as late as possible, which leads to simpler reduction tree and wider adder in final stage[40]. Thus, we can design approximate multipliers by using our SARA design instead of CRA in the final stage.

Three types of 16×16 multipliers (array multiplier, Wallace multiplier and Dadda multiplier) as well as behavioral multiplier are synthesized and implemented by using Nangate 45nm Open

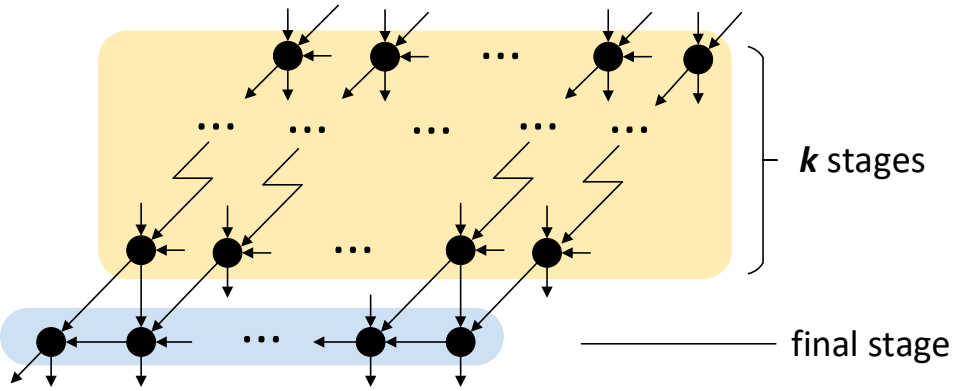


Figure 2.22: Basic structure of multiplier.

Cell Library. Their error data are obtained from 100K-run Monte Carlo simulation with uniform distribution of operands. In approximate multiplier the final stage uses SARA4 which consists of sub-adders with bit-width of 4 bits, while the accurate one uses CRA. Figure 2.23 and 2.24 present the tradeoff between error and PDP. Most of approximate multipliers configured in approximate mode have better PDP compared with the accurate multipliers. The variance of error between different approximate mode in approximate multiplier has similar trend as SARA. Total error increases as more bits are configured in approximate mode. Approximate array multiplier shows larger error than approximate Wallace/Dadda multiplier at the same PDP level. It's because array multiplier has larger critical delay from internal stages in step 2 than Wallace/Dadda multiplier.

Figure 2.25 and 2.26 show the error versus EDP for both accurate and approximate multipliers. As more MUXes are set to propagate approximate carry, the average error in output increases to about 10^7, which as well achieves best EDP. The worst-case error rate of approximate Dadda multiplier is about 30%, while it comes to about 17% for approximate array multiplier and Wallace multiplier. As shown in Figure 2.26, when approximate multipliers are working in completely accurate mode (error rate equals 0), EDP is larger than that of their accurate counterpart. In summary, The experimental results show that our technique can be successfully extended to high speed multiplier designs. And due to the simple but effective structure of SARA it provides an easy way

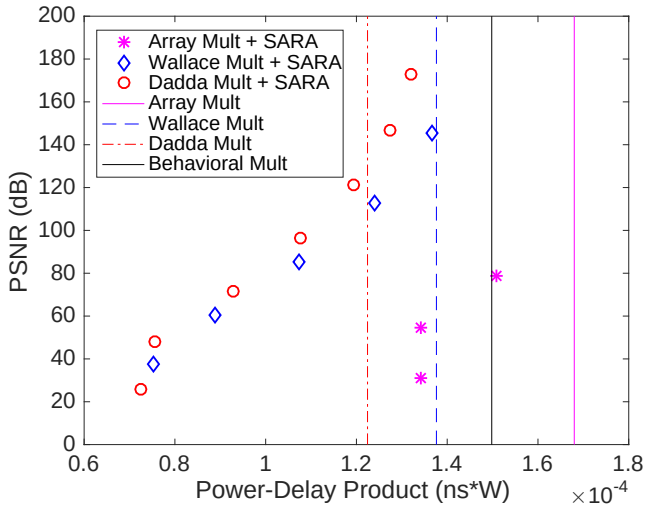


Figure 2.23: Multiplier: PSNR versus power-delay product.

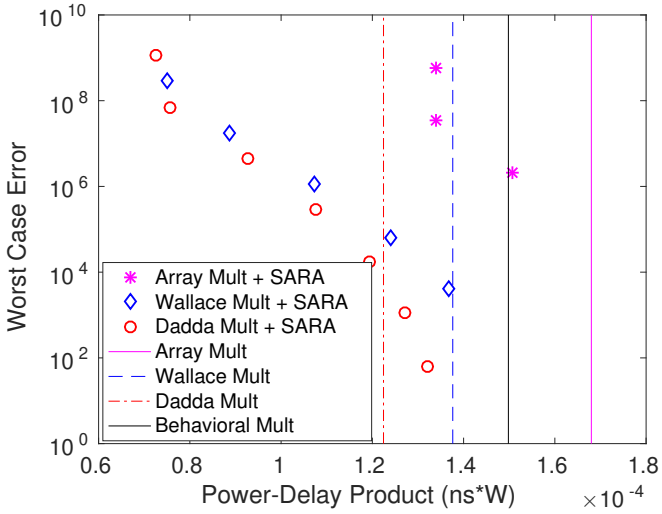


Figure 2.24: Multiplier: The worst case error versus power-delay product.

for us to convert conventional multiplier into approximate design.

2.7.2 DCT Computation in Image Processing

The discrete cosine transform (DCT) has been recognized as the basic in many transform coding methods for image and video signal processing. It is used to transform the pixel data of image or video into corresponding coefficients in frequency domain. Since human visual system is more

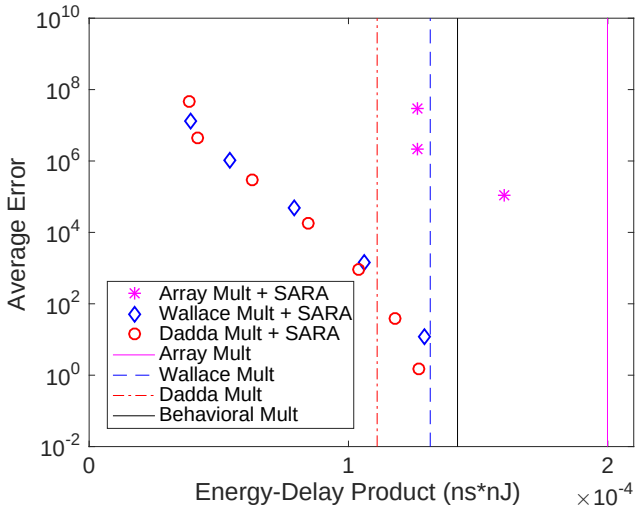


Figure 2.25: Multiplier: Average error versus energy-delay product.

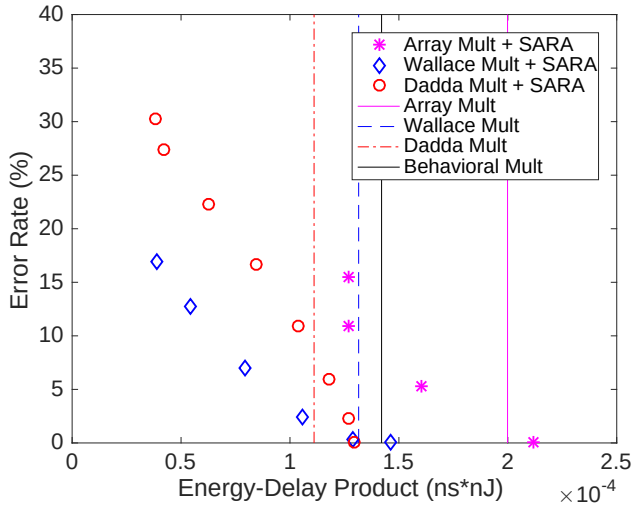


Figure 2.26: Multiplier: Error rate versus energy-delay product.

sensitive to the changes in low frequency, the lost of accuracy in high-frequency components does not heavily degrade the quality of image processed by DCT. In addition, those components in different frequency have different tolerances to the degradation in original data. It is a good example to show the reconfigurability of our design by applying them in VLSI implementation of DCT computing in JPEG image compression.

Figure 2.27: Comparison of image lenna: (a) accurate adder; (b) SARA4; (c) SARA8; (d) SARA4-DAR2; (e) GDA; (f) RAP-CLA.

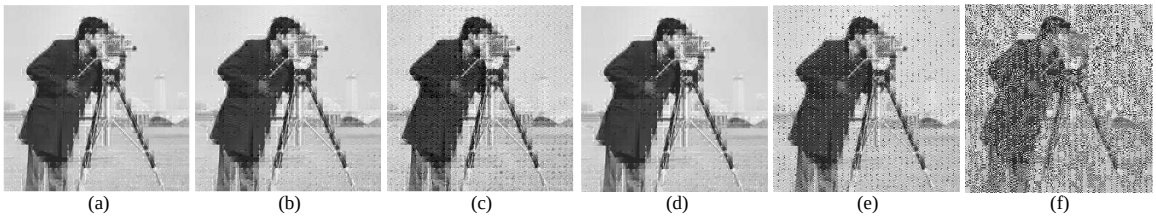


Figure 2.28: Comparison of image cameraman: (a) accurate adder; (b) SARA4; (c) SARA8; (d) SARA4-DAR2; (e) GDA; (f) RAP-CLA.

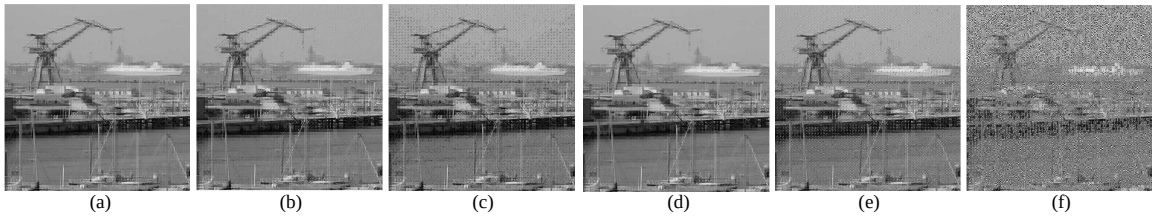


Figure 2.29: Comparison of image kiel: (a) accurate adder; (b) SARA4; (c) SARA8; (d) SARA4-DAR2; (e) GDA; (f) RAP-CLA.

The two-dimensional DCT is implemented by the row-column decomposition technique, which contains two stages of 1-D DCT[41, 42, 43]. The 2-D DCT of size $N \times N$ could be defined as

$$Z = C^t X C \tag{2.24}$$

where C is a normalized Nth-order matrix and X is the data matrix. Generally the image is di-

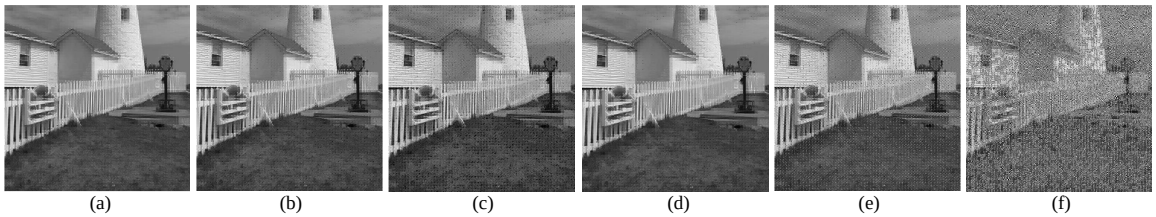


Figure 2.30: Comparison of image house: (a) accurate adder; (b) SARA4; (c) SARA8; (d) SARA4-DAR2; (e) GDA; (f) RAP-CLA.

vided into several $N \times N$ blocks and each block is transformed by 2-D DCT into frequency domain components. The VLSI implementation of DCT computing contains a set of ROM and Accumulator Components (RACs) which can be implemented by multipliers and adders[41, 42, 43]. In this application we use approximate adders to replace those accurate ones in RACs to implement an imprecise but low power circuit for image processing which contains DCT computing.

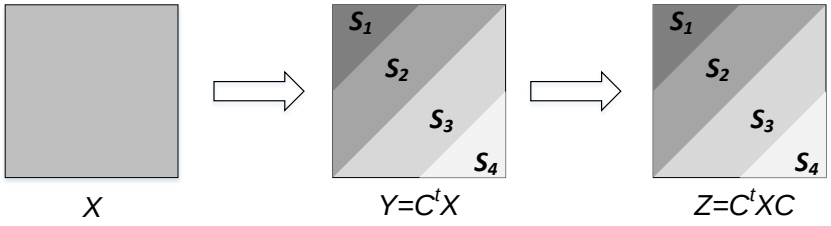


Figure 2.31: 2 dimensional descrete cosine transform.

We replace the adders in circuits with different configurations of SARA, SARA-DAR, GDA as well as RAP-CLA. The results are obtained by numerical simulations on 4 images (Fig. 2.27-2.30) in Matlab. As we know, after DCT process data in different frequency domain have different level of error tolerance. As shown in Figure 2.31, matrix components in the upper-left corner correspond to lower frequency coefficients which are sensitive to human vision, while those components in lower-right corner might allow more errors.

Table 2.4: Image Quality Comparison in PSNR

	lenna	cameraman	kiel	house	AVERAGE
Accurate	39.85	38.23	37.68	37.35	38.27
SARA4	38.32	37.50	36.83	36.53	37.30
SARA8	35.33	35.07	34.92	34.81	35.03
SARA4-DAR2	39.45	37.90	37.43	37.00	37.97
GDA	34.53	34.55	34.88	34.20	34.54
RAP-CLA	33.38	33.44	33.51	33.39	33.43

To utilize this feature for better energy-accuracy tradeoff, we make following configuration for different designs.

1. **SARA4**: SARA4 with 4, 3, 2, 1 consecutive segments working in accurate mode are used to compute components in S_1, S_2, S_3 and S_4 respectively.

2. **SARA8**: SARA8 with 1 segments in accurate mode are used to compute components in S_1, S_2, while another configuration with all segments in approximate mode are for S_3, S_4.

3. **SARA4-DAR2**: DAR counterpart of SARA4 with detection window of 2 bits.

4. **GDA**: $GDA_{4,1}, GDA_{3,1}, GDA_{2,1}, GDA_{1,1}$ (same notation as [13]) are used to compute components in S_1, S_2, S_3 and S_4 respectively.

5. **RAP-CLA**: since RAP-CLA can work in one approximate mode, we use RAP-CLA with window size of 20, 16, 12, 8 to compute components in S_1, S_2, S_3 and S_4.

The image processing results are shown in Table 2.4. PSNR in the table is defined via the mean squared error (MSE). Given an $m \times n$ image I and its restored image K, MSE and PSNR are defined as

$$MSE = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} [I(i,j) - K(i,j)]^2 \tag{2.25}$$

$$PSNR = 20 \cdot \log(MAX_I) - 10 \cdot \log(MSE), \tag{2.26}$$

where MAX_I is the maximum pixel value of the image. SARA4-DAR2 has the highest PSNR

for every image among all configurable adders, which is close to the quality of accurate adder. Comparing SARA8 with GDA, they have similar PSNR and similar delay, but SARA8 has less power consumption according to the analysis in the previous section. SARA4-DAR2 achieves better image quality than SARA4, but might result in more power due to additional logic for self-configuration. The image quality for different adders in DCT computing can also be demonstrated in Figure 2.27 to 2.30. According to human vision, SARA and its DAR counterpart show better image quality than GDA and RAP-CLA in JPEG compression processing.

2.8 Conclusion

In this chapter, we propose a simple accuracy reconfigurable adder design SARA. It has significantly lower power/energy-delay product than the latest previous work when comparing at the same accuracy level. In addition, SARA has considerable lower area overhead than almost all the previous works. The accuracy-power-delay efficiency is further improved by a delay-adaptive reconfiguration technique. We demonstrate the efficiency of our adder in the applications of multiplication circuits and DCT computing circuits for image processing.

3. LAYOUT RECOGNITION ATTACKS ON SPLIT MANUFACTURING*

3.1 Introduction

Split manufacturing is a security technique against untrusted foundries. By having only front-end-of-line (FEOL) layers manufactured at an untrusted foundry while the back-end-of-line (BEOL) is fabricated at a trusted foundry, attackers in the untrusted foundry do not have complete information to perform attacks such as Trojan insertion, piracy, and overproduction [14, 15, 16]. The same principle can be applied to 3D ICs, where different dies are manufactured by different foundries, since each 3D IC contains two or more independently manufactured ICs which are vertically stacked on top of each other [17]. Despite the security enhancement, split manufacturing still has a significant risk of being successfully attacked [18, 19, 20].

There are two attack scenarios in split manufacturing. (i) The attacker at FEOL foundry does not have circuit netlist and attempts to reverse engineering the entire design for stealing intellectual property, and conducting piracy and overproduction; (ii) The attacker has circuit netlist and tries to recognize critical components on the layout for inserting Trojans. Most existing attack models are proposed for (i), such as the work in [19], and there are also defense methods against the attack models, such as placement perturbation [19] and routing perturbation [44].

Table 3.1: Comparison of different attack methods.

	SAT bijective [17]	Network-flow attack [19]	Structural pattern matching
Target to attack	Cells	Wires	Cells
Use of structural information	$\surd$	$\times$	$\surd$
Use of design convention	$\times$	$\surd$	$\surd$
Knowledge of circuit netlist	$\surd$	$\times$	$\surd$
Recovery of BEOL connections	$\times$	$\times$	$\surd$

*Republished with permission of ACM (Association for Computing Machinery), from Wenbin Xu, Lang Feng, Jeyavijayan (JV) Rajendran, Jiang Hu, Layout Recognition Attacks on Split Manufacturing, Proceedings of the 24th Asia and South Pacific Design Automation Conference, 2019; permission conveyed through Copyright Clearance Center, Inc.

Table 3.2: Comparison of different defense methods.

	Placement perturbation [19]	Routing perturbation [44]	K-security [17]
Modification on placement	$\surd$	$\times$	$\surd$
Modification on routing	$\times$	$\surd$	$\surd$
Defense for physical attack	$\surd$	$\surd$	$\surd$
Defense for logical attack	$\times$	$\times$	$\surd$

With access to the netlist, the attack in scenario (ii) may seem to be easier than that in scenario (i). However, (ii) requires much more understanding of the layout design than (i). More specifically, the attack in (i) only needs to reproduce the overall design without understanding its layout details while (ii) must recognize layout components and match them exactly to the netlist. With BEOL information missing, the layout-to-netlist matching is not trivial at all. However, the investigation on scenario (ii) is mostly restricted to only defense techniques. For example, a previous work [17] proposed a method using a metric called k-security to protect the layout by obfuscation. The metric k-security means that for every group of components of which the connections are visible in FEOL, it will have at least another $k-1$ groups (k is an integer larger than 1) which are identical to it logically. They also proposed a SAT-based algorithm [17] for selecting wires to be manufactured in BEOL to achieve the k-security. Moreover, another work [21] improves the algorithm in [17] and realizes a shorter runtime for achieving k-security. More details about the comparison of different typical attack methods and that of different typical defense methods are shown in Table 3.1 and 3.2.

To the best of our knowledge, this work provides the first systematic study on the attack in split manufacturing when untrusted foundry has access to netlist and intends to insert Trojans. Our work focuses on recognizing critical components in FEOL layouts rather than actually inserting Trojans. We propose a new attack technique using structural pattern matching assisted by hints from design conventions. We also implemented the SAT-based bijective mapping attack [17]. The experiment results show that the structural pattern matching attack is more efficient than the bijective mapping attack. For split manufacturing without k-security defense, the proposed attack techniques usually

achieve near 100% success rate. They are also applied to designs with k-security defense to confirm the effectiveness of such defense. Moreover, the evaluation by actual attacks reveals richer and more tangible interpretation for k-security in split manufacturing. The main contributions of this study are:

- We proposed an attack method based on structural pattern matching for layout recognition. Our attack method achieves shorter runtime and/or better performance than the SAT-based bijective mapping attack for circuits with and without k-security defense.
- The attack method we proposed uses both logical information and hints from design conventions, which are seldom considered at the same time within a same attack before.
- Our work on attack techniques provides an alternative way to evaluate the defense of split-manufactured ICs for Trojan insertion.

3.2 Preliminary

3.2.1 Attack Scenario

In this work, we consider the scenario that an attacker is at an untrusted FEOL foundry and intends to insert Trojan into split-manufactured ICs. The attacker needs to understand circuit functions in the FEOL layout so that effective Trojan site can be decided. The attacker is assumed to know the following information:

1. Layout of transistor and FEOL metal layers.
2. Entire circuit netlist or part of the netlist that is related to Trojan insertion. The attacker may obtain this information by stealing from design companies or collaborating with an observer in design stage [17, 21].
3. The technology library containing information about logic gates: layout structure, delay, capacitance load, wire capacitance and design specifications.

The assumption on the knowledge of the gate-level netlist for the attack is possible. Current circuit design flow cannot and is not intended to defend against any attackers in the design stage who have the full knowledge of the netlist. An observer in the design stage might have chances to collaborate with an attacker in the foundry. The attacker can easily access to the gate-level netlist via the sharing from the observer. On the other hand, the attacker can reverse engineer the FEOL layouts to obtain the incomplete netlist. Then, the attacker will try to identify the cells in FEOL layouts that are determined as targets for Trojan insertion.

3.2.2 Related Work

Split manufacturing was proposed to improve IC security against reverse engineering and Trojan insertion [45, 46]. There are several research works showing the benefits of split manufacturing on digital ICs [47, 48, 49, 50, 51] and analog ICs [52]. The concept of split manufacturing can be extended to 3D or 2.5D IC designs such that different dies are manufactured at different foundries [53].

For the attack scenario where attackers only have incomplete designs, several works provide different techniques to restore missing BEOL wires so that the design can be reverse engineered [18, 19, 20]. Their attack is evaluated with a number of correctly restored wires even though the functions of related layout components are still unknown. To defend against such kind of attack, several strategies based on placement or routing are introduced [19, 44, 54]. Different from heuristic approaches for enhancing security, the work in [55] proposed a theoretical model based on information theory.

The work in [17] shows a different attack scenario where attackers have access to circuit netlist and intend to recognize layout for Trojan insertions. It mentions SAT-based bijective mapping attack without detailed elaboration. It proposes the concept of k-security, which leads to k identical options when attackers attempt to recognize layout. Indeed, this approach causes up to 200% area overhead. The recent work [21] proposes to restrict k-security to a small and critical portion of the circuit so that the overhead can be reduced. It inserts dummy cells to obfuscate the design further and enhance the security. However, the defense techniques in both [17] and [21] are evaluated by

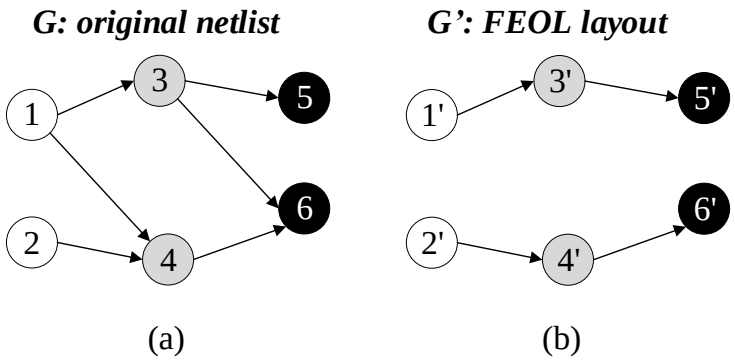


Figure 3.1: Graph representations of (a) circuit netlist and (b) its FEOL layout. Each vertex indicates a logic gate, whose logic type is represented by grayscale.

k-security without actual attacks.

Although the overall attack and defense in split manufacturing have been recently studied, there is no investigation on layout recognition attack driven by Trojan insertion, to the best of our knowledge.

3.2.3 SAT-based Bijective Mapping

SAT-based bijective mapping was briefly mentioned as a Trojan-driven attack to split manufacturing [17]. Generally, either circuit netlist or FEOL layout can be modeled as a graph, where each vertex represents a logic gate, and edges indicate wire nets. Normally, the netlist graph and its corresponding FEOL layout graph should have the same vertices, while the netlist graph contains more BEOL edges that are not available in the layout graph. Given a netlist graph $G = (V, E)$ and its layout graph $G' = (V', E')$ as shown in Figure 3.1, the goal of layout recognition attack is to find the vertex $v \in V$ that corresponds to each vertex $v' \in V'$. In bijective mapping, such correspondence is designated by Boolean variables

$$\phi_{ij} = \begin{cases} 1, & \text{if } v_i \in V \text{ can be mapped to } v'_j \in V' \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

Then, the bijective mapping should be subject to three constraints: 1) mapping v in G to v' in G', 2) mapping v' in G' to v in G, and 3) mapping e' in G' to e in G.

SAT-based bijective mapping is to solve the matching between the netlist and FEOL layouts.

1) Each vertex v_i in G should be mapped to one vertex v'_j in G' which has the same logic type as v_i:

$$F_1 = \prod_{v_i \in V_L} \sum_{v'_j \in V'_L} (\phi_{i,j} \prod_{v'_k \in V'_L, k \neq j} \overline{\phi}_{i,k}), \forall L \tag{3.2}$$

where $V_L \subset V$ and $V'_L \subset V'$ indicate the vertex subsets of logic type L.

2) Each vertex v'_j in G' should be mapped to one vertex v_i with same logic type in G:

$$F_2 = \prod_{v'_j \in V'_L} \sum_{v_i \in V_L} (\phi_{i,j} \prod_{v_k \in V_L, k \neq i} \overline{\phi}_{k,j}), \forall L \tag{3.3}$$

3) Each edge $(v'_j, v'_l) \in E'$ should be mapped to one edge $(v_i, v_k) \in E$ such that $\phi_{ij} = \phi_{kl} = 1$:

$$F_3 = \prod_{(v'_j, v'_l) \in E'} \sum_{(v_i, v_k) \in E} \phi_{ij} \cdot \phi_{kl} \tag{3.4}$$

Thus, the bijective mapping problem is reduced to a SAT instance constrained by

$$F = F_1 \wedge F_2 \wedge F_3. \tag{3.5}$$

Thus, the bijective mapping problem is reduced to a SAT instance [17]. By solving it via SAT solver, one can obtain one possible assignment for the set of ϕ_{ij}. When multiple assignments can be found, SAT or bijective mapping alone cannot decide which is the correct one. As such, an arbitrary feasible assignment solution is taken.

In the example shown in Figure 3.1, we can construct the SAT constraints as

$$\begin{aligned} F_1 =& (\phi_{11}\overline{\phi}_{12} + \overline{\phi}_{11}\phi_{12})(\phi_{21}\overline{\phi}_{22} + \overline{\phi}_{21}\phi_{22})(\phi_{33}\overline{\phi}_{34} + \overline{\phi}_{33}\phi_{34}) \\ & (\phi_{43}\overline{\phi}_{44} + \overline{\phi}_{43}\phi_{44})(\phi_{55}\overline{\phi}_{56} + \overline{\phi}_{55}\phi_{56})(\phi_{65}\overline{\phi}_{66} + \overline{\phi}_{65}\phi_{66}) \end{aligned} \tag{3.6}$$

$$F_2 = (\phi_{11}\overline{\phi}_{21} + \overline{\phi}_{11}\phi_{21})(\phi_{12}\overline{\phi}_{22} + \overline{\phi}_{12}\phi_{22})(\phi_{33}\overline{\phi}_{43} + \overline{\phi}_{33}\phi_{43}) \\ (\phi_{44}\overline{\phi}_{34} + \overline{\phi}_{44}\phi_{34})(\phi_{55}\overline{\phi}_{65} + \overline{\phi}_{55}\phi_{65})(\phi_{56}\overline{\phi}_{66} + \overline{\phi}_{56}\phi_{66}) \tag{3.7}$$

$$F_3 = (\phi_{11}\phi_{33} + \phi_{21}\phi_{43})(\phi_{12}\phi_{34} + \phi_{22}\phi_{44}) \\ (\phi_{33}\phi_{55} + \phi_{43}\phi_{65})(\phi_{34}\phi_{56} + \phi_{44}\phi_{66}) \tag{3.8}$$

$$F = F_1 \wedge F_2 \wedge F_3. \tag{3.9}$$

By solving Equation (3.9) via SAT solver, one satisfiable assignment is

$$q_1 = \phi_{11}\overline{\phi}_{12}\overline{\phi}_{21}\phi_{22}\phi_{33}\overline{\phi}_{34}\overline{\phi}_{43}\phi_{44}\phi_{55}\overline{\phi}_{56}\overline{\phi}_{65}\phi_{66}. \tag{3.10}$$

The SAT-based bijective mapping is straightforward in recognizing the layouts secured by split manufacturing. The advantage of this method is that it‘s a brute force attack which enumerates all the possible mappings. It could not miss any possible matching for cells between netlist and FEOL layout. The results obtained by this method must be highly reliable. In addition, it’s simple enough to implement whenever a SAT solver is available. However, its drawback is also obvious in practice.

1. **Recognition capability**: When multiple feasible mapping solutions exist, it does not provide any clue on which one is the truly correct layout recognition. Hence, such attack can easily fail on circuits with k-security protection.

2. **Scalability**: SAT is a well-known NP-complete problem. Therefore, a solver finds feasible solution using exponential time in the worst case. Despite the tremendous progress on SAT solving techniques, the fundamental scalability challenge is not solved.

3. **Flexibility**: Even for small circuits, the SAT-based bijective mapping can be effective only when it applies with complete netlist. Even if only a small portion of the layout needs to be recognized, it is applied to the entire circuit.

3.3 Layout Recognition Attack by Structural Pattern Matching

We propose a new attack technique to recognize split manufactured layout in a more scalable manner than the SAT-based bijective mapping. Our technique is inspired by structural pattern matching [56], a technology mapping technique that determines which library cell can implement a subfunction in a given Boolean network. This is similar to the attack scenario described in Section 3.2.1, where one needs to match components in FEOL layout with those in the netlist. However, there are several significant differences between the pattern matching in the attack and that in technology mapping.

- In attacking split manufacturing, the netlist is to match with *incomplete* layout where BEOL information is not available. This is quite different from technology mapping and much more difficult to handle.

- Technology mapping is to match library cells, which are typically small. However, security attack is to match larger subcircuit or entire circuit. Such difference entails different data representation approaches.

- Technology mapping intends to map one library cell with many parts in a circuit design. However, the security attack is to identify a unique matching between netlist and layout. Therefore, a partial matching in a local region is often inconclusive.

Overall, the pattern matching in the layout recognition attack is much more challenging than that in technology mapping. We develop techniques to improve the attack by exploiting hints from design conventions.

3.3.1 Pattern Table

The pattern table is to represent logic and structural relationship in a Boolean network, and plays a similar role as the pattern table in pattern matching for technology mapping [56]. The format of our pattern table is the sparse matrix of the pattern table used in [56]. A pattern table is generated from a circuit netlist according to different gates, for example, $AND2$ gate and $OR2$

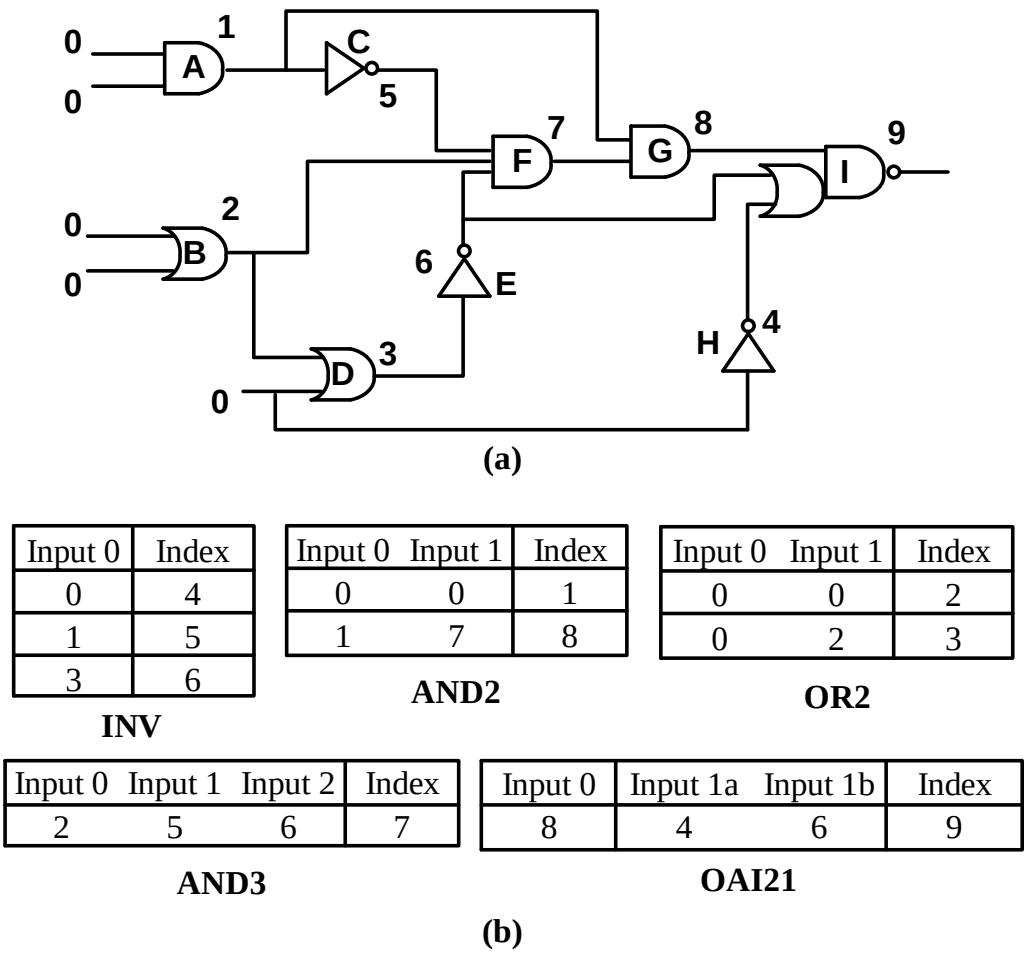


Figure 3.2: Example of the pattern table: (a) circuit netlist and (b) its pattern tables associated with input pins of logic gates and indices.

gate have separated pattern tables. The elements in the pattern table are pattern indices. Each logic gate is assigned with a pattern index. The index of a gate g is uniquely associated with its gate type and indices of its fanin gates. Figure 3.2 shows an example of a circuit netlist and its pattern tables. In this example, gate G is represented by pattern $8(AND2, 1, 7)$ where 8 is its pattern index, and 1 and 7 are indices of its fanin gates. Such information will be stored as the second row of $AND2$ pattern table in Figure 3.2(b). For an $AND2$ gate, the logic function of its two inputs are identical. This is represented by removing the solid lines between the columns of input 0 and input 1. Otherwise, if two inputs are not identical, like input 0 and input 1a of $OAI21$ gate

in Figure 3.2, we used the solid line to separate the columns of input 0 and input 1a. If there is no solid line between two columns, that means the order of input pattern indices does not matter. Given a complete or partial circuit netlist, we generate pattern tables for all the gates through a topological order traversal. The heuristic of generating pattern table for a specific netlist is shown in Algorithm 1.

There is a difference in handling pattern indices between matching an entire circuit and a subcircuit. When matching an entire circuit, we assume that circuit I/O pins have already been matched to put more focus on matching of gates. This is a reasonable assumption, as the foundry can obtain chip I/O information from product specification. In this case, each of corresponding I/O vertices in the netlist has its own unique index. When to match only a subcircuit, we cannot assume the knowledge of those I/O nodes. As such, their indices are uniformly 0 like in technology mapping [56], which are treated as don't care. By doing so, we only match the internal logic structure and function of the subcircuit.

3.3.2 Matching a Subcircuit with FEOL Layout

In this section, we describe how to match a subcircuit with FEOL layout. For an entire circuit represented by a graph G, a subcircuit can be treated as a subgraph $G' \subset G$. By the pattern table generation defined by Section 3.3.1, each $v' \in G'$ should be annotated with a pattern index.

For FEOL layout, a pre-processing is performed to identify logic gates from transistors according to the cell library. Next, we attempt to find pattern index for each gate in the layout. The pattern indices of all layout gates are initialized to 0. Later, if a layout gate is successfully matched with a node in the netlist, its index is set to be the same as that in the netlist. Figure 3.3 shows the layout for the circuit netlist of Figure 3.2, in which FEOL layers contain all the gates and connections in solid lines while BEOL layers include the connections in dashed lines. In this example, the pattern table of A' is $(AND2, 0, 0)$, which matches with the $1(AND2, 0, 0)$ in the netlist. Therefore, layout gate A' is also annotated with $1(AND2, 0, 0)$.

The key challenge here is that the layout information is not complete. In Figure 3.3, the dashed lines indicate BEOL connections, which are invisible to the attacker. For example, one input

Algorithm 1: Generating pattern table

Input : Circuit netlist $G = (V, E)$
Output: Pattern table T

$idx := 0$
for $v \in V$ **do**
  if v *is leaf node* **then**
    $g := \phi(v)$; `// Get cell type of` v
    if *entity* $E(0, \ldots, 0)$ *not found in* $T(g)$ **then**
      idx++
      append $E(0, \ldots, 0) = idx$ to $T(g)$
      assign pattern index idx to v
    end
    else assign pattern index $E(0, \ldots, 0)$ to v
  end
end
while $\exists v \in V$ *not assigned* **do**
  for $v \in V$ **do**
    if *all fanins of* v *are assigned* **then**
      $g := \phi(v)$; `// Get cell type of` v
      foreach *fanin* i *of* v **do** Get pattern index I_i
      if *entity* $E(\ldots, I_i, \ldots)$ *not found in* $T(g)$ **then**
        idx++
        append $E(\ldots, I_i, \ldots) = idx$ to $T(g)$
        assign idx to v
      end
      else assign $E(\ldots, I_i, \ldots)$ to v
    end
  end
end
return T

connection of gate F' is at BEOL. In this case, our attack keeps a set of candidate possibilities and prune them according to the pattern tables and simple logic reasoning. For example, the input of AND3 can only be 2, 5 or 6 according to the pattern table in Figure 3.2. Then, all of them are considered for gate F'. Since pattern 5 has already been connected to F', we can exclude the possibility that it is connected to the middle input pin of F'. Likewise, the pattern index for a layout gate may have multiple candidate possibilities due to the missing connection information. In Figure 3.3, the inverter E' has 3 possible input connections "0/1/3", which lead to three possible

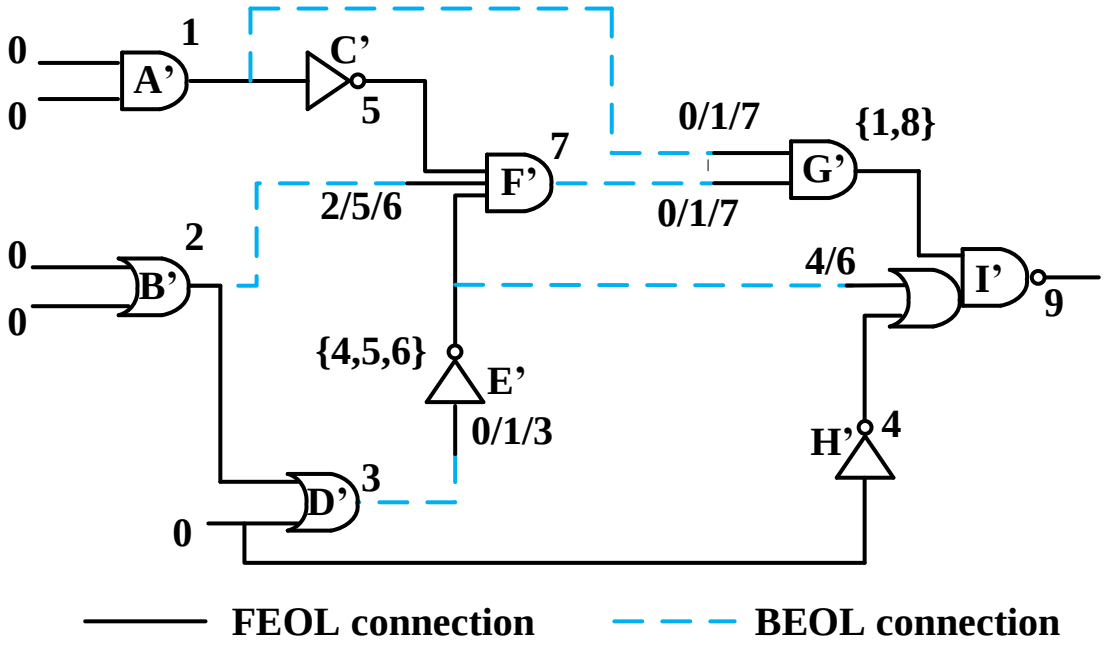


Figure 3.3: Matching cells with FEOL layout by using pattern tables (solid line: FEOL connection, dashed line: BEOL connection).

pattern indices 4, 5 and 6. From the pattern table in Figure 3.2(b), we know 4 cannot be an input to AND3, and therefore we can remove 4. Since pattern 5 has already been input of F' from C', E' cannot be 5, and thus we can identify the pattern index for E' as 6. This identification also tells that the middle input to F' must be from pattern index 2, i.e., gate B'.

3.3.3 Pruning by Hints from Design Conventions

Due to the missing BEOL information, the attack must guess and keep a set of candidate solutions. Even after the simple pruning described in Section 3.3.2, there could still be multiple candidate pattern indices for a layout gate. For example, in Figure 3.3, layout gate G' can be matched with either index 1 or index 8. We use the hints from design conventions as below to perform further pruning, and then the final algorithms for pattern matching, given a layout and pattern table, is described in Algorithm 2.

1. **Load capacitance constraint**: To ensure the signal integrity, a gate in the technology library should honor a limited load capacitance on its fanouts. When we are examining a potential connection in BEOL layers, those that violate the load capacitance constraint can be excluded from being a candidate.

2. **Timing constraint**: Digital circuit is sensitive to setup/hold time constraint. We can obtain an estimate to the actual arrival time and required time on each node of the path through an educated guess on clock period. If a candidate connection violates the timing constraints, it should be excluded from possible BEOL connections in matching cells.

3. **Directionality of the dangling wires**: Although BEOL connections are hidden, the directionality of dangling wires at lower FEOL layers can still suggest the direction for reconnection. For example, if a source gate has a dangling wire pointing toward the south of the layout, those candidates of sink gates located in its north are most likely disregarded.

If there are still multiple candidates after all pruning, we choose the one with minimum BEOL wirelength.

3.3.4 Propagating Candidates along Subcircuit

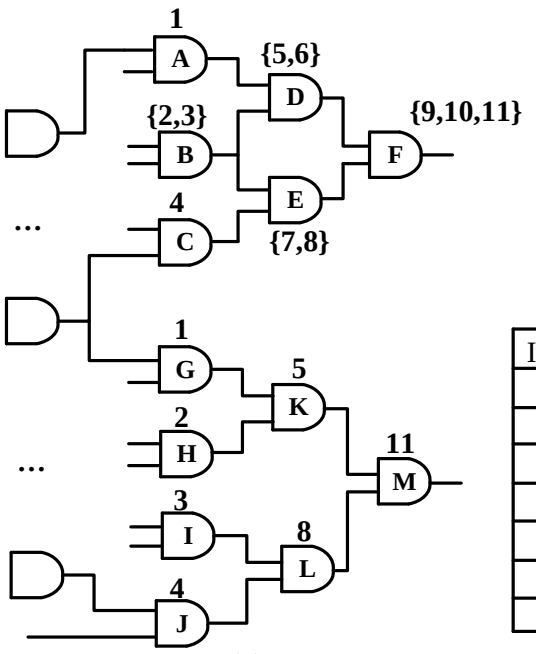
When multiple candidate indices of a layout gate cannot be immediately pruned to a single one, they are propagated toward circuit outputs. When a candidate is combined with a fanout gate, and the combined pattern cannot be found in the netlist pattern tables, this candidate can be pruned out.

Since the topology of a subcircuit is generally a DAG (Directed Acyclic Graph), the candidate propagation on it faces the history consistency issue. Please look at the example in Figure 3.4, which has two candidates 2 and 3 for gate B. When they are propagated to gate D, we have two candidates, 5 that results from combining 1 of gate A with 2, and 6 that is from combining 1 with 3. Likewise, two candidates 7 and 8 are obtained at gate E. When the candidates from D and E are propagated to gate F, there are four combinations. Some of the combinations are illegal. For example, 11 that combines 5 from D and 8 from E is illegal, because this combination implies

Algorithm 2: Matching gates in FEOL layout

Input : FEOL layout $G = (V, E)$, pattern table T
Output: matched candidates in G

$MaxIdx := max(T)$ **foreach** $v \in V$ **do** $option(v) = \{0, \ldots, MaxIdx\}$
while *any updates in pattern index on* V **do**
  Reset all pattern indices on V
  foreach $v \in V$ **do** assign pattern index 0
  for $v \in V$ **do**
    if v *is leaf node* **then**
      $g := \phi(v)$ `// Get cell type of` v
      if *entity* $E(0, \ldots, 0)$ *found in* $T(g)$ **then** assign pattern index $E(0, \ldots, 0)$ to v
    end
  end
  foreach $v \in V$ *that has dangling input pins* **do**
    $g := \phi(v)$ `// Get cell type of` v
    foreach *net* e *connected to dangling input pins* **do**
      foreach *input index* I_i *in* $T(g)$ **do**
        foreach $u \in V$ *that satisfies* $I_i \in option(u)$ *&& output pin is dangle* **do**
          Check the physical constraints for connection between u with v
          if *no violations* **then** assign pattern index I_i to e
        end
      end
    end
  end
  while $\exists v \in V$ *not assigned* **do**
    for $v \in V$ **do**
      if *all fanins of* v *are assigned* **then**
        $g := \phi(v)$ `// Get cell type of` v
        foreach *fanin* i *of* v **do** Get pattern index I_i
        for *entity* $E(\ldots, I_i, \ldots)$ *found in* $T(g)$ **do**
          assign pattern index $E(\ldots, I_i, \ldots)$ on v
        end
      end
    end
  end
  Copy pattern index on V to $option$
end
$List \longleftarrow \varnothing$
for $v \in V$ **do**
  if $MaxIdx$ *is pattern index of* v **then** append v to $List$
end
return $List$



Input 0	Input 1	Index
1	2	5
1	3	6
2	4	7
3	4	8
5	7	9
6	8	10
5	8	11

(b)

Figure 3.4: Example of propagating candidates along subcircuit: (a) the subcircuit from layout and (b) the pattern table from netlist.

that B must be both 2 and 3 simultaneously. One can track the whole propagation history, but such tracking would cost either huge runtime or memory storage. Hence, we choose not to track the history with the risk of keep illegal candidate. However, we can show that the probability of finally keeping the illegal candidate is very low. If the pattern $(AND2, 5, 8)$ does not exist in the netlist pattern table, the combination of 5 and 8 on AND gate can be directly pruned. Otherwise, like in Figure 3.4(a), $(AND2, 5, 8)$ is a legal pattern and exists in the pattern table. Such probability that a legal pattern has the same pattern table as an illegal candidate is very low. This probability can be further reduced by matching tree-only patterns first.

3.3.5 Matching of the Entire Circuit

There are two different ways of recognizing the layout of an entire circuit. One is to build a single pattern table of the entire netlist and use it to match the layout. Its drawback is that the table can be either too huge or very slow to search. Thus, we divide the netlist into subcircuits and

use relatively small pattern tables of these subcircuits to match the layout. The overall program flow is shown in Algorithm 3. This approach has another advantage. Sometimes, there is no need to recognize the entire layout. For instance, an attacker plans to insert a Trojan into encryption circuit. Then, only the encryption part of the layout needs to be recognized.

Given a list of subcircuits with pattern tables, we match each of them one by one on the layout. Once some layout gates are recognized, we use the pattern tables of the fanout of the recognized gates for further matching. If the complete netlist is available, I/O pins can be treated as matched. As such, the matching starts from primary inputs and moves toward primary output in topological order. The matching terminates when no more layout gate can be further recognized.

Algorithm 3: Layout recognition attack flow based on structural pattern matching

Input : Circuit netlist $G = (V, E)$, FEOL layout $G' = V', E')$, fanin cone depth w
Output: mapping of V to V'

$\forall v \in V, map(v) = \varnothing$
while *any updates in* map **do**
  foreach $v \in V$ *that* $map(v) == \varnothing$ **do**
    $H = ExtractFaninCone(G, v, w)$
    $T = GeneratePatternTable(H)$
    $cand = MatchFEOLCell(G', T)$
    Prune matched candidates in $cand$
    if $|cand| == 1$ **then**
      $map(v) = cand$
      Match fanin and fanout of all matched v
    end
  end
  foreach $v \in V$ *that* $map(v) \neq \varnothing$ **do**
    Match the fanin and fanout of v and $map(v)$
  end
  Reconnect nets between matched cells
end
return map

3.4 Experiment Results

3.4.1 Experiment Setup

We evaluate our technique by using ISCAS'85 and ITC'99 benchmark suites. All the designs are synthesized by Synopsys Design Compiler using 45nm technology library. Placement and routing are implemented by Cadence SoC Encounter. The bijective mapping attack is solved by an off-the-shelf SAT solver iSAT3 [57]. The structural pattern matching attack is implemented with C language. The experiments run at Intel Xeon CPU with 2.8GHz frequency and CentOS Linux operating system. We evaluate the effectiveness of the attack methods by identifying the matching ratio that shows the percentage of correctly matched cells. The testcases are all simulated to be split manufactured, and then further prepared in two ways, one without k-security defense and the other with k-security defense [17]. To limit the overhead, k-security is applied to a small portion (about 5 or10 percent) of the circuits with defense.

3.4.2 Experiments on Cases without k-security Defense

Table 3.3 shows the experimental results on cases with k-security defense. Although our structural pattern matching algorithm is a heuristic, it can reach the matching ratio similar to the SAT-based bijective mapping attack and achieve 100% matching ratio in many cases. In only one design (c7552), the structural pattern matching has slightly lower matching ratio because of its heuristic nature. By incorporating the hints from design conventions, the structural pattern matching sometimes (c2670) provides more accurate attack than bijective mapping. The runtime advantage of the structural pattern matching is obvious in larger cases, and it is usually several times faster than bijective mapping. In a few big cases, the bijective mapping could not finish after 48 hours, because the SAT instance contains huge variables and constraints.

3.4.3 Experiments on Cases with k-security Defense

We implemented the benchmarks with a certain portion (5% or 10%) of cells being protected by k-security. Each design has 6 different secured layouts: 5/10% cells in 2-security, 5/10% cells in 3-security and 5/10% cells in 4-security. Both the matching ratio (MR) and runtime of attacks

Table 3.3: Experiment results on cases without k-security defense ("MR" indicates the ratio of correctly matched cell).

Design	#cells	Split layer	SAT bijective mapping		Structural pattern matching	
			MR(%)	Runtime	MR(%)	Runtime
c432	109	M3	100.00	0.9s	100.00	0.2s
c1355	222	M3	100.00	5.1s	100.00	0.9s
c1908	197	M3	100.00	3.2s	100.00	0.4s
c2670	374	M3	95.72	28.7s	97.06	14.2s
c3540	588	M3	100.00	56.9s	100.00	1m37.3s
c5315	819	M4	100.00	4m1.8s	100.00	21.9s
c6288	1889	M3	-	>48h	100.00	10.1s
c7552	834	M3	100.00	2m51.4s	99.76	9.5s
c880	192	M3	100.00	2.5s	100.00	0.4s
b07	258	M3	100.00	10.7s	100.00	5.8s
b11	345	M3	100.00	13.0s	100.00	1.9s
b13	175	M3	100.00	12.9s	100.00	0.7s
b14	2743	M4	100.00	87m9.9s	100.00	2m34.5s
b15	5533	M5	-	>48h	100.00	164m16.4s
b17	17161	M6	-	>48h	98.51	1511m29.9s

are shown in Table 3.4.

Our structural pattern matching attack mostly obtains higher matching ratio than the expected success rate that k-security defense can guarantee. Such results are boldface entries in the table. For example, benchmark c432 with 10% cells defended by 2-security is expected to have about 95% matching ratio under attack. This is because cells with k-security protection can be successfully recognized with the probability $1/k$. The experimental result shows that the theoretical guarantee of k-security cannot always be realized in practice, especially under our structural pattern matching attack, which significantly outperforms the SAT-based bijective mapping attack. On the other hand, the runtime advantage of structural pattern matching attack becomes less obvious compared with cases without k-security defense. Some cases show more runtime than the SAT-based bijective mapping attack, because we divide the netlist into subcircuits and use them to match the layout individually.

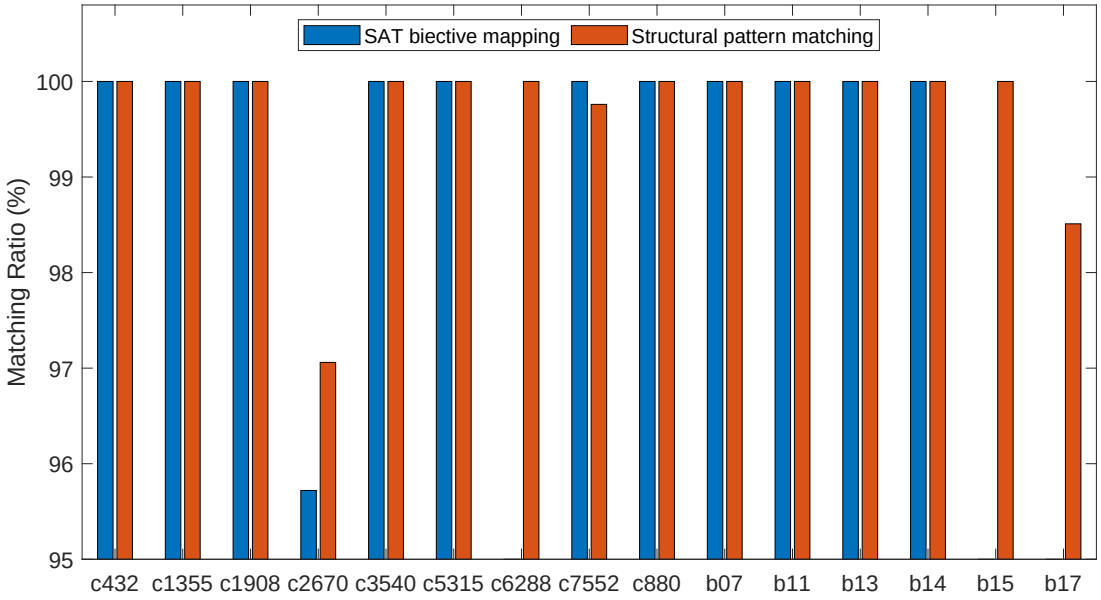


Figure 3.5: Experiment results of cases without k-security defense.

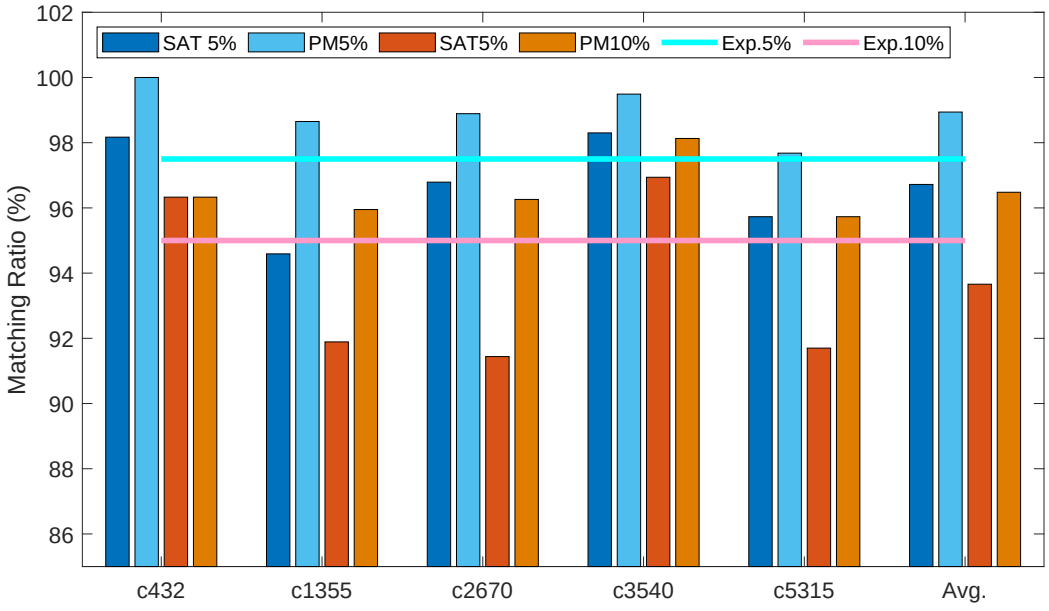


Figure 3.6: Experiment results of cases with 2-security defense.

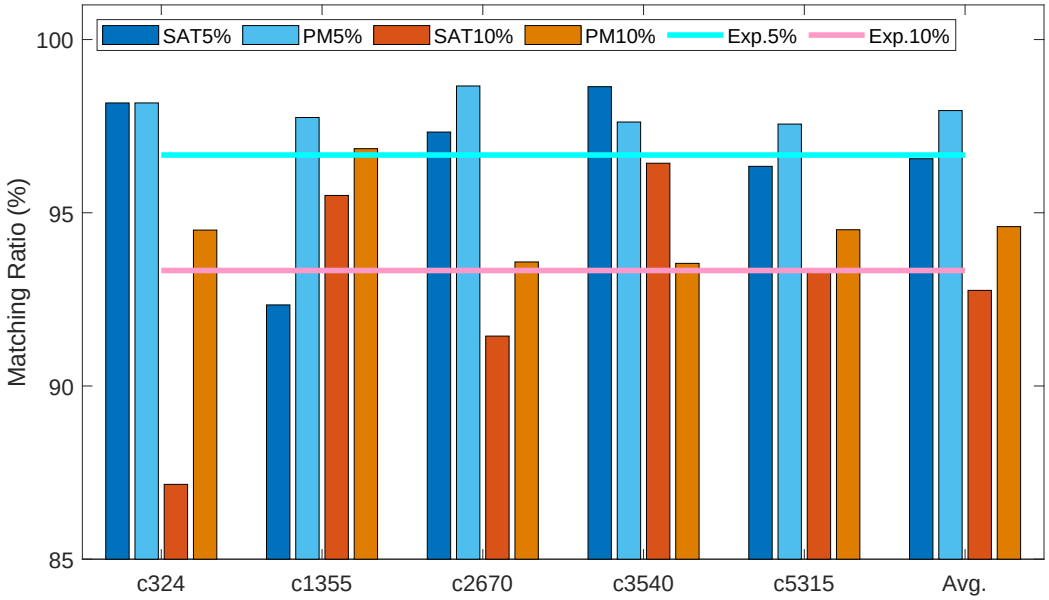


Figure 3.7: Experiment results of cases with 3-security defense.

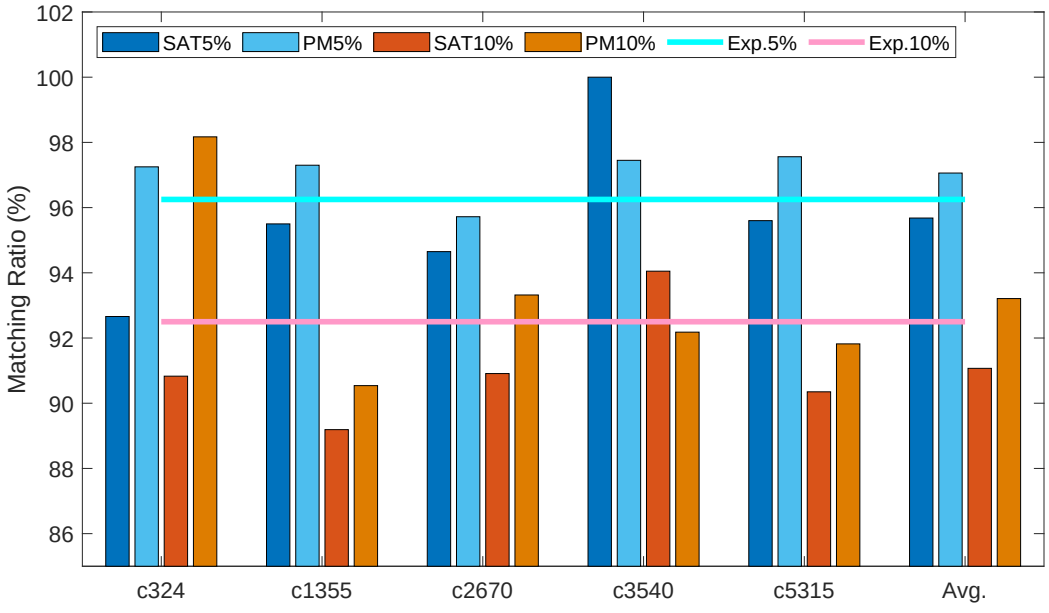


Figure 3.8: Experiment results of cases with 4-security defense.

3.5 Discussions

3.5.1 Comparison with Other Attack Methods

In this work, we assume that the attacker targets to identify cells in the FEOL layouts with the existence of complete netlist. In an orthogonal work [19], network-flow attack focus on restoring the BEOL connections without knowledge of functionality/netlist of the target design. Both structural pattern matching and SAT-based bijective mapping use the structural information for the attack. In addition, structural pattern matching attack utilizes hints from design conventions exposed by FEOL layouts to reduce the solution space, which is also used in the network-flow attack. In the flow of our method, partial BEOL connections are recovered based on the matched cells, which implements similar effect of the network-flow attack.

3.5.2 Extension to Attacks with Incomplete Netlist

Due to the nature of structural pattern matching, our attack flow can be easily extended to another attack scenario where the attacker only has partial netlist. In such scenario, SAT-based bijective mapping should be refined because there is only an injective mapping from the cells in netlist to those in FEOL layout. Even when only a small portion of the netlist is provided, the structural pattern matching attack can be easily extended to recognize the layout by generating pattern table based on the incomplete netlist.

3.6 Conclusion

In this chapter, we consider the scenario where attackers have netlist information and attempt to recognize split manufactured layout for Trojan insertion, which was not fully discussed in existing works. We develop a new attack technique based on structural pattern matching to address the drawbacks of earlier mentioned SAT-based bijective mapping attack. Experiment results show that it is more efficient and more scalable than the bijective mapping attack. Unlike the theoretical metric of k-security, our proposed attack method provides an alternative way to evaluate the security of split manufactured ICs for Trojan insertion in practice.

Table 3.4: Experiment results on cases with k-security defense ("MR" indicates the ratio of correctly matched cell). Number in bold indicates higher ratio than the expected success rate that k-security can guarantee.

Design	Defense type	#cell secured	SAT bijective mapping		Structural pattern matching	
			MR(%)	Runtime	MR(%)	Runtime
c432	2-secure	5%	**98.17**	0.9s	**100.00**	0.3s
		10%	**96.33**	3.1s	**96.33**	0.6s
	3-secure	5%	**98.17**	1.3s	**98.17**	1.8s
		10%	87.16	1.0s	**94.50**	5.9s
	4-secure	5%	92.66	0.9s	**97.25**	0.8s
		10%	90.83	1.2s	**98.17**	1.3s
c1355	2-secure	5%	94.59	6.6s	**98.65**	1.0s
		10%	91.89	5.9s	**95.95**	3.0s
	3-secure	5%	92.34	5.6s	**97.75**	1.8s
		10%	**95.50**	5.2s	**96.85**	2.1s
	4-secure	5%	95.50	5.5s	**97.30**	5.8s
		10%	89.19	5.4s	90.54	19.0s
c2670	2-secure	5%	96.79	37.1s	**98.89**	8.0s
		10%	91.44	36.8s	**96.26**	16.5s
	3-secure	5%	**97.33**	32.1s	**98.66**	10.6s
		10%	91.44	33.5s	**93.58**	21.5s
	4-secure	5%	94.65	37.7s	95.72	10.2s
		10%	90.91	36.2s	**93.32**	47.0s
c3540	2-secure	5%	**98.30**	58.1s	**99.49**	2m16.9s
		10%	**96.94**	55.2s	**98.13**	11m29.3s
	3-secure	5%	**98.64**	55.9s	**97.62**	58.6s
		10%	**96.43**	57.8s	**93.54**	9m47.6s
	4-secure	5%	**100.00**	1m10.2s	**97.45**	1m25.8s
		10%	**94.05**	1m0.2s	92.18	17m12.2s
c5315	2-secure	5%	95.73	4m3.8s	**97.68**	1m11.2s
		10%	91.70	3m48.6s	**95.73**	3m28.8s
	3-secure	5%	96.34	3m59.5s	**97.56**	54.2s
		10%	93.28	3m51.4s	**94.51**	2m18.5s
	4-secure	5%	95.60	3m44.5s	**97.56**	1m3.4s
		10%	90.35	4m19.1s	91.82	4m0.2s
Average	2-secure	5%	96.72	1m9.3s	**98.94**	43.5s
		10%	93.66	1m5.9s	**96.48**	3m3.64s
	3-secure	5%	96.56	1m6.9s	**97.95**	25.4s
		10%	92.76	1m5.8s	**94.60**	2m31.1s
	4-secure	5%	95.68	1m7.8s	**97.06**	33.2s
		10%	91.07	1m12.4s	**93.21**	4m27.9s

4. HIERARCHICAL CONSTRAINT-DRIVEN ANALOG LAYOUT AUTOMATION

4.1 Introduction

Analog/mixed-signal ICs today is more essential than ever before. The analog/mixed-signal modules become important components in many modern SoC design applications such as mobile electronics, wireless communications, consumer electronics, energy infrastructure, and automotive systems. It has been reported that the annual growth rate of analog/mixed-signal ICs in the past five years has become 8.9% faster than all other major IC product categories [58]. Such increasing market demands of analog ICs require new design methodology to accelerate the design cycle and reduce design efforts.

Modern emerging SoC designs highly rely on EDA tools. Such kind of automation not only efficiently enhance the scalability of large-scale designs which is almost impossible to be handled by manual work, but also can significantly improve the productivity of circuit engineers. Although analog ICs have attracted more attention in modern SoC systems, the development of design automation tools that assist layout synthesis of analog/mixed-signal circuits are still far behind their digital counterparts. The difficulties in automating analog designs come from the fact that analog ICs, especially analog design layouts, involve a large amount of expert knowledge which requires sophisticated constraints and manual layouts.

The layout synthesis automation of analog designs can be classified into two categories, analog layout migration and analog layout generation [59, 60]. Analog layout migration flows first extract the layout templates from existing designs, then migrate the design knowledge to target designs in a new process. By contrast, analog layout generation flows synthesize the layouts without any existing analog layouts, but they require some design constraints that the design layouts should obey. The advantages of analog layout generation flows are obvious. 1) The layout can be generated without any dependency on existing manual layouts. 2) Analog layout migration flows can only migrate the existing design, whose schematic topology is the same as the target design,

to layouts in a new process. In case that new design schematic is provided or new process requires different knowledge to ensure performance, analog layout migration flows would lose their efficiency. However, how to prepare proper design constraints and automatically generate regular layout which satisfies those constraints is a huge challenge. Our proposed hierarchical constraint-driven analog layout methodology is one of the layout generation flows based on primitive devices and electrical/geometrical constraints.

Analog circuits are very sensitive to parasitic, crosstalk and power supply noise. Various effects such as process variation or thermal effect can degrade the analog signals to the point where system fails or errors are introduced. Improper analog layouts might further degrade the circuit performance because the mismatch can be amplified by irregular geometric structures. Sufficient design constraints will help physical synthesis generate qualified layouts which minimize those harmful effects. For example, the differential pairs, one of common components in differential signal processing circuits, require that two complementary signals should be propagated along electrically-equivalent routes. Although it is impossible to propagate complementary signals on the same route, the layouts with perfect symmetric routes can minimize the parasitic mismatches. Another example is that the thermal radiating module or power source will induce thermal gradients on the chip. Those thermally-sensitive matched devices need to be placed on the same isotherm to reduce the thermal mismatch between them.

Different kinds of design constraints for analog circuits are summarized as below.

- Symmetry: As mentioned above, symmetry is an important constraint that can guarantee matched devices on layouts. It can reduce both the effect of parasitic mismatches and process variations. A symmetry constraint in placement requires a set of devices and/or device pairs to be placed in symmetry with respect to an axis. In routing, a symmetry constraint corresponds to the matched nets which are symmetrically routed. Not only the coordinates of routing paths but also the wire length, metal layers, via number should be exactly matched.

- Common centroid: A common centroid constraint is used to minimize systematic and random mismatch among a set of compactly connected device units. For example, capacitors in

analog layout are built from small capacitor units in a two dimensional array with a common centroid. Then the two dimensional array has the properties of coincidence, symmetry, dispersion and compactness. For routing, the matched nets connecting the devices in a common centroid group should be routed with respect to the common centroid of the group.

- Proximity: To restrict the distance/spacing between multiple devices, a proximity constraint is generally required in placement. Such constraints can improve the matching quality of matched devices and reduce the wirelength of the critical nets.

- Boundary: By restricting devices to the boundary of a specified proximity group, a boundary constraint can help reduce the wire length of critical nets on those paths across the hierarchical boundary. Hence, the parasitic of critical nets on chip level can be reduced and better performance can be achieved.

- Thermal: With a thermal constraint, the thermal radiating devices such as power source should be placed along a symmetry axis evenly dividing the layout, so that the isothermal lines are symmetric across the symmetry axis. At the same time, those thermal-sensitive matched devices should be placed symmetrically about the axis to guarantee the same distance from the radiating devices. Then the thermal mismatches between matched devices can be reduced.

- Route matching: A route matching constraint is used to ensure the matching of a set of routing paths but is less restrict than any symmetry or common centroid constraint. It requires equivalent segment numbers for all paths, equivalent lengths of wire segments and identical layer assignments of wire segments.

- Shielding: In analog/mixed-signal circuits, some modules/interconnections such as inputs to high gain stages and charge storage nodes in sampled data circuits are sensitive to noise, while other modules/interconnections such as outputs of high gain stages and clocks in sampled data circuits can induce lots of noise. A shielding constraint aims to insert shielding,

generally some nets connecting to power/ground, around wire segments of sensitive nets to isolate the coupling noise.

- Multi-track routing: The critical nets with multi-track routing constraint can be routed with multiple parallel tracks. As a result, the interconnect current densities and parasitic can be reduced to achieve better circuit performance.

To address required design constraints for analog/mixed-signal layouts, we present our hierarchical layout synthesis methodology including constraint-driven placement and routing. The proposed layout synthesis flow works on primitive analog modules such as differential pairs, capacitors or resistors instead of tiny transistor units. It means we assume that the layouts of devices or building blocks for placement and routing have been created. To support complicated analog circuits with multiple design hierarchies, we develop a hierarchical framework which is implemented in the bottom-up manner. Moreover, a two-stage placement methodology is presented to handle the placement of designs with mixed-size blocks.

4.2 Related Work

Similar to standard physical synthesis of digital circuits, the analog layout generation flow includes placement and routing with corresponding placement and routing constraints. A variety of works have been presented in the field of analog placement [23, 24, 25, 27, 61, 62, 63] and analog routing [28, 29, 32, 64]. The analog placement tries to determine the physical position of devices with specified placement constraints and layout requirements. At the same time, the layout area and estimated wirelength should be minimized. The analog routing completes the physical interconnections by assigning actual metal tracks according to routing constraints. Both the placement and routing take into consideration the design constraints to achieve better circuit performance, while the physical resource of the layouts can be saved.

4.2.1 Analog Placement

Generally the constraint-driven placement for analog circuits includes two types of methodologies. One is based on simulated annealing which searches feasible placement solutions and eval-

uates corresponding performance and satisfaction of design constraints by cost function/penalty. Another type, which has attracted more research interests recently, is to use analytical placement algorithm. This method has been successfully demonstrated in placing digital circuits. Different from simulated annealing based placement, such technique uses the non-linear conjugate gradient algorithm to achieve the feasible solutions. Then the global placement results will be further refined by legalization and detailed placement.

The selection of topological representation is important to simulated annealing based placement. The representation should be efficient and flexible in both representing the relative position of building blocks and exploring feasible solutions with various design constraints. Different topological representation such as sequence pair [23, 25, 27], O-tree [24, 65], B*-tree [66, 67, 68, 69], and transitive closure graph(TCG) [70, 71] are studied for non-slicing floorplans in placement. By using efficient topological representation and simultaneously taking multiple design constraints into consideration, the simulate annealing based placement explores the search space with different solutions.

A B*-tree based simulated annealing scheme [72] is introduced to optimize non-slicing floorplans. For each node in the ordered binary tee, its adjacent blocks on the right-hand side are stored in the right subtree, while the left subtree represents other adjacent blocks located above. It's further extended to support multiple placement constraints [66, 67, 68, 69]. The placement method based on sequence pair and corresponding constraint graphs handles particular kinds of geometric constraints simultaneously, including preplace constraint, symmetry constraint, boundary constraint, abutment constraint, and alignment constraint[23, 24]. It is improved to support common centroid constraint by using centre-based corner block list[25, 27]. Abhishek et al. develop a placement algorithm which satisfies the current flow and symmetry constraints by using Parallel Current Path-Sequence Pair[61].

Recently, the analytical placement scheme, which has gained great success in digital placement, are introduced to the analog layout automation[62]. The placement constraints are formulated into penalty terms in the objective of non-linear conjugate gradient algorithm. In this work, the place-

ment can be further improved by allowing overlaps between mutually exclusive layers. Another work presents a hierarchical and analytical placement for high performance analog circuits[63]. It allows different placement variants in the bottom-up flow, which expand the exploration of search space to achieve better placement results.

4.2.2 Analog Routing

To generate regular routing with better matching of parasitic, the grid-based routing scheme becomes a suitable choice. The analog routing searches feasible paths by using Dijkstra or A* search algorithm which is common in digital routing. In addition to minimizing wire length, number of bends and number of vias, specific routing constraints such as symmetry or common centroid net constraint, route matching constraint, shielding constraints will be satisfied in the final routing results.

Practical routing methods address the symmetry net in an straightforward way [64]. For each pair of symmetry nets, one net will be routed first and the resulting route will be mirrored around the symmetry axis to complete the whole pair. To guarantee no overlaps or design rule check (DRC) violations after mirroring, all the obstacles on both sides of the symmetry axis should be addressed when the first route is produced. Other works implement common centroid routing by assigning symmetrical wire tracks among common centroid array and completing the devices to their respective adjacent wire tracks [73, 74].

Some works take considerations of route matching constraint by using pattern-based routing method [28, 29] or exactly matched maze routing algorithm [75, 76]. Various routing methods have been studied to determine the wire width according to the current density requirement of nets [30, 31]. A*-search algorithm is used to bind the shielding metals to the sensitive nets to obey shielding constraints [77]. A constructive Integer Linear Programming (ILP) based routing [32] is proposed to balance the performance and time consumption. This approach first generates multiple routing candidates for each net, then annotates them with different kinds of routing constraints. The optimal solution with one candidate for each net will be selected by ILP.

4.3 Constraint-driven Placement

The placement method we select for analog layout generation uses sequence pair as topological representation. The corresponding constraint graphs are used to represent the relative locations of devices in floorplans. The placement constraints can be simultaneously handled by adding new vertices and weighted edges in the constraint graphs. The framework of simulated annealing is used to determine the optimal solution by searching a certain amount of feasible solutions. This methodology was first presented in some previous works [23, 24, 27].

4.3.1 Topological Representation

4.3.1.1 Sequence Pair

Sequence pair is one of the most efficient topological representation for non-slicing floorplans[78]. We use sequence pair to present different placement solutions in our implementation. Any rectangle packing containing a set of blocks can be represented as an ordered pair of block sequence (positive sequence and negative sequence). For example, the sequence pair $\Gamma = \{\Gamma_+, \Gamma_-\} = \{abcd, adbc\}$ represents the packing of the block set $\{a, b, c, d\}$. If block x is in the i-th position of Γ_+, we denote $\Gamma_+^{-1}(x) = i$. The relative positions between two blocks are implied by their orders in both positive and negative sequence.

- If $\Gamma = \{\ldots a \ldots b \ldots, \ldots a \ldots b \ldots\}$, then block a is on the left of block b.
- If $\Gamma = \{\ldots a \ldots b \ldots, \ldots b \ldots a \ldots\}$, then block a is above block b.

Any changes in the sequence pair can result in a new packing that has a different floorplan. Hence, simulated annealing algorithm can easily explore the search space by disturbing the sequence pair.

4.3.1.2 Constraint Graph

To obtain a compact placement from the floorplan imposed by a sequence pair, we use constraint graphs to present the horizontal and vertical relationships between blocks. Then, the exact positions of blocks can be calculated based on the constraint graphs. One important advantage of

using constraint graphs is its ability to deal with different kind of geometric placement constraints (see Section 4.3.2).

To represent the two-dimensional relationships in the floorplan, we need two constraint graphs, horizontal constraint graph G_h in horizontal direction and vertical constraint graph G_v in vertical direction. A horizontal/vertical constraint graph is a directed graph where the vertices represent the blocks and the weighted edges represent the horizontal/vertical relationship between blocks. When block a is on the left of block b, we have an edge from vertex a to vertex b in the horizontal constraint graph. The weight of the edge corresponds to the minimum distance from the left edge of a to the left edge of b. Similarly, when block c is below block d, we have an edge between them in the vertical graph with the weight corresponding to the minimum distance from the lower edge of c to the lower edge of d.

We add two dummy vertices in the constraint graphs, one as source vertex and the other as sink vertex. An edge from source vertex to vertex i indicates that there are no blocks on the left of block i or no blocks below block i. Similarly, if there are no blocks on the right of block i or no blocks above block a, there should be an edge from vertex i to sink vertex. According to the constraint graphs, we can calculate the location of each block in the compact packing. The x coordinate of the lower-left corner of a block i is the length of the longest path from source vertex to vertex i in horizontal constraint graph, while the y coordinate is the length of the longest path in vertical constraint graph. We can also calculate the width/height of the entire packing according to the coordinates of the sink vertex in horizontal/vertical constraint graph.

Given a specific sequence pair Γ, we can build the constraint graphs by the following rules.

- Add an edge from vertex a to vertex b in horizontal constraint graph if $\Gamma_+ = \ldots a \ldots b \ldots$ and $\Gamma_- = \ldots a \ldots b \ldots$.

- Add an edge from source vertex to vertex a in horizontal constraint graph if there are no blocks on the left of block a.

- Add an edge from vertex a to sink vertex in horizontal constraint graph if there are no blocks

on the right of block a.

- Add an edge from vertex a to vertex b in vertical constraint graph if $\Gamma_+ = \ldots a \ldots b \ldots$ and $\Gamma_- = \ldots b \ldots a \ldots$.

- Add an edge from source vertex to vertex a in vertical constraint graph if there are no blocks below block a.

- Add an edge from vertex a to sink vertex in vertical constraint graph if there are no blocks above block a.

4.3.2 Handling of Placement Constraints

To make a feasible placement satisfying required design constraints, we first identify those infeasible solutions by performing an initial check on the candidate sequence pairs. The initial check is necessary but cannot guarantee that all the qualified sequence pairs will result in feasible solutions. Then, we handle different kinds of placement constraints by adding new vertices and/or edges in the constraint graphs according to their topological relationships.

As we will discuss in this section, new edge pairs will be added in constraint graphs to address the required placement constraints. These newly added edges between vertices can be categorized into two sets, forward edges and backward edges. The forward edges are used to calculate the longest path for the block locations, which do not conflict with edges in the original constraint graph. The backward edges are used to calculate the constraint penalty in cost function (see Section 4.3.3.4), which have opposite direction of forward edges.

4.3.2.1 Symmetry Block Constraint

The initial check for symmetry block constraint is a sufficient symmetry-feasible condition in sequence pair [22, 79]. Given the sequence pair $\Gamma = \{\Gamma_+, \Gamma_-\}$ of a placement containing a symmetry block group γ that consists of several symmetric block pairs and self-symmetric blocks

with respect to a common vertical axis, the sequence pair is symmetric-feasible if

$$\Gamma_+^{-1}(x) < \Gamma_+^{-1}(y) \Longleftrightarrow \Gamma_-^{-1}(sym(y)) < \Gamma_-^{-1}(sym(x)), \forall x, y \in \gamma, x \neq y \tag{4.1}$$

where $sym(x)$ is the symmetry block of x. If block x is a self-symmetric block in group γ, $sym(x)$ is x itself. According to this condition, any two blocks a and b belonging to symmetry group γ in Γ_+ must have different order from their symmetric counterparts in Γ_-. Similarly, a sufficient symmetry-feasible condition for a symmetry group γ' with a common horizontal axis should be

$$\Gamma_+^{-1}(x) < \Gamma_+^{-1}(y) \Longleftrightarrow \Gamma_-^{-1}(sym(x)) < \Gamma_-^{-1}(sym(y)), \forall x, y \in \gamma', x \neq y. \tag{4.2}$$

According to this condition, any two blocks a and b belonging to symmetry group γ' in Γ_+ must have the same order as their symmetric counterparts in Γ_-.

After checking the symmetry-feasible condition, we will augment the constraint graphs to enforce the symmetry block constraint. For each symmetry group G_i, there can be r_i self-symmetric blocks $S_1, S_2, \ldots, S_{r_i}$ and s_i symmetry pairs $(A_1, B_1), (A_2, B_2), \ldots, (A_{s_i}, B_{s_i})$. To show how the constraint graphs are augmented according to the symmetry constraint, we assume that G_i has a common vertical axis. First, we should augment the vertical constraint graph to force each symmetry pair aligned horizontally. Since each symmetry pair (A_j, B_j) where $j = 1, 2, \ldots, s_i$ should contain two identical blocks, a pair of edges, $e(A_j, B_j)$ and $e(B_j, A_j)$ of weight 0 will be inserted in the vertical constraint graph.

To represent the symmetry axis of G_i, we insert a dummy node d_i in the horizontal constraint graph. For each self-symmetric block S_j where $j = 1, 2, \ldots, r_i$, a pair of edges $e(S_j, d_i)$ and $e(d_i, S_j)$ with weight of $w(S_j)/2$ and $-w(S_j)/2$ respectively will be added to ensure that S_j is placed symmetrically on the axis ($w(S_j)$ is the width of block S_j). In order to ensure that each symmetry pair (A_j, B_j) can be placed symmetrically around the axis, four edges, $e(A_j, d_i)$, $e(d_i, A_j)$, $e(d_i, B_j)$ and $e(B_j, d_i)$, are added with weights of x_{ij}, $-x_{ij}$, $x_{ij} - w(B_j)$ and $w(B_j) - x_{ij}$ respectively. Here we assume that at least block A_j is not on the right of block B_j according to

the sequence pair. x_{ij} which should be greater than $w(B_j)$ is the parameter we should adjust. Figure 4.1 shows an example of a simple symmetry group and its corresponding constraint graphs.

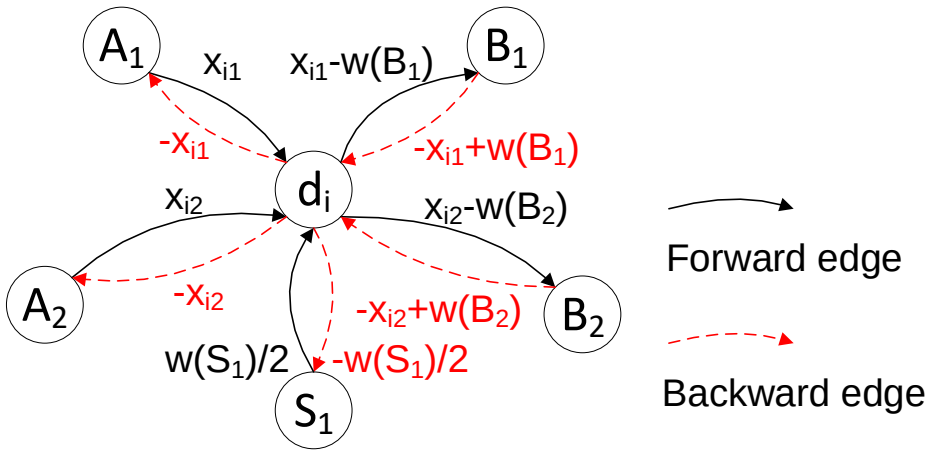


Figure 4.1: Example of constraint graph with symmetry constraint.

After adding new dummy nodes and weighted edges for all symmetry groups, we will determine the value of x_{ij} so that no positive cycle exists and $max(x_{ij})$ is minimized. We use the heuristic in previous work [24, 27] which dynamically adjusts the value of variables according to possible slacks. After determining the edge weights of each symmetry group individually, we will check if any positive cycle exists in the constraint graphs. If any positive cycle exists, we will claim that no feasible placement satisfying all symmetry constraints can be obtained according to the given topological representation.

4.3.2.2 Alignment Constraint

To guarantee the sequence pair satisfies horizontal alignment constraints, at least those blocks within constraints should not be placed vertically in the initial floorplan. Hence, the sufficient feasible condition for horizontal alignment should be

$$\Gamma_+^{-1}(x) < \Gamma_+^{-1}(y) \Longleftrightarrow \Gamma_-^{-1}(x) < \Gamma_-^{-1}(y) \tag{4.3}$$

where x and y are two blocks in the horizontal alignment constraint. Similarly, the sufficient feasible condition for vertical alignment can be

$$\Gamma_+^{-1}(x) < \Gamma_+^{-1}(y) \Longleftrightarrow \Gamma_-^{-1}(y) < \Gamma_-^{-1}(x). \tag{4.4}$$

The constraint graphs should be augmented by adding new edges to reflect the alignment constraint. For horizontal(vertical) alignment, a pair of edges, $e(x, y)$ and $e(y, x)$ with weight of 0 must be inserted in the vertical(horizontal) constraint graph.

4.3.2.3 Abutment Constraint

The sufficient feasible condition for abutment constraint depends on the required relationship of abutment between blocks.

- Case 1: If block x should be abut with block y horizontally and on the left of y, the feasible sequence pair must be $\Gamma = \{\ldots x \ldots y \ldots, \ldots x \ldots y \ldots\}$.
- Case 2: If block x should be abut with block y horizontally and on the right of y, the feasible sequence pair must be $\Gamma = \{\ldots y \ldots x \ldots, \ldots y \ldots x \ldots\}$.
- Case 3: If block x should be abut with block y vertically and above y, the feasible sequence pair must be $\Gamma = \{\ldots x \ldots y \ldots, \ldots y \ldots x \ldots\}$.
- Case 4: If block x should be abut with block y vertically and below y, the feasible sequence pair must be $\Gamma = \{\ldots y \ldots x \ldots, \ldots x \ldots y \ldots\}$.

The constraint graphs can be modified according to different scenarios below. Note that we use $w(x)$ and $h(x)$ to denote the width and height of block x respectively.

- Case 1: A pair of edges, $e(x, y)$ and $e(y, x)$ with weights of $w(x)$ and $-w(x)$ will be added to the horizontal constraint graph.
- Case 2: A pair of edges, $e(y, x)$ and $e(x, y)$ with weights of $w(y)$ and $-w(y)$ will be added to the horizontal constraint graph.

- Case 3: A pair of edges, $e(y, x)$ and $e(x, y)$ with weights of $h(y)$ and $-h(y)$ will be added to the vertical constraint graph.

- Case 4: A pair of edges, $e(x, y)$ and $e(y, x)$ with weights of $h(x)$ and $-h(x)$ will be added to the vertical constraint graph.

4.3.2.4 Boundary Constraint

The sufficient feasible condition for boundary constraint is determined by the required relationship between the block and boundary.

- Case 1: If block x is required to abut with the left boundary, there should be no block that can be before x in both Γ_+ and Γ_-.

- Case 2: If block x is required to abut with the right boundary, there should be no block that can be after x in both Γ_+ and Γ_-.

- Case 3: If block x is required to abut with the bottom boundary, there should be no block that can be before x in Γ_+ and after bx in Γ_-.

- Case 4: If block x is required to abut with the top boundary, there should be no block that can be after x in Γ_+ and before bx in Γ_-.

Similar to the abutment constraint, a pair of edges between two vertices should be inserted in constraint graphs. The only difference is that we use source and sink vertex to represent the left/bottom and right/top boundary in horizontal/vertical constraint graph respectively. The modification in the constraint graphs follows the rules below. Note that s denotes the source vertex and t denotes the sink vertex in the graph.

- Case 1: A pair of edges, $e(s, x)$ and $e(x, s)$ with weight of 0 will be added in horizontal constraint graph.

- Case 2: A pair of edges, $e(x, t)$ and $e(t, x)$ with weights of $w(x)$ and $-w(x)$ will be added in horizontal constraint graph.

- Case 3: A pair of edges, $e(s, x)$ and $e(x, s)$ with weight of 0 will be added in vertical constraint graph.

- Case 4: A pair of edges, $e(x, t)$ and $e(t, x)$ with weights of $h(x)$ and $-h(x)$ will be added in vertical constraint graph.

4.3.2.5 Proximity Constraint

It's easy to handle the proximity constraint in our method. Neither an initial check nor any modification on constraint graphs is required. We only need to add a term corresponding to the distances between blocks in such constraint in the cost function. More details are discussed in Section 4.3.3.4.

4.3.3 Simulated Annealing Algorithm

By successfully generating constraint graphs from sequence pair and placement constraints, we can obtain feasible placement solutions. Then the simulated annealing algorithm will be used to evaluate every feasible solution and select the optimal one with minimum cost. Moreover, in order to achieve better performance in runtime, we introduce parallel computing to accelerate the exploration of feasible solutions in large and complicated cases.

4.3.3.1 Overall Flow

We use simulated annealing as our search engine to find the optimal solution. The overall flow is shown in Figure 4.2. Simulated annealing starts from a possible solution and makes the decision to move to its neighboring solution based on the acceptance probability. It stops when the solution is good enough to accept or certain amount of iterations has been tried. In our algorithm, each iteration will start from a possible sequence pair that might be an initial solution (Section 4.3.3.3) or generated from the previous solution by perturbations (Section 4.3.3.2). Then the initial check will be performed to identify infeasible sequence pair that cannot satisfy specified constraints. The construction of constraint graphs according to placement constraints will follow the initial check if no violation can be found, otherwise another iteration will start. After checking positive

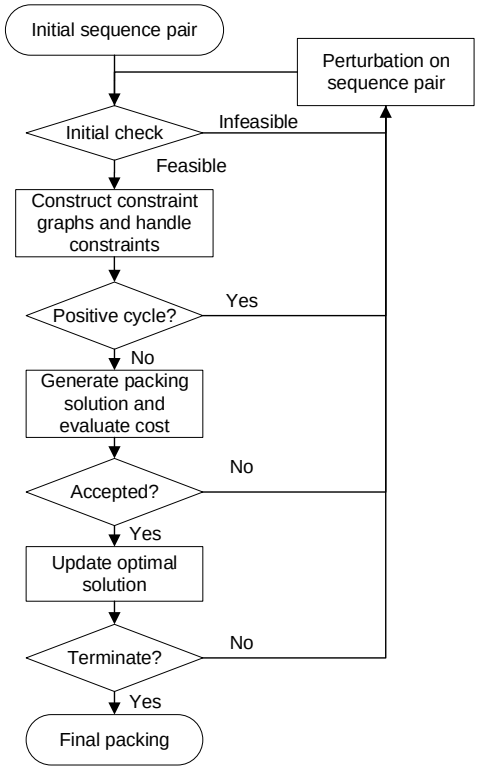


Figure 4.2: Overview of simulated annealing algorithm.

cycles in constraint graphs, the resulting packing solution will be evaluated by the cost function (Section 4.3.3.4) and simulated annealing process will determine whether it is acceptable. Once the new solution is accepted, another round of iteration will start if the termination condition is not satisfied.

4.3.3.2 Perturbations

The perturbation will generate the neighboring sequence pair based on the existing solution. Such perturbation should introduce minimal alterations of the last solution and also provide sufficient variations in searching directions. We randomly choose the perturbation from the following set of moves in the simulated annealing process.

- Moving an asymmetric block: The position of an asymmetric block can be changed without any impact on the feasibility of symmetry constraints. A random asymmetric block can be

moved in the positive sequence Γ_+ or the negative sequence Γ_- or even both of them.

- Rotating an asymmetric block: An asymmetric block will be randomly chosen and its orientation will be changed to a new one.

- Swapping two symmetry groups: Two symmetry groups will be randomly selected and all the blocks of the two groups will be swapped. To ensure that the resulting sequence pair is symmetry-feasible, the relative ordering of blocks in each group will be retained. For example, two symmetry groups $G_1 = \{a_1, a_2, a_3\}$ and $G_2 = \{b_1, b_2\}$ occupy the positions $\{2, 3, 6\}$ and $\{1, 5\}$ in the sequence respectively. After the swapping, G_2 will occupy the positions $\{2, 3\}$ and G_1 will occupy the positions $\{1, 5, 6\}$.

- Swapping two blocks in the same symmetry group: Two blocks that are not symmetric to each other in the symmetry group will be randomly selected. First we swap them in the sequence Γ_+. Then their symmetric counterpart (the other block of the symmetry pair or the self-symmetric block itself) will be swapped in the sequence Γ_-.

- Rotating a symmetry group: In this move, the orientation of one symmetry group randomly selected will be changed. For example, if the symmetry group has a common horizontal axis, after this move the common axis will become vertical. Only the order of blocks in negative sequence Γ_- needs to be reversed.

- Rotating a self-symmetric block in the symmetry group: a self-symmetric block in a symmetry group (if it has) will be randomly selected and its orientation will be changed.

- Rotating a symmetry pair in the symmetry group: a symmetry pair in a symmetry group will be randomly selected. Unlike the self-symmetric blocks or asymmetry blocks, the rotation of the symmetry pair should be simultaneously changed according to the symmetry style. For example, if the symmetry pair is mirror-symmetric with respect to a vertical axis, one block is configured with orientation of north, then its symmetry counterpart should has the orientation of flip-north.

4.3.3.3 Initial Solution

In order to generate an initial solution which can satisfy the symmetry constraints, the initial sequence pair should be constructed according to the sufficient condition in Section 4.3.2.1. Then other asymmetric blocks can be appended to the end of the sequence pair. For example, given a symmetry group G with a vertical common axis which includes r self-symmetric blocks $S_1, S_2, \ldots, S_r$ and s symmetry pairs $(A_1, B_1), (A_2, B_2), \ldots, (A_s, B_s)$ and other asymmetric blocks $C_1, C_2, \ldots, C_w$, the initial sequence pair can be

$$\Gamma_+ = \{A_1, \ldots, A_s, X, S_1, \ldots, S_r, B_s, \ldots, B_1, C_1, \ldots, C_w\} \tag{4.5}$$

$$\Gamma_- = \{A_1, \ldots, A_s, S_r, \ldots, S_1, X, B_s, \ldots, B_1, C_1, \ldots, C_w\} \tag{4.6}$$

where X represents the common axis. If the symmetry group has a horizontal common axis, the initial sequence pair becomes

$$\Gamma_+ = \{A_1, \ldots, A_s, X, S_1, \ldots, S_r, B_s, \ldots, B_1, C_1, \ldots, C_w\} \tag{4.7}$$

$$\Gamma_- = \{B_1, \ldots, B_s, X, S_1, \ldots, S_r, A_s, \ldots, A_1, C_1, \ldots, C_w\}. \tag{4.8}$$

4.3.3.4 Cost Function

We use the following cost function to evaluate the candidate solutions in the simulated annealing process.

$$Area + \lambda \times WireLength + \gamma \times Penalty + \beta \times Proximity + \sigma \times AspectRatio \tag{4.9}$$

The first term $Area$ represents the total area of the rectangle packing which is the product of width and height. The second term $WireLength$ is the half-parameter wirelength (HPWL) that estimates the total wirelength of the candidate solution. The third term $Penalty$ is a penalty that evaluates

any violations of the specified constraints. It can be calculated by

$$Penalty = \sum_{i=1}^{N} [min(pos(y) - pos(x) - weight(e_i), 0)]^2, e_i = e(x, y) \tag{4.10}$$

where $weight(e_i)$ is the weight of the edge e_i in the constraint graphs. The term $pos(y) - pos(x)$ represents the actual distance from block x to y in the placement. Since $weight(e_i)$ represents the minimum distance from x to y restricted by constraints, a violation exists if $pos(y) - pos(x)$ is less than $weight(e_i)$. So the penalty term is the sum of squares of all negative $pos(y) - pos(x) - weight(e_i)$. The forth term $Proximity$, which is the sum of the distances between constrained blocks in both horizontal and vertical direction, corresponds to the proximity constraints . The last term $AspectRatio$ tries to make the packing as square as possible, which is expressed as the ratio between width and height of the packing.

$$AspectRatio = \begin{cases} width/height, & \text{if } width \geqslant height \\ height/width, & \text{otherwise.} \end{cases} \tag{4.11}$$

The parameters λ, γ, β and σ can be adjusted to specify the relative priority of different terms for the optimization.

4.4 Constraint-driven Routing

In modern SoC system, although analog circuits have less modules than digital circuits, some elementary modules such as big capacitors and inductors will occupy huge areas. It results in that some interconnections might have very long routes on the layouts. Since analog routing requires multiple types of constraints such as symmetry net constraint, shielding constraint and multi-track constraint, it increases the difficulty to solve the congestion by directly applying sequential detailed routing. To address such potential problems in analog routing, we present a two-stage process which includes global routing and detailed routing. The global routing will first create the routing channels with low congestion to guide the detailed router. Then the detailed routing can assign the

actual metals for each interconnection.

4.4.1 Global Routing

4.4.1.1 Basic Flow

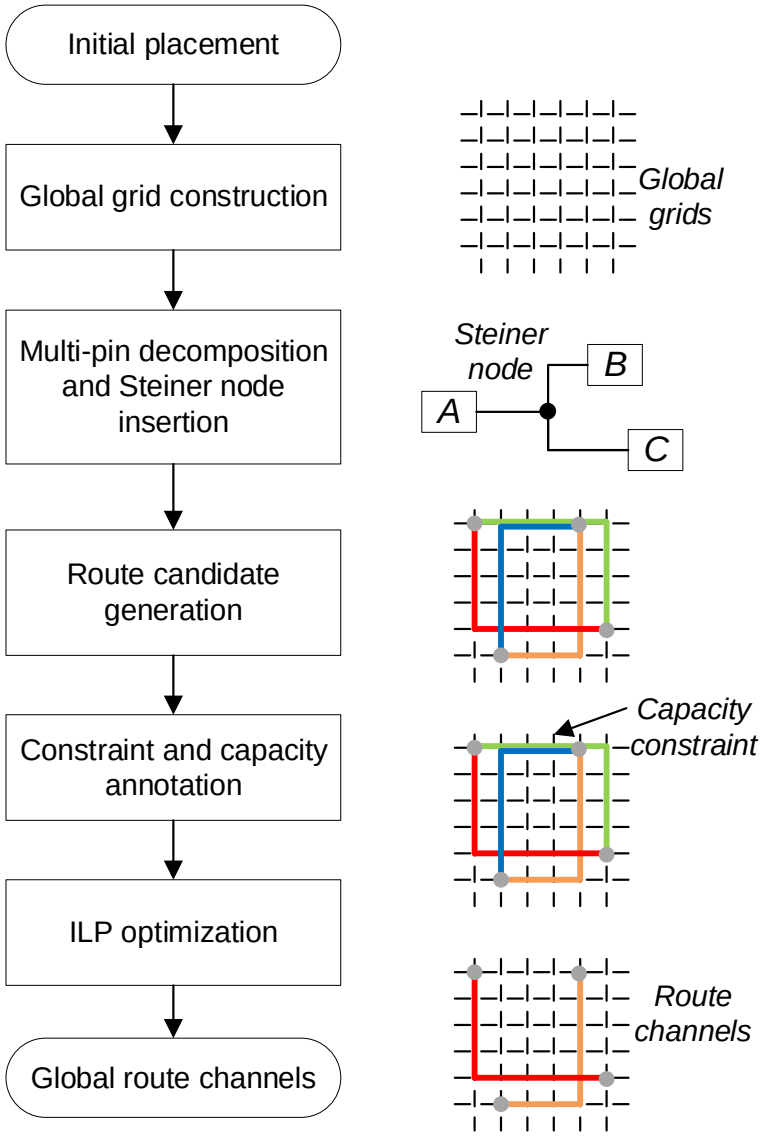


Figure 4.3: Flow of global route.

The basic flow of global routing is shown in Figure 4.3. We use an ILP-based methodology which has been demonstrated in the previous work [32]. First, we create the global grids whose pitches are several times of the actual routing grids. Then we decompose the multi-pin nets into several two-pin segments by adding Steiner nodes. For each two-pin segment we obtain, a certain amount of route candidates are generated by using Dijkstra algorithm. Each route candidate will be annotated with the routing capacity of the global grids, which becomes the constraint in ILP. Then the global routing problem can be formulated into an ILP instance which tries to minimize the total wirelength and total number of bends. At last, we select the optimal combination of route candidates by using ILP solver. The optimal solution, global route channels, will be fed to the detailed router to generate the physical metal segments.

4.4.1.2 Global Grids

In order to fully utilize the routing resource and generate regular routing layouts, we adapt the grid-based routing methodology. Generally the routing directions are specified alternately from the lowest metal layer to the highest metal layer. For example, if metal 1 has the vertical routing direction, metal 2 should have the horizontal routing direction, then metal 3 will be specified with the vertical routing direction and so on for other layers. For each metal layer, we can construct the routing template according to the design rules. The width/pitch of the routing template can be tuned based on required current density or other constraints. On the same metal layer the width/pitch of different tracks can be different. To simplify the process, each metal layer has the same metal width/pitch in our model. As a result, the routing resource can be modeled by metal templates shown in Figure 4.4.

Based on the template model, the global grids can be constructed in such a sparse way that the pitch is multiple times of that of actual route grids. The resulting global grids are shown in Figure 4.5(a). The solid lines represent global tracks on different metal layers and dash lines denote the potential Vias connecting neighboring layers. We can create an undirected graph based on the global grids we have. The vertices in the graph are the intersections of solid lines and dash lines, and the edges represent the possible connections between them in global routing. However,

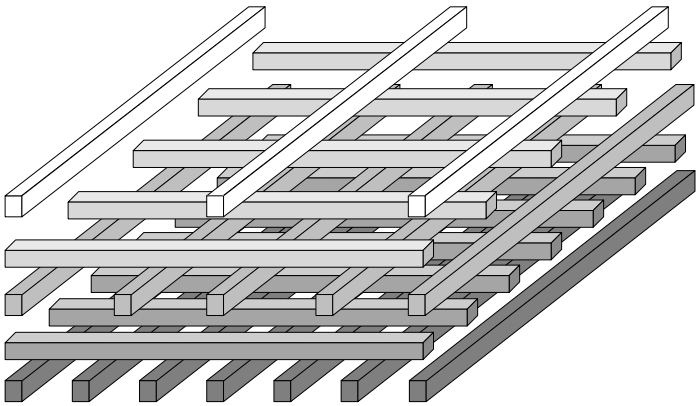


Figure 4.4: Model of metal templates.

sometimes the pitches between neighboring layers are different, which results in some direct connections might violate the minimum length rule. In this scenario, we need to replace those edges with new edges to indicate the feasible routes which can satisfy the design rule. For the case shown in Figure 4.5(b), not all the direct connections between vertices on the middle layer are valid. Only those connections indicated by curved arrows will become edges in the graph. Moreover, every edge in the graph will be labeled with a capacity which corresponds to the number of available metal tracks around it. For example, if every global grid includes 4 metal tracks and two tracks have already been occupied by other metal segments, the corresponding edge should have the capacity of 2.

4.4.1.3 Multi-pin Net Decomposition

For the nets connecting to more than two terminals, we create global routes by reducing them into two-pin nets. Thus we need to insert some Steiner nodes to break the nets into several two-pin segments. The insertion of Steiner nodes also create a rectilinear minimum Steiner tree. Such decomposition can be implemented by some practical rectilinear minimum Steiner tree algorithms [80].

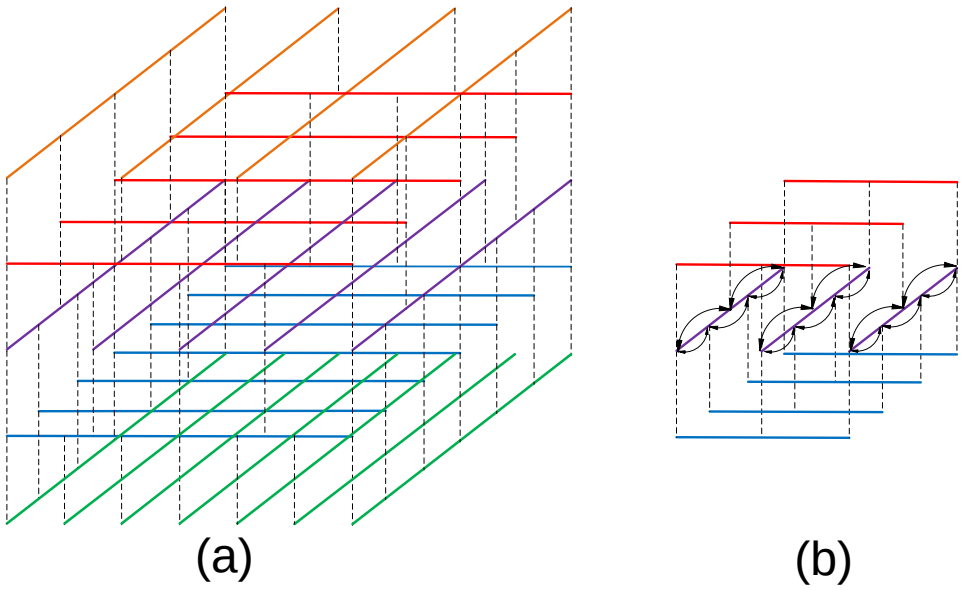


Figure 4.5: Scheme of global grids in (a) full view and (b) detailed view.

4.4.1.4 Route Candidate Generation

Given the undirected graph of global grids and the terminals of each two-pin segment, we can generate several route candidates by finding the shortest path connecting the terminals in the graph. For each two-pin segment, we first identify those vertices in the graph according to the physical locations of the terminals. If the terminal does not cover any vertex, the closest vertex will be selected. Then we use Dijkstra algorithm to find the shortest path from one terminal to the other in the graph. Since we will generate a certain amount of route candidates for each two-pin segment, we perform several rounds of Dijkstra algorithm. After each iteration, the weight of the edges along the path will be increased to increase diversity for candidates.

4.4.1.5 Constraint Annotation

The constraint annotation generates the constraints of the ILP instance. The constraints can be categorized into two types, one type comes from the available capacity in global grids and the other is from the specified routing constraints. The first type of constraints try to control the routing congestion across the design area. For each edge in the graph, the total load (number of

used metal tracks) cannot exceed its max capacity. For each edge e_i with capacity of $cap(e_i)$, the ILP constraint of its relative candidates $cand_j$ is

$$\sum_{cand_j \bigcap e_i \neq \varnothing} x_j \cdot load(cand_j) \leqslant cap(e_i) \tag{4.12}$$

where x_j is the variable corresponding to $cand_j$ and $load(cand_j)$ is the load of $cand_j$.

The routing constraints can also be formulated by another type of ILP constraints. For symmetry net constraint and route matching constraint, two route candidates should be simultaneously selected if they are perfectly symmetric or matched. For two symmetric/matched route candidates $cand_i$ and $cand_j$, they should be constrained by

$$x_i \cdot x_j = 1 \tag{4.13}$$

where $x_i(x_j)$ is the variable corresponding to $cand_i(cand_j)$. For shielding constraint and multi-track routing constraint, the actual width of constrained candidates might occupy multiple metal tracks. It might result in overlapping with other candidates. For two overlapped candidates $cand_i$ and $cand_j$ in this case, they should be constrained by

$$x_i + x_j \leqslant 1. \tag{4.14}$$

4.4.1.6 ILP Optimization

Given the route candidates and related constraints in Section 4.4.1.5, the global routing will choose the optimal combination of route candidates satisfying the constraints and minimize the total wirelength and the total number of bends. The objective in ILP optimization becomes

$$min \sum x_i \cdot len(x_i) + \sum x_i \cdot bend(x_i) \tag{4.15}$$

where $len(x_i)$ and $bend(x_i)$ are the estimated wire length and bend of candidate $cand_i$ respectively.

4.4.2 Detailed Routing

4.4.2.1 Basic Flow

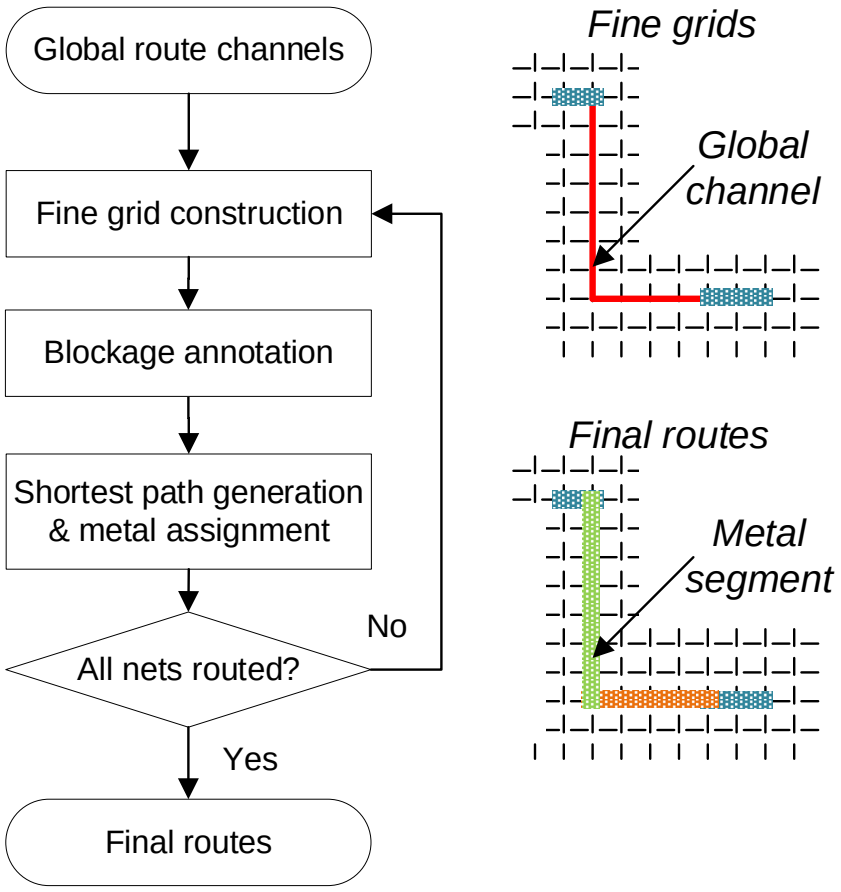


Figure 4.6: Flow of detailed routing.

Figure 4.6 shows the basic flow of the detailed routing. Unlike the global routing, the detailed routing is performed on each net sequentially. First, the fine routing grids based on the maximum metal width and minimum metal spacing in design rules are created. To guide the detailed routing with the global channels, the fine grids are created around the global channels and terminals for connections. The bandwidth of fine grids along the global channels is determined by how many

metal tracks we allow to use. In the example of Figure 4.6, we use 3 metal tracks when fine grids are created along the global channels. Note that the fine grids might not be restricted to the exact metal layers of global channels. If the global channels are on metal 2, we can create fine grids on metal 1/2/3 to give some spaces for detailed routing. After constructing the fine routing grids, we can obtain the undirected graph for shortest path searching.

Then, we annotate the routing blockages, which belong to existing metals of the blocks or other routed nets, in the graph. There are two ways to make such kind of annotations. One is to remove the corresponding vertices and their connected edges, and the other is to reduce the capacity of those edges to 0. Either type of annotation will create a reduced graph so that any path in the graph has no overlap with other existing metals. At last, we use Dijkstra algorithm to find the shortest path in the graph and assign metal segments along the path.

For multi-pin nets connecting more than two terminals, we first pick two terminals and complete the connection between them. Then the resulting metal segments including the two terminals connected by them become a dummy terminal. Another terminal will be selected and its connection to the dummy terminal will be routed. Again, a new dummy terminal will be obtained by taking into account all the metal segments that have been created. Such iterative approach will stop when all the terminals have been connected. In the example of Figure 4.7(a), a three-pin net has three terminals A, B and C. We first assign metals to complete the connection between A and B. The new metals including terminal A and B become the dummy terminal D' as shown in Figure 4.7(b). Then the interconnection between terminal C and D' is routed, which results in the final routes in Figure 4.7(c).

4.4.2.2 Handling of Symmetry Net Constraints

We use a straightforward method to handle the symmetry net constraint [64]. Instead of applying detailed routing on nets in the symmetry constraint individually, we do detailed routing for those nets simultaneously by taking the environment of both sides into consideration. For two nets in the symmetry constraint, we pick one of them to create fine grids and annotate blockages. During the blockage annotation, not only the blockages around the selected net are annotated, but

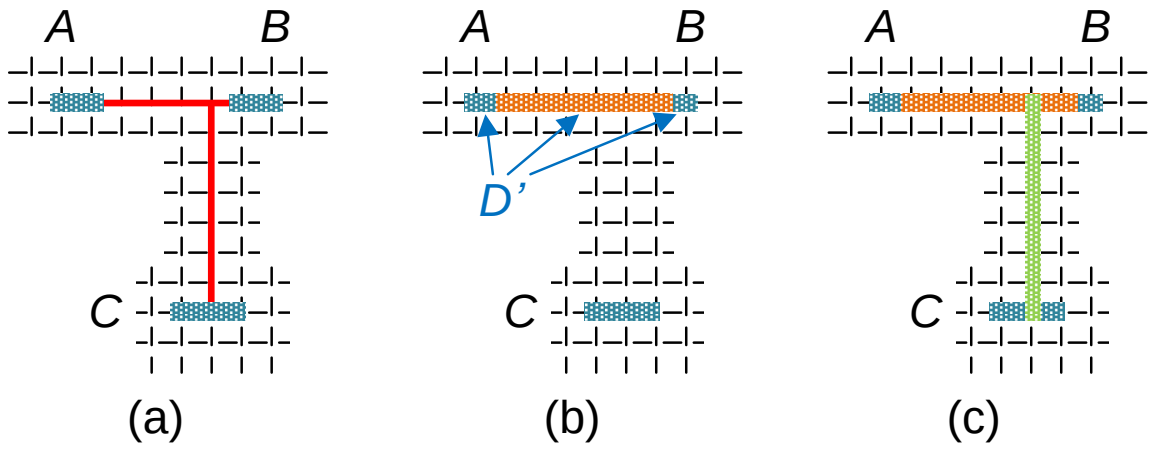


Figure 4.7: Example of detailed routing on a three-pin net.

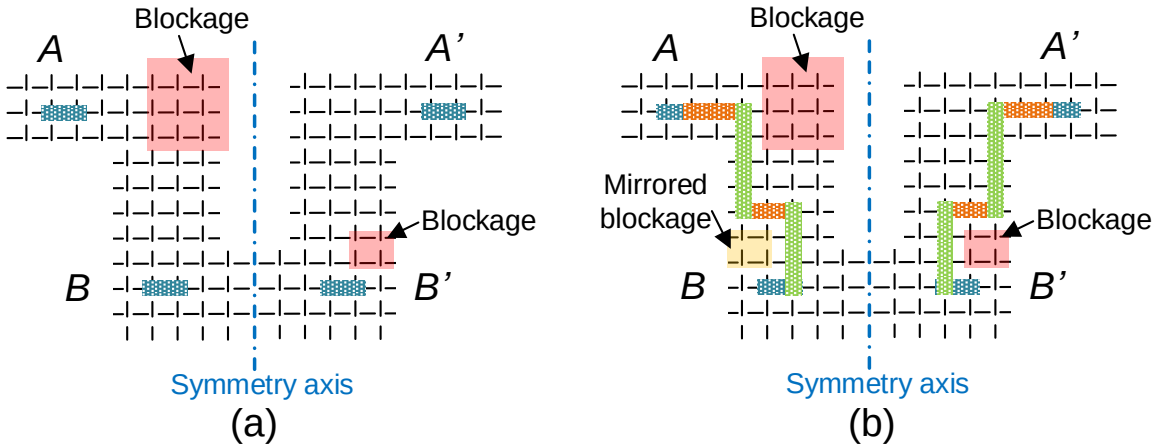


Figure 4.8: Example of symmetry nets (a) before and (b) after detailed routing.

also the blockages around its symmetry counterpart will be mirrored and annotated. After the detailed routing, the complete routes of the selected net will be mirrored to form the actual routes for its counterpart. For the example in Figure 4.8(a), symmetry nets $net(A, B)$ and $net(A', B')$ have different routing blockages around them (red rectangles in the figure). Then the blockage of

$net(A', B')$ is mirrored with respect to the symmetry axis (yellow rectangle in the figure). The detailed routing annotates both the blockages when it works on $net(A, B)$. The resulting routes are presented in Figure 4.8(b).

4.5 Bottom-up Hierarchical Flow

Analog designers tend to draw hierarchical designs to handle sophisticated cases with better circuit performance. For example, matched blocks can be placed in the same macro, or symmetry/common centroid layout can be easier to achieve by grouping some devices first. To follow such kind of design methodology, we develop a hierarchical layout synthesis flow which works in bottom-up manner. As placement and routing are integrated into the hierarchical flow, a comprehensive infrastructure is required to manage the design data across different hierarchies. Such central database is constructed based on the data categories and their corresponding hierarchical level. Both the placer and router can interact with the database to access and update the data in the flow.

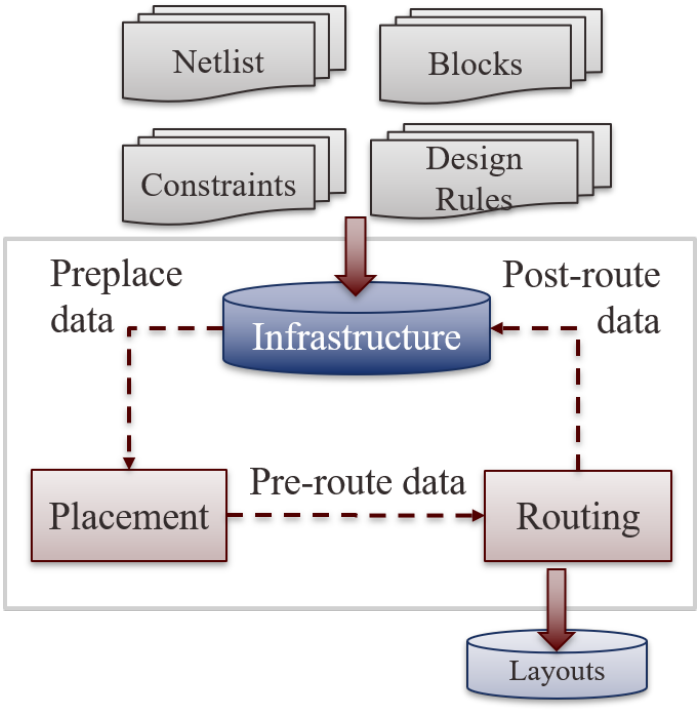


Figure 4.9: Infrastructure in hierarchical process and the data flow.

As presented in Figure 4.9, the infrastructure in the hierarchical process imports all the data such as netlist, building blocks, user-defined constraints and design rules. The design data will be classified into individual nodes according their hierarchical level. All the nodes are in a hierarchy tree where the descendant of one node is its prerequisite in the design hierarchy. Thus there are three types of nodes: leaf nodes corresponding to the macros at bottom level, intermediate nodes representing the middle-level macros, and a root node that is the top level of the design. Based on the hierarchy tree, we use depth-first search (DFS) algorithm to sort the nodes in the dependency order. Then the hierarchical design can be proceeded by sequentially placing and routing the ordered nodes. For each node, the placement engine gets pre-place data from the central database and pass the pre-route data to routing. After routing, the post-route data including all the resulting layouts will be stored back to the central database. Such data flow between the infrastructure, placement and routing are shown in Figure 4.9.

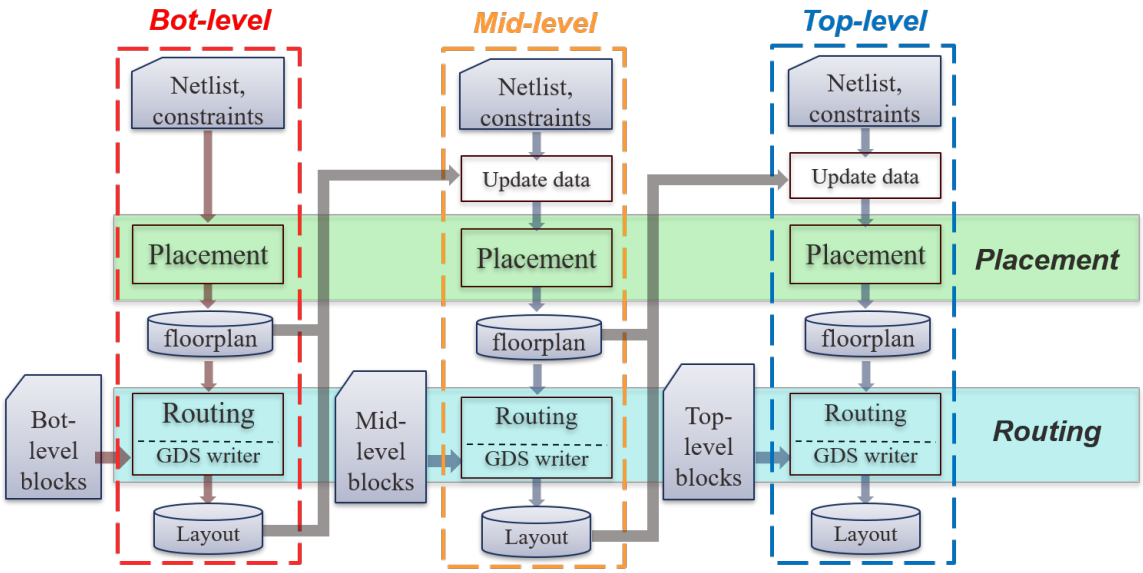


Figure 4.10: Scheme of the bottom-up hierarchical flow.

The overall hierarchical flow is presented in Figure 4.10. Each hierarchy uses the same placement and routing engine that have been introduced in previous sections. Except for the basic

functions, the hierarchical flow need to resolve two problems. 1) Interface between hierarchies in the flow: For the nodes at bottom/middle level, we will commit their post-route layouts into new building blocks after placement and routing. Then their upper hierarchies can treat them as general building blocks so that the same layout synthesis method can be applied without any revision. In the flow, we need to update the nodes at middle/top level with physical information of their lower hierarchies before working on them. 2) Ports of lower hierarchies on the layout: when the lower hierarchy is committed into a new block, how can we generate the physical terminals of block pins (originally ports in the lower hierarchy)? In our method, we do not create terminals for ports when working on the lower hierarchy. Instead, we specify the metal segments of nets connecting to those ports as physical terminals. As a result, the upper hierarchy can choose the optimal location to connect the pins of its lower blocks. The pseudo-code of the whole hierarchical flow is shown in Algorithm 4.

Algorithm 4: Hierarchical flow for analog layout synthesis.

Input : Circuit netlist N, building blocks B, constraints C, design rules DR
Output: Layouts

```
1 DB = ConstructDatabase(N, B, C, DR)
2 Q = SortHierNode(DB)
3 while Q ≠ ∅ do
4 |   Node = Q.pop()
5 |   Placement(Node)
6 |   Routing(Node)
7 |   UpdateParentData(Node)
8 end
9 WriteLayout(Node)
```

4.6 Mixed-size Block Placement

Although the annealing based placement in Section 4.3 can gracefully handle multiple kinds of design constraints, the search space can grow dramatically in large cases, which leads to degradation in runtime performance. Given an analog design with N building blocks, the number of block

sequence including all the blocks is $N!$. Thus the max number of sequence pairs representing different layout topology should be $(N!)^2$. For a design including 10 blocks, the search space will contain more than 1.3×10^{13} placement solutions (without considering any constraints). Meanwhile, we observe the fact that the building blocks in analog circuits vary in size. Some modules such as capacitors and inductors may occupy huge areas due to their capacity, while some transistors can be very small. It's quite inefficient to simultaneously disturb all the blocks in simulated annealing, since the movement of tiny blocks will have very small impact on the entire placement. In order to further improve the performance of our placement, we present a two-stage placement method to handle analog designs with mixed-size blocks.

4.6.1 Overall Flow

The mix-sized block placement includes two stages, one is the macro placement which will create floorplans containing only large blocks, and the second stage is the full cell placement considering all the blocks. In macro placement, the large blocks will be placed to create a suitable foundation for full cell placement. Although only large blocks are included, the interconnections between blocks and related design constraints will be retained. The macro placement will identify large blocks, reduce the design, and then do simulated annealing based placement. In full cell placement, the rest blocks will be inserted into the floorplan created by macro placement. The floorplan will be improved without changing the relative relationship of large blocks. The full cell placement consists of three steps, design restoration, analytical placement and legalization. The overall flow of mixed-size placement is presented in Figure 4.11.

4.6.2 Macro Placement

First of all, we need to identify all the large blocks in the design. To handle all kind of designs in various scale, we set a threshold to identify large blocks. For example, if the average block size is 200 units and the threshold is 1.2 times of the average block size, all those blocks larger than 240 units should be identified as large blocks.

The reduced design will be generated by removing small blocks from the original designs and

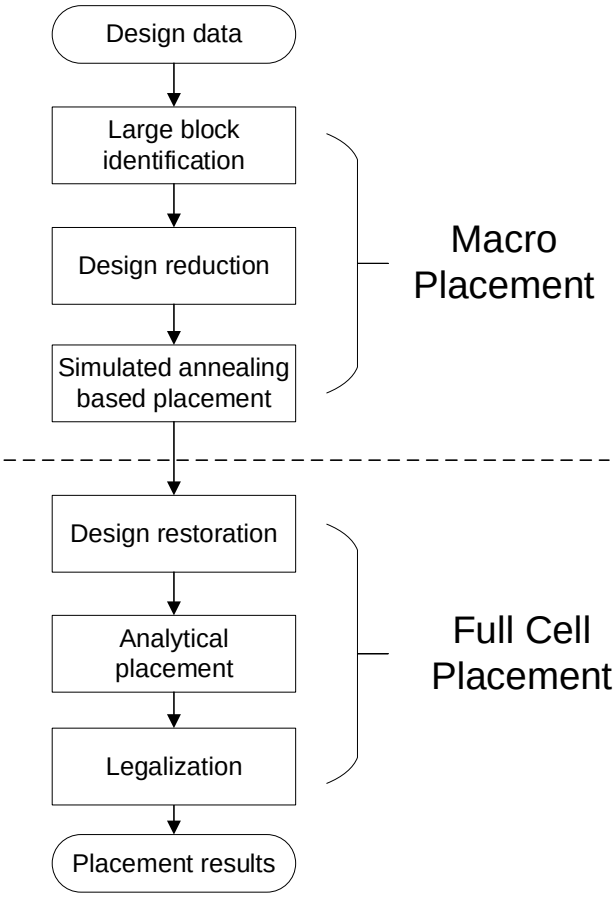


Figure 4.11: Scheme of the mixed-size block placement.

retaining the interconnections between large blocks. But the removal of small blocks can result in the loss of connection information, which may degrade the wirelength estimation in macro placement. Thus, if there is a path between two large blocks going through one or more small blocks in the original design, we add a dummy connection between them in the reduced design. In the example of Figure 4.12(a), the nodes in gray represents the large blocks and the white nodes represent the small blocks. Since there is a path from block A to E through small block B, we add a dummy connection from A to E which is indicated as the dash arc in Figure 4.12(b).

Besides the netlist, the design constraints also need to be updated to reflect the reduction of the design. For example, for symmetry block constraints, only those symmetry pairs and self-

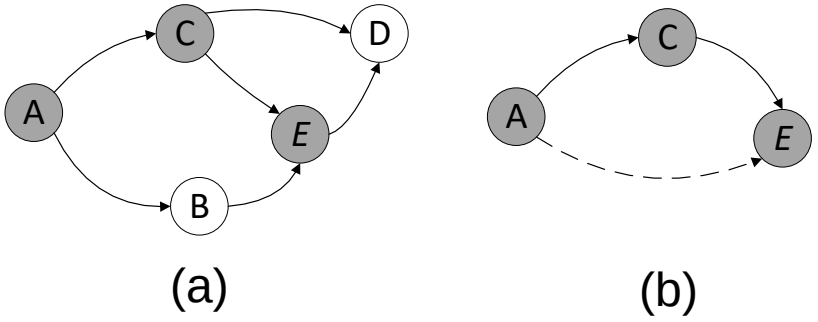


Figure 4.12: Example of design reduction: (a) original design v.s. (b) reduced design.

symmetric blocks that are identified as large blocks will be kept. Otherwise, those constraints on small blocks will be removed.

After the design reduction, we can reuse our simulated annealing based placement algorithm in Section 4.3 to create placement for large blocks. Although we have removed small blocks in this stage, we need to create a suitable floorplan for full cell placement so that small blocks can be inserted between large blocks. Thus, we will augment the constraint graphs of reduced design to reserve sufficient spaces for small block insertion. For any dummy connection between large blocks, we will label it with the total half-perimeter of small blocks on the such connection. Then a weighted edge will be added in constraint graphs to reflect the label. For the example in Figure 4.12, the dummy connection between A and E will be labeled with the half-perimeter of block B. And an edge with weight of the same value will be added in the constraint graphs.

4.6.3 Full Cell Placement

After macro placement, we obtain a floorplan of large blocks. It is used as an initial solution for full cell placement by inserting the other blocks in suitable locations. First, the original design will be restored. At the same time, the design constraint will also be recovered. Given the floorplan by macro placement, we allocate small blocks in the space between large blocks. Then we use the analytical methodology to create global placement for all the blocks.

The analytical method is based on the non-linear global placement model, which has been

demonstrated in previous work [81, 82, 83]. It simultaneously considers the wirelength, overlaps between blocks, boundary constraint and other design constraints. For a net n_k connecting to multiple block pins, its wirelength in half-perimeter estimation can be

$$\max_{i \in n_k}(x'_i) - \min_{i \in n_k}(x'_i) + \max_{i \in n_k}(y'_i) - \min_{i \in n_k}(y'_i) \tag{4.16}$$

where x'_i / y'_i is the x /y coordinate of the pin connecting to n_k. The coordinates of any block pin are constrained by its corresponding block as

$$\begin{aligned} x'_i &= x_i + \Delta x_i \\ y'_i &= y_i + \Delta y_i \end{aligned} \tag{4.17}$$

where (x_i,y_i) is the location of block i and (Δx_i, Δy_i) represents the bias of the block pin from the block center. Then, the total wirelength of a given placement should be

$$WL = \sum_{n_k}(\max_{i \in n_k}(x_i + \Delta x_i) - \min_{i \in n_k}(x_i + \Delta x_i) + \max_{i \in n_k}(y_i + \Delta y_i) - \min_{i \in n_k}(y_i + \Delta y_i)). \tag{4.18}$$

The overlaps between different blocks are modeled as the product of maximum overlaps in x direction and y direction [62, 84]. The overlap between block i and j in x direction can be calculated by

$$O^x_{i,j} = \max(\min(x_i + w_i - x_j, x_j + w_j - x_i, w_i, w_j), 0) \tag{4.19}$$

where x_i /y_i is the x /y coordinate of the lower-left corner of block i and w_i is the width of the block. Similarly, the overlap between two blocks in y direction should be

$$O^y_{i,j} = \max(\min(y_i + h_i - y_j, y_j + h_j - y_i, h_i, h_j), 0) \tag{4.20}$$

where h_i is the height of block i. Then the total overlaps in a given placement can be calculated by

$$OL = \sum_{i,j \in L, i \neq j} O^x_{i,j} \cdot O^y_{i,j} \tag{4.21}$$

for each pair of blocks in the set L.

The result of macro placement gives a floorplan with a desired bounding box. The full cell placement should keep all the blocks stay inside the box. Thus we model the violations of the boundary constraint as a penalty

$$BND = \sum_{i \in L} (\max(x_L - x_i, 0) + \max(x_i + w_i - x_H, 0) + \max(y_L - y_i, 0) + \max(y_i + h_i - y_H, 0)) \tag{4.22}$$

where (x_L,y_L) and (x_H, y_H) are the coordinates of the lower-left and upper-right corner of the bounding box.

For other design constraints such as proximity constraint, we can simply evaluate the violation of the constraint as penalty terms. For example, the symmetry constraint requires that each symmetry pair in the symmetry group should be placed around a common symmetry axis and each self-symmetric block should be on the same axis. Given a set of symmetry group G and g^p_k (g^s_k) representing the set of symmetry pairs (self-symmetric blocks) in group g_k respectively, the penalty for the violation of symmetry constraint with a common horizontal axis can be calculated by

$$SYM_x = \sum_{g_k \in G} (\sum_{i,j \in g^p_k} ((x_i + x_j - 2x^c_k)^2 + (y_i - y_j)^2) + \sum_{i \in g^s_k} (x_i - x^c_k)^2) \tag{4.23}$$

where (x_i,y_i) is the location of block i and x^c_k is the coordinate of the common symmetry axis. The penalty for the symmetry constraint with a vertical axis can be calculated in the similar method.

In the analytical placement, a unconstrained non-linear conjugate gradient algorithm is used to minimize the objective which includes all the terms listed above.

$$\lambda_{WL} \times WL + \lambda_{OL} \times OL + \lambda_{BND} \times BND + \lambda_{SYM} \times (SYM_x + SYM_y) \tag{4.24}$$

The analytical placement gradually adjusts the location of each block according to the conjugate gradient of the objective until the objective is small enough or the maximum number of iterations is reached. To smooth the $max(min)$ functions in the objective, we use the log-sum-exponential model (LSE) [85] below to approximate those non-differentiable functions. α is the parameter for smoothing.

$$\begin{aligned} \max_i(x_i) &\leftarrow \alpha \log \sum_i \exp(x_i/\alpha) \\ \min_i(x_i) &\leftarrow -\alpha \log \sum_i \exp(-x_i/\alpha) \end{aligned} \tag{4.25}$$

The conjugate gradient algorithm in our full cell placement uses dynamic step-size control in [82] to speed up the convergence.

After the analytical process, we get the result of a global placement. Then we do legalization to finalize the placement which has optimal area and satisfies all the design constraints. First, we construct the constraint graphs with minimum edges by using the plane sweep algorithm in [86]. Then the constraint graphs will be augmented by the design constraints as Section 4.3.2. At last, the location of each block in the final placement will be determined by its longest path in the constraint graphs.

4.7 Experiment Results

4.7.1 Experiment Setup

We implement our hierarchical placement and routing in C++ except for a few specific algorithms solved by the dynamically-linked solvers, FasterSteiner [87] and lp_solve [88]. In the simulated annealing based placement, the temperature is set to 1×10^6 at the beginning and is progressively decreased at the rate of 0.95 until the minimum temperature 1×10^{-6} is reached. The number of iterations at each temperature step is 80. The parameters in cost function are initially set to $\lambda = 1000$, $\gamma = 30$, $\beta = 100$ and $\sigma = 1000$ and will be dynamically adjusted after the first 100 iterations to ensure all the terms in cost function are approximately equal. In global routing, we create the global grids whose pitches are 4 times of the actual grids. In order to decompose multi-pin nets and determine Steiner nodes, we use the open solver FastSteiner [87]. The Mixed Integer

Linear Programming (MILP) solver lp_solve [88] is used to get the results of global routing. For mixed-size block placement, the parameters of the objectives are set to $\lambda_{WL} = 1500$, $\lambda_{OL} = 10$, $\lambda_{BND} = 2500$, $\lambda_{SYM} = 1$ and the scale α in the LSE model is set as a big value (1000$\sim$2000). All the experiments are run on CentOS Linux system with 2.8GHz Intel Xeon core. A set of real analog designs including operating amplifier (opamp) and switch capacitor filter (SCF) are used as our benchmark circuits. Most of them have multiple design hierarchies. The basic information of all the benchmarks are summarized in Table 4.1. The column of hierarchy shows the level of each hierarchy in the whole design. For example, the hierarchy-1 of SCF is the bottom level in the hierarchy, while the hierarchy-3 is its top level.

Table 4.1: Benchmark circuits for experiments.

Design	Hierarchy	#Blocks	#Nets	#Constraints
opamp	1	5	16	16
trackhold	1	12	7	9
CCC	1	8	9	2
	2	2	9	0
SCF	1	14	19	30
	2	8	9	8
	3	5	12	19
BS_AMP1	1-1	4	6	5
	1-2	3	3	2
	2	2	4	1
	3	5	11	0
BS_AMP2	1	3	3	2
	2	2	4	1
	3	8	11	5

4.7.2 Experiment Results of Hierarchical Flow

We test our hierarchical layout synthesis flow on all the benchmarks. The results are shown in Table 4.2. It is demonstrated that our hierarchical flow can support designs in multiple hierarchies with different kinds of design constraints. We can observe that the runtime spent on the whole flow

increases as the design become larger. We also observe that the design constraints have impact on the runtime. For example, although the overall area of design opamp is only one third of that of BS_AMP1/BS_AMP2, the runtime is 4 times of BS_AMP series. It is because opamp has more design constraints required for qualified performance. Even though the number of blocks is less, it needs more time to search the feasible placement solution.

Table 4.2: Results of hierarchical flow on benchmarks.

Design	Hierarchy	Area(μm^2)	HPWL(μm)	Runtime(s)
opamp	1	10.65	39.04	3.88
trackhold	1	152.35	154.86	45.02
CCC	1	322.26	66.50	12.00
	2	650.27	98082	
SCF	1	100.45	176.39	154.83
	2	330	183.75	
	3	1817.75	880.17	
BS_AMP1	1-1	12.05	33.10	0.92
	1-2	2.06	2.02	
	2	3.05	2.93	
	3	36	10.91	
BS_AMP2	1	1.44	1.59	0.76
	2	2.40	2.71	
	3	29.24	22.61	

The layouts of design opamp is shown in Figure 4.13. All the blocks are constrained self-symmetric to a common symmetry axis in one symmetry group. According to the final layouts, all blocks are placed symmetrically about the same axis and are self-symmetric to themselves. At the same time, the placement is compact so that the rectangle area is minimized. The experiment results including placement and routing of a more complicate design, SCF, are illustrated in Figure 4.14. The design constraint of SCF requires that more than 20 blocks are constrained in one symmetry group. According to the layouts, we can find that one self-symmetric module is placed in the center of the layouts. The other blocks which are symmetry pairs around the self-symmetric block are placed in mirror symmetry. The layout satisfies all the constraints specified by users.

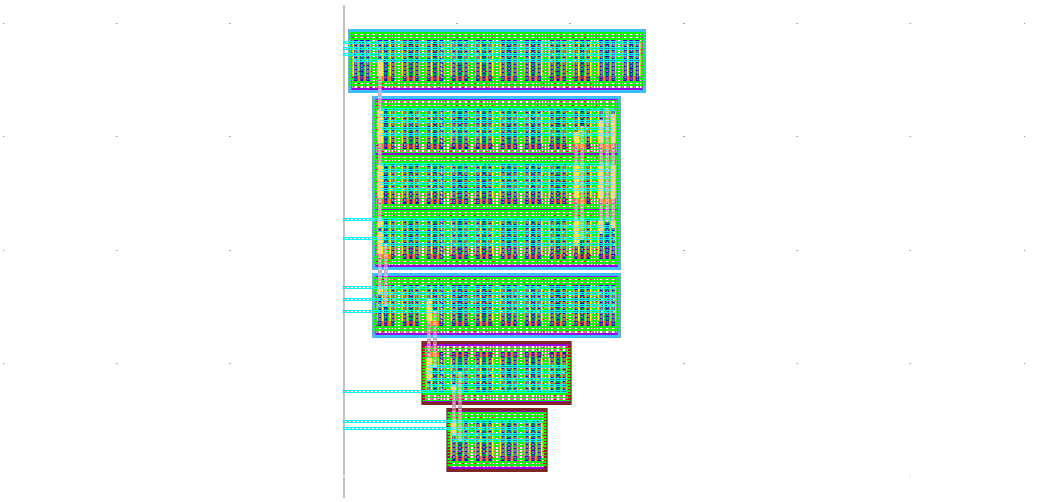


Figure 4.13: Layouts of design opamp.

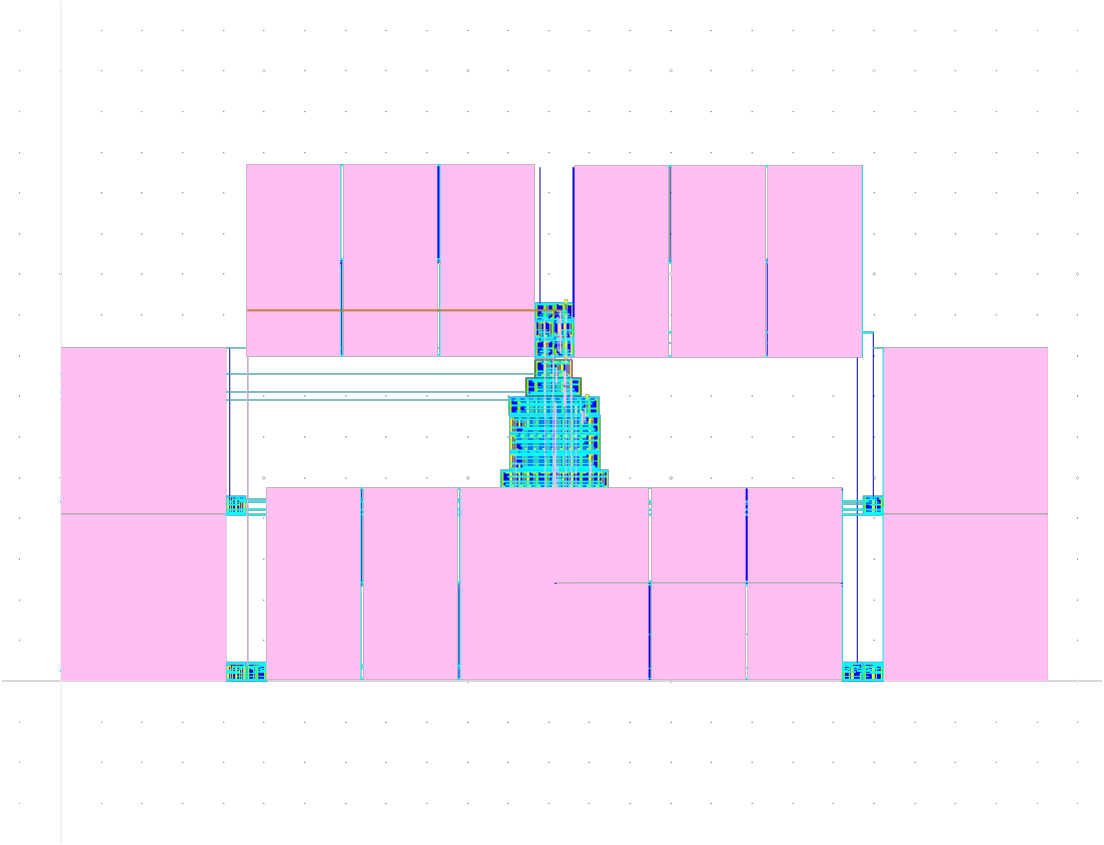


Figure 4.14: Layouts of design SCF.

4.7.3 Experiment Results of Mixed-size Block Placement

We examine our mixed-size block placement on design trackhold. The experiment data of both mixed-size block placement and simulated annealing based placement are shown in Table 4.3. The mixed-size placement achieves a smaller layout with slight degradation in HPWL and runtime, compared with simulated annealing based placement.

Table 4.3: Comparison of performance between different placement on design trackhold.

Placement Method	Area(μm^2)	HPWL(μm)	Runtime(s)
Simulated annealing based placement	152.36	189.81	12.33
Mixed-size block placement	144.85	230.31	15.16

Figures 4.15 to 4.17 show the placement results in different stages of mixed-size block placement. Only the large blocks are placed in Figure 4.15. Notice that the spaces between blocks are reserved for the insertion of small blocks. We can also observe that large blocks are placed symmetrically, which obeys the original design constraints. Figure 4.16 shows the result after analytical placement. The small blocks are placed among large blocks and are placed symmetrically about the same common axis of large blocks. No overlaps can be found between any blocks. All the design constraints are satisfied according to the objective of analytical placement. The final placement after legalization is shown in Figure 4.17. We obtain a compact floorplan which minimizes the area of rectangle bounding box. Comparing to the result of simulated annealing based placement in Figure 4.18, the mixed-size placement achieves less dead area and better aspect ratio.

4.8 Conclusion

In this chapter, we develop the layout automation flow for analog/mixed-signal ICs. A hierarchical layout synthesis flow which works in bottom-up manner is presented. To ensure the qualified layouts for better circuit performance, we use constraint-driven placement and routing which employs the expert knowledge via design constraints. The constraint-driven placement uses

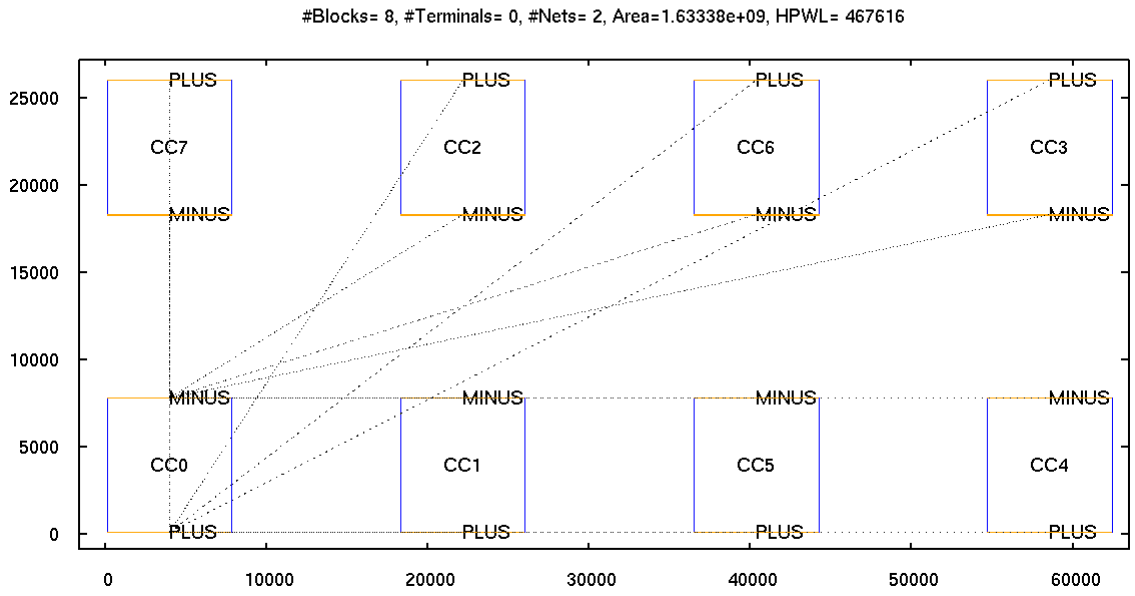


Figure 4.15: Mixed-size placement result after macro placement.

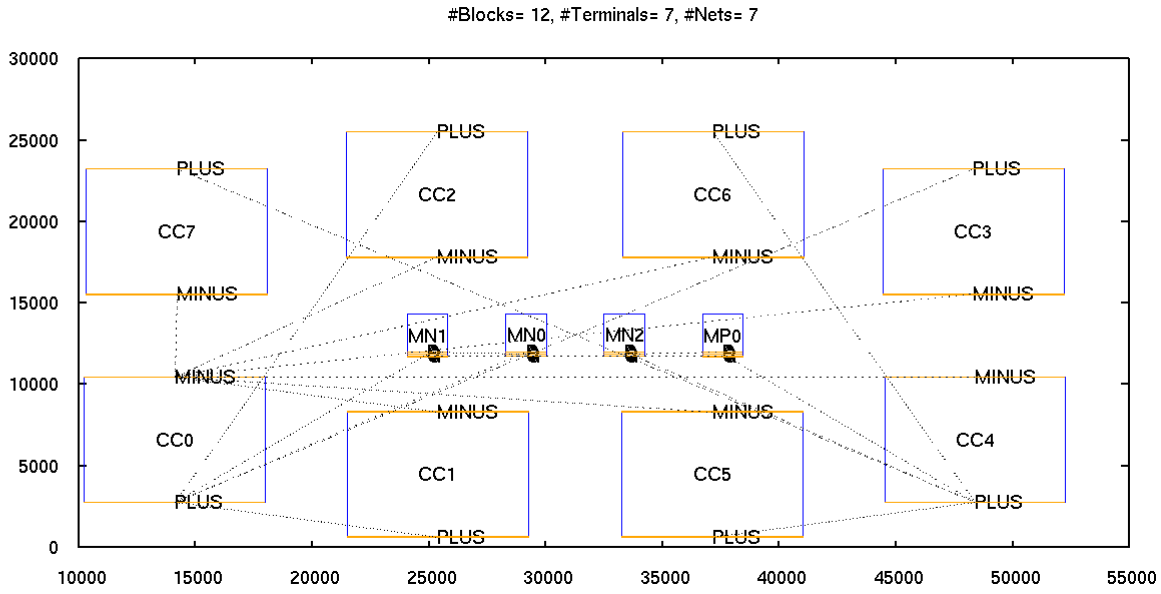


Figure 4.16: Mixed-size placement result after analytical placement.

the simulated annealing process to find the optimal solution. The packing represented by sequence pairs and constraint graphs can simultaneously handle different kinds of placement constraints. The constraint-driven routing consists of two stages, ILP-based global routing and sequential de-

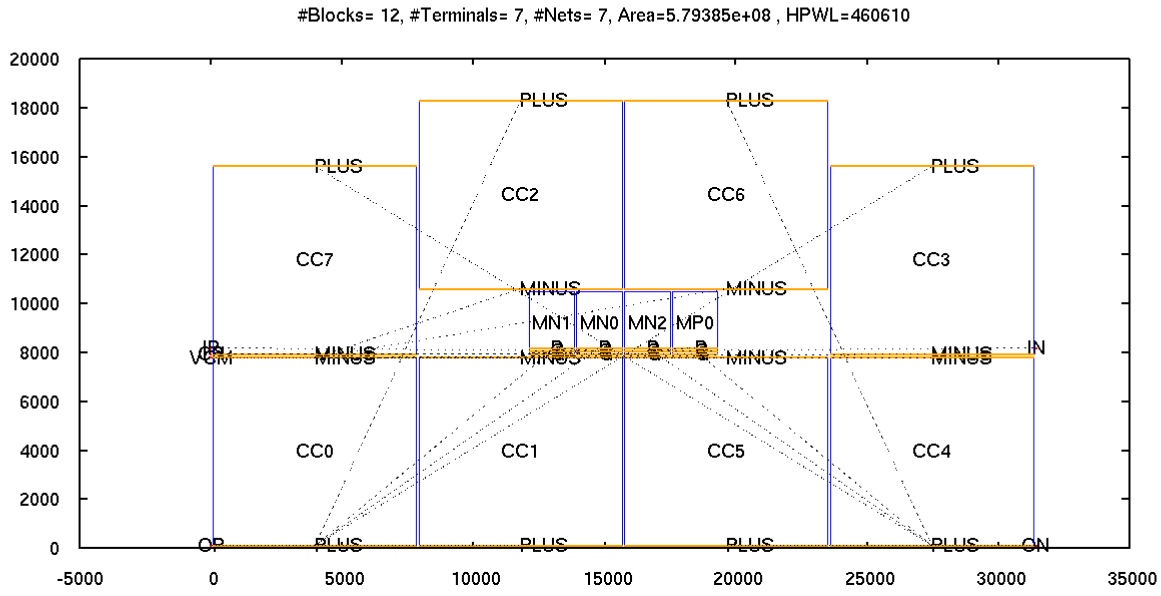


Figure 4.17: Mixed-size placement result after legalization.

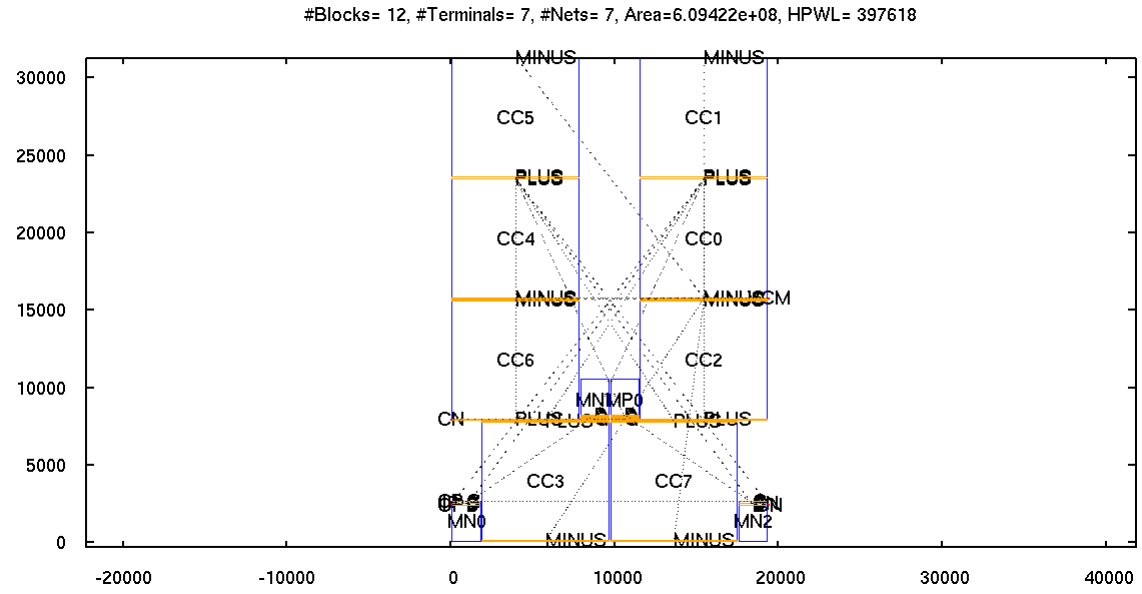


Figure 4.18: Simulated annealing based placement result.

tailed routing. The experiment results demonstrate that our flow can handle complicated hierarchical designs with multiple design constraints. Furthermore, the placement can be further improved by using mixed-size block placement which works on large blocks in priority.

5. SUMMARY AND CONCLUSIONS

In this dissertation, we present three practical techniques to improve the performance and evaluate security of circuit designs. Simple accuracy reconfigurable adder design provides a flexible and efficient solution in approximate computing to trade-off computation quality and power consumption. The circuit performance can be reconfigured during the runtime according to the required applications. It further explores the probability of developing low-power designs in circuit level. Layout recognition attacks focus on studying the security of physical layouts during designing and manufacturing the circuits. Although split manufacturing together with k-security defense raises the threshold to attack designs, structural pattern matching with conventional design hints improves the correctness of attacks. It also provides an alternative way to evaluate the circuit security in practice. In another work, the analog layout synthesis can be automated in a hierarchical framework, which targets for fully design automation with no human in loop. The constraint-driven placement and routing methods generate the layouts according to the required design constraints, which follow the designer experiences and expert knowledge. The mixed-size block placement further improve the layouts for designs with blocks in various size. All the proposed techniques have been demonstrated by solid experiments. They are proved to be efficient in practice by their scalability to large-scaled designs and extent-ability to real applications.

REFERENCES

[1] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *SIGPLAN Notices*, pp. 301–312, 2012.

[2] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[3] J. Han and M. Orshansky, "Approximate computing: an emerging paradigm for energy-efficient design," in *European Test Symposium*, pp. 1–6, 2013.

[4] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: a new paradigm for arithmetic circuit design," in *Conference on Design, Automation and Test in Europe*, pp. 1250–1255, 2008.

[5] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in *Conference on Design, Automation and Test in Europe*, pp. 1257–1262, 2012.

[6] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *International Symposium on Integrated Circuits*, pp. 69–72, 2009.

[7] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 18, no. 8, pp. 1225–1229, 2010.

[8] N. Zhu, W. L. Goh, and K. S. Yeo, "Ultra low-power high-speed flexible probabilistic adder for error-tolerant applications," in *International SoC Design Conference*, pp. 393–396, 2011.

[9] G. Liu, Y. Tao, M. Tan, and Z. Zhang, "CASA: correlation-aware speculative adders," in *International Symposium on Low Power Electronics and Design*, pp. 189–194, 2014.

[10] J. Hu and W. Qian, "A new approximate adder with low relative error and correct sign calculation," in *Conference on Design, Automation and Test in Europe*, pp. 1449–1454, 2015.

[11] J. Miao, K. He, A. Gerstlauer, and M. Orshansky, "Modeling and synthesis of quality-energy optimal approximate adders," in *International Conference on Computer-Aided Design*, pp. 728–735, 2012.

[12] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Design Automation Conference*, pp. 820–825, 2012.

[13] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *International Conference on Computer-Aided Design*, pp. 48–54, 2013.

[14] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[15] Y. Xie, C. Bao, and A. Srivastava, "Security-aware 2.5D integrated circuit design flow against hardware IP piracy," *Computer*, vol. 50, no. 5, pp. 62–71, 2017.

[16] S. Dupuis, P. S. Ba, G. D. Natale, M. L. Flottes, and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *International On-Line Testing Symposium*, pp. 49–54, 2014.

[17] F. Imeson, A. Emtenan, S. Garg, and M. Tripunitara, "Securing computer hardware using 3D integrated circuit (IC) technology and split manufacturing for obfuscation," in *USENIX Security Symposium*, pp. 495–510, 2013.

[18] J. Rajendran, O. Sinanoglu, and R. Karri, "Is split manufacturing secure," in *Conference on Design, Automation and Test in Europe*, pp. 1259–1264, 2013.

[19] Y. Wang, P. Chen, J. Hu, and J. Rajendran, "The cat and mouse in split manufacturing," in *Design Automation Conference*, pp. 1–6, 2016.

[20] J. Magaña, D. Shi, J. Melchert, and A. Davoodi, "Are proximity attacks a threat to the security of split manufacturing of integrated circuits," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 12, pp. 3406–3419, 2017.

[21] M. Li, B. Yu, Y. Lin, X. Xu, W. Li, and D. Z. Pan, "A practical split manufacturing framework for trojan prevention via simultaneous wire lifting and cell insertion," in *Asia and South Pacific Design Automation Conference*, pp. 265–270, 2018.

[22] F. Balasa and K. Lampaert, "Symmetry within the sequence-pair representation in the context of placement for analog design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 7, pp. 721–731, 2000.

[23] E. F. Young, C. C. Chu, and M. Ho, "Placement constraints in floorplan design," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 12, no. 7, pp. 735–745, 2004.

[24] Y.-C. Tam, E. F. Young, and C. Chu, "Analog placement with symmetry and other placement constraints," in *International Conference on Computer- Aided Design*, pp. 349–354, 2006.

[25] Q. Ma, E. F. Young, and K.-P. Pun, "Analog placement with common centroid constraints," in *International Conference on Computer- Aided Design*, pp. 579–585, 2007.

[26] L. Xiao and E. F. Young, "Analog placement with common centroid and 1-D symmetry constraints," in *Asia and South Pacific Design Automation Conference*, pp. 353–360, 2009.

[27] Q. Ma, L. Xiao, Y.-C. Tam, and E. F. Young, "Simultaneous handling of symmetry, common centroid, and general placement constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 1, pp. 85–95, 2011.

[28] M. M. Ozdal and R. F. Hentschke, "Exact route matching algorithms for analog and mixed signal integrated circuits," in *International Conference on Computer- Aided Design*, pp. 231–238, 2009.

[29] M. M. Ozdal and R. F. Hentschke, "An algorithmic study of exact route matching for integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 12, pp. 1842–1855, 2011.

[30] H.-C. Ou, H.-C. C. Chien, and Y.-W. Chang, "Non-uniform multilevel analog routing with matching constraints," in *Design Automation Conference*, pp. 549–554, 2012.

[31] K.-H. Ho, H.-C. Ou, Y.-W. Chang, and H.-F. Tsao, "Coupling-aware length-ratio-matching routing for capacitor arrays in analog integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 161–172, 2015.

[32] C.-Y. Wu, H. Graeb, and J. Hu, "A pre-search assisted ILP approach to analog integrated circuit routing," in *International Conference on Computer Design*, pp. 244–250, 2015.

[33] V. Chippa, A. Raghunathan, K. Roy, and S. Chakradhar, "Dynamic effort scaling: managing the quality-efficiency tradeoff," in *Design Automation Conference*, pp. 603–608, 2011.

[34] S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, vol. 37, no. 3, pp. 67–73, 2004.

[35] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits," in *Conference on Design, Automation and Test in Europe*, pp. 1367–1372, 2013.

[36] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel, "An area-efficient consolidated configurable error correction for approximate hardware accelerators," in *Design Automation Conference*, pp. 1–6, 2016.

[37] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Design Automation Conference*, pp. 1–6, 2015.

[38] V. Benara and S. Purini, "Accurus: a fast convergence technique for accuracy configurable approximate adder circuits," in *IEEE Computer Society Annual Symposium on VLSI*, pp. 577–582, 2016.

[39] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "RAP-CLA: a reconfigurable approximate carry look-ahead adder," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 8, pp. 1089–1093, 2016.

[40] W. J. Townsend, E. E. Swartzlander, and J. A. Abraham, "A comparison of dadda and wallace multiplier delays," in *Advanced Signal Processing Algorithms, Architectures, and Implementations*, vol. 5205, pp. 552–561, 2003.

[41] S. Yu and E. Swartziander, "DCT implementation with distributed arithmetic," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 985–991, 2001.

[42] M.-T. Sun, T.-C. Chen, and A. M. Gottlieb, "VLSI implementation of a 16*16 discrete cosine transform," *IEEE Transactions on Circuits and Cystems*, vol. 36, no. 4, pp. 610–617, 1989.

[43] D. Gong, Y. He, and Z. Cao, "New cost-effective VLSI implementation of a 2-D discrete cosine transform and its inverse," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 4, pp. 405–415, 2004.

[44] Y. Wang, P. Chen, J. Hu, and J. Rajendran, "Routing perturbation for enhanced security in split manufacturing," in *Asia and South Pacific Design Automation Conference*, pp. 605–510, 2017.

[45] R. W. Jarvis and M. G. McIntyre, "Split manufacturing method for advanced semiconductor circuits," *US Patent no. 7195931*, 2004.

[46] Intelligence Advanced Research Projects Activity, "Trusted Integrated Circuits (TIC) Program." `https://www.fbo.gov/utils/view?id=b8be3d2c5d5babbdffc6975c370247a6`, 2011.

[47] K. Vaidyanathan, R. Liu, E. Sumbul, Q. Zhu, F. Franchetti, and L. Pileggi, "Efficient and secure intellectual property (IP) design for split fabrication," in *International Symposium on Hardware-Oriented Security and Trust*, pp. 13–18, 2014.

[48] K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu, and L. Pileggi, "Building trusted ICs using split fabrication," in *International Symposium on Hardware-Oriented Security and Trust*, pp. 1–6, 2014.

[49] K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu, and L. Pileggi, "Detecting reliability attacks during split fabrication using test-only BEOL stack," in *Design Automation Conference*, pp. 1–6, 2014.

[50] C. Otero, J. Tse, R. Karmazin, B. Hill, and R. Manohar, "Automatic obfuscated cell layout for trusted split-foundry design," in *International Symposium on Hardware-Oriented Security and Trust*, pp. 56–61, 2015.

[51] S. Mitra, H.-S. Wong, and S. Wong, "Stopping hardware trojans in their tracks." `http://spectrum.ieee.org/semiconductors/design/stopping-hardware-trojans-in-their-tracks`, 2015.

[52] Y. Bi, J. Yuan, and Y. Jin, "Beyond the interconnections: split manufacturing in RF designs," *Electronics*, pp. 541–564, 2015.

[53] J. Valamehr, T. Sherwood, R. Kastner, D. Marangoni-Simonsen, T. Huffmire, C. Irvine, and T. Levin, "A 3-D split manufacturing approach to trustworthy system development," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 4, pp. 611–615, 2013.

[54] S. Patnaik, J. Knechtel, M. Ashraf, and O. Sinanoglu, "Concerted wire lifting: enabling secure and cost-effective split manufacturing," in *Asia and South Pacific Design Automation Conference*, pp. 251–258, 2018.

[55] A. Sengupta, S. Patnaik, J. Knechtel, M. Ashraf, S. Garg, and O. Sinanoglu, "Rethinking split manufacturing: an information-theoretic approach with secure layout techniques," in *International Conference on Computer-Aided Design*, pp. 329–326, 2017.

[56] M. Zhao and S. S. Sapatnekar, "A new structural pattern matching algorithm for technology mapping," in *Design Automation Conference*, pp. 371–376, 2001.

[57] F. Winterer, L. Winterer, and M. Sauer, "iSAT3." `https://projects.informatik.uni-freiburg.de/projects/isat3/`, 2014.

[58] "Analog unit shipments outpacing growth of all IC product segments." `http://www.icinsights.com/news/bulletins/Analog-Unit-Shipments-Outpacing-Growth-Of-All-IC-Product-Segments`, 2014. IC Insights.

[59] P.-H. Wu, M. P.-H. Lin, T.-C. Chen, C.-F. Yeh, X. Li, and T.-Y. Ho, "A novel analog physical synthesis methodology integrating existent design expertise," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 199–212, 2014.

[60] M. P.-H. Lin, Y.-W. Chang, and C.-M. Hung, "Recent research development and new challenges in analog layout synthesis," in *Asia and South Pacific Design Automation Conference*, pp. 617–622, 2016.

[61] A. Patyal, P.-C. Pan, K. Asha, H.-M. Chen, H.-Y. Chi, and C.-N. Liu, "Analog placement with current flow and symmetry constraints using PCP-SP," in *Design Automation Conference*, pp. 1–6, 2018.

[62] B. Xu, S. Li, C.-W. Pui, D. Liu, L. Shen, Y. Lin, N. Sun, and D. Z. Pan, "Device layer-aware analytical placement for analog circuits," in *International Symposium on Physical Design*, pp. 19–26, 2019.

[63] B. Xu, S. Li, X. Xu, N. Sun, and D. Z. Pan, "Hierarchical and analytical placement techniques for high-performance analog circuits," in *International Symposium on Physical Design*, pp. 55–62, 2017.

[64] L. Xiao, E. F. Young, X. He, and K.-P. Pun, "Practical placement and routing techniques for analog circuit designs," in *International Conference on Computer-Aided Design*, pp. 675–679, 2010.

[65] Y. Pang, F. Balasa, K. Lampaert, and C.-K. Cheng, "Block placement with symmetry constraints based on the O-tree non-slicing representation," in *Design Automation Conference*, pp. 464–467, 2000.

[66] C.-W. Lin, J.-M. Lin, C.-P. Huang, and S.-J. Chang, "Performance-driven analog placement considering boundary constraint," in *Design Automation Conference*, pp. 292–297, 2010.

[67] H.-F. Tsao, P.-Y. Chou, S.-L. Huang, Y.-W. Chang, M. P.-H. Lin, D.-P. Chen, and D. Liu, "A corner stitching compliant B*-tree representation and its applications to analog placement," in *International Conference on Computer-Aided Design*, pp. 507–511, 2011.

[68] M. Strasser, M. Eick, H. Gräb, U. Schlichtmann, and F. M. Johannes, "Deterministic analog circuit placement using hierarchically bounded enumeration and enhanced shape functions," in *International Conference on Computer-Aided Design*, pp. 306–313, 2008.

[69] P.-H. Lin and S.-C. Lin, "Analog placement based on novel symmetry-island formulation," in *Design Automation Conference*, pp. 465–470, 2007.

[70] L. Zhang, C.-J. R. Shi, and Y. Jiang, "Symmetry-aware placement with transitive closure graphs for analog layout design," in *Asia and South Pacific Design Automation Conference*, pp. 180–185, 2008.

[71] R. He and L. Zhang, "Symmetry-aware TCG-based placement design under complex multi-group constraints for analog circuit layouts," in *Asia and South Pacific Design Automation Conference*, pp. 299–304, 2010.

[72] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: a new representation for non-slicing floorplans," in *Design Automation Conference*, pp. 458–463, 2000.

[73] M. P.-H. Lin, Y.-T. He, V. W.-H. Hsiao, R.-G. Chang, and S.-Y. Lee, "Common-centroid capacitor layout generation considering device matching and parasitic minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 7, pp. 991–1002, 2013.

[74] M. P.-H. Lin, V. W.-H. Hsiao, and C.-Y. Lin, "Parasitic-aware sizing and detailed routing for binary-weighted capacitors in charge-scaling dac," in *Design Automation Conference*, pp. 1–6, 2014.

[75] M. M. Ozdal and R. F. Hentschke, "Maze routing algorithms with exact matching constraints for analog and mixed signal designs," in *International Conference on Computer-Aided Design*, pp. 130–136, 2012.

[76] M. M. Ozdal and R. F. Hentschke, "Algorithms for maze routing with exact matching constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 1, pp. 101–112, 2013.

[77] Q. Gao, Y. Shen, Y. Cai, and H. Yao, "Analog circuit shielding routing algorithm based on net classification," in *International Symposium on Low Power Electronics and Design*, pp. 123–128, 2010.

[78] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing-based module placement," in *International Conference on Computer-Aided Design*, pp. 472–479, 1995.

[79] S. Kouda, C. Kodama, and K. Fujiyoshi, "Improved method of cell placement with symmetry constraints for analog IC layout design," in *International Symposium on Physical Design*, pp. 192–199, 2006.

[80] A. B. Kahng, I. I. Măndoiu, and A. Z. Zelikovsky, "Highly scalable algorithms for rectilinear and octilinear Steiner trees," in *Asia and South Pacific Design Automation Conference*, pp. 827–833, 2003.

[81] A. B. Kahng, S. Reda, and Q. Wang, "Aplace: a general analytic placement framework," in *International Symposium on Physical design*, pp. 233–235, 2005.

[82] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "NTUplace3: an analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1228–1240, 2008.

[83] H.-C. Ou, K.-H. Tseng, J.-Y. Liu, I.-P. Wu, and Y.-W. Chang, "Layout-dependent effects-aware analytical analog placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 8, pp. 1243–1254, 2015.

[84] S. Kuwabara, Y. Kohira, and Y. Takashima, "An effective overlap removable objective for analytical placement," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 96, no. 6, pp. 1348–1356, 2013.

[85] W. C. Naylor, R. Donelly, and L. Sha, "Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer," Oct. 9 2001. US Patent 6,301,693.

[86] J. Doenhardt and T. Lengauer, "Algorithmic aspects of one-dimensional layout compaction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 5, pp. 863–878, 1987.

[87] J. A. Roy, A. B. Kahng, and I. Mandoiu, "FastSteiner: highly scalable rectilinear and octilinear minimum Steiner tree algorithms." `https://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/FastSteiner`, 2005.

[88] K. Eikland and P. Notebaert, "Mixed integer linear programming (MILP) solver lpsolve." `http://lpsolve.sourceforge.net/5.5`, 2016.