The way ahead is long. I see no ending, yet high and low I'll search with my will unbending.

– Qu, Yuan, ancient Chinese poet.

University of Alberta

A Fast Circuit Similarity-Based Placement Engine for Field Programmable Gate Arrays

by

Xiaoyu Shi

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

Master of Science

Department of Computing Science

Fall 2011
Edmonton, Alberta

To my dear parents, Guang Shi and Youcheng Yu,
and my lovely fiancee, Yuanfang Sun,
I would never achieve this without their endless love.

Abstract

This thesis work presents a novel and efficient circuit similarity algorithm to detect both the local and global topological similarity between two circuits (configurations). Targeting placement, one of the most time-consuming phases in Computer Aided Design (CAD), our proposed circuit similarity algorithm, namely CSBP (Circuit Similarity-Based Placement), is able to significantly accelerate placement based on this similarity. Moreover, circuit similarity can also be applied to other CAD phases, such as routing and verification. We have applied CSBP to incremental design for FPGAs and design space exploration for FPGAs, respectively. For incremental design, we generate the placement of a logically optimized netlist based on the placement of the original netlist and the circuit similarity between the original and the optimized logic-level netlists using both mild resynthesis and aggressive resynthesis. Experimental results show CSBP is averaged 31X faster than the state-of-the-art Versatile Place and Route (VPR) with comparable wire length and estimated critical delay. For design space exploration, experimental results show CSBP accurately depicts the "shape" of a design space and pinpoints the optimal designs at both logic level and algorithm level. Moreover, a turbo version of CSBP performs an average of 30X (up to 100X) faster than VPR's while still achieving comparable placement results.

Acknowledgements

I would first like to express my deepest gratitude to my co-supervisors, Dr. Osmar R. Zaiane and Dr. Bryan Y. Hu, for their great support and valuable guidance in this work. Despite of being extremely busy serving as the scientific director of Alberta Ingenuity Center for Machine Learning (AICML), Dr. Zaiane always managed to give me considerable time to answer my questions and provide insightful ideas on my project. It was a great pleasure working with him and his wisdom and personality would have a tremendous influence on my future career. A big thank-you goes to Dr. Hu whom I have known back in the UCLA days in 2008. He was always there to help me out when I ran out of ideas to attack the problem. He is not only a great mentor, but also an amazing person, a great friend, and has been my role model ever since.

I would also like to thank Dr. Guohui Lin, who also provided a great deal of insightful advice into this research. I am especially thankful for Dr. Lin's help in revising the iterative graph similarity algorithm to circuit similarity algorithm.

I would like to thank many other people who have contributed to this project in various ways, including Dahua Zeng, who implemented the iterative graph similarity algorithm module, Bret Hoehn, who helped me carefully proofread the papers over and over again and many others in the department whom I don't have enough space to list in here, for making my graduate study at University of Alberta a pleasant, rewarding and unforgettable experience.

Last but not the least, I would like to thank my family, who have always been supportive of my education, and words could not express how grateful I am for having them in my life. They are always proud of me, as I am always proud of them.

This project was generously supported by AICML and Natural Sciences and Engineering Research Council of Canada (NSERC). I am grateful for their financial support.

Table of Contents

List of Tables

List of Figures

List of Symbols

v_i	A node in graph G 19
$\mathbf{v'_j}$	A node in graph G' 19
$\mathbf{X^{(n)}_{i,j}}$	Similarity score between node i in graph $G(V)$ and node j in graph $G'(V')$ in iteration n 19
$\mathbf{t}$	The iteration number 19
$\mathbf{n(v)}$	The set of all adjacent nodes of node v 19
π	An injective map from $n(v_i)$ to $n(v'_j)$, if $\|n(v_i)\| < \|n(v'_j)\|$ or An injective map from $n(v'_j)$ to $n(v_i)$, if $\|n(v_i)\| \geq \|n(v'_j)\|$ 19
α	A weight constant within interval (0,1) 19
ε	A terminating threshold for iterations 19
$\mathbf{M}$	An upper bound for number of iterations 19
$\mathbf{k_v : V \to V'}$	A predefined inter-similarity between two nodes 19
$\mathbf{k_e : E \to E'}$	A predefined inter-similarity between two edges, where (v_i, v) is an edge in graph G and $(v'_j, \pi(v))$ is an edge in graph G' 19
$\mathbf{in(v)}$	The set of all adjacent nodes that have an edge entering node v 19
$\mathbf{out(v)}$	The set of all adjacent nodes that have an edge leaving node v 19

List of Acronyms

AIG	And-Inverter Graph
ASIC	Application Specific Integrated Circuit
CAD	Computer Aided Design
CLB	Configurable Logic Block
CS	Circuit Similarity
CSBP	Circuit Similarity-Based Placement
CS-t	Circuit Similarity-turbo
DAG	Directed Acyclic Graph
DSP	Digital Signal Processing
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
I/O	Input/Output
IC	Integrated Circuit
ICP	Incremental Placement
IPR	Incremental Physical Resynthesis
LUT	Look-Up Tables
PI/PO	Primary Input/Primary Output
PLB	Programming Logic Block
QPF	Quadratic Placement for FPGAs
RTL	Register Transfer Level
SIMD	Simple Instruction Multiple Data
VHDL	VHSIC Hardware Description Language
VLSI	Very-Large-Scale Integration
VPR	Versatile Place and Route

List of Publications

[65] X. Shi, D. Zeng, Y. Hu, G. Lin and O. R. Zaiane. Accelerating FPGA Design Space Exploration Using Circuit Similarity-Based Placement. *International Conference on Field-Programmable Technology*, pages 373–376, 2010.

[66] X. Shi, D. Zeng, Y. Hu, G. Lin and O. R. Zaiane. Enhancement of incremental design for FPGAs using circuit similarity. *The International Symposium on Quality Electronic Design*, pages 243–250, 2011.

[67] X. Shi, Y. Hu, G. Lin and O. R. Zaiane. CSBP: A Fast Circuit Similarity-Based Placement for FPGA Incremental Design and Design Space Exploration. Integration, the VLSI Journal (In Review), 2011.

Chapter 1

Introduction

1.1 Overview

Field Programmable Gate Arrays (FPGAs) are programmable integrated circuits that can be reprogrammed after manufacture. The prime advantages provided by FPGAs are their fast manufacturing turnaround time, low start-up costs and ease of design that involves less financial risks [51]. However, since the size of FPGAs has reached the million gates level, modern FPGA designs suffer from long compilation time, and placement is one of the most time-consuming phases.

In a typical FPGA incremental design cycle, several iterations of synthesis need to be performed before delivering the final design. There are several phases of a design process in which iterative repetitions are common, including the initial checks of the Register Transfer Level (RTL) code, constraint verification, timing closure, and in-system debugging [20]. Each of these steps requires a time-consuming resynthesis of the FPGA design. Several methodologies have been devised to save the recompilation time [25, 33, 70, 2]. However, these methods either need to keep the internal node boundaries or have to use extra information (*i.e.* name matching), which are usually destroyed during aggressive resynthesis. Therefore, a faster and more effective tool is required for FPGA incremental design.

On the other hand, an FPGA design offers a variety of customizations by varying design parameters. Those parameters include decisions at the algorithm level (*e.g.*, Simple Instruction Multiple Data (SIMD) or pipeline) or at the architecture level (*e.g.*, cache and bus structures); options at high-level synthesis (*e.g.*, scheduling and resource binding tradeoff) or combinations of various logic synthesis and optimization (*e.g.*, the Berkeley ABC toolset [9]). Efficiency of a design space exploration tool is of paramount importance for designers to quickly identify a small set of favorable design parameter combinations (*i.e.*, configurations) for a multi-objective design. Some methods have been devised to minimize the number of configurations to be evaluated in the design space exploration [18, 29, 8, 42]. However, FPGA application designers still heavily rely on the general CAD tools (*e.g.*, Altera Quartus or Xilinx ISE) to generate every single configuration due to the lack of efficient FPGA design space exploration tools.

A unique property shared by both the incremental design problem and the design space exploration problem is the existence of *similarities* between netlists of different circuits (configurations). For the incremental design problem, functional changes or optimizations in RTL or the logic level are small, and they generally result in a "similar" topology of the modified netlist compared with the original one [52]. For the design space exploration problem, such similarities include both *local similarity* and *global similarity*. Local similarity is due to the use of common primitives (*e.g.*, Digital Signal Processing (DSP) modules or macros) in different configurations. Global similarity exists because the characteristics of the application are shared by all configurations that implement it.

1.2 Thesis Statements

In this thesis, we propose a circuit similarity algorithm and put forward a novel Circuit Similarity-Based Placement (CSBP) framework. With the significant speedup for placement runtime and satisfactory placement quality, we apply CSBP to accelerate FPGA incremental design and design space exploration, respectively. We address the following hypotheses in this thesis:

- Hypothesis 1: The proposed circuit similarity algorithm can efficiently detect the intrinsic topological similarities between circuits (configurations) in both logic level (including both mild and aggressive resynthesis) and algorithm level.
- Hypothesis 2: The proposed CSBP flow can significantly accelerate the FPGA placement process with comparable placement quality compared to the state-of-the-art VPR tool.
- Hypothesis 3: The proposed CSBP can be applied to accelerate and facilitate the FPGA incremental design process and FPGA design space exploration process.

1.3 Contribution of this Research

In this thesis work, we use *circuit similarity*[1] to accelerate placement in incremental design and exploration of the design space. To quantitatively represent such a similarity, the proposed circuit similarity algorithm employs *graph similarity* [75], a widely applied technique in social network analysis [44] and chem-informatic domains, to measure the topological similarity of two circuits. We present an iterative algorithm to compute the circuit similarity (both local and global) between the modified and original netlists, and identify the correspondence of nodes/edges. Based on such circuit similarity, the placement and routing of the modified netlist can be derived from the layout of the original netlist obtained in the previous iteration. Unlike many existing algorithms [33, 2], which require radical changes to existing CAD tools, our approach simply inserts a plugin to the existing CAD tools and preserves the "push-button" feature in the commercial FPGA CAD tools.

[1] Note that the same notion of "circuit similarity" was used in [32], but it was defined based on the Boolean functions of logic gates in a circuit.

The proposed circuit similarity CAD flow first produces an initial placement and routing solution for the modified logic-level netlist (*e.g.*, functional changes, optimizations or design parameters changes), based on the layout of the original netlist and the circuit similarity between the original and modified logic-level netlists. Based on this initial solution, an efficient refinement is then performed as a fine-grain tuning for further improvement of the layout quality. Note that such a refinement procedure does not require a new implementation since the existing placement and routing tools can be used with lower optimization strength (*e.g.*, lower initial temperature in the simulated annealing-based placement or fewer iterations in the negotiation-based routing). The essential information obtained from the previous design iterations is automatically captured and quantified by a runtime-efficient "similarity detection" phase. We also take advantage of such "similarity" property to perform an efficient yet accurate estimation of the design space.

We have applied the proposed Circuit Similarity-Based algorithm to Placement, namely CSBP, and verified its effectiveness on incremental design for FPGAs and design space exploration for FPGAs, respectively. For incremental design, we have used CSBP to generate the placement for a logically resynthesized netlist (*i.e.*, both mild resynthesis and aggressive reynthesis) based on the placement of the original netlist and the circuit similarity between the original and the modified logic-level netlists. Tested on the 20 largest MCNC benchmark circuits [73], experimental results show CSBP produces a much higher quality initial placement than VPR's [57] initial placement in terms of bounding box costs and delay costs on both mild "imfs" resynthesis and aggressive "rwsat2" resynthesis. Our CSBP is 31X faster on average than the VPR placement, while producing comparable wire length and estimated critical delay. For design space exploration, we have performed experiments at the logic-level and algorithm-level, respectively. At both levels, experimental results show that CSBP captures the characteristics of the design space with accurately estimated wire length and critical delay, and pinpoints the best designs. Therefore, the design highlights and optimization efforts from the previous designs are preserved by CSBP, and can be reused in future design iterations, thus reducing the engineering effort in the FPGA cycle. Moreover, CSBP achieves an average of 30X speedup over VPR's placement.

Two conference papers resulting from this work have been published [66, 65], and one journal paper is under review [67]. One report of invention is being filed for U.S. patent. One open-source software, circuit similarity-based placement toolset including a circuit similarity placement tool and a circuit similarity visualization tool, is developed and publicly available for download[2].

1.4 Thesis Outline

The thesis proceeds as follows. Chapter 2 provides preliminaries and background for this work. This chapter briefly reviews FPGAs and the major steps in a typical FPGA CAD flow, including synthesis, placement, routing and timing and delay analysis. This chapter also reviews the fundamentals of

[2]Download address: http://webdocs.cs.ualberta.ca/ xshi/soft.html

graph similarity algorithms and two case studies we present in this thesis: incremental design and design space exploration. Chapter 3 illustrates the circuit similarity-based CAD flow with a motivating example and describes the circuit similarity algorithm in detail. Chapter 4 presents two case studies using CSBP for incremental design and design space exploration, respectively, and experimentally demonstrates the efficiency and the effectiveness of the proposed circuit similarity algorithm. The thesis concludes in Chapter 5. Appendix A lists the resynthesis optimization commands used in ABC toolset.

Chapter 2

Preliminaries

2.1 Overview of FPGAs

Field Programmable Gate Array (FPGA), a programmable integrated circuit, has gained great popularity in the circuit design since its first introduction in 1984. The key advantage provided by FPGAs is their reconfigurable ability compared to Application Specific Integrated Circuits (ASICs), where a device is custom built for a particular design. By properly programming, FPGAs can implement any functionality of ASICs without sending the chips to manufacturers for re-fabrication. Therefore, FPGAs can significantly reduce the manufacturing turnaround time, save start-up costs and simplify the design that involves less financial risks [51].

An FPGA chip consists of Input/Output (IO) blocks and other core programmable blocks. An island style FPGA architecture is shown in Figure 2.1. The generic structure of island style FPGA includes four main parts: Configurable Logic Blocks (CLBs), which are the basic logic blocks, implement the logic functions of the circuit. IO blocks are the connections of FPGA and external devices. The connection block is used to connect a CLB to the routing channels while the switch block is used to connect the routing channels [58].

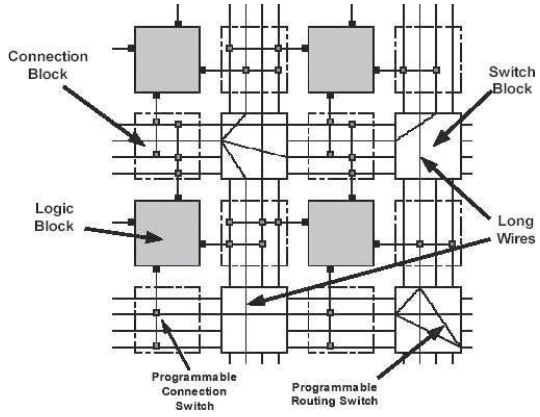


Figure 2.1: An island-style FPGA architecture [58].

FPGA programmability carries a price. In contrast to ASICs, the circuit speed of FPGAs is gen-

erally slower and the circuit area of FPGAs is generally larger. Moreover, since the size of FPGAs has reached the million gates level, modern FPGA designs suffer from long compilation time, and placement is one of the most time-consuming phases. Extensive studies have been performed to search for better FPGA architectures in order to reduce the speed and area penalties, and improve the efficiency of the placement as a single synthesis phase [57, 62, 71].

2.2 Overview of FPGA CAD Flow

The Computer Aided Design (CAD) or design automation process for FPGAs plays a critical role in the field of modern FPGA design technology. With the rapid growth of the FPGA size, implementing a circuit normally requires millions of programmable switches or configuration bits to be properly set [58]. It is extremely difficult for Integrated Circuit (IC) designers to manually trace down all the states for each switch or gate. Thus, CAD tools are devised to make the design process easier and traceable for IC designers. More specifically, FPGA IC designers usually use high level hardware description languages, such as VHSIC[1] Hardware Description Language (VHDL) or Verilog [17], to implement the circuit functionalities. In order to process the highly abstracted hardware descriptions, CAD tools are used to convey those high level designs into a netlist, which is a formatted file consisting of basic Programming Logic Blocks (PLBs) specifying the states of each programming logic units and the connections among them. The conversion from hardware descriptions to netlists is called synthesis, which can be further broken down into three sub-phases: technology optimization, technology mapping and packing as shown in Figure 2.2. After synthesis, the FPGA circuit designs can be then placed and routed into particular FPGA architectures. Finally, timing and delay analysis can be performed before downloading the designs to the FPGA circuit boards. In the following subsections, we will briefly describe the fundamentals and summarize the state-of-the-art techniques of synthesis, placement, routing and timing and delay analysis, respectively.

2.2.1 Synthesis

Synthesis phase is a crucial step in FPGA CAD flow. First, synthesis tools automatically map a FPGA circuit design that is described using a high level hardware description language to a library of simple gates. This gives IC designers freedom to choose among different implementation options. For the synthesis methods, it can be divided into two categories: sequential synthesis and combinational synthesis. In this work, we focus on combinational synthesis, which the presence of latches or flip-flops are explicitly specified by IC designers. There are extensive studies on synthesis methodologies for homogeneous FPGAs, but very limited methods are proposed for heterogeneous FPGAs [27, 60, 4]. Therefore, an emerging trend of research is to speedup the synthesis process for heterogeneous FPGAs with mixed size Look-Up Tables (LUTs). For example, using functional and

[1] VHSIC stands for Very-High-Speed Integrated Circuits, which was a U.S. government program aiming to develop high speed circuits [16].

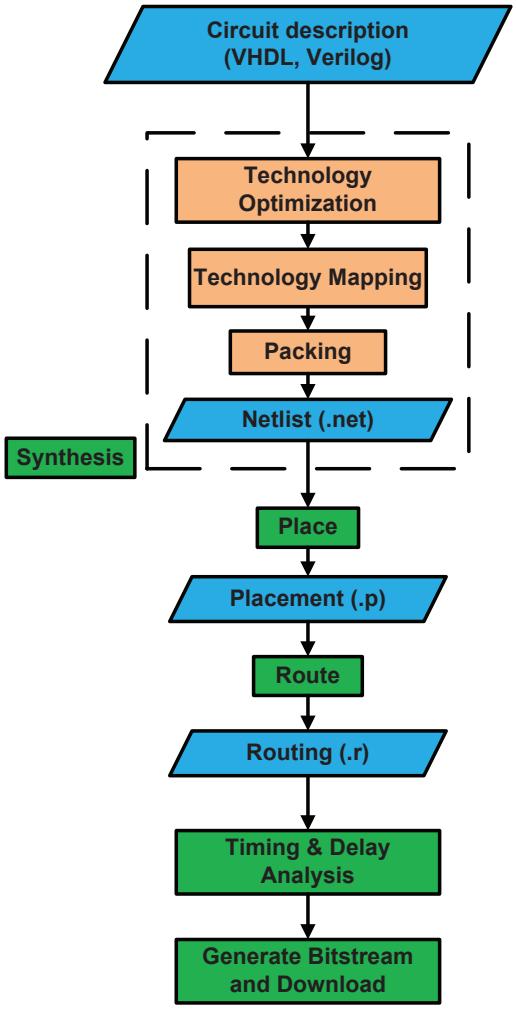


Figure 2.2: Overall CAD flow for FPGAs.

architectural symmetries in SAT-based Boolean matching to speedup heterogeneous FPGAs [69]. The goal of the synthesis phase is to minimize the number of logic blocks that are needed to implement the circuits and to maximize the circuit speed. Synthesis itself is complicated, enough to be further divided into three sub-phases: technology optimization, technology mapping and packing.

Technology optimization

Technology optimization attempts to generate an optimal abstract representation of the logic circuits by removing redundant logic and simplifying logic as much as possible [6, 48]. There are two main operations performed in technology optimization: network restructuring and node minimization. Network restructuring operations include decomposition, extraction, factoring, resubstitution and collapsing [6]. Node minimization operations include logic algebraic simplification and node reduction using Boolean minimization techniques [48].

Technology mapping

Technology mapping maps the optimized netlist of basic gates into a restricted set of circuit elements. There are generally three FPGA architectures, namely LUT-based logic blocks, multiplexers and wide AND/OR arrays. In this work, we only consider the most popular LUT-based[2] logic blocks architectures since previous research suggests that LUTs are an area-efficient method of implementing combinational functions [49]. In this case, technology mapping maps the optimized netlists into LUTs based on the logic Boolean functions. Technology mapping for LUT-based FPGAs can be further divided into two major categories: library-based technology mapping and direct approaches. LUT-based technology mapping is an active research area and both categories have been extensively studied by previous research [49, 48, 31, 27], so we would skip the details of these techniques and heuristics.

Packing

Packing groups several LUTs and registers (or flip-flops) into one logic blocks after technology mapping phase. Given certain circuit specifications, such as the maximum number of LUTs a logic block can contain and the number of input/output signals a logic block can contain, the essence of packing phase is to optimize the inner connections between LUTs so that the number of logic blocks needed is minimized and the number of signals needed to be routed among blocks is reduced. Packing problem can be classified as a form of clustering problem, which is essentially the same problem as partitioning problem [58]. Partitioning uses a top-down hierarchical design methodology to divide the circuits into smaller, more trackable components while clustering uses a bottom-up design methodology to group the basic components (*i.e.* LUTs in this context) into different logic blocks. The partitioning method has been extensively studied and can be divided into four major categories: move-based approaches, geometric representations, combinational formulations and clustering approaches [12]. The clustering method has also been extensively studied, among which spectral and labeling methods are the top popular two [58, 34, 21].

2.2.2 Placement

Placement in FPGA determines the physical locations and inter connections of each logic block in the circuit design, which now becomes one of the bottlenecks of the CAD processes. Since this work essentially focuses on the placement engine, the classic placement methods that have been used for the past two decades along with some modern placement techniques in the last 5-10 years are thoroughly reviewed. More specifically, these placement methodologies are grouped into four different categories, namely simulated annealing approach, min-cut approach, quadratic approach and parallel approach. The methodology of each algorithm is briefly described, with an emphasis

[2] A K-LUT device consists of K inputs, one output, and 2^K configuration bits that serve as truth table entries. Therefore, a K-LUT can implement any K-input function by probably configuring its 2^K bits.

on the comparison of performances and evaluation of its advantages and disadvantages. In particular, simulated annealing approach is stated in detail since the circuit similarity-based placement engine in this work is based on simulated annealing. Generally in the placement step, the netlist of logic blocks is placed into FPGA circuit [58]. The optimization goal of placement is to place the blocks in a proper location so that the objective function is minimized. There are three common optimization criteria for placement, time-driven requirement, wire-length-driven requirement and path-driven requirement. Time-driven placement attempts to minimize the delay in the circuit while wire-length-driven placement targets to minimize the total wire used. Path-driven placement focuses on trying to put the logic blocks on the critical path of the circuit so that both timing and wire length can be optimized. New challenges have emerged since the size of FPGAs has reached million gates level. The design and development using FPGAs suffer from a significant amount of placement time as turnaround time is crucial. For the next generation of CAD tools for FPGAs, fast and quality placement methods are critical and highly demanded.

Simulated annealing placement

An annealing process allows molecules to cool down in a controlled manner by temperature in order to find their best fit in the system. Simulated annealing placement mimics the annealing process used to gradually cool down molten metal to high quality metal objects. An initial placement is created by randomly placing the logic blocks in the circuit. A large number of block swaps is made to gradually reduce the cost. In this sub-section, the well-known Versatile Place and Route (VPR) tool using simulated annealing is reviewed [57].

The simulated annealing algorithm starts with random "move", which swaps the logic blocks [58]. Multiple cost functions can be defined to address different design specifications, such as wire-length driven criteria, time-driven criteria and path-driven criteria. The cost functions can also be a linear combination of those criteria in a reasonable computation time so that all the criteria can be considered at the same time [57].

Algorithm 1 Pseudo-code of similarity annealing-based placer [58].

```
S = randomPlacement();
T = initialTemperature();
R_limit = initialR_limit();
while (exitCriteria() == false) do
  while (innerLoopCriteria() == false) do
    S_new = generateViaMove(S, R_limit);
    ΔC = Cost(S_new) - Cost(S);
    r = random(0, 1);
    if (r < e^(−ΔC/T)) then
      S = S_new;
  T = updateTemperature();
  R_limit = updateR_limit();
```

As shown in Algorithm 1, simulated annealing starts with a random placement of each logic block in the circuit. After the initial placement, a certain number of moves are performed per temperature based on the following [58]:

$$\text{MovesPerTemperature} = InnerNum \times (N_{blokcs}^{4/3})$$

where InnerNum can be controlled by the user and its default value is 10. After each swap, a move is either accepted or rejected depending on whether the cost is reduced or not at certain temperature. If the cost decreases, the move is always accepted. However, if the cost increases, there is still probability for the move to be accepted. The probability is given by $e^{-\Delta C/T}$, where ΔC is the change in the cost function that the move courses, and T is the current temperature. This hill-climbing ability allows simulated annealing method not to converge to local minima and thus to reach global optimization [57].

A good annealing schedule is essential to the final results. With the motivation of increasing the amount of time spent at temperatures where a significant of moves are being accepted, the following temperature update schedule is used in VPR:

$$\text{T}_{\text{new}} = \alpha T_{old}$$

where α is defined as shown in Table 2.1. Note that $R_{accepte}$ is the percentage of the move that has been accepted at the previous temperature.

R_{accept}	α
$R_{accept} > 0.96$	0.5
$0.8 < R_{accept} \leq 0.96$	0.9
$0.15 < R_{accept} \leq 0.8$	0.95
$R_{accept} \leq 0.15$	0.8

Table 2.1: Temperature update schedule

Even with a good annealing schedule, millions of block swaps still need to be evaluated at each temperature. The most time consuming and computationally intensive part is calculating the cost change caused by the swap. It is crucial to speed up this part as fast as possible. VPR applies some heuristics to optimize this process, such as using incremental net bounding box update and a changing range of distant limit [58].

There are several advantages of the simulated annealing-based placer. Firstly, it outperforms the other placers as long as direct comparisons can be made [57]. The FPGA CAD tool VPR, which uses the simulated annealing method, has become the state-of-the-art tool in the research community. Secondly, simulated annealing placer has an open cost function which can be defined as either wire-length-driven, time-driven or path-driven. The cost function can also be the linear combination of

the above types though the weights need to be carefully chosen. Thirdly, simulated annealing can achieve global minimum due to its hill-climbing ability. However, simulated annealing is very slow because of its computationally expensive and time consuming evaluation of each move. In addition, due to the inherent sequential nature of simulated annealing, it is very hard to run it in parallel using multi-core CPUs or clusters.

Quadratic placement

Quadratic placement method uses the squared wire length as the objective function. It tries to minimize the cost by solving the linear equations deprived from the models [74]. Although quadratic placement only considers the squared wire length, it can efficiently finish the placement process with nearly no quality lost. As a result, quadratic placement is widely used in the Very-Large-Scale Integration (VLSI) placement [74].

Different cost functions can be defined as linear equations in terms of wire length. One of the classic algorithm proposed in [74] can be divided into three stages.

- In stage One, by repeatedly building up, modifying and solving linear equations, a good initial placement can be obtained. This stage is performed until no significant improvement can be achieved.
- In stage Two, instead of building and solving linear equations, nodes can be directly moved to reduce the total wire length since stage one has already given a reasonably good initial placement. The process in stage two is much faster than stage 1 so more iterations can be performed to achieve a better refinement.
- In stage Three, simulated annealing can be used to further refine the placement with low temperature.

The main advantage of the quadratic placement technique is that it significantly reduces the runtime with almost no quality lost compared to VPR. According to the results reported in [74], among the 20 MCNC benchmark circuits [24], Quadratic Placement for FPGAs (QPF) runs 5.8 times faster than VPR on average while the wire length obtained by QPF is only 1.9% more than VPR. By using better algebraic method to solve the linear equation, the run time could be further reduced. However, since the squared wire length is the only factor considered in the objective function, the timing part of the placement can not be shown in the quadratic placement.

Min-cut placement

Partitioning-based placement algorithms have been fast and hence scalable for large ASIC placement and have also been applied to FPGAs. One of the recent partitioning-based placement methods, the min-cut placement, recursively applies bi-partitioning to map the netlist of a circuit into the FPGA

layout region. It minimizes the number of cuts of the nets, leaving the highly connected logic blocks in one partition [14].

Delay optimization is very important in circuit design. Effective delay minimization on large circuits is possible only by accounting for performance as early as possible in the design flow. Min-cut placement targets delay minimization on the placement stage, which is an early step in the CAD design process.

The min-cut placer employs the fundamental divide-and-conquer method. A circuit is recursively bi-partitioned in a breadth-first manner. The cut direction (horizontal or vertical) is determined based on the criticality of the nets crossing the four borders so that the total cut numbers are minimized [14]. This recursive process is repeated until each partition contains only a few blocks to group the highly connected blocks together so that the placement cost is decreased. The goal of min-cut is to find a proper partition that cuts fewest wires in the net.

All the edges in the net are weighted with timing criticality, as well as terminal alignment of critical nets [14]. The algorithm can be divided into three steps. In the first step, min-cut uses the cutting-edge multilevel partitioner hMetis [63] as its partitioning engine. During the partitioning process, a tight connection between the circuit graph and placement is maintained, which represents coordinates of all blocks on the FPGA fabric. Recursive partitioning is done until each leaf partition has only a few blocks. In some cases, some leaf nodes might contain more nodes than it can accommodate, so overlaps must be removed. In step two, overlaps are removed by using a greedy technique, which moves blocks to the closest best aligned partition. Finally, the placement is refined by using a low temperature simulated annealing method to further minimize the delay.

The advantage of the min-cut placement technique is that it minimizes the delay in the placement stage, which lays the foundation for designing a better performance circuit. In addition, the run time reported in [14] shows that an average 3-4 times speed up is gained compared to VPR on 20 MCNC benchmarks with a slight degradation in the quality. However, the results of min-cut rely on how well the partition is performed. Current research is focused on finding some heuristics to better partition the circuit. Also, min-cut placer may not be able to reach the global minimum because of some of the greedy strategies it uses.

Parallel placement

As the scale of modern FPGAs has reached millions of logic blocks, more efficient and scalable FPGA placement algorithms are needed. Parallelization is an appealing solution for providing fast placements due to the rapid development of multi-core CPUs in recent years. For the parallel placement approaches, we focus on the simulated annealing-based methods since they outperform the other placers. The main drawback is their slow runtime. We divide modern simulated annealing based parallel FPGA placers into three categories: parallel move approach, area based approach and deterministic parallel approach.

A: Parallel Move Approach Since there are a myriad number of moves at each temperature, the motivation of the parallel move approach is trying to accelerate the simulated annealing process by performing several moves at the same time. There are three possible cases after each move is done. **(1)** the move is accepted, and two blocks are swapped **(2)** the move is accepted and a block is moved to an empty location **(3)** the move is rejected. Moves can be done in parallel only if they do not move the same block or move to the same location. However, ensuring the above can only guarantee there are no move collisions. Net cost collision might still happen when two moves that move blocks of the same net may evaluate the bounding box incorrectly as each move can not take into account the fact that the other move is changing the bounding box cost.

Generally, there are two ways to deal with the move collision and the net cost collision:

- Ignoring the errors in the cost function is the easiest way to resolve collisions, but it has negative effects on the accuracy of the cost which interferes with the acceptance of moves, hence adversely affects the results.
- Finding the disjoint moves is an alternative way that not only moving different blocks, but also belonging to different nets. However, the over restricted moves result in a smaller swap space and the synchronization overheads tend to overwhelm the gain in parallelism.

Both of these two methods show negative speedups [7]. This is due to the overhead of synchronization outweighs the advantages of parallelization. Nevertheless, the thought of trying to parallelize the moves inspires many other parallel FPGA placement methods.

B: Area Based Approach The area based approach is motivated to solve the collision illustrated in the parallel move approach by partitioning the area of FPGA and assigning the partitioned areas to different processors. The whole circuit is partitioned into several parts, and each processor is in charge of one partition.

The moves evaluated are much less restricted compared to the move parallel approach. However, collisions could still happen because multiple processors may move blocks belonging to the same net across the partition. These errors can be tolerated because the net collisions that span over two or more partitions rarely happen. Moreover, with cooling temperature, the swaps tend to occur between nearby blocks. Since each processor can only move blocks within its own partitioned area, to allow the placement to reach global minimum, the partition must be carefully chosen so that each block has the freedom to move to any arbitrary location in FPGA. The area based approach uses both horizontal and vertical partitions to ensure the global minimum could be reached [7].

The experimental results show a non-linear speed up has been gained compared to the sequential placer and the cost is not degraded with the increasing processors. This is due to the less synchronization requirements.

C: Deterministic Parallel Approach One of the constrains of parallelism is the non-determinism of

the results. This constrain is seldom studied in the past work (an exception is [10]), but is vital in a commercial context for the following two reasons [62]:

- When IC designers use a commercial FPGA placement tool, they must be able to reproduce the problem when a bug is reported. Non-determinism makes this extremely difficult because the results are varied at each run.
- In the release testing stage of building a placer, it would be terribly difficult to look into failing tests since the results changed randomly.

The algorithm proposed in [62] parallelizes the placement while keeping the results deterministic. The deterministic parallel approach partitions a move into two stages: processing and finalization. As shown in Figure 2.3, during the processing stage, each processor proposals a move and evaluates it. This takes the vast majority of time and thus occurs in parallel. In order to avoid collisions and maintain the deterministic property, the calculated moves are put into a queue and a dependency checker is needed to ensure there is no collision. If collision occurs, it will re-propose the moves that have collided. Note that the finalization part can be done by any of the idle processor. In our example shown in Figure 2.3, $C0$ is idle when all the moves in queue have been checked, thus $C0$ perform the finalization job.

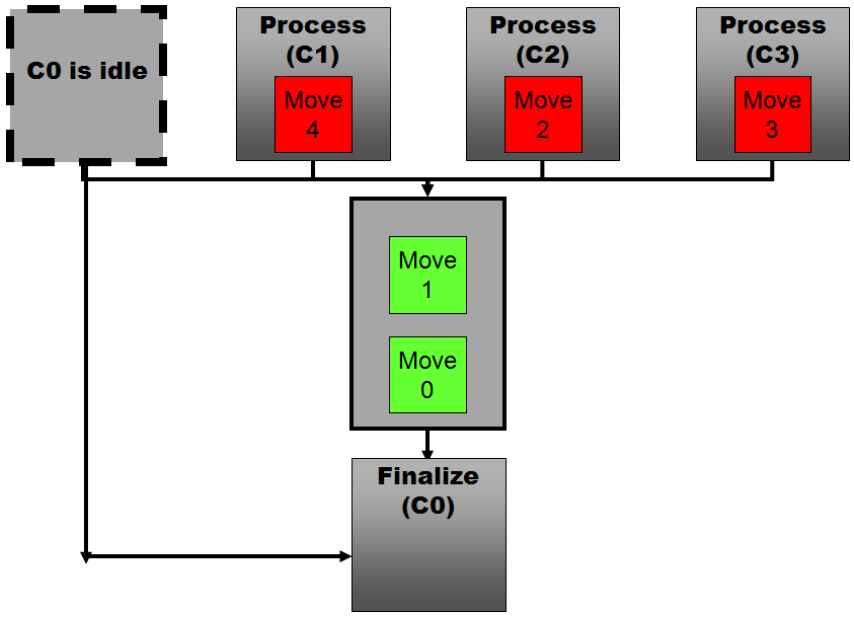


Figure 2.3: An illustration of the deterministic parallel approach.

There are several advantages of the deterministic parallel approach. Firstly, speedup can be linear given the assumption that the finalization time is negligible. Secondly, a move is now processed entirely by one processor, which improves the memory locality. Thirdly, the results are deterministic and serial equivalent, which tremendously benefits the IC design process.

Incremental placement

The time spent on placement is still a dominate part for the entire FPGA complication process, especially with the size and logic capacity of FPGAs increasing dramatically [15]. Moreover, the recompilation time of the entire circuit design for small changes or localized improvements is also time-consuming, taking hours or even days to re-execute the entire CAD flow [36]. Therefore, incremental placement is devised as a more scalable and efficient placement methodology. In most of the multi-iteration design processes, changes are generally small so that incremental placement is able to speedup the compilation process by considering only the changed part as opposed to the traditional full compilation of the entire FPGA CAD flow.

Incremental placement for FPGAs is a new and challenging field and not so much work is published thus far. In general, most incremental placement algorithms keeps a "reference placement" from previous iterations and a list of modified logic blocks, including the removed ones and newly added ones. The unmodified logic blocks are normally replaced into their previous locations. The newly added logic blocks[3] are overlapped onto the physical location of the existing logic blocks in order to maintain placement locality while a further step of "overlap re-legalization" is needed to fix those overlapped blocks [36, 43].

We will briefly mention several most cited work about incremental placement. The Incremental Placement (ICP) engine is primarily focused on improving timing through small logic level netlist changes [19]. ICP tries to shift the non-critical logic elements in the reference placement to satisfy the preferred locations requests targeting numerous FPGA architectures. Incremental Physical Resynthesis (IPR) presents a new way to optimize timing by utilizing a series types of physical optimizations, such as cell repacking, signal re-routing and logic restructuring, with a quadratic initial placement and a future overlap re-legalization process [45]. "iPlace" algorithm is proposed as an efficient incremental placement method based on shifting, compaction and annealing [36]. "iPlace" first places the new design into a super-grid which is larger than the previous FPGA physical size, followed by a compaction scheme to re-legalize the illegal blocks. "iPlace" ends with a low temperature simulated annealing as a further refinement.

In this work, we propose an enhancement for incremental placement using circuit similarity [66]. We will discuss the details of our method in Chapter 3.

