國立臺灣大學電機資訊學院電子工程學研究所

碩士論文

Graduate Institute of Electronics Engineering
College of Electrical Engineering & Computer Science
National Taiwan University

Master Thesis

針對大型異質現場可程式化邏輯閘陣列之巨集暨解析擺置

Packing and Analytical Placement for Large-Scale Heterogeneous FPGAs

陳昱臻

Yu-Chen Chen

指導教授：張耀文 博士

Advisor: Yao-Wen Chang, Ph.D.

中華民國 103 年 1 月

January 2014

Acknowledgements

I would like to express my gratitude to my advisor, Professor Yao-Wen Chang, for his tremendous advice, instruction, and guidance throughout my pursuit of the Master's degree. I learned many important things from him, including research methodologies, presentation skills, technical writing, and most importantly the attitude towards things. Besides, I greatly appreciate the committee members of my oral defense, Professor Hung-Ming Chen, Professor Shih-Hsu Huang, and Professor Jiun-Lang Huang for their valuable comments and suggestions.

I am grateful to every member of the Electronic Design Automation Laboratory in National Taiwan University for many inspiring discussions and useful comments on my research and constant encouragement during my studies in graduate school. Especially, I would like to thank Sheng-Yen Chen, who helped me enormously with my research. I would also like to thank Kuan-Hsien Ho and Po-Ya Hsu for the hard but wonderful time we have been through together in the Gate Sizing Contest.

My deepest appreciation goes to my dear parents and my younger brother. Without their love and constant support, the completion of this thesis would not have been possible.

Yu-Chen Chen

National Taiwan University
January 2014

針對大型異質現場可程式化邏輯閘陣列之巨集暨解析擺置

研究生：陳昱臻　　指導教授：張耀文 博士

國立臺灣大學電子工程學研究所

摘要

隨著現場可程式邏輯閘陣列的演進，其晶片大量採用複雜元件如隨機存取記憶體以及數位訊號處理器以實現當代電路設計廣泛使用的矽智產。而複雜元件電路往往伴隨著資料路徑。在過去，電路巨集與擺置的問題雖然已被廣泛研究，資料路徑電路在電路巨集與擺置上的考量卻大多被忽視。然而，未考慮資料路徑之巨集與擺置將破壞電路的規律性，從而降低效能。除此之外，隨著電路設計複雜度的急遽上升，可擴展性已成為主要考量。另外，多數巨集之演算法過度聚集元件，也造成擺置上的困難並限制了擺置的最佳化。然而，學術界廣為使用的標竿電路卻遠小於現實電路，不足以顯現出演算法之可擴展性。

因此，在這篇論文中，我們針對上述問題提出了一個考慮密集資料

路徑結構的巨集與擺置演算法。我們所提出的巨集演算法針對複雜元件、資料路徑以及一般元件分別採用不同的巨集方法。我們的巨集方法提供後續的擺置更大的彈性和潛力以得到更好的最佳化。此外，我們提出 V 型架構來加強演算法的可擴展性。而我們所提出的擺置演算法使用非線性最佳化的方式降低線長並針對密集資料路徑結構作處理以獲得高品質的擺置。實驗結果顯示出，我們所提出的方法相較於先前的研究，可以有效地得到非常好的電路擺置結果，並且可以在大型異質標竿電路中得到驗證。

關鍵詞：實體設計、資料路徑、電路巨集、電路擺置、異質電路

PACKING AND ANALYTICAL PLACEMENT FOR LARGE-SCALE HETEROGENEOUS FPGAS

Student: Yu-Chen Chen **Advisor: Dr. Yao-Wen Chang**

Graduate Institute of Electronics Engineering
National Taiwan University

Abstract

As *field programmable gate arrays* (FPGAs) have evolved, heterogeneous components such as *random access memory blocks* (RAMs) and *digital signal processing blocks* (DSPs) are widely applied to effectively implement *intellectual property* (IP) cores extensively used in modern circuits. The RAMs and DSPs are often accompanied with datapath-intensive circuits. Although FPGA packing and placement for general logic blocks have been studied extensively, not much work addresses the optimization of datapath. Packing and placement without considering datapath could break the regularity and lead to significantly worse results. Besides, scalability has become a first-order metric for modern FPGA design, mainly due to the dramatically increasing design complexity. Most commonly used academic benchmark suites, however, are relatively much smaller than the latest commercial FPGA chips, which may not demonstrate scalability well. Furthermore, most of the existing works on packing tend to be over-packed, which increases the difficulty for placement and limits the placement optimization.