2.2.3 Routing

Routing is one of the most basic yet important phases in FPGA design [15]. Similar to ASIC routing problem, FPGA routing configures which programmable switches should be switched on or off in order to connect all the logic blocks subject to timing and other constraints [58]. However, FPGA

[3]There are two cases regarding to the number of the newly added logics blocks. If the number of newly added blocks is less than the number of the removed blocks, the problem is considered "trivial" by replacing the deleted ones with the newly added ones. We emphasize on the other case, where the number of newly added blocks is larger than the number of the removed blocks.

routing is more restricted and challenging compared to ASIC routing since it can only use prefabricated routing resources, such as limited wire segments, programmable switches and multiplexers.

A routing-resource graph is often created as an abstract representation of the FPGA routing architecture [35]. Each vertex in the routing-resource graph represents a wire segment or a logic block pin and each edge represents the programmable switch that connects the two vertices. The routing-resource graph can be either modeled as a directed graph [54] (*e.g.*, connecting the bi-directional switch, such as a transistor, with directed edges) or as an undirected graph [37](*e.g.*, connecting the unidirectional switch, such as a buffer, with undirected edges).

In essence, the optimization goal of routing problem is trying to find a shortest path in the routing-resource graph between the nodes without exceeding the maximum number of wire segments. Moreover, an additional optimization goal is to make nets on or near the shortest path fast [58], which is called time-driven whereas delay-driven only considers about routability.

Most of the FPGA routers use a two-step routing algorithms, with a combination of global routing as the first step and detailed routing as the second step. Global routing determines the coarse routing topology of each net in terms of logic block pins and channel segments while detailed routing determines the wire each net uses in the routing channel or region [15, 58]. The combined global-detailed routers have the potential to reach the maximum optimization and significantly simplify the generally routing problem, which is NP-hard [15].

In terms of popular open-source academic FPGA routing CAD tools, up to date, the VPR router is the most successful one that uses the PathFinder negotiated congestion-delay algorithm, which is a combined global-detailed router [13]. VPR router iteratively rips-up and re-routes every net in the circuit until all congestion is resolved. Another important reason for VPR routing tool being widely recognized is that it provides the IC designer flexibility to specify a simple yet reasonable set of FPGA architecture parameters and generates the corresponding routing-resource graph automatically.

2.2.4 Timing and Delay Analysis

Timing and delay analysis is the last step before bitstream generation in the FPGA CAD flow shown in Figure 2.2. Timing analysis is a design automation program that provides an assist for debugging timing issues in hardware [50]. Timing analysis is mainly used for two cases: determining the circuit speed for fully placed and routed circuits and estimating the slack of each source-sink connections during various CAD phases (mostly used in placement and routing) in order to not slow down the overall circuit performance [58].

Delay analysis computes the delay of a routed circuit from a net source to any of its sinks. Delay analysis is primarily used for two purposes: determining the speed of a routed circuit and analyzing the delay of different net topologies during routing [58]. Several delay models have been proposed to meet the requirements of delay analysis. For instance, the Prefield-Rubinstein delay model was

used to determine an upper and lower bound on the a RC-tree to each net sink [38, 30]. Alternatively, the Elmore delay model was defined in a combination of a common model of buffer delay in order to allow its use with circuits that contain buffers as well as resistors and capacitors [64], which is considered as the most widely used delay estimation model in the FPGA research community [28].

2.3 Graph Similarity

Given two graphs (or networks), there are multiple ways to define their similarity. The characteristics of commonly used measures of similarity are summarized in Table 2.2, where column "Global Topo" indicates whether a measure considers the global topological information, which is important to find the correspondence between nodes of two graphs. Some measures have already been used for FPGA design automation, *e.g.*, J. Cong *et al.* applied the edit distance measure to FPGA resource optimization [26]. Our circuit similarity algorithm employs the iterative method, which has relatively low computational complexity and considers the global topological information.

Table 2.2: Summary of notions of similarity.

Measure	Description	Time Complexity	Global Topo
Isomorphism [46]	Identifying a bijection between the nodes of two graphs which preserves (directed) adjacency.	NP-Hard	Yes
Edit distance [11]	Given a cost function on edit operations (*e.g.*, addition/deletion of nodes and edges), determine the minimum cost transformation from one graph to another.	NP-Hard	Yes
Common subgraph [39]	Identifying the 'largest' isomorphic subgraphs of two graphs.	NP-Hard	Yes
Iterative methods [59]	Two graph elements (*e.g.*, edges or nodes) are similar if their neighborhoods are similar.	Cubic	Yes
Statistical methods [47]	Assessing aggregate measures of graph structure (*e.g.*, degree distribution, diameter, betweenness measures).	Linear	No

Different algorithms, including similarity flooding [53], simRank [22], and the coupled node-edge [75], have been proposed to compute the graph similarity based on the iterative definition. In this work, we use an iterative graph similarity algorithm for molecular graphs [41], which takes advantage of graph sparsity, one of the properties of a circuit graph. Before presenting this algorithm, Table 2.3 describes all frequently used variables.

Table 2.3: Summary of variables in iterative similarity algorithm

Variable	Description
$X_{i,j}^{(n)}$	Similarity score between node i in graph $G(V)$ and node j in graph $G'(V')$ in iteration n
v_i	A node in graph G
v'_j	A node in graph G'
t	The iteration number
$n(v)$	The set of all adjacent nodes of node v
π	An injective map from $n(v_i)$ to $n(v'_j)$, if $\|n(v_i)\| < \|n(v'_j)\|$ An injective map from $n(v'_j)$ to $n(v_i)$, if $\|n(v_i)\| \geq \|n(v'_j)\|$
α	A weight constant within interval (0,1)
ε	A terminating threshold for iterations
M	An upper bound for number of iterations
$k_v : V \to V'$	A predefined inter-similarity between two nodes
$k_e : E \to E'$	A predefined inter-similarity between two edges, where (v_i, v) is an edge in graph G and $(v'_j, \pi(v))$ is an edge in graph G'
$in(v)$	The set of all adjacent nodes that have an edge entering node v
$out(v)$	The set of all adjacent nodes that have an edge leaving node v

The iterative similarity algorithm is summarized in Algorithm 2. In each iteration (t), the algorithm computes the *similarity score*, $X_{i,j}^{(t)}$, between each node pair (v_i, v'_j), where $v_i \in G$ and $v'_j \in G'$. The similarity score of a node pair is a real value between 0 and 1. The higher the similarity score of a node pair is, the more likely these two nodes are matched together. This score is updated based on the values of their adjacent node pairs obtained in the previous iteration and the predefined inter-similarity between two nodes/edges. The predefined similarity is used to capture non-topological

connections between two graphs. The algorithm terminates when the difference between the total similarity scores in two consecutive iterations is smaller than ε, or the number of iterations reaches an upper bound M.

Algorithm 2 Similarity of G and G'

Initialize $X_{i,j}^{(0)}$
while $|\sum X^{(t)} - \sum X^{(t-1)}| > \varepsilon$ and $t < M$ **do**
if $|n(v_i)| < |n(v'_j)|$ **then**

$$X_{i,j}^{(t)} = (1-\alpha)k_v(v_i, v_j^{'}) + \alpha \max_{\pi} \frac{1}{|n(v_j^{'})|} \sum_{v \in n(v_i)} X_{v,\pi(v)}^{(t-1)} k_e((v_i, v), (v_j^{'}, \pi(v)))$$

else

$$X_{i,j}^{(t)} = (1-\alpha)k_v(v_i, v_j^{'}) + \alpha \max_{\pi} \frac{1}{|n(v_i)|} \sum_{v^{'} \in n(v_j^{'})} X_{\pi(v^{'}),v^{'}}^{(t-1)} k_e((v_i, \pi(v^{'})), (v_j^{'}, v^{'}))$$

2.4 Incremental Design for FPGA

The key to incremental design methodology is *design preservation* [68], *i.e.*, maximally preserving and taking advantage of the engineering effort in the previous design iterations. A commonly employed method for design preservation is to partition a design and avoid a recompilation of unchanged partitions in the next iteration. This method can yield a significant reduction in iteration time but due to the strict hierarchical boundaries, synthesis cannot perform any cross-boundary optimizations where a partition exists. To break this hard hierarchy boundary constraint for improving the quality of the design, Xilinx SmartGuide [68] employs naming and local topological matching to identify the correspondence between two netlists resulting from the previous and the current iterations, respectively. Based on this correspondence, the layout from the previous iteration can be reused in the current iteration, therefore leading to better quality and saving the recompilation time. However, in modern synthesis algorithms (*e.g.*, ABC), the internal node boundaries are usually destroyed by structural hashing (transforming a logic network into an And-Inverter Graph (AIG)), and aggressive optimization and logic restructuring performed in the netlist make it difficult to produce the naming matching between the original and modified netlists. As shown in Figure 3.3, the naming matching-based correspondence does not work in this example since only two nodes (node 7 and node 8) out of nine internal nodes have the same names in the original and the modified networks. In contrast, the proposed circuit similarity algorithm employs the topological similarity detection technique and is able to identify a more comprehensive correspondence between the two networks.

2.5 Design Space Exploration for FPGA

For multi-objective optimization, a pareto-optimal point represents a design point or a configuration for which no other configuration is better in the objective function space. A common goal of design space exploration in multi-objective optimization is to find pareto-points, which benefit IC designers seeking to make appropriate design tradeoffs for given constraints. Pareto-point exploration is time-consuming due to the exponential increase of the number of configurations with regard to the tuneable parameters and the long runtime required by the current FPGA CAD tools. For example, current configurable soft cores (*e.g.*, Xilinx Microblaze cores) can have thousands of configurations [55] and the runtime for evaluating one configuration of Microblaze by Xilinx platform studio is about 15 minutes [18]. Obviously, a straightforward evaluation of all configurations for pareto-points is infeasible.

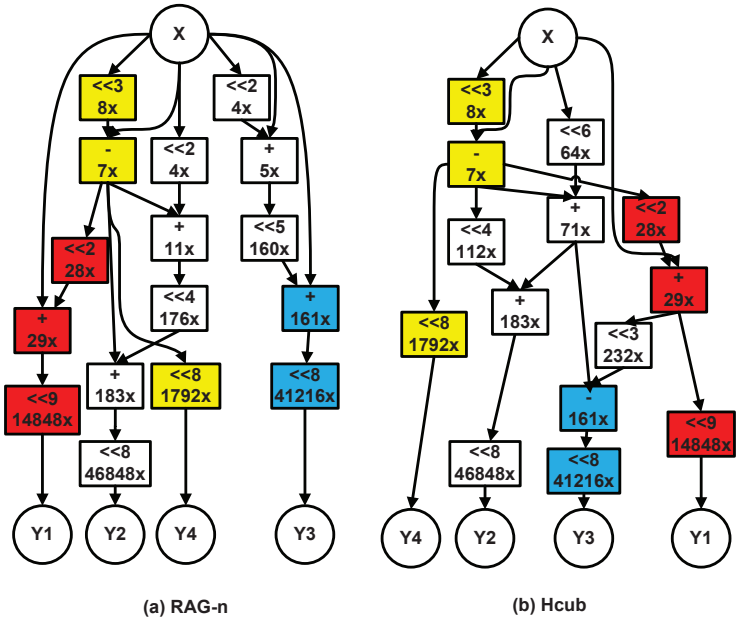


Figure 2.4: Constant multiplier blocks generated by CMU SPIRAL (integer constants: 58, 183, 161, 7; bit width is 8).

A key insight of the design space exploration problem is that there exists similarity among different configurations. This feature is illustrated by an example of an algorithm-level design space exploration problem with two implementation algorithms (*i.e.*, RAG-n [1] and Hcub [72]) of a constant multiplier block. The algorithm-level schematics of these two implementations (generated by CMU SPIRAL [40]) are shown in Figure 2.4. They both implement the following constant multiplier block:

$$Y_1 = 58 \cdot X, Y_2 = 183 \cdot X, Y_3 = 161 \cdot X, Y_4 = 7 \cdot X \tag{2.1}$$

where X is the input and $Y_1, \cdots, Y_4$ are outputs, and the precision (bit width) is 8 bits. Although there is a significant difference between the structures of these two configurations at the first glance,

they both use adders, subtractors and shifters as building blocks (primitives), which lead to a local similarity. When these algorithm level designs are mapped to FPGAs, such local similarity results in similarities of local clusters that contain LUTs or DSPs used to implement these primitives. In addition, both configurations generate the constant multiplication for equation (1), which results in global similarity. Specifically, the I/O (X and Ys) of both configurations are identical; there are identical internal structures (*e.g.*, subgraph $28x \rightarrow 29x \rightarrow 14848x$ is shared by both implementations) as highlighted in Figure 2.4 using different colors; both configurations are sparse Directed Acyclic Graph (DAG) structures, and they are topologically similar.

Besides pareto-points, which only characterize a set of "good configurations" in the design space, people may also be interested in finding the "shape" of the entire design space, which includes both "good configurations" and "bad configurations". The knowledge of "bad configurations" is helpful for algorithm developers to diagnose the design and for CAD tool designers to analyze the tool working flow. For instance, the combination of two logic optimizations may result in a poorer design than applying them individually. A full profiling of the design space will reveal such phenomenon and help CAD tool designers improve the tool.

2.6 Summary

In this Chapter, we first briefly reviewed FPGA and its key advantages compared to ASIC. Properly programmed, FPGAs can implement any functions of ASICs with lower cost and faster manufacture turnaround time. The main phases of the FPGA CAD flow are reviewed, including synthesis, placement, routing and timing and delay analysis. For each phase, we described the fundamentals and summarized both the classic and state-of-the-art techniques with a particular focus on placement techniques. Moreover, specific for this thesis work, we reviewed the details of the graph similarity algorithms and compared the pros and cons for each algorithm. We particularly focused on the iterative method used for molecular graphs which later in Chapter 3 we adapt it to a circuit similarity algorithm. We also reviewed the essential background for incremental design process and design space exploration process, which later in Chapter 4 we use as two applications to perform experiments on.

The next Chapter will present a motivation example of the circuit similarity algorithm and provide in depth details and applications of the proposed algorithm.

Chapter 3

Circuit Similarity

3.1 Motivating Example

Following the flow in Figure 3.1, we use an example to illustrate the procedures of circuit similarity flow. In the first design iteration, given a logic-level network G shown in Figure 3.2(a), where each node denotes a LUT and each edge denotes an interconnection between LUTs, the placement (Figure 3.3(a)) of network G can be obtained by performing a time-consuming and highly-optimized placement (*e.g.*, VPR). Suppose a change of RTL code is made due to optimizations or design parameter changes after the first iteration, and the RTL and logic-level synthesis is performed in the following iteration, resulting in a modified network, G', as shown in Figure 3.2(b). To produce the placement of network G', circuit similarity flow first computes the similarity between networks G and G', and finds the correspondence of nodes in these two networks (Figure 3.3(a) right). Based on such node correspondence, the initial placement (Figure 3.3(b)) of network G' can be determined using the placement of network G (Figure 3.3(a)), for example, if node V' in network G' corresponds to node V in network G, V' is assigned the same coordinates as node V.

Note that the detection of similarity and the correspondence of two networks is generally much faster than the replacement of the entire network. Therefore, circuit similarity flow is more efficient than the *from-scratch* design flow, which replaces the entire circuit.

Taking advantage of circuit similarity flow's efficiency, we also illustrate a logic-level design space exploration problem using CSBP as an example. The design parameters explored in this example are the logic synthesis options in the Berkeley ABC tool set [9]. The purpose of the design exploration is to identify the impact of different combinations of logic optimizations on the wire length and timing. MCNC benchmark "des" is used as the application to be implemented. The reference configuration is synthesized using the following ABC script:

b; rs; rs -K 6; b; rsz; rsz -K 6; b; rsz -K 5; b

where each command is a logic optimization in ABC. For example, "b" means balance the AIG and "rs" means the logic rewriting using Boolean substitution[1]. The placement of the reference

[1] A complete list of the ABC commands can be found at Appendix A

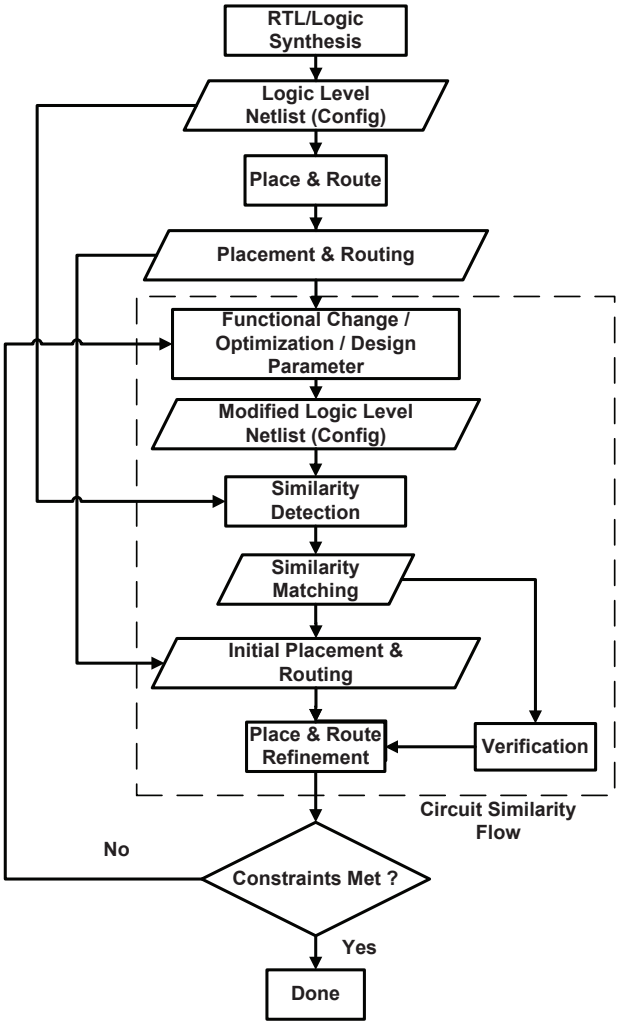
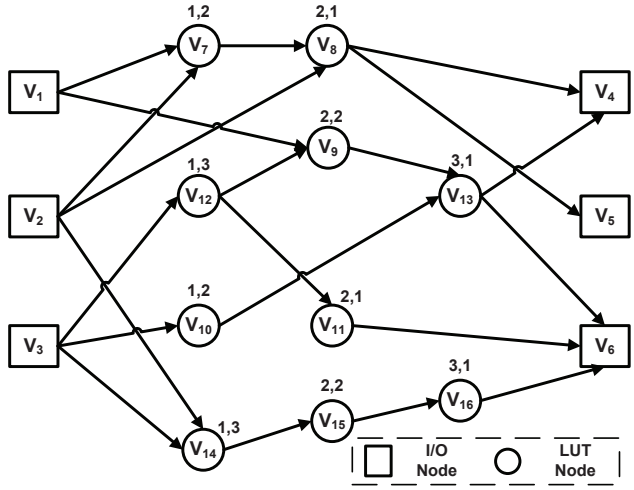


Figure 3.1: CAD flow using circuit similarity.

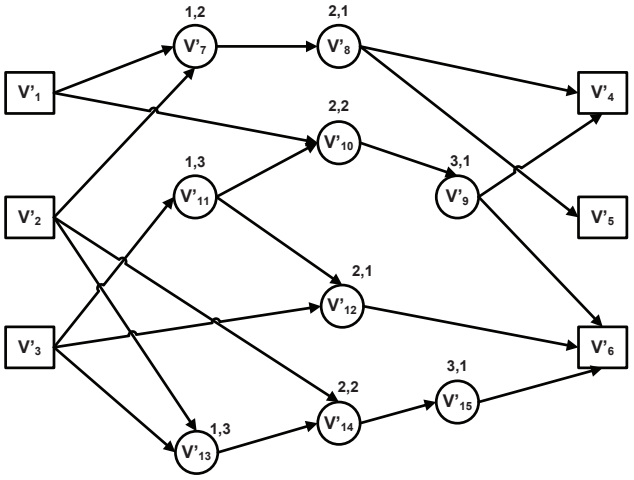
configuration is generated by VPR and the layout is shown in Figure 3.4(a) with nets toggled in VPR Graphical User Interface (GUI). Next we generate a new configuration, which is synthesized by the following ABC script:

st; rw -l; b -l; rw -l; rf -l; fraig; rw -l; b -l; rw -l; rf -l

The similarity of the netlists for the reference and the new configuration is obtained by a circuit similarity algorithm (detailed in Section 3.2). Based on this similarity, the layout of the initial placement is shown in Figure 3.4(b), which is obtained based on the placement of the reference configuration. As circled in the figures, CSBP captures the main characteristics of the topological similarity between the reference and the new configurations, and thus results in a well optimized initial layout. Given this initial placement, a low-temperature annealing process is used to refine the placement and final placement result is shown in Figure 3.4(c). For comparison, Figure 3.4(d) shows the layout of the random initial placement produced by VPR. Obviously, the initial placement generated by



(a) G, network before RTL code change



(b) G', network after RTL code change

Figure 3.2: Logic-level networks before and after optimization (the label above each node describes the level and reverse level of the node).

circuit similarity has significantly less wire length compared to the random placement. This shows that CSBP successfully finds the internal corresponding topologies of both netlists, and therefore makes a good decision on the relative position of Clustered Logic Blocks (CLBs) placement. It is also interesting to compare the highlighted topologies of the layouts of the final placement between the new configuration and the reference configuration (Figure 3.4(c) and Figure 3.4(a)). Although different logic optimizations are applied, the resulting layout shares similarities. The final placement produced by VPR is shown in Figure 3.4(e) for comparison. Table 3.1 shows the numerical comparisons of these layouts, where delay cost quantifies the delay of a route from a net source to any of its sinks. The placement generated by our similarity-aware algorithm results in comparable wire length, better critical path delay and less placement runtime compared to the placement generated by VPR.

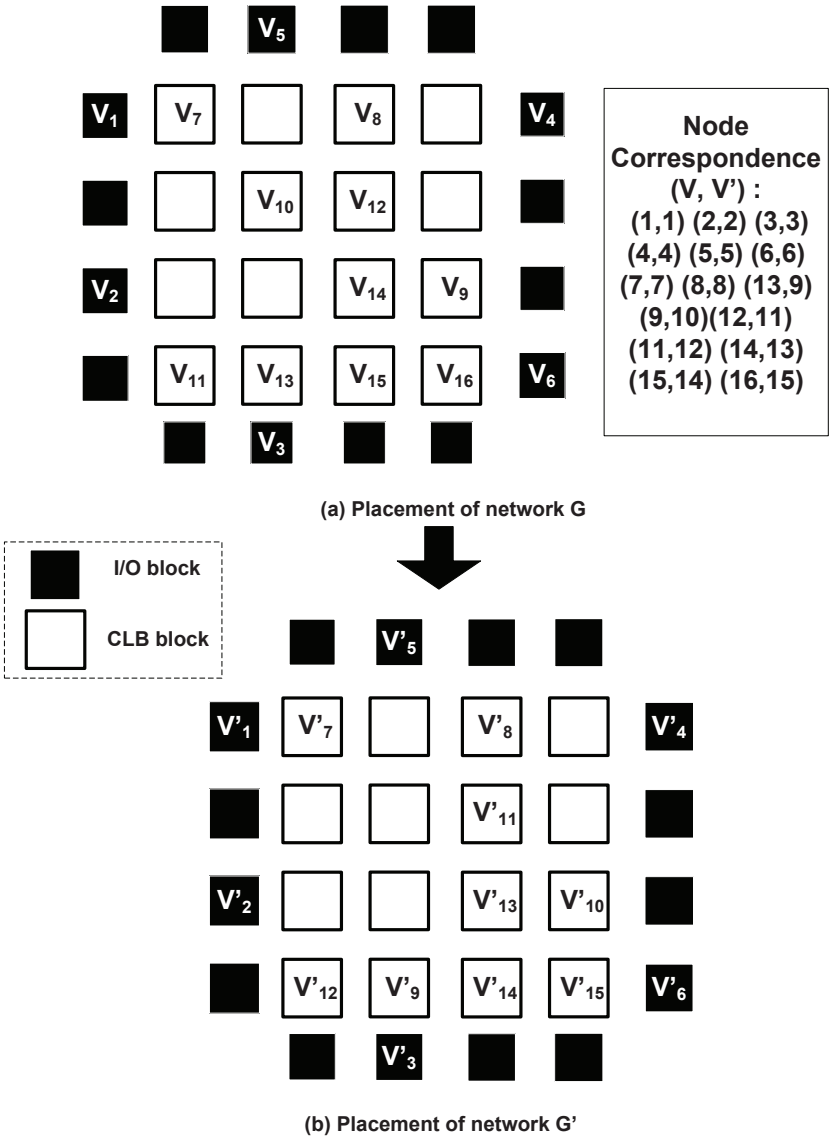


Figure 3.3: The placement of the original and modified networks.

3.2 Circuit Similarity Algorithm

As described in Chapter 2, Section 2.3, Algorithm 2 is designed for undirected molecular graphs [41], and the computational complexity is too expensive to handle real circuits. In this section, we first adapt Algorithm 2 to consider a directed circuit graph and then present two techniques to significantly improve both time and space efficiency of the circuit similarity detection.

One unique constraint for circuit similarity detection in incremental design is that the matching of the corresponding Primary Inputs (PIs) and Primary Outputs (POs) of the two circuits must be guaranteed. Therefore, the similarity score for a pair of corresponding PI/PO nodes is set to 1 and is not updated during the iteration. As a result, such a predefined PI/PO matching effectively provides extra hints for the iterative similarity detection process and generates better matching between the

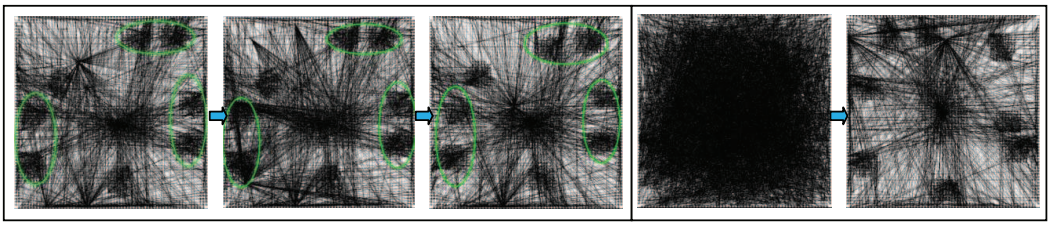


Figure 3.4: Placement results for circuit "des". (reference configuration has 1245 CLBs and 1501 nets, the new configuration has 1215 CLBs and 1471 nets)

Table 3.1: Status of layouts of Figure 3.4.

Layout	Wire	Delay cost	Critical delay	Runtime (s)
CS-init	306	5.93E-05	-	-
VPR-init	1087	1.40E-04	-	-
CS-final	237	5.08E-05	8.28E-08	13.38
VPR-final	221	4.98E-05	1.01E-07	28.42

two circuits. Intuitively, for those node pairs close to PI/PO nodes, higher scores will be obtained because of the propagation of the constant similarity score set in PI/PO node pairs. Note that other hints such as internal registers and naming matching information obtained in logic synthesis can also be used as the predefined matching to enhance both the quality and speed of the circuit similarity detection.

Algorithm 3 Circuit similarity algorithm of G and G'.

Initialize $X_{i,j}^{(0)} = 1$

while $|\sum X^{(t)} - \sum X^{(t-1)}| > \varepsilon$ and $t < M$ **do**

for each node in G **do**

for each node in G' **do**

if $|in(v_i)| \geq |in(v'_j)|$ and $|out(v_i)| \geq |out(v'_j)|$ **then**

$$X_{i,j}^{(t)} = (1-\alpha)X_{i,j}^{(t-1)} + \alpha\frac{1}{|out(v_i)|+|in(v_i)|}[\max_{\pi}(\sum_{v' \in out(v'_j)} X_{\pi(v'),v'}^{(t-1)}) + \max_{\pi}(\sum_{v' \in in(v'_j)} X_{\pi(v'),v'}^{(t-1)})]$$

else if $|in(v_i)| \geq |in(v'_j)|$ and $|out(v_i)| \leq |out(v'_j)|$ **then**

$$X_{i,j}^{(t)} = (1-\alpha)X_{i,j}^{(t-1)} + \alpha\frac{1}{|out(v_i)|+|in(v_i)|}[\max_{\pi}(\sum_{v' \in out(v_i)} X_{v,\pi(v)}^{(t-1)}) + \max_{\pi}(\sum_{v' \in in(v'_j)} X_{\pi(v'),v'}^{(t-1)})]$$

else if $|in(v_i)| \leq |in(v'_j)|$ and $|out(v_i)| \geq |out(v'_j)|$ **then**

$$X_{i,j}^{(t)} = (1-\alpha)X_{i,j}^{(t-1)} + \alpha\frac{1}{|out(v_i)|+|in(v_i)|}[\max_{\pi}(\sum_{v' \in out(v'_j)} X_{\pi(v'),v'}^{(t-1)}) + \max_{\pi}(\sum_{v' \in in(v_i)} X_{v,\pi(v)}^{(t-1)})]$$

else

$$X_{i,j}^{(t)} = (1-\alpha)X_{i,j}^{(t-1)} + \alpha\frac{1}{|out(v_i)|+|in(v_i)|}[\max_{\pi}(\sum_{v' \in out(v_i)} X_{v,\pi(v)}^{(t-1)}) + \max_{\pi}(\sum_{v' \in in(v_i)} X_{v,\pi(v)}^{(t-1)})]$$

For those internal nodes without predefined similarity, we replace k_v with $X_{i,j}^{(t)}$, and k_e with 1. Instead of updating similarity scores based on all the neighbors, we perform the update for edges that leave the nodes and edges that enter the nodes, separately. More specifically, as shown in Algorithm 3, given the two graphs, we initialize the similarity scores of all pairs of nodes to 1. In each iteration, for instance, if $|in(v_i)| \geq |in(v_j^{'})|$ and $|out(v_i)| \geq |out(v_j^{'})|$, the similarity score $X_{i,j}^{(t)}$ is updated as follows:

$$X_{i,j}^{(t)} = (1-\alpha)X_{i,j}^{(t-1)} + \alpha \frac{1}{|out(v_i)|+|in(v_i)|}[\max_{\pi}(\sum_{v^{'} \in out(v_j^{'})} X_{\pi(v^{'}),v^{'}}^{(t-1)}) + \max_{\pi}(\sum_{v^{'} \in in(v_j^{'})} X_{\pi(v^{'}),v^{'}}^{(t-1)})]$$

In our experiments, we find $\alpha = 0.75$ consistently produces a high quality matching. For the two circuit graphs in Figure 3.2, the obtained similarity score matrix is shown in Table 3.2 (PI/PO nodes are not shown). Clearly, the topologically similar node pairs (*e.g.*, node 7 in graph G and node 7 in graph G') have scores close to 1. This matrix describes a complete bipartite graph, where the weight associated with each edge denotes the similarity score of two nodes.We can now compute a maximum matching in this bipartite graph to obtain a node matching between the two graphs. The min-cost network flow [56] is used to compute the maximum matching in our experiments, and the resulting node matching is given in Figure 3.3(a) top right.

Table 3.2: Similarity score matrix for two graphs in Figure 3.2

	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}	V_{13}	V_{14}	V_{15}	V_{16}
V_7'	0.92	0.25	0.48	0.15	0	0	0	0.42	0.06	0
V_8'	0	0.73	0	0	0.05	0	0.39	0	0.17	0.06
V_9'	0	0.39	0	0	0.4	0	0.73	0	0.06	0.48
V_{10}'	0.48	0	0.89	0.25	0.3	0.12	0.14	0.06	0.33	0.09
V_{11}'	0	0	0.11	0.48	0	0.86	0	0.36	0.17	0
V_{12}'	0	0	0.3	0.34	0.64	0.25	0.39	0.34	0.15	0.42
V_{13}'	0.48	0.25	0.07	0.4	0	0.36	0	0.88	0.06	0
V_{14}'	0.4	0.39	0.29	0.15	0.15	0.18	0.12	0.46	0.59	0.06
V_{15}'	0	0.12	0.09	0	0.63	0	0.36	0	0.27	0.82

3.3 Performance Enhancement

In practice, it is infeasible to compute the similarity scores of all $|V| \cdot |V'|$ node pairs for large circuits. In this subsection, we present two pruning techniques to reduce the number of pairs that need to be updated so that we can reduce both the runtime and storage.

Support Constraint. Two internal nodes are less likely to be matched if they share few predefined matchings in their supports. A *support* of a node is the set of nodes with predefined matchings in the transitive fanin or fanout cone of this node. For example, the nodes with predefined matchings are PIs and POs in two graphs in Figure 3.2. The support for node V_7 is $SP(V_7) = \{V_1, V_2, V_4, V_5\}$, while the support for node V_{15}' is $SP(V_{15}') = \{V_2', V_3', V_6'\}$. The *support similarity* of V_7 and V_{15}' is the

sum of similarity scores of all $V \rightarrow V'$ node pairs in their supports: $X_{SP(V_7),SP(V'_{15})} = X_{V_2,V'_2} = 1$. On the other hand, the support similarity of V_7 and V'_7 is 4. Therefore, V_7 is more likely to be matched with V'_7 than with V'_{15}. Formally, for two nodes $v \in G$ and $v' \in G'$, the support constraint requires

$$\min\left(\frac{X_{SP(v),SP(v')}}{|SP(v)|}, \frac{X_{SP(v),SP(v')}}{|SP(v')|}\right) \geq \beta$$

where $\beta \in (0,1]$ is a constant. If the support constraint of the two nodes is not satisfied, we do not update their similarity score in the iteration. For example, if $\beta = 1$, *i.e.*, we only keep the pairs of nodes that have exactly the same supporting PIs and POs, 54 node pairs (*e.g.*, $(V_7, V'_{11}), (V_7, V'_{12})$) in Figure 3.2 can be pruned.

Level Constraint. If only combinatorial resynthesis is involved in an incremental design process, we can convert a circuit into a DAG by removing all registers and adding the register inputs (outputs) as POs (PIs). Given a DAG, a topological sort and reverse topological sort can label each internal node v with two values (shown above each node in Figure 3.2), *i.e.*, $level(v)$ and $rlevel(v)$, where $level(v)$ ($rlevel(v)$) denotes the length of the longest path from PIs (node v) to node v (POs). Two nodes with significantly different ($level$, $rlevel$) values are less likely to be matched. Formally, for two nodes $v \in G$ and $v' \in G'$, the level constraint requires

$$|level(v) - level(v')| \leq B_l, |rlevel(v) - rlevel(v')| \leq B_r$$

where B_l and B_r are two nonnegative constant integers. For example, if B_l and B_r are both set to be one, 22 node pairs (*e.g.*, (V_7, V'_9) and (V_7, V'_{15})) in Figure 3.2 can be pruned.

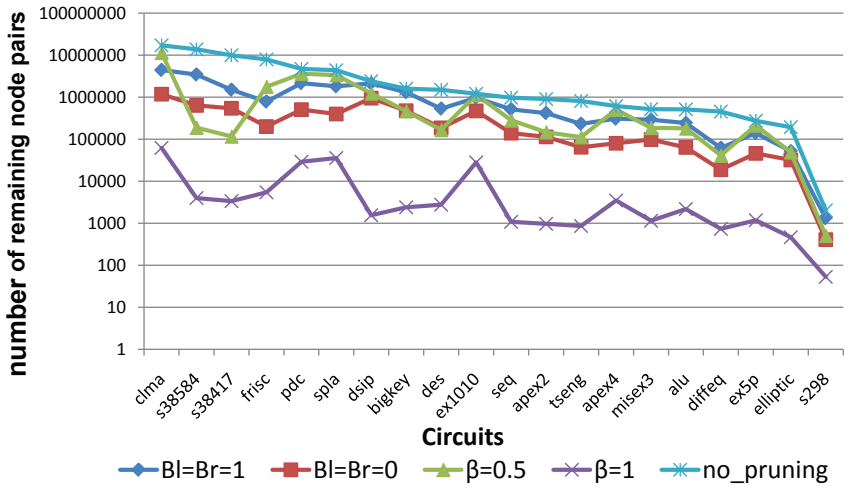


Figure 3.5: Effectiveness of the present pruning techniques

We have tested the above two pruning techniques on the 20 standard MCNC benchmark circuits. For each circuit, we run two logic synthesis algorithms, one with ABC command "if -K 4" and the other with "if -K 4; imfs" (an area-oriented resynthesis engine which destroys the internal name matching [3].), and generate two logic-level netlists. Figure 3.5 compares the number of node

pairs that need to be updated in the iterative similarity with the following five schemes: (a) without pruning ("no_pruning"), (b) using a weak level constraint-based pruning ("B_l=B_r=1"), (c) using a strong level constraint-based pruning ("B_l=B_r=0"), (d) using a weak support constraint-based pruning ("β=0.5"), and (e) using a strong support constraint-based pruning ("β=1"). As shown in Figure 3.5, the pruning techniques reduce the number of node pairs by 3 to 4 orders of magnitude compared with the total number of node pairs. More specifically, the strong level constraint-based pruning ("B_l=B_r=0") and the strong support constraint-based pruning ("β=1") can prune approximately 90% and 99% node pairs, respectively.

Table 3.3: Similarity score matrix for two graphs in Figure 3.2 with pruning ($\beta = 0.5$, $B_l = B_r = 0$)

	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}	V_{13}	V_{14}	V_{15}	V_{16}
V'_7	0.92	0	0	0	0	0	0	0	0	0
V'_8	0	0.73	0	0	0	0	0	0	0	0
V'_9	0	0	0	0	0	0	0.73	0	0	0.48
V'_{10}	0	0	0.89	0	0	0	0	0	0.33	0
V'_{11}	0	0	0	0	0	0.86	0	0.36	0	0
V'_{12}	0	0	0	0	0.64	0	0	0	0	0
V'_{13}	0	0	0	0	0	0.36	0	0.88	0	0
V'_{14}	0	0	0.29	0	0	0	0	0	0.59	0
V'_{15}	0	0	0	0	0	0	0.36	0	0	0.82

Table 3.3 shows the similarity score matrix obtained after applying these two pruning techniques on the similarity detection of G and G' in Figure 3.2. The iterative circuit similarity with pruning results in a very sparse matrix, while the most significant elements in this matrix are well preserved. For example, (V_{11}, V'_8) and (V_9, V'_7) are pruned due to the support and level constraints, respectively while (V_9, V'_{14}) is not pruned simply because it does not satisfy either of the constraints. Nevertheless, the most useful node pairs are preserved after pruning and the same node matching can be obtained compared to the completely computed similarity score matrix shown in Table 3.2. As a result of the sparsity of the similarity matrix, the maximum matching algorithm (min-cost network flow) is significantly faster. In Chapter 4, we will show that these pruning techniques do not significantly affect the quality of the similarity detection.

3.4 Applications of Circuit Similarity

As shown in Figure 3.1, circuit similarity detection flow can be employed to discover the topological correspondence between the original netlist and the modified netlist. Such information is then used to improve the efficiency of time-consuming CAD phases including placement, routing and verification, *e.t.c.*

Based on the layout results of the original netlist, such correspondence provides guidance and improves the efficiency of placement and routing of the modified netlist. Specifically, the node

matching between the two netlists produces a good initial placement of the modified netlist; the edge matching[2] between the two netlists gives a high-quality initial routing. Overlaps and congestions can be resolved by an efficient layout refinement phase.

Circuit similarity can also be applied to accelerate the sequential verification process, since sequential optimizations such as retiming only have a limited impact on the structure of a circuit. Particularly, the topological similarity can be employed to filter the redundant register pairs in the sequential equivalence checking before performing the computationally expensive functional equivalence checking and the state traversal. Similar ideas have been presented by van Eijk [61], but global topological similarity was not employed.

In this thesis work, we use the proposed circuit similarity to speed up placement which allows faster incremental design and design space exploration. More specifically, given an original network G, its placement can be obtained by performing a highly-optimized placement (*e.g.*, VPR). For another network, G', which is optimized by resynthesis scripts or generated by other design parameters, its placement is generated by first computing the similarity between networks G and G', and finding the matching of nodes in these two networks. The node matching between the two networks gives a good candidate for the initial placement of the modified network (G'). Therefore, based on such node correspondence, a high-quality initial placement of network G' can be determined using the placement of network G. For example, if node V' in network G' corresponds to node V in network G, V' is assigned the same coordinates as node V. Further refinement (*e.g.*, low-temperature simulated annealing) is applied to the initial placement of G' to gain better results.

3.5 Summary

In this chapter, we first illustrated the circuit similarity-based CAD flow using a motivating example in Section 3.1. The details of the circuit similarity algorithm are described in Section 3.2. In essence, the circuit similarity algorithm updates the circuit similarity scores for each node based on the similarity scores of the the incoming nodes and outgoing nodes in each iteration. In order to improve the efficiency and reduce storage requirement, we devised two pruning techniques, support constraint and level constraint. Support constraint takes advantage of the predefined matchings of the transitive fanin/fanout of one node while level constraint utilizes the topological/reverse topological sort of a graph. Finally, we briefly discussed the various applications CSBP could be applied to, such as placement, routing and sequential verification.

The next Chapter experimentally demonstrates the effectiveness and efficiency of the CSBP algorithm on incremental design case and design space exploration case, respectively.

[2]The edge correspondence can be obtained using the coupled node-edge algorithm [75].

Chapter 4

Experimental Results and Discussions

We use incremental design for FPGAs and design space exploration for FPGAs as two case studies to experimentally demonstrate the efficiency and effectiveness of the proposed CSBP, respectively. For the incremental design case study, we first explain the experimental CAD flow and settings. Then we present the comprehensive experimental results in terms of initial placement quality, final placement quality and runtime comparison. Moreover, we compare the experimental results of CSBP performing on two completely different resynthesis scripts, one using mild resynthesis (*i.e.*, "imfs") and the other using aggressive resynthesis (*i.e.*, "rwsat2"). For the design space exploration case study, we devise the experiments into two parts: logic-level design space exploration and algorithm-level design space exploration. For logic-level design space exploration, we present in-depth experimental results and analysis in terms of initial placement quality, final placement quality, design space shape characterization and runtime comparison. For algorithm-level design space exploration, we present the results using a wire length-delay space.

4.1 Case Study on Incremental Design

4.1.1 Experimental CAD Flow and Settings

In this case study, we consider an island-style FPGA architecture, which includes an array of CLBs interconnected by programmable routing. Figure 4.1 shows the two CAD flows that are compared in our experiments. Both flows include two design iterations and they share the first iteration, which starts from a logic-level netlist (a BLIF file). A technology mapping (using ABC command "if -K 4") is first performed on this netlist to map it into a 4-LUT-based network. After the mapping, T-VPack [57] is performed with "no_cluster" parameter to generate a CLB-based network, where each CLB contains one LUT and one flip-flop. The timing-driven placement in VPR is then used to produce the placement result (a ".p" file), and the timing-driven routing with a detailed timing analysis is finally performed.

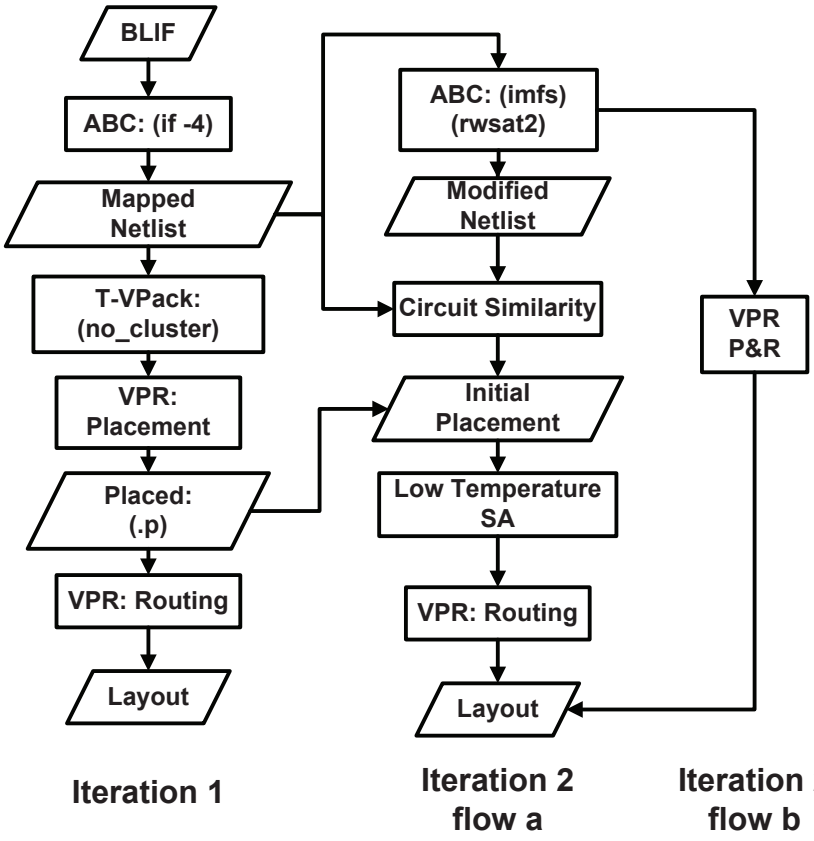


Figure 4.1: CSBP CAD flows used in incremental design.

In the second iteration, we perform a logic-level optimization on the mapped netlist using ABC command "imfs", an area-oriented resynthesis engine for network mapped into K-LUTs [3]. This optimization performs a window-based traversal in the logic-level netlist and rewrites the local network in each window using Boolean resubstitution based on logic don't-cares. Even though "imfs" is considered a mild resynthesis optimization compared to "rwsat2" optimization, the names of internal nodes (LUTs) in the modified netlist are not preserved after the optimization. Table 4.1 shows the characteristics of the logic-level netlist before (column "original") and after "imfs" optimization (column "imfs"). CIs (COs) include the PIs (POs) and register outputs (inputs). We will employ the proposed CSBP to discover the node correspondence purely based on topological information of the original and the "imfs"-modified netlists.

Starting from the modified netlist, we compare the following two flows: (a) CSBP flow and (b) *from-scratch* flow, as shown in Figure 4.1. Flow (b) uses VPR to replace the entire modified netlist. Flow (a) first computes the circuit similarity between the original and the modified netlists and uses it to generate an initial placement, which is further refined by a low-temperature annealing process using the VPR placement (*innerNum* is set to 0.1 in VPR for the moves per temperature. See Chapter 2, Section 2.2.2). As stated in Chapter 3, Section 3.3, based on different pruning settings

Table 4.1: Characteristics of the original and "imfs"-modified netlists.

			CLB#	
Circuit	CI#	CO#	original	imfs
alu4	14	8	719	713
apex2	38	3	963	946
apex4	9	19	788	786
bigkey	452	421	1261	1261
clma	94	115	4193	4100
des	256	245	1232	1222
diffeq	332	308	674	673
dsip	452	421	1554	1551
elliptic	196	196	441	438
ex1010	10	10	1103	1101
ex5p	8	63	541	505
frisc	905	1002	2844	2793
misex3	14	14	735	711
pdc	16	40	2211	2145
s298	17	20	45	45
s38417	1490	1568	3161	3150
s38584	1297	1564	3723	3717
seq	41	35	997	982
spla	16	46	2126	2060
tseng	435	507	941	867

and annealing parameters, we develop two versions of circuit similarity. A high-quality version, CS, uses $\beta = 0.5, B_l = B_r = 1$ and $inner\_num = 1$[1]. A turbo version, CS-t, uses $\beta = 1, B_l = B_r = 0$ and $inner\_num = 0.1$. Both CS and CS-t are evaluated in our experiments.

Our proposed CSBP is implemented in C and evaluated on the 20 largest MCNC benchmarks. All results are collected averaged over five runs and benchmarked on a Linux server with dual-core 2.19GHz CPU and 5GB memory. The CS2 package [23] is used to solve the min-cost network flow for the maximum matching problem.

4.1.2 Experimental Results for "imfs"

Quality of the initial placement. Table 4.2 compares the initial placement generated by the proposed circuit similarity (column "CS_init") and the one generated by VPR (a random initial placement) in terms of bb_cost (bounding box cost) and delay cost, two key measures of the placement process. The bb_cost and delay cost of the final placement produced by VPR (column "VPR_final") are also presented in the table for reference. Clearly, CSBP generates the initial placement with a quality very close to the final placement. This result shows that the topological node correspondence extracted by the circuit similarity algorithm indeed discovers the intrinsic connection between

[1] A parameter in VPR which controls the number of moves at each temperature. See Chapter 2, Section 2.2.2.

the original and modified logic-level netlists, and thus provides a reliable guidance to generate the placement for the modified netlist.

Table 4.2: Comparisons of initial solutions of different CAD flows for "imfs" in incremental design.

	bb cost			delay cost		
Circuit	CS_init	VPR_init	VPR_final	CS_init	VPR_init	VPR_final
alu4	94	219	94	2.43E-05	3.59E-05	2.40E-05
apex2	146	364	142	2.92E-05	5.43E-05	2.88E-05
apex4	133	273	130	2.76E-05	4.29E-05	2.87E-05
bigkey	245	1979	245	1.17E-04	2.45E-04	1.17E-04
clma	842	3078	683	2.05E-04	4.49E-04	1.90E-04
des	235	1084	221	4.99E-05	1.39E-04	4.73E-05
diffeq	209	934	209	3.65E-05	1.06E-04	3.65E-05
dsip	430	2158	400	1.65E-04	3.27E-04	1.68E-04
elliptic	83	335	82	1.41E-05	3.73E-05	1.40E-05
ex1010	155	392	153	4.28E-05	6.66E-05	4.17E-05
ex5p	95	153	70	1.46E-05	2.14E-05	1.24E-05
frisc	2373	9616	2206	4.47E-04	1.17E-03	4.85E-04
misex3	100	228	95	2.25E-05	3.71E-05	2.16E-05
pdc	489	1170	431	1.02E-04	1.79E-04	9.49E-05
s298	3	6	3	5.79E-07	8.26E-07	5.79E-07
s38417	1130	19113	957	4.01E-04	2.07E-03	3.74E-04
s38584	1620	20477	1369	5.01E-04	2.41E-03	4.95E-04
seq	164	380	163	3.34E-05	5.93E-05	3.22E-05
spla	490	1102	407	9.40E-05	1.65E-04	8.45E-05
tseng	298	1700	275	7.76E-05	1.95E-04	7.54E-05
geomean	229	829	211	5.34E-05	1.14E-04	5.19E-05
ratio	108%	392%	1	103%	220%	1

Quality of the final placement. A low-temperature annealing is applied to the initial placement generated by our CSBP flow. The quality of post-routing results produced by flow (a) and flow (b) shown in Figure 4.1 are compared. In flow (a), two different settings are tested: (i) a high-quality version (column "CS") and (ii) a turbo version (column "CS-t"). Wire length, the device area (in the number of minimal width transistors) and the critical path delay are compared between the circuits produced by the two versions of flow (a) and flow (b) as shown in Table 4.3, 4.4 and 4.5, which shows that both versions of CSBP produce layout with a quality very close to the results produced by *from-scratch* flow. The comparison between CS and CS-t shows the effectiveness of the proposed pruning techniques (in Chapter 3, Section 3.3). Clearly, CS-t, geared with aggressive pruning and significantly lower annealing effort, still produces placement with comparable quality to CS and VPR.

Table 4.3: Comparisons of post-routing wire length of different CAD flows for "imfs" in incremental design.

	Wire length		
Circuit	CS	CS-t	VPR
alu4	11091	13289	10882
apex2	16716	19149	16122
apex4	16396	18664	16424
bigkey	26520	39827	26632
clma	74664	95025	71487
des	26504	28066	25540
diffeq	22721	23984	23195
dsip	47265	55024	39454
elliptic	8736	9783	8329
ex1010	18132	20640	17881
ex5p	8570	9618	8079
frisc	243989	259561	233522
misex3	11379	13532	10733
pdc	50685	62485	50442
s298	319	363	321
s38417	70544	100379	76544
s38584	106385	134489	103074
seq	19300	21233	18940
spla	50101	61598	46630
tseng	27260	30195	27758
geomean	23214	27352	22602
ratio	103%	121%	1

Table 4.4: Comparisons of post-routing area of different CAD flows for “imfs” in incremental design.

	Area		
Circuit	CS	CS-t	VPR
alu4	2.63E+06	3.48E+06	2.42E+06
apex2	3.98E+06	4.57E+06	3.68E+06
apex4	3.76E+06	4.27E+06	3.76E+06
bigkey	2.53E+07	1.84E+07	2.53E+07
clma	1.86E+07	2.36E+07	1.86E+07
des	9.48E+06	9.48E+06	9.48E+06
diffeq	1.53E+07	1.53E+07	1.34E+07
dsip	2.88E+07	2.19E+07	2.88E+07
elliptic	5.07E+06	5.07E+06	5.07E+06
ex1010	4.48E+06	4.82E+06	4.48E+06
ex5p	2.09E+06	2.42E+06	2.09E+06
frisc	2.01E+08	1.69E+08	1.85E+08
misex3	2.83E+06	3.51E+06	2.83E+06
pdc	1.36E+07	1.62E+07	1.29E+07
s298	6.53E+04	8.19E+04	6.53E+04
s38417	3.47E+08	3.47E+08	3.47E+08
s38584	3.43E+08	2.67E+08	3.79E+08
seq	4.57E+06	5.18E+06	4.86E+06
spla	1.36E+07	1.49E+07	1.17E+07
tseng	2.91E+07	2.52E+07	2.91E+07
geomean	1.07E+07	1.11E+07	1.05E+07
ratio	102%	106%	1

Table 4.5: Comparisons of post-routing critical delay of different CAD flows for "imfs" in incremental design.

	Critical delay (s)		
Circuit	CS	CS-t	VPR
alu4	7.65E-08	6.58E-08	7.45E-08
apex2	8.35E-08	7.61E-08	7.21E-08
apex4	1.09E-07	8.80E-08	8.24E-08
bigkey	1.34E-07	1.35E-07	1.43E-07
clma	1.32E-07	1.67E-07	1.34E-07
des	9.91E-08	1.11E-07	7.46E-08
diffeq	1.24E-07	1.16E-07	1.23E-07
dsip	1.27E-07	1.26E-07	1.31E-07
elliptic	8.56E-08	8.44E-08	8.89E-08
ex1010	1.16E-07	9.57E-08	9.82E-08
ex5p	6.73E-08	6.11E-08	6.52E-08
frisc	3.36E-07	3.44E-07	3.48E-07
misex3	7.10E-08	6.95E-08	6.02E-08
pdc	1.45E-07	1.68E-07	1.40E-07
s298	2.44E-08	2.79E-08	2.02E-08
s38417	3.80E-07	3.80E-07	4.16E-07
s38584	4.47E-07	4.31E-07	4.34E-07
seq	7.86E-08	8.04E-08	6.78E-08
spla	1.33E-07	1.66E-07	1.43E-07
tseng	1.44E-07	1.44E-07	1.51E-07
geomean	1.18E-07	1.18E-07	1.12E-07
ratio	106%	106%	1

Runtime comparison. Table 4.6 compares the runtime of the placement in the two versions of flow (a) with flow (b). It shows that CS-t achieves 31X speedup on average (up to 91X), compared with the *from-scratch* VPR placement. Since computing the similarity between two circuits is much faster than replacing them from scratch, more speedup is expected when applying CSBP to larger circuits. As a matter of fact, the two largest circuits in the MCNC benchmarks (s38417 and s38584) achieve the highest speedup as shown in Table 4.6. Similar results can also be found at Table 4.12 and 4.16. This essentially proves that the circuit similarity-based algorithm has a premium scalability and thus can be applied to very large scale circuits. In practice, one can use CS-t as a quick estimation of the solution quality for an iteration during the incremental design. If the quality is within a satisfied range, the VPR placement can be performed for a better quality.

Table 4.6: Comparisons of placement runtime of different CAD flows for "imfs" in incremental design.

	Placement runtime (s)		
Circuit	CS	CS-t	VPR
alu4	4.47 (4x)	0.48 (33x)	15.68
apex2	5.8 (4x)	0.63 (38x)	23.86
apex4	13.15 (1x)	0.63 (30x)	18.81
bigkey	27.3 (2x)	2.13 (27x)	58.41
clma	174.8 (2x)	12.66 (33x)	413.64
des	16.83 (2x)	1.64 (25x)	41.31
diffeq	6.65 (4x)	0.67 (42x)	28.13
dsip	78.92 (1x)	4.63 (16x)	75.8
elliptic	3.46 (3x)	0.31 (37x)	11.56
ex1010	51.77 (1x)	5.92 (5 x)	27.9
ex5p	6.14 (2x)	0.3 (33x)	9.87
frisc	127.78 (4x)	8.72 (52x)	453.57
misex3	4.88 (3x)	0.42 (37x)	15.49
pdc	90.48 (1x)	4.01 (23x)	92.82
s298	0.06 (5x)	0.02 (16x)	0.32
s38417	193.95 (5x)	10.79 (91x)	977.24
s38584	224.21 (5x)	12.66 (81x)	1022.41
seq	8.27 (3x)	0.68 (41x)	28.02
spla	74.56 (1x)	3.81 (22x)	84.96
tseng	17.71 (3x)	1.11 (44x)	48.53
geomean	18.63 (2 x)	1.45 (31 x)	44.98
ratio	41%	3%	1

4.1.3 Experimental Results for "rwsat2"

As mentioned in Section 4.1.1, we perform "rwsat2", a more aggressive logic-level optimization, on the mapped netlist in the experimental CAD flow shown in Figure 4.2.1. The "rwsat2" optimization consists of a set of commands as following:

st; rw -l; b -l; rw -l; rf -l; fraig; rw -l; b -l; rw -l; rf -l

where each command (alias) is a logic optimization in ABC, *e.g.*, "st" (structural hashing) aggressively destroys the initial boundaries among internal nodes (LUTs); "rw" (rewrite) and "rf" (refactor) reconstruct the network by reducing the AIG size and level; "fraig" (functionally-reduced AIG) changes the current network structure and transforms into a functionally-reduce AIG [5][2]. Therefore, in the modified netlist, the name matchings among the nodes are not preserved and the structure of the network is completely changed. Table 4.7 shows the characteristics of the logic-level netlist before (column "original") and after "rwsat2" optimization (column "rwsat2"). CIs (COs) include the PIs (POs) and register outputs (inputs). We will employ the proposed circuit similarity to discover the node correspondence purely based on topological information of the original and the "rwsat2"-modified netlists. The rest of the experimental CAD flow and settings is identical to the case study on "imfs" as stated in Section 4.1.1.

[2] A complete list of these commands can be found in Appendix A.

Table 4.7: Characteristics of the original and "rwsat2"-modified netlists.

			CLB#	
Circuit	CI#	CO#	original	rwsat2
alu4	14	8	719	691
apex2	38	3	963	914
apex4	9	19	788	771
bigkey	452	421	1261	1261
clma	94	115	4193	4063
des	256	245	1232	1215
diffeq	332	308	674	665
dsip	452	421	1554	1553
elliptic	196	196	441	439
ex1010	10	10	1103	851
ex5p	8	63	541	515
frisc	905	1002	2844	2320
misex3	14	14	735	629
pdc	16	40	2211	1969
s298	17	20	45	39
s38417	1490	1568	3161	3122
s38584	1297	1564	3723	3646
seq	41	35	997	932
spla	16	46	2126	1935
tseng	435	507	941	883

Quality of the initial placement. Table 4.8 compares the initial placement generated by the proposed circuit similarity (column "CS" and "CS-t") and the one generated by VPR (a random initial placement) in terms of bb_cost (bounding box cost) and delay cost. The results are conclusions are similar to the "imfs" experiments. Both CS and CS-t produce the initial placement with a much better quality than VPR's by reducing 40% of the bb_cost and 31% of the delay cost, respectively. This again demonstrates the circuit similarity algorithm is able to discover the underlying structural characteristic on the aggressive resynthesized netlists which naming matching is destroyed and structure is dramatically changed.

Table 4.8: Comparisons of initial solutions of different CAD flows for "rwsat2".

	initial bb cost			initial delay cost		
Circuit	CS	CS-t	VPR	CS	CS-t	VPR
alu4	145	155	212	2.70E-05	2.95E-05	3.55E-05
apex2	213	249	341	3.38E-05	4.24E-05	5.25E-05
apex4	231	222	259	3.49E-05	3.66E-05	4.09E-05
bigkey	1294	1575	2004	2.23E-04	2.39E-04	2.49E-04
clma	1737	1615	3020	3.36E-04	3.28E-04	4.62E-04
des	306	798	1087	5.93E-05	1.13E-04	1.40E-04
diffeq	818	901	922	9.52E-05	1.04E-04	1.03E-04
dsip	448	578	2184	1.64E-04	1.75E-04	3.30E-04
elliptic	149	244	338	1.73E-05	2.56E-05	3.72E-05
ex1010	277	259	297	3.95E-05	3.99E-05	4.30E-05
ex5p	118	134	156	1.82E-05	2.03E-05	2.31E-05
frisc	5374	6530	8696	7.57E-04	8.49E-04	1.07E-03
misex3	129	162	192	2.36E-05	2.83E-05	3.23E-05
pdc	860	789	1015	1.23E-04	1.31E-04	1.55E-04
s298	3	4	5	5.50E-07	6.06E-07	7.27E-07
s38417	9254	14905	19358	1.07E-03	1.74E-03	2.15E-03
s38584	11948	20781	19983	1.39E-03	2.37E-03	2.33E-03
seq	246	297	355	3.77E-05	4.72E-05	5.47E-05
spla	799	765	986	1.13E-04	1.27E-04	1.50E-04
tseng	801	1323	1736	1.17E-04	1.56E-04	2.00E-04
geomean	472	585	785	7.41E-05	8.96E-05	1.08E-04
ratio	60%	75%	1	69%	83%	1

Quality of the final placement. In order to provide some different perspectives for results, we compare the final bounding box, final delay cost and estimated critical delay between two versions of CSBP for "rwsat2" results. Table 4.9 compares final bounding box cost and Table 4.10 compares the final delay cost. Both CS and CS-t produce quality very close to the results produced by *from-scratch* flow. Table 4.11 compares the estimated critical delay, CS and CS-t reduce it by 4% and 1%, respectively, compared to *from-scratch* flow. The comparison between CS and CS-t again shows the effectiveness of the proposed CSBP method and the pruning techniques.

Table 4.9: Comparisons of final placement bounding cost of different CAD flows for “rwsat2”.

	Final bounding box cost		
Circuit	CS	CS-t	VPR
alu4	92	111	91
apex2	137	157	136
apex4	135	148	131
bigkey	262	401	272
clma	766	956	690
des	237	250	221
diffeq	224	237	216
dsip	468	518	426
elliptic	91	96	84
ex1010	147	167	142
ex5p	80	88	76
frisc	2308	2376	2052
misex3	87	97	81
pdc	416	480	402
s298	3	3	3
s38417	1091	1287	978
s38584	1401	1685	1352
seq	156	174	153
spla	396	481	389
tseng	276	298	272
geomean	218	249	208
ratio	105%	120%	1

Table 4.10: Comparisons of final delay cost of different CAD flows for “rwsat2” .

	Final delay cost		
Circuit	CS	CS-t	VPR
alu4	2.34E-05	2.48E-05	2.24E-05
apex2	2.83E-05	3.00E-05	2.77E-05
apex4	2.71E-05	2.87E-05	2.70E-05
bigkey	1.13E-04	1.11E-04	1.11E-04
clma	2.22E-04	2.38E-04	1.93E-04
des	5.08E-05	5.23E-05	4.98E-05
diffeq	3.79E-05	3.91E-05	3.68E-05
dsip	1.57E-04	1.57E-04	1.65E-04
elliptic	1.41E-05	1.45E-05	1.42E-05
ex1010	2.71E-05	2.91E-05	2.78E-05
ex5p	1.45E-05	1.54E-05	1.42E-05
frisc	4.41E-04	4.39E-04	4.37E-04
misex3	1.98E-05	2.06E-05	1.84E-05
pdc	8.80E-05	9.05E-05	8.56E-05
s298	5.34E-07	5.40E-07	5.05E-07
s38417	3.14E-04	3.25E-04	3.68E-04
s38584	5.25E-04	5.49E-04	4.95E-04
seq	3.08E-05	3.25E-05	3.04E-05
spla	7.83E-05	8.78E-05	8.18E-05
tseng	7.86E-05	7.98E-05	7.73E-05
geomean	5.00E-05	5.20E-05	4.94E-05
ratio	101%	105%	1

Table 4.11: Comparisons of estimated critical delay of different CAD flows for “rwsat2”.

	Estimated critical delay (s)		
Circuit	CS	CS-t	VPR
alu4	5.61E-08	6.27E-08	5.88E-08
apex2	6.77E-08	7.30E-08	6.45E-08
apex4	6.27E-08	6.50E-08	6.27E-08
bigkey	1.32E-07	1.29E-07	1.59E-07
clma	1.85E-07	1.93E-07	1.29E-07
des	8.28E-08	8.04E-08	1.01E-07
diffeq	1.17E-07	1.13E-07	1.38E-07
dsip	1.26E-07	1.22E-07	1.48E-07
elliptic	8.05E-08	8.73E-08	8.98E-08
ex1010	6.80E-08	7.51E-08	7.22E-08
ex5p	5.06E-08	5.47E-08	5.54E-08
frisc	3.13E-07	3.19E-07	3.16E-07
misex3	5.19E-08	6.27E-08	5.48E-08
pdc	9.70E-08	1.06E-07	1.03E-07
s298	1.78E-08	1.78E-08	1.61E-08
s38417	3.65E-07	3.65E-07	3.83E-07
s38584	4.37E-07	4.13E-07	5.09E-07
seq	5.53E-08	6.12E-08	5.40E-08
spla	1.02E-07	1.11E-07	9.25E-08
tseng	1.46E-07	1.40E-07	1.67E-07
geomean	9.84E-08	1.02E-07	1.03E-07
ratio	96%	99%	1

Runtime comparison. Table 4.12 compares the runtime of the placement for the "rwsat2" resynthesized circuits. The results and conclusions are similar to "imfs" resynthesized circuits. Note that a timeout is invoked if CSBP takes longer than the original netlist. It shows that CS-t achieves 28X speedup on average and up to 93X on the largest circuit, which again proves the scalability of CSBP, compared with the *from-scratch* VPR placement. Moreover, note that the pruning setting in CS-t makes the similarity detection phase in CS-t approximately 10X faster than that in CS, and the reduction of *inner_num* also makes the annealing process in CS-t approximately 10X faster. Even with such a significant speedup, the quality of the placement produced by CS-t is still comparable to that produced by CS, which again proves the effectiveness of the proposed pruning techniques and the quality of the initial placement generated by CSBP.

Table 4.12: Comparisons of final placement runtime of different CAD flows for "rwsat2".

	Placement runtime (s)		
Circuit	CS	CS-t	VPR
alu4	6.89 (1 x)	0.36 (28 x)	9.97
apex2	8.65 (2 x)	0.49 (31 x)	15.01
apex4	*4.16 (3 x)	0.54 (22 x)	12.03
bigkey	35.33 (1 x)	1.74 (25 x)	43.18
clma	*43.64 (3 x)	5.85 (24 x)	139.37
des	13.38 (2 x)	1.28 (22 x)	28.42
diffeq	3.11 (6 x)	0.52 (36 x)	18.73
dsip	*11.58 (4 x)	5.21 (10 x)	52.09
elliptic	3.54 (2 x)	0.24 (34 x)	8.11
ex1010	*8.38 (2 x)	0.49 (32 x)	15.56
ex5p	*2.44 (3 x)	*1.84 (5 x)	8.53
frisc	22.17 (8 x)	4.23 (44 x)	185.37
misex3	6.3 (2 x)	0.25 (40 x)	10.02
pdc	*19.67 (3 x)	2.7 (21 x)	56.97
s298	0.05 (5 x)	0.01 (24 x)	0.24
s38417	38.66 (11 x)	5.05 (84 x)	426.19
s38584	29.23 (14 x)	4.52 (93 x)	421.8
seq	14.3 (1 x)	0.51 (37 x)	18.75
spla	*17.21 (3 x)	2.21 (24 x)	53.04
tseng	5.99 (6 x)	0.82 (43 x)	35.02
geomean	8.4 (3 x)	0.96 (28 x)	27.26
ratio	31%	4%	1

4.2 Case Study on Design Space Exploration

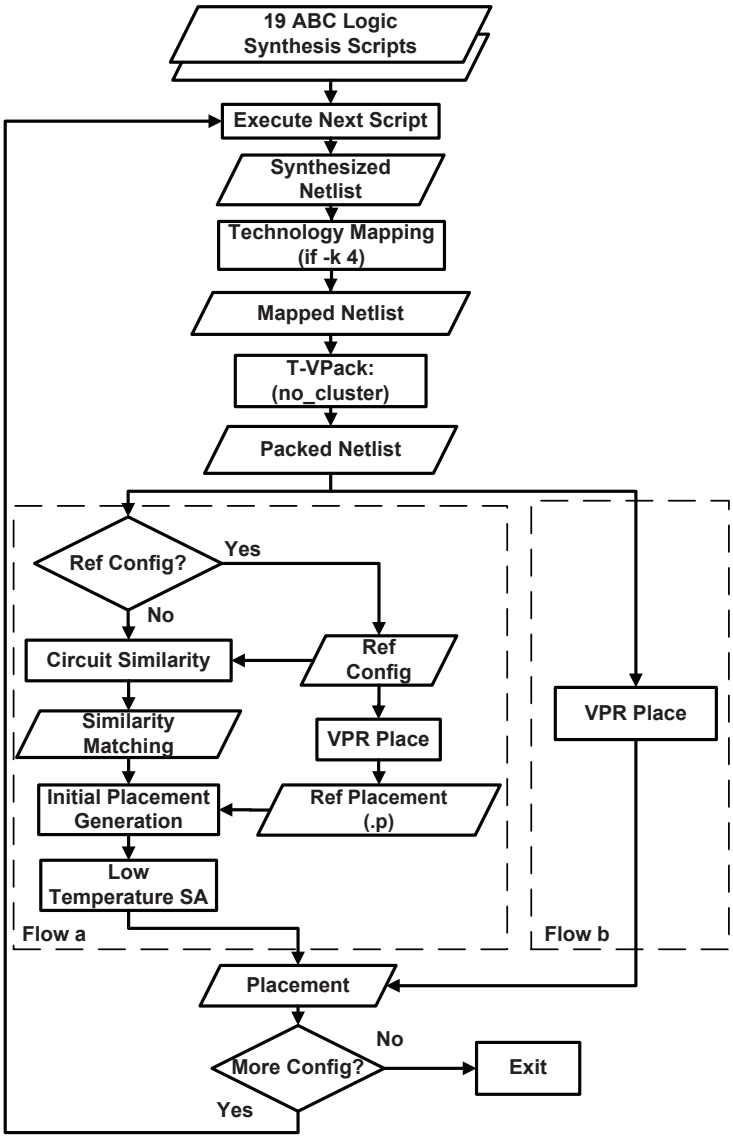


Figure 4.2: CAD flows used in the experiments for design space exploration.

In this section, we apply CSBP to design space exploration at logic level and algorithm level, respectively.

4.2.1 Logic-Level Design Space Exploration

Experimental CAD Flow and Settings The objective of this design space exploration is to identify the influence of logic-level optimization in a post-layout design. In our experiment, the design parameters are logic synthesis and optimization commands in Berkeley ABC [9]. We use 19 synthesis scripts provided in abc.rc from the ABC package, *i.e.*, there are 19 configurations in this design

Table 4.13: Characteristics of the logic-level design space for 20 MCNC applications over 19 configurations.

			Block Number		Level	
Circuit	CI#	CO#	min	max	min	max
alu4	14	8	652	710	7	10
apex2	38	3	773	926	8	11
apex4	9	19	754	805	7	10
bigkey	452	421	924	1263	3	4
clma	94	115	3731	4221	12	17
des	256	245	1157	1245	6	10
diffeq	332	308	648	712	12	15
dsip	452	421	1106	1554	3	4
elliptic	196	196	375	441	8	11
ex1010	10	10	851	1100	6	10
ex5p	8	63	460	519	6	11
frisc	905	1002	2173	2788	19	26
misex3	14	14	545	680	6	9
pdc	16	40	1836	2159	8	14
s298	17	20	36	43	3	4
s38417	1490	1568	3063	3252	9	10
s38584	1297	1564	3568	3715	8	11
seq	41	35	891	982	6	9
spla	16	46	1718	2074	8	14
tseng	435	507	744	938	12	14

space exploration case. In the rest of this sub-section, we follow the same names of each script used in ABC as the index, *e.g.*, the two scripts shown in Chapter 3, Section 3.1 are named "resyn3" and "rwsat2", respectively. Interested readers may refer to Appendix A for more details.

The experimental CAD flow is shown in Figure 4.2. Starting from 19 ABC logic synthesis scripts, we have the resulting synthesized netlists stored in BLIF file format. Next, the same procedures are performed as stated in Section 4.1.1 until the netlists are packed. After this point, we again compare two CAD flows: (a) CSBP flow and (b) VPR *from-scratch* flow, as shown in Figure 4.2. Flow (a) first selects the largest configuration (*i.e.*, the one with largest number of CLBs) as the reference. Then the reference configuration is placed using VPR and produces a reference placement (".p" file). The reference configuration and its placement are then used to guide the initial placement of the new configuration by finding the similarity between the new configuration and the reference configuration. The same annealing process is used to further refine the placement results as Section 4.1.1.

As in Section 4.1.1, we evaluate both CS and CS-t in this design space exploration case, and use the same server and settings to benchmark the results.

Experimental Results Table 4.13 shows the minimal and maximal CLB number and level for the design space of each application. The number of CLBs and levels vary widely in different configurations.

Quality of the initial placement. Table 4.14 shows the initial placement quality of CS and CS-t compared to VPR's initial results. We show one representative circuit "dsip" as an example. The results for the other circuits are similar. The "Configuration" column in Table 4.14 lists the 19 ABC scripts' names. The bounding box cost ("initial bb cost" column) and the delay cost ("initial delay cost" column) are compared. The initial placement results generated by CS and CS-t are significantly better than VPR's random initial placement results. CS improves the bb cost and delay cost by 76% and 48% compared to VPR, respectively. This again demonstrates that CSBP is able to find the underlying structural similarities among different configurations, and thus provides a quality placement for the design space exploration.

Table 4.14: Initial placement quality comparison of circuit "dsip" for 19 designs.

	initial bb cost			initial delay cost		
Configuration	CS	CS-t	VPR	CS	CS-t	VPR
resyn	363	934	1817	1.02E-04	1.49E-04	2.22E-04
resyn2	363	934	1819	1.02E-04	1.49E-04	2.19E-04
resyn2a	453	453	2184	1.69E-04	1.69E-04	3.11E-04
resyn3	443	494	2189	1.64E-04	1.70E-04	3.07E-04
compress	448	448	2203	1.66E-04	1.66E-04	3.31E-04
compress2	363	936	1785	1.03E-04	1.52E-04	2.21E-04
choice	1943	1978	2020	2.72E-04	2.74E-04	2.72E-04
choice2	1940	1977	1990	2.72E-04	2.74E-04	2.75E-04
rwsat	371	930	1804	1.03E-04	1.53E-04	2.26E-04
rwsat2	448	578	2184	1.64E-04	1.75E-04	3.30E-04
shake	424	518	2151	1.44E-04	1.51E-04	2.93E-04
share	365	936	1778	1.02E-04	1.50E-04	2.19E-04
src_rw	458	793	2185	1.48E-04	1.70E-04	2.96E-04
src_rs	452	458	2159	1.46E-04	1.48E-04	2.96E-04
src_rws	544	841	2172	1.53E-04	1.72E-04	2.93E-04
resyn2rs	365	937	1770	1.03E-04	1.52E-04	2.23E-04
compress2rs	363	936	1809	1.03E-04	1.53E-04	2.22E-04
resyn2rsdc	427	719	2156	1.65E-04	1.83E-04	3.10E-04
compress2rsdc	373	933	1772	1.04E-04	1.53E-04	2.24E-04
geomean	483	802	1989	1.39E-04	1.69E-04	2.65E-04
ratio	24%	40%	1	52%	64%	1

Quality of the final placement. A low-temperature annealing is applied to the initial placement results generated by CSBP. Table 4.15 compares the final placement results of circuit "dsip" for 19 designs. We evaluate the final wire length and the critical delay. For wire length, CS and CS-t produce the results close to VPR's final results with 32% and 53% overhead, respectively. For critical

delay, CS and CS-t achieve better results than VPR, reducing it by 18% and 20%, respectively. This shows the effectiveness of CSBP that generates an optimized initial placement which in turn leads to an optimized final placement.

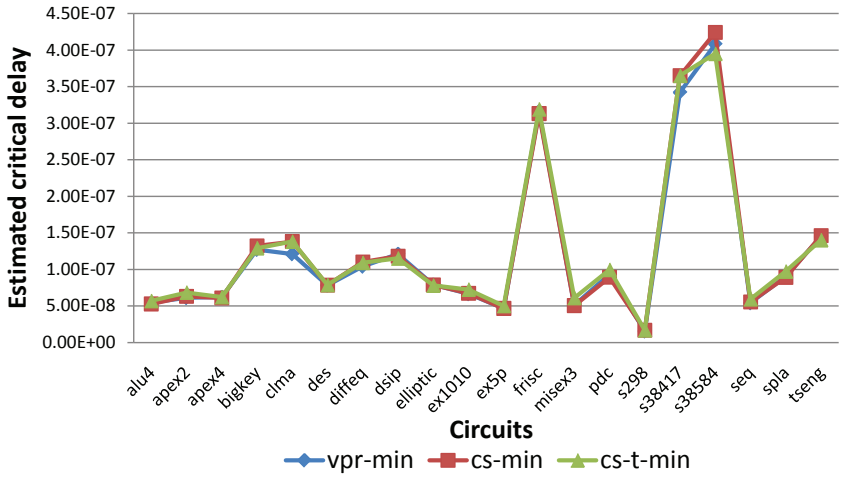


Figure 4.3: Minimal estimated critical delay design space shape of 20 circuits on 19 designs.

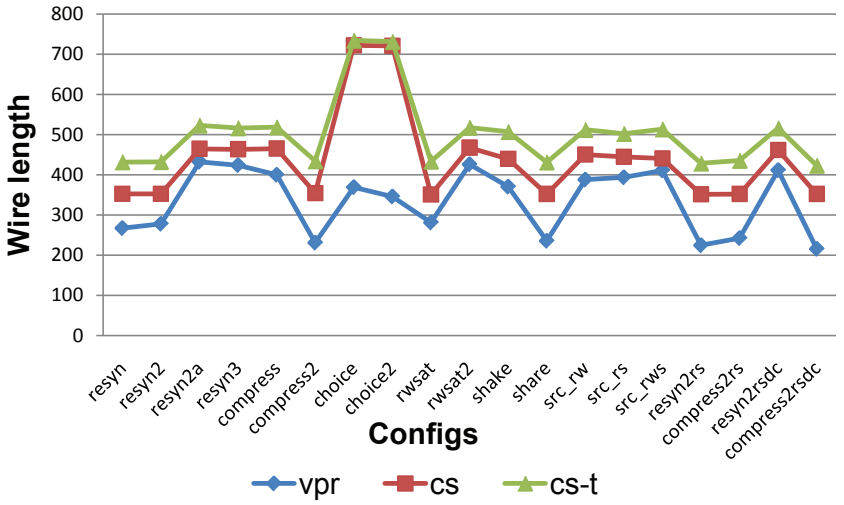


Figure 4.4: Final wire length design space shape comparison of VPR, CS and CS-t on circuit “dsip”.

Design space shape characterization. We compare the minimal, median and maximal wire length and critical delay produced by CS and CS-t to VPR. Figure 4.3 shows the minimal critical delay curves of all 19 designs for 20 circuits using CS, CS-t and VPR. The almost identical curves prove that both CS and CS-t can precisely pinpoint the minimal critical delay design. Similarly, the median and maximum curves of both CS and CS-t follow close to VPR’s. Moreover, the shape of most configurations is accurately matched as well. As an example, Figure 4.4 shows the wire length curve for circuit “dsip”. Note that “choice” and “choice2” include a repetitive call of ABC synthesis command “fraig_store”, which stores the current functionally-reduced network as one “synthesis snapshot” for

Table 4.15: Final placement quality comparison of circuit "dsip" for 19 designs.

	Wire length			Critical delay		
Configuration	CS	CS-t	VPR	CS	CS-t	VPR
resyn	353	432	267	1.23E-07	1.18E-07	1.63E-07
resyn2	353	432	278	1.26E-07	1.17E-07	1.51E-07
resyn2a	465	523	432	1.25E-07	1.25E-07	1.34E-07
resyn3	464	516	424	1.25E-07	1.22E-07	1.28E-07
compress	465	519	400	1.25E-07	1.19E-07	1.63E-07
compress2	354	434	231	1.19E-07	1.18E-07	1.81E-07
choice	722	735	369	1.30E-07	1.30E-07	1.27E-07
choice2	721	732	346	1.30E-07	1.30E-07	1.54E-07
rwsat	351	433	281	1.21E-07	1.15E-07	1.40E-07
rwsat2	468	518	426	1.26E-07	1.22E-07	1.48E-07
shake	440	507	370	1.20E-07	1.21E-07	1.77E-07
share	352	431	236	1.22E-07	1.18E-07	1.64E-07
src_rw	451	512	388	1.24E-07	1.25E-07	1.72E-07
src_rs	445	502	394	1.21E-07	1.22E-07	1.24E-07
src_rws	441	514	411	1.25E-07	1.22E-07	1.65E-07
resyn2rs	352	429	225	1.18E-07	1.16E-07	1.63E-07
compress2rs	352	435	242	1.24E-07	1.19E-07	1.73E-07
resyn2rsdc	462	516	412	1.23E-07	1.24E-07	1.21E-07
compress2rsdc	352	423	216	1.22E-07	1.16E-07	1.53E-07
geomean	429	496	324	1.24E-07	1.21E-07	1.52E-07
ratio	132%	153%	1	82%	80%	1

later technology mapping. Such a choice-based logic optimization may significantly change the topology of the netlist. In the future, we will investigate improvements to this problem. Although CS-t produces longer wire length than VPR, Figure 4.4 shows that CS-t closely captures the relative wire length of each configuration which is essentially useful in the design space exploration.

Runtime Comparison. Table 4.16 compares the total runtime of placing 19 designs of each MCNC application using CS, CS-t and VPR. Column "Ref" shows the time to place the reference configuration. Column "CS" and "CS-t" shows the time to generate the placement for the remaining 18 configurations. Note that a timeout is invoked if CSBP takes longer than the original netlist. Without considering the time for placing the reference configuration, CS achieves averaged 3X speedup while CS-t achieves averaged 30X speedup with up to 100X compared to from-scratch VPR placement. Since the computation of the reference placement is a one-time cost for a design space exploration problem of one application, the time used for the reference placement should be amortized and becomes negligible as the number of configurations increases.

We make use of the significant speedup of CS-t and use it to perform quick design space exploration. For instance, the total time of exploring the whole design space of 20 MCNC applications with 19 designs is more than 8 hours using VPR (place only). In contrast, it only takes 37 min-

utes (including the time for the reference placement) using our CS-t. More significant speedup is expected when larger design space is explored.

Table 4.16: Comparison of total runtime (s) for logic-level design space exploration. The '*' marked time is measured with a timeout.

Circuit	CS	CS-t	VPR	Ref
alu4	113.99 (2x)	6.13 (31x)	188.62	10.37
apex2	122.85 (2x)	7.7 (35x)	267.46	17.03
apex4	187.18* (1x)	8.77 (28x)	246.03	14.89
bigkey	630.69 (1x)	26.71 (27x)	720.96	37.85
clma	1782.7* (1x)	101.42 (25x)	2532.65	143.91
des	183.37 (3x)	20.32 (27x)	549.24	29.38
diffeq	56.09 (7x)	9.13 (42x)	387.89	19.78
dsip	626.48* (1x)	55.92 (14x)	776.66	47.12
elliptic	47.06 (4x)	4.37 (40x)	173.6	9.88
ex1010	229.28* (1x)	72.73 (5x)	336.51	18.54
ex5p	95.55* (1x)	4.18 (33x)	136.49	7.87
frisc	373.98 (8x)	75.43 (42x)	3178.44	176.56
misex3	122.87 (1x)	4.68 (36x)	170.32	11.44
pdc	747.53* (1x)	43.8 (22x)	975.6	55.11
s298	0.94 (5x)	0.32 (14x)	4.45	0.27
s38417	564.45 (15x)	82.99 (100x)	8318.7	445.85
s38584	541.19 (15x)	84.73 (96x)	8155.16	443.59
seq	218.33 (2x)	9 (37x)	337.44	18.52
spla	745.35* (1x)	39.71 (23x)	923.75	53.36
tseng	86.72 (7x)	13.71 (43x)	587.89	33.35
geomean	186.97 (3x)	16.64 (30x)	497.81	28.33
total	7476.6 (4x)	671.75 (43x)	28967.86	1594.67

4.2.2 Algorithm-Level Design Space Exploration

Experimental CAD Flow and Settings We now demonstrate the effectiveness of CSBP at the algorithm-level for the design space exploration. Specifically, the design is a constant multiplier as shown in Figure 4.5, where (a) shows a multiplier block implements a parallel multiplication of a variable with a fixed set of constants, *i.e.* $c_1, c_2, \ldots c_n$ and (b) presents a very simple example - a multiplier block for the parallel multiplication with the constants 23 and 81. The design parameter in this exploration is the fractional bits, which controls the precision of the constants, from 7 to 25, resulting in a design space containing 18 configurations[3]. Given a fractional bit setting, we use the CMU SPIRAL multiplier block generator to generate the RTL design of each configuration based on the Hcub algorithm [72]. The following constants (accurate to two decimal places) are used for all configurations: 0.23, 0.71, 0.63, 0.03, -0.19, 0.03, 0.03, -0.01. Once we obtain the RTL design, we use Altera Quartus to perform RTL elaboration and generate a BLIF file from a

[3] $Bits = 16$ is abandoned since ABC crashed when synthesized it. So there are 18 configurations in total.

verilog (.v) file. Other experimental settings are the same as described in Section 4.1.1. Table 4.17 presents the number of CLBs and level for algorithm-level design space. Since those configurations vary in algorithm level, the topological structure and circuit size differ considerably compared to logic-level variations. Therefore, it is more challenging to find the similarities between these design configurations.

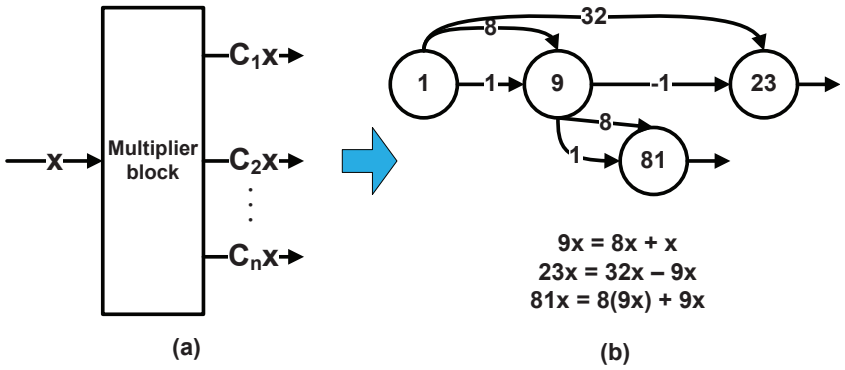


Figure 4.5: A simple example of a multiplier block with constants 23 and 81 [2].

Table 4.17: Characteristics of the algorithm-level design space of 18 configurations using CMU SPIRAL.

Bits	CLB#	Level	*Bits*	CLB#	Level
7	501	35	*17*	2222	57
8	697	38	*18*	2356	52
9	814	37	*19*	2339	60
10	920	41	*20*	2577	56
11	1115	42	*21*	2398	54
12	938	43	*22*	2625	55
13	1085	41	*23*	2832	55
14	1293	48	*24*	3289	56
15	1352	48	*25*	3234	62

Experimental Results Figure 4.6 shows the wire length-critical path delay space produced by CS and VPR-based placement for the 18 configurations (using $Bits = 24$ as reference). The label besides each point indicates the corresponding configuration. For example, "B7" means this point corresponds to the configuration using $Bits = 7$. Figure 4.7 shows the same design space using CS. From these two figures, we can clearly see that CS and VPR find the same pareto-points, *i.e.*, optimal configurations of this design space, such as $B7$, $B8$ and $B9$. In addition, the overall shapes of the two design spaces match well. This demonstrates that our circuit similarity not only works well at low level logic synthesis, but also at high level algorithm level. Moreover, in terms of runtime, CS and CS-t, respectively, achieve 7X and 30X speedup compared to VPR.

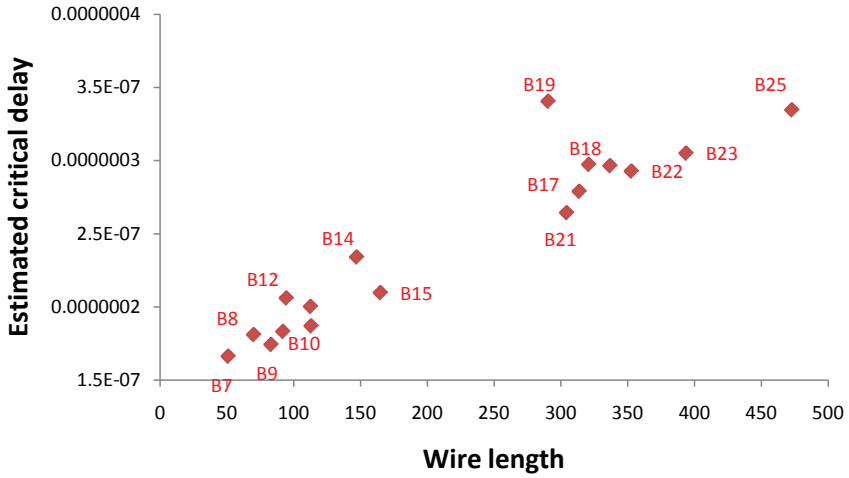


Figure 4.6: Wire length-delay space of VPR for 18 configurations.

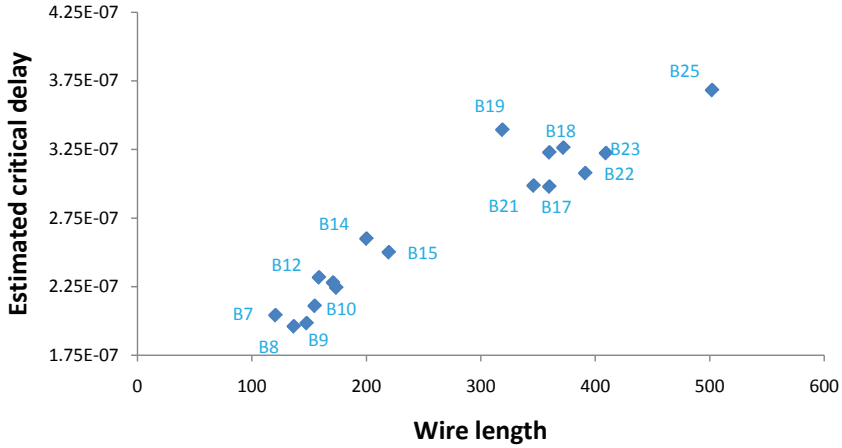


Figure 4.7: Wire length-delay space of CS for 18 configurations.

4.3 Summary

In this Chapter, we presented the experimental CAD flow and results on FPGA incremental design case and FPGA design space exploration case, respectively. For the incremental design case study, we first explained the CAD flow and experimental settings. In essence, the CAD flow consists of two iterations, where the first iteration performs a standard synthesis, mapping, placement and routing flow, while the second iteration uses the proposed similarity-based flow compared with "from-scratch" VPR flow. We conducted our experiments on two optimization scripts: a mild "imfs" resynthesis script and an aggressive "rwsat2" resynthesis scripts. We compared the CSBP placement quality against VPR using the core measurements, such as wire length, critical delay, area, bounding box cost, delay cost and runtime. Our experimental results showed that CSBP generates comparable or even better placement results in the above criteria with averaged more than 30X speedup. By increasing the circuit size, more speedup is achieved up to 100X. The results demonstrate CSBP is

able to detect the similarity among both minor and radical changes in the incremental design process and significantly saves runtime in particular to very large scale circuits.

For the design space exploration case study, we performed our experiments on logic-level design space and algorithm-level design space. For logic-level design space, the changes are due to the logic-level optimization to netlists or gates, which generally results in local similarity among different circuit designs. In contrast, for algorithm-level design space, since the changes are due to different high level optimization algorithms, the resulting circuit designs generally share global similarities. For both case studies, we also first presented the CAD flow and experimental settings. For the logic-level case study, we constructed the design space with 19 configurations on each of the 20 MCNC benchmarks. We compared the initial and final placement quality and runtime to VPR, and experimentally showed CSBP is able to significantly improve the initial placement quality with 76% reduction in bounding box cost and 48% reduction in delay cost. Moreover, CSBP generates comparable final placement results with 30X speedup compared to VPR. We also characterized the shape of the design space in terms of minimal, median and maximal wire length and critical delay. The almost identical cures produced by CSBP and VPR proved CSBP is able to accurately capture the design characteristics of a design space and provides insights for IC designers. For algorithm-level case study, we used CMU SPIRAL multiplier block generator to generate different RTL designs using different algorithms and configurations. We focused on the wire length-critical path delay space produced by CS and VPR and demonstrated that CSBP is able to comprehensively understand the underlying structural similarities among circuits and precisely pinpoint the most optimized designs in a design space thus provides significant convenience to IC designers.

The next Chapter concludes this thesis and points out the future research direction for applying CSBP to more applications.

Chapter 5

Conclusions

In this thesis, we have presented an efficient circuit similarity algorithm. Based on this algorithm, we have proposed a Circuit Similarity-Based Placement (CSBP), and applied it to incremental design for FPGAs and design space exploration for FPGAs. For incremental design, experimental results show that CSBP is able to accurately capture the similarity from the previous design iterations for both mild and aggressive synthesises and thus reduce the engineering effort. For design space exploration, experimental results show that CSBP can precisely depict the shape of the design space and closely match the design curves. Compared with the state-of-the-art VPR tool, CSBP is averaged 31X and 30X faster while preserving the wire length and critical delay in incremental design and design space exploration, respectively. Moreover, up to 100X speedup can be achieved when circuit gets larger, therefore, demonstrates the CSBP's premium scalability. The significant speedup is achieved because of the high-quality initial placement generated based on circuit similarity.

We first addressed our Hypothesis 1 by proposing the "circuit similarity" concept based on the iterative graph similarity algorithm used for undirected molecular graphs, and adapted it to consider the unique circuit properties and reduce the expensive computational complexity. Different from molecular graph similarity algorithm, we update the circuit similarity scores for one node based on the edges that leave the nodes and edges that enter the nodes, respectively. In addition, since primary inputs and primary outputs are fixed in incremental design and design space exploration processes, the matchings for those nodes are fixed and not updated in each iteration. In order to improve the efficiency of the circuit similarity algorithm, we devised two pruning techniques, namely support constraint and level constraint to reduce the number of node pairs needed to be updated in each iteration. We experimentally demonstrated that with a strong level constraint pruning and a strong support constraint pruning, and approximately 90% and 99% of the nodes can be pruned.

We then addressed our Hypothesis 2 by proposing a circuit similarity-based CAD flow for FPGAs. In essence, the flow produces a highly optimized initial placement for the modified logic-level/algorithm-level netlists based on the layout of the original netlist and the circuit similarity between the original and modified logic-level/algorithm-level netlists. Based on this initial solution, an efficient refinement is then performed as a fine-grain tuning for further improvement of the layout

quality. Note that such a refinement procedure does not require a radical change of the existing CAD tools and can be inserted as a simple plug-in. The essential information obtained from the previous design iterations is automatically captured and quantified by a runtime-efficient "similarity detection" phase. The circuit similarity based CAD flow can be applied to various FPGA applications. We used the circuit similarity-based flow to accelerate the FPGA incremental design process and FPGA design space exploration process.

We finally addressed our Hypothesis 3 by implementing the circuit similarity-based placement and experimentally compared the results in two FPGA applications: incremental design and design space exploration. For the incremental design case study, we chose two optimization synthesis scripts to perform our experiments, a mild resynthesis "imfs" and an aggressive resynthesis "rwsat2". We compared the key criteria against VPR tool, such as bounding box cost, delay cost, wire length, area, critical delay and runtime, and CSBP is able to generate comparable or better placement quality with averaged more than 30X speedup and up to 100X on very large circuits. For the design space exploration case study, in addition to those criteria compared in incremental design, we also investigated the design space characterization by comparing the wire length-critical path delay space. The experimental results show that CSBP is able to capture the intrinsic structural similarities among different circuit designs, and thus provides a useful insight for design space exploration.

There are some promising and interesting future projects resulting from this thesis work, including integrating predefined matchings (*e.g.*, the naming matching) into the CSBP flow to further enhance both the efficiency and the quality of the design and applying the circuit similarity algorithm to routing and sequential verification for FPGAs.

Bibliography

[1] A. G. Dempster and M. D. Macleod . Use of Minimum-adder Multiplier Blocks in FIR Digital Filters. *IEEE Transactions in Circuits and Systems-II: Analog and Digital Signal Processing*, 42:569–577, 1995.

[2] A. Ling, S. Brown and J. Zhu. Towards Automated ECOs in FPGAs. *International Symposium on Physical Design*, 2000.

[3] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang. SAT-based Logic Optimization and Resynthesis. *International Workshop on Logic and Synthesis*, 2007.

[4] A. Mishchenko, S. Chatterjee, J. H. Jiang and Robert Brayton. Integrating Logic Synthesis, Technology Mapping, and Retiming. *In Proceedings of the International Workshop on Logic and Synthesis*, 2005.

[5] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton. FRAIGs: A Unifying Representation for Logic Synthesis and Verification. *Technical Report, EECS Dept., UC Berkeley*, 2005.

[6] A. Sangiovanni-Vincentelli, A. El Gamal and J. Rose. Synthesis Method for Field Programmable Gate Arrays. *Proceedings of the IEEE*, pages 1057–1083, 1993.

[7] A.Nayak A.Choudhary, M.Haldar and P.Banerjee. Parallel Algorithms for FPGA Placement. *Proceedings of the 10th Great Lakes Symposium on VLSI*, pages 86–94, 2000.

[8] A.M. Smith, J. Das, S.J.E. Wilton. Wirelength Modeling for Homogeneous and Heterogeneous FPGA Architectural Development. *ACM International Symposium on FPGAs*, 2009.

[9] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/~alanmi/abc/, Release 70930.

[10] J.Chandy S.Kim B.Rankumar, S.Parkers and P.Banerjee. An Evaluation of Parallel Simulated Annealing Strategies with Application to Standard Cell Placement. *Technology Computer-Aided Design*, 16:398–410, 1997.

[11] H. Bunke. Error Correcting Graph Matching: On the Influence of the Underlying Cost Function. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21:917–922, 1999.

[12] C. Alpert and A. Kahng. Recent Directions in Netlist Partitioning: A Survey. *Integration, the VLSI Journal*, 19:1–81, 1995.

[13] C. Ebeling, L. McMurchie, S. A. Hauck and S. Burns. Placement and Routing Tools for the Triptych FPGA. *IEEE Transactions on VLSI*, pages 473–482, 1995.

[14] P.Maidee C.Ababei and K.Bazargan. Time-driven Partitioning-based Placement for Island Style FPGAs. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 24:395–406, 2005.

[15] D. Chen, J. Cong and P. Pan. FPGA Design Automation: A Survey. *Found. Trends Electron. Des. Autom.*, 1:139–169, 2006. ISSN 1551-3076.

[16] D. J. Creasey. *Advanced Signal Processing*. IEE Telecommunications Series, 1985. ISBN 0863410375.

[17] D. M. Miller and M. A. Thornton. *Multiple Valued Logic: Concepts and Representations*. Morgan and Claypool Publishers, 2008. ISBN 9781598291902.

[18] D. Sheldon and F. Vahid. Making Good Points: Application-Specific Pareto-Point Generation for Design Space Exploration using Statistical Methods. *ACM International Symposium on FPGAs*, 2009.

[19] D. Singh and S. Brown. Incremental Placement for Layout Driven Optimizations on FPGAs. *International Conference on Computer Aided Design*, pages 752–759, 2002.

[20] D. Zacher. Turbo-Charge Your FPGA Design Iterations. http://chipdesignmag.com/display.php?articleId=3497, 2007.

[21] E. Lawler, K. Levitt and J. Turner. Module Clustering to Minimize Delay in Digital Networks. *IEEE Transactions on Computers*, pages 47–57, 1966.

[22] G. Jeh and J. Widom. A Measure of Structural-context Similarity. *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining*, 2002.

[23] A. V. Goldberg. An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm. *Journal of Algorithms*, 22:1–29, 1997.

[24] S.Yildiz M.Markov I. Villarrubia, P.Parakh and Madden. Benchmarking for Large Scale Placement and Beyond. *Proceedings of the International Symposium on Physical Design*, pages 95–103, 2003.

[25] J. Cong and M. Sarrafzadeh. Incremental Physical Design. *International Symposium on Physical Design*, 2000.

[26] J. Cong and W. Jiang. Pattern-based Behavior Synthesis for FPGA Resource Reduction. *Proc. 16th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2008.

[27] J. Cong and Y. Ding. Flowmap: An Optimal Technology Mapping Algorithm For Delay Optimization in Lookup-table based FPGA Designs. *Technology Computer-Aided Design*, 1994.

[28] J. Cong, L. He, C. Koh and P. Madden. Performance Optimization of VLSI Interconnect Layout. *Integration, The VLSI Journal*, 21:1–94, 1996.

[29] J. Das, S.J.E. Wilton, W. Luk, P.H.W. Leong. Modeling Post-Techmapping and Post-Clustering FPGA Circuit Depth. *International Conference on Field-Programmable Logic*, 2009.

[30] J. Rubinstein, P. Penfield and M. Horowitz. Signal Delay in RC Tree Networks. *IEEE Transactions on Computer Aided Design*, pages 202–211, 1983.

[31] K. C. Chen, J. Cong, Y. Ding, A. Kahng and P. Trajmar. DAG-Map: Graph Based FPGA Technology Mapping For Delay Optimization. *IEEE Design and Test Magzine*, pages 7–20, 1992.

[32] K. Chang, D. A. Papa, I. L. Markov and V. Bertacco. InVerS: An Incremental Verification System with Circuit Similarity Metrics and Error Visualization. *International Symposium on Quality Electronic Design*, 2007.

[33] K. Chang, I. L. Markove and V. Bertacco. Fixing Design Errors with Counterexamples and Resynthesis. *IEEE Journal on Technology in Computer-Aided Design*, 27:184–188, 2008.

[34] L. Hagen and A. Kahng. Fast Spectral Methods for Ratio Cut Partitioning and Clustering. *International Conference on Computer Aided Design*, pages 10–13, 1991.

[35] L. Mcmurchie and C. Ebeling. PathFinder: A Negotiation-based Performance-driven Router for FPGAs. *In Proceedings of International Symposium on Field Programmable Gate Arrays*, 1995.

[36] D. Leong. *Incremental Placement for Field-Programmable Gate Arrays*. PhD thesis, University of British Columbia, 2004.

[37] M. Alexander, J. Cohoon, J. Ganley and G. Robins. An Architecture-Independent Approach to FPGA Routing Based on Multi-Weighted Graphs. *European Design Automation Conference*, pages 259–264, 1994.

[38] M. Khellah, S. Brown and Z. Vranesic. Modeling Routing Delays in SRAM-based FPGAs. *Proceedings Canadian Conference on VLSI*, pages 13–18, 1993.

[39] M. L. Fernandez and G. Valiente. A Graph Distance Metric Combining Maximum Common Subgraph and Minimum Common Supergraph. *Pattern Recognition Letters*, 22:735–758, 2001.

[40] M. Püschel, B. Singer, J. Xiong, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso and R. W. Johnson . SPIRAL: A Generator for Platform-Adapted Libraries of Signal Processing Algorithms . *Journal of High Performance Computing and Applications*, 18:21–45, 2004.

[41] M. Rupp, E. Proschak and G. Schneider. Kernel Approach to Molecular Similarity Based on Iterative Graph Similarity. *Journal of Chemical Information and Modeling*, 47:2280–2286, 2007.

[42] M. Xu and F. Kurdahi. Area and Timing Estimation for Lookup Table Based FPGAs. *European Design and Test Conference*, 1996.

[43] N. Togawa, K. Hagi and M. Yanagisawa. An Incremental Placement and Global Routing Algorithm for Field Programmable Gate Arrays. *Asia and South Pacific Design Automation Conference*, pages 519–526, 1998.

[44] O. Macindoe and W. Richards. Graph Comparison using Fine Structure Analysis. *Proceedings of IEEE Conference on Social Computing*, 2010.

[45] P. Suaris, L. Liu, Y. Ding and N. Chou. Incremental Physical Resynthesis for Timing Optimization. *International Symposium on Field-Programmable Gate Arrays*, pages 99–108, 2004.

[46] M. Pelillo. Matching Free Trees, Maximal Cliques and Monotone Game Dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:1535–1541, 2002.

[47] R. Albert and A. L. Barabasi. Statistical Mechanics of Complex Networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[48] R. Brayton, G. Hachtel and A. Sangiovanni-Vincentelli. Multilevel Logic Synthesis. *Proceedings of the IEEE*, pages 264–300, 1990.

[49] R. Francis, J. Rose and Z. Vranesic. Chortle-crf: Fast Technology Mapping for Lookup Table-based FPGAs. *Design Automation Conference*, pages 227–233, 1991.

[50] R. Hitchcock, G. Smith and D. Cheng. Timing Analysis of Computer Hardware. *IBM Journal of Research and Development*, pages 100–105, 1983.

[51] S. Brown and J. Rose. FPGA and CPLD Architectures: A Tutorial. *IEEE Design and Test of Computers*, 12:42–57, 1996.

[52] S. Krishnaswamy, H. Ren, N. Modi, R. Puri. DeltaSyn: An Efficient Logic Difference Optimizer for ECO Synthesis. *International Conference on Computer Aided Design*, 2009.

[53] S. Melnik, H. Garcia-Molina and A. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[54] S. Nag and R. Rutenbar. Performance-driven Simultaneous Place and Route for Island-Style FPGAs. *International Conference on Computer Aided Design*, pages 332–338, 1995.

[55] T. Givargis and F. Vahid. Platune: A Tuning Framework for System-on-a-Chip Platforms. *IEEE Transactions on Computer Aided Design*, 21:1317–1327, 2002.

[56] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0-262-03293-7.

[57] V. Betz and J. Rose. *VPR: A New Packing, Placement and Routing Tool for FPGA Research*. 1997.

[58] V. Betz, J. Rose and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999. ISBN 0792384601.

[59] V. Blondel, A. Gajardo, M. Heymans, P. Senellart and P. Van Dooren. A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching. *Society for Industrial and Applied Mathematics Review*, 46:647–666, 2004.

[60] V. Manohararajah, S. D. Brown and Z. Vranesic. Heuristics for Area Minimization in LUT-based FPGA Technology Mapping. *In Proceedings of the International Workshop on Logic and Synthesis*, pages 14–21, 2004.

[61] C. A. J. van Eijk. Sequential Equivalence Checking based on Structural Similarities. *IEEE Transactions on Computer-Aided Design*, 19:814–819, 2000.

[62] A.Ludwin V.Betz and K.Padalia. High-quality, Deterministic Parallel Placement for FPGAs on Commodity Hardware. *ACM/Sigda International Symposium on FPGAs*, pages 14–23, 2008.

[63] R.Aggarwal V.Kumar, G.Karypis and S.Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. *Proceedings ACM/IEEE Design Automation Conference*, 1997.

[64] W. Elmore. The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers. *Journal of Applied Physics*, pages 55–63, 1948.

[65] X. Shi, D. Zeng, Y. Hu, G. Lin and O. R. Zaiane. Accelerating FPGA Design Space Exploration Using Circuit Similarity-Based Placement. *International Conference on Field-Programmable Technology*, pages 373–376, 2010.

[66] X. Shi, D. Zeng, Y. Hu, G. Lin and O. R. Zaiane. Enhancement of incremental design for FPGAs using circuit similarity. *The International Symposium on Quality Electronic Design*, pages 243–250, 2011.

[67] X. Shi, Y. Hu, G. Lin and O. R. Zaiane. CSBP: A Fast Circuit Similarity-Based Placement for FPGA Incremental Design and Design Space Exploration. Integration, the VLSI Journal (In Review), 2011.

[68] Xilinx Corporation. SmartCompile Technology: SmartGuide. *Xilinx Press Release*, 2008.

[69] Y. Hu, V. Shih, R. Majumdar and L. He. Exploiting Symmetry in SAT-Based Boolean Matching for Heterogeneous FPGA Technology Mapping. *The International Conference on Computer-Aided Design*, 2007.

[70] Y. S. Yang, S. Sinha, A. Veneris and R. K. Brayton. Automating Logic Rectification by Approximate SPFDs. *Asia-South Pacific Design Automation Conference*, 2007.

[71] Y. Sankar and J. Rose. Trading Quality for Compile Time: Ultra-fast Placement for FPGAs . 1999.

[72] Y. Voronenko and M. Pschel . Multiplierless Multiple Constant Multiplication. *ACM Transactions on Algorithms*, 3(2), 2007.

[73] S. Yang. Logic Synthesis and Optimization Benchmarks User Guide, Version 3.0. *Technical Report, Microelectronics Center of North Carolina*, 1991.

[74] Y.Xu and M.A.S. Khalid. Qfd: Efficient quadratic placement for fpgas. *International Conference on Field Programmable Logic and Application*, pages 555–558, 2005.

[75] L. Zager. *Graph Similarity and Matching*. PhD thesis, Massachusetts Institute of Technology, 2005.

Appendix A

A Complete List of Commands in ABC

Here is a complete list of resynthesis optimization commands used in ABC tool from file abc.rc in the download package.

resyn ”b; rw; rwz; b; rwz; b”

resyn2 ”b; rw; rf; b; rw; rwz; b; rfz; rwz; b”

resyn2a ”b; rw; b; rw; rwz; b; rwz; b”

resyn3 ”b; rs; rs -K 6; b; rsz; rsz -K 6; b; rsz -K 5; b”

compress ”b -l; rw -l; rwz -l; b -l; rwz -l; b -l”

compress2 ”b -l; rw -l; rf -l; b -l; rw -l; rwz -l; b -l; rfz -l; rwz -l; b -l”

choice ”fraig_store; resyn; fraig_store; resyn2; fraig_store; fraig_restore”

choice2 ”fraig_store; balance; fraig_store; resyn; fraig_store; resyn2; fraig_store; resyn2; fraig_store; fraig_restore”

rwsat ”st; rw -l; b -l; rw -l; rf -l”

rwsat2 ”st; rw -l; b -l; rw -l; rf -l; fraig; rw -l; b -l; rw -l; rf -l”

shake ”st; ps; sat -C 5000; rw -l; ps; sat -C 5000; b -l; rf -l; ps; sat -C 5000; rfz -l; ps; sat -C 5000; rwz -l; ps; sat -C 5000; rfz -l; ps; sat -C 5000”

share ”st; multi -m; fx; resyn2”

src-rw ”st; rw -l; rwz -l; rwz -l”

src-rs ”st; rs -K 6 -N 2 -l; rs -K 9 -N 2 -l; rs -K 12 -N 2 -l”

src-rws ”st; rw -l; rs -K 6 -N 2 -l; rwz -l; rs -K 9 -N 2 -l; rwz -l; rs -K 12 -N 2 -l”

resyn2rs ”b; rs -K 6; rw; rs -K 6 -N 2; rf; rs -K 8; b; rs -K 8 -N 2; rw; rs -K 10; rwz; rs -K 10 -N 2; b; rs -K 12; rfz; rs -K 12 -N 2; rwz; b”

compress2rs ”b -l; rs -K 6 -l; rw -l; rs -K 6 -N 2 -l; rf -l; rs -K 8 -l; b -l; rs -K 8 -N 2 -l; rw -l; rs -K 10 -l; rwz -l; rs -K 10 -N 2 -l; b -l; rs -K 12 -l; rfz -l; rs -K 12 -N 2 -l; rwz -l; b -l”

resyn2rsdc ”b; rs -K 6 -F 2; rw; rs -K 6 -N 2 -F 2; rf; rs -K 8 -F 2; b; rs -K 8 -N 2 -F 2; rw; rs -K 10 -F 2; rwz; rs -K 10 -N 2 -F 2; b; rs -K 12 -F 2; rfz; rs -K 12 -N 2 -F 2; rwz; b”

compress2rsdc ”b -l; rs -K 6 -F 2 -l; rw -l; rs -K 6 -N 2 -F 2 -l; rf -l; rs -K 8 -F 2 -l; b -l; rs -K 8 -N 2 -F 2 -l; rw -l; rs -K 10 -F 2 -l; rwz -l; rs -K 10 -N 2 -F 2 -l; b -l; rs -K 12 -F 2 -l; rfz -l; rs -K 12 -N 2 -F 2 -l; rwz -l; b -l”