Therefore, to solve the issues above, we propose efficient and effective packing and analytical placement algorithms in this thesis. Our packing algorithm is composed of three stages to handle different structures of heterogeneous components and datapath blocks. Our packing provides a more placement-friendly netlist than VPR's, which gives a great potential for placers to achieve significant quality improvement. A V-shaped framework is proposed to enhance the scalability while considering more exact design constraints than existing works. Moreover, our wirelength-driven analytical placement algorithm applies effective nonlinear optimization techniques and utilizes the regularity of the datapaths to achieve scalability and high quality. A complex-block-alignment function is proposed to better handle the heterogeneity, and a multilevel framework is applied to further enhance the scalability of our placement algorithm.

Experimental results show that our method is scalable while preserving high quality. Our approach achieves a 199.80× speedup, compared to the VPR's latest packing flow (VPR 7.0), and a 3.07× speedup with 6% shorter wirelength, compared to the VPR's latest placement flow, based on a set of modern large-scale FPGA benchmarks, Titan23 benchmark suites. The overall flow achieves a 18.30× speedup with 50% shorter wirelength, compared to the VPR's packing and placement flow.

Keywords: Physical Design, Datapath, Packing, Placement, Heterogeneous

Table of Contents

Acknowledgements ii

Abstract (Chinese) iii

Abstract v

List of Tables ix

List of Figures x

Chapter 1. Introduction 1

1.1 Introduction to FPGA . . . 3

1.1.1 FPGA Architecture . . . 3

1.1.2 FPGA Design Flow . . . 4

1.2 Datapath Circuit in FPGA . . . 5

1.3 FPGA Packing . . . 6

1.4 FPGA Placement . . . 10

1.5 Related Work . . . 14

1.6 Motivation . . . 15

1.7 Our Contributions . . . 18

1.8 Thesis Organization . . . 19

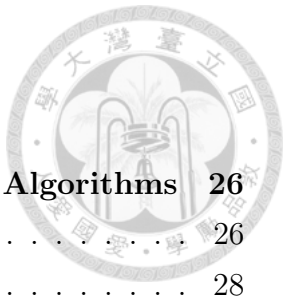
Chapter 2. Preliminaries 20

2.1 Problem Formulation . . . 20

2.2 Analytical Placement . . . 21

2.2.1 Wirelength Model . . . 23

2.2.2 Density Model . . . 24



Chapter 3. The Heterogeneous Packing and Placement Algorithms 26
3.1 Algorithm Overview 26
3.2 Heterogeneous Packing 28
3.2.1 Identification Packing 29
3.2.2 Datapath Extraction and Packing 30
3.2.3 General Logic Packing 33
3.2.3.1 Affinity Calculation 33
3.2.3.2 Packing Order 34
3.2.4 V-Shaped Framework 35
3.3 Heterogeneous Analytical Placement 36
3.3.1 Multilevel Mixed-Size Prototyping 37
3.3.2 Look-Ahead Legalization 41
3.3.3 Datapath Placement 43
3.3.4 General Logic Refinement 43
3.3.5 Detailed Placement 45

Chapter 4. Experimental Results 46

Chapter 5. Conclusions and Future Work 55

Bibliography 57

List of Tables

4.1 Statistics of the Titan23 benchmarks. 47

4.2 Comparison of packing results. 49

4.3 Comparison of block number and average nets of a block after packing. 50

4.4 Comparison of packing quality using the same placer, NTUFPGA. . 50

4.5 Comparison of placement quality using the same packer, VPR 7.0. . 50

4.6 Comparison of packing and placement results. 51

List of Figures

1.1 A sample of heterogeneous FPGAs. 2

1.2 Basic components of FPGAs. (a) A typical BLE consists of a k-input LUT, a FF, and a MUX. (b) A CLB contains several BLEs. 3

1.3 A typical FPGA design flow. 4

1.4 An example of a datapath circuit. 5

1.5 An example of packing. 7

1.6 Illustrations of complex blocks. (a) A RAM block. (b) A DSP block. (c) A k-bit adder. (d) A k-bit shift-register chain. 9

1.7 Homogeneous FPGA placement. 10

1.8 Heterogeneous FPGA placement. 14

1.9 Comparison of different packing densities. (a) Dense packing. (b) Sparse packing. 16

2.1 The FPGA placement flow. 22

2.2 An illustration of the bell-shaped function. 25

3.1 The overall flow of the proposed algorithms. 27

3.2 An example of RAM configuration. 29

3.3 Comparison of the placement results. (a) Placement with both M9K and M144K blocks. (b) Placement with only M9K blocks. 30

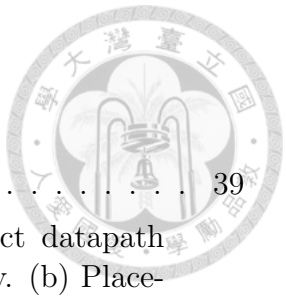
3.4 The flow of the proposed datapath extraction algorithm. 31

3.5 The attribution code of a net. 32

3.6 An example of a datapath extraction and packing. 32

3.7 The flow of the proposed V-shaped framework. 35

3.8 The mismatch between global placement and legalization. (a) The result of global placement. (b) The result after legalization. 38



3.9 An illustration of our guiding function for DSPs. 39

3.10 An example of conducting a more regular and compact datapath placement. (a) Placement without considering regularity. (b) Placement considering regularity. (c) Placement considering regularity and further refinement. 44

4.1 A result of our global placement. 52

4.2 A result of our legalization. 53

4.3 A result of our detailed placement. 53

4.4 A result of VPR placement. 54

Chapter 1

Introduction

Field programmable gate arrays (FPGA) is a compelling proposition against *application specific integrated circuit* (ASIC) in the market due to their dramatically increasing logic density, short time-to-market, and flexibility. Being field programmable, FPGAs demand for very fast, scalable, yet high-quality place-and-route tools to cope with the high complexity of modern large-scale design. Besides, modern circuits extensively use *intellectual property* (IP) cores, which are accompanied with datapath-intensive circuits. Most packing and placement works on FPGA, however, do not address the optimization of datapath, which could break the regularity and lead to significantly worse results. Therefore, in this thesis, we propose efficient and effective packing and analytical placement algorithms for large-scale heterogeneous FPGAs. In the following sections, we first give a brief introduction to FPGAs in Section 1.1 and introduce datapath circuits in Section 1.2. Next, we describe FPGA packing and placement problems in Section 1.3 and Section 1.4, respectively. Then, a survey of related work is given in Section 1.5. After that, we explain our motivation in Section 1.6 and summarize our contributions in Section 1.7. Finally, in Section 1.8, we show the organization of the rest of this thesis.

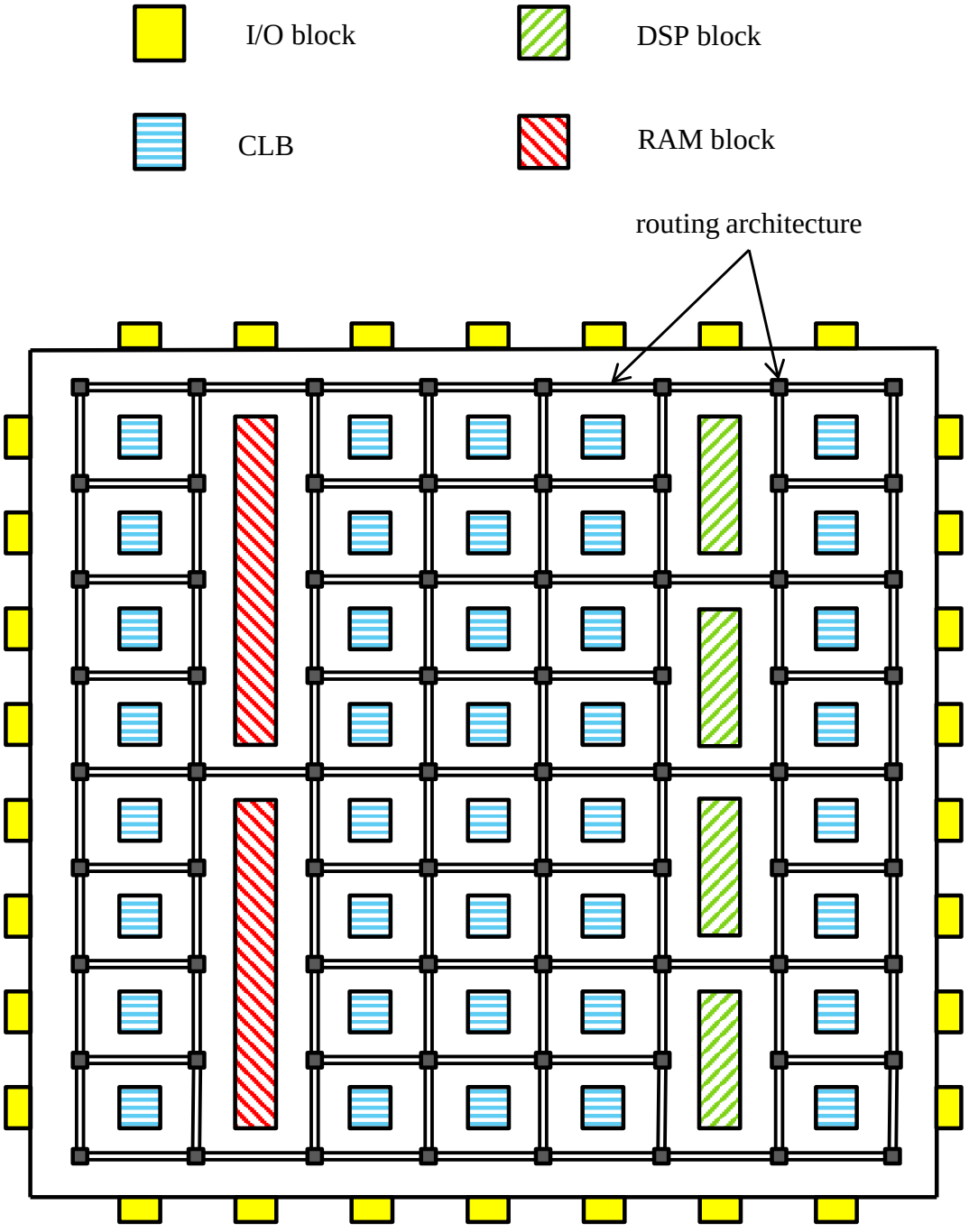


Figure 1.1: A sample of heterogeneous FPGAs.

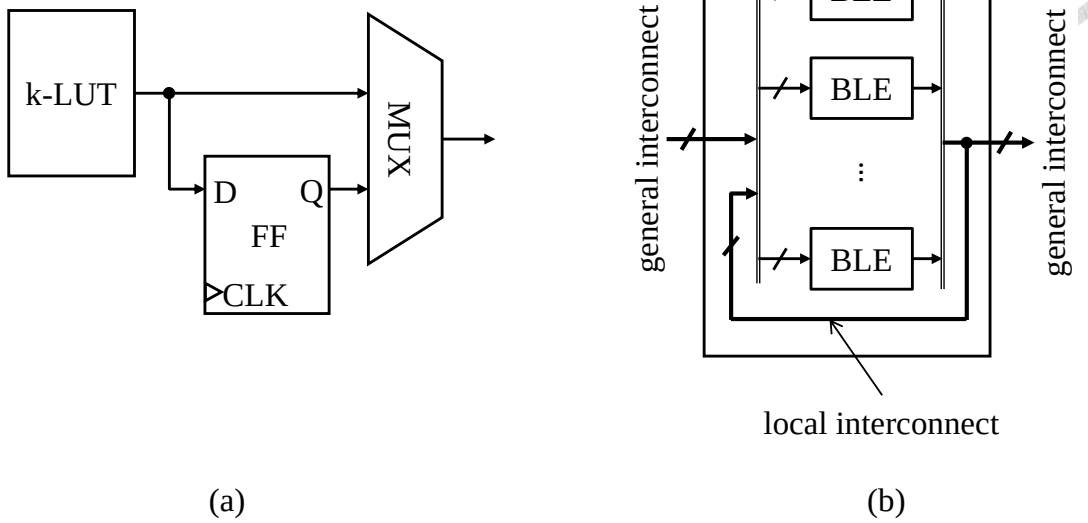


Figure 1.2: Basic components of FPGAs. (a) A typical BLE consists of a k-input LUT, a FF, and a MUX. (b) A CLB contains several BLEs.

1.1 Introduction to FPGA

In this section, we first introduce the FPGA architecture in Section 1.1.1 and show the FPGA design flow in Section 1.1.2.

1.1.1 FPGA Architecture

A typical heterogeneous FPGA, as illustrated in Figure 1.1, has five main components: *configurable logic blocks* (CLBs), *random access memory blocks* (RAMs), *digital signal processing blocks* (DSPs), programmable routing architecture, and input/output (I/O) blocks. In a heterogeneous FPGA, a two-dimensional array of CLBs, RAMs, and DSPs is surrounded by general routing resources bounded by I/O blocks [1, 2]. A CLB usually contains several *basic logic elements* (BLEs). We say that a CLB containing N BLEs has *cluster size N*. Moreover, a BLE is composed of a simple combinational logic such as a lookup table (LUT) and a sequential

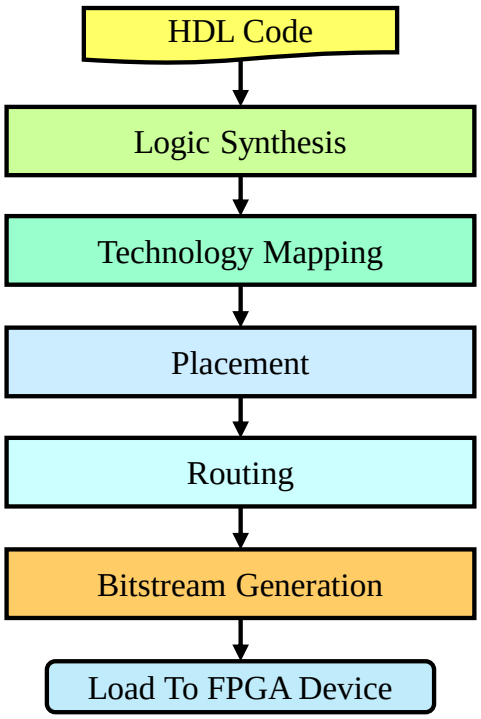


Figure 1.3: A typical FPGA design flow.

logic like a flip-flop (FF). A K-input LUT can implement any function of K inputs. Figures 1.2 (a) and (b) depict a BLE and a CLB, respectively.

1.1.2 FPGA Design Flow

A typical FPGA design flow consists of five main stages [23]: (1) logic synthesis, (2) technology mapping, (3) placement, (4) routing, and (5) bitstream generation. Logic synthesis converts the high-level logic written in hardware description language (HDL) into a gate-level circuit. After logic synthesis, circuit gates will be grouped to best fit the FPGA logic resources such as CLBs through technology mapping. Then, the physical locations of blocks in the technology-mapped netlist will be determined in the placement stage, and connections between CLBs, RAMs, DSPs, and I/O blocks will later be determined during the routing stage. Finally,

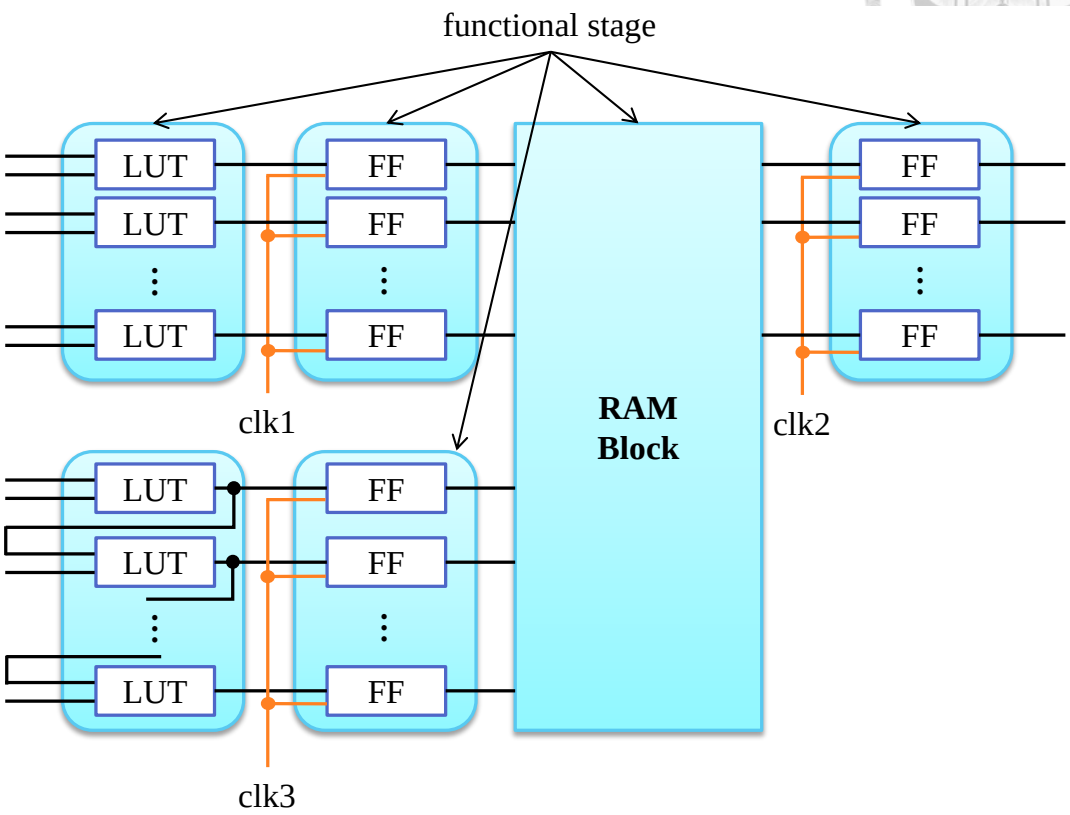


Figure 1.4: An example of a datapath circuit.

bitstream generation generates a binary file which will decide all of the FPGA programming points to configure the CLBs and interconnections. Figure 1.3 shows the FPGA design flow.

1.2 Datapath Circuit in FPGA

In FPGA circuits, complex blocks (such as RAMs, DSPs, and arithmetic logic blocks) are used to efficiently implement IP cores. These complex blocks tend to have regular structures in their forward and backward stages. Figure 1.4 gives an example of a datapath circuit in FPGA. The circuit contains four functional stages, and each functional stage is composed of several logic blocks which are the same

type. A functional stage usually has high-degree controlling nets that should be aligned together to achieve better performance.

To achieve better packing and placement performance, datapath regularity is widely considered to obtain more physically compact placement results and shorter wirelength. Blocks in datapath circuits, however, are rarely packed together in general packing algorithms because (1) they are often connected by high-degree nets, and (2) they tend to have few interconnections with each other. This could break the regularity and lead to significantly worse results. Therefore, for better packing and placement results, it is of particular importance to spare no effort on such datapath-intensive circuits. Earlier works such as Callahan *et al.* [10] proposed a netlist extraction algorithm and a datapath placement algorithm. Ye *et al.* [52] proposed a packing algorithm for datapath blocks. In this thesis, we propose the first datapath extraction, packing, and placement algorithms to better exploit the regularity of the datapaths in heterogeneous FPGA circuits.

1.3 FPGA Packing

Packing (also known as clustering) is a crucial step of the FPGA CAD flow. Traditionally, packing falls between technology mapping and placement, but in modern flows, packing is tightly integrated with technology mapping or placement. After technology mapping, gates in a netlist are mapped into LUTs and FFs. As shown in Figure 1.5, grouping LUTs and FFs that connect to the same net into a coarse-grained block could decrease the net degree. Moreover, a net with all its terminals in the same coarse-grained block will not be considered in the placement stage, which could reduce the problem size of placement, and further optimize metrics such as area and routability. Furthermore, packing BLEs into CLBs needs to satisfy the constraints on the cluster size N, the maximum distinct inputs I, and/or the controlling

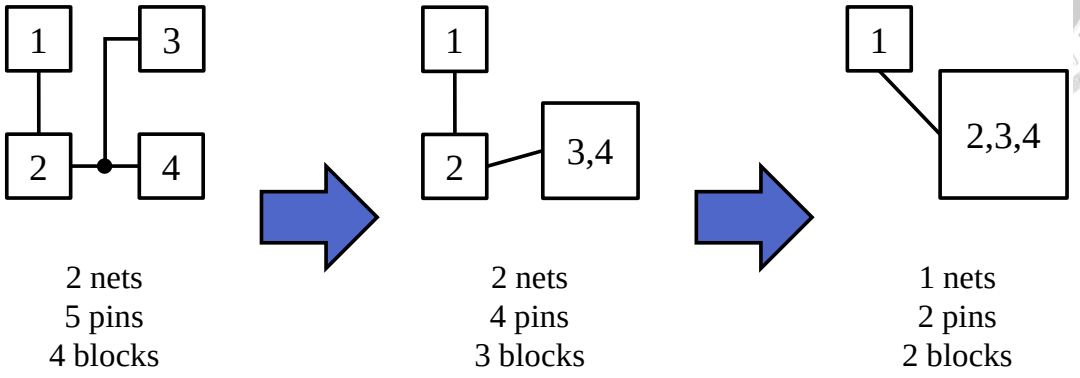


Figure 1.5: An example of packing.

signals. The exact requirements are specified based on the given architecture.

The methodologies applied by numerous packing works can be classified into four categories:

1. **Seed-based approach:** This approach is first proposed in VPack [6] and is widely applied. The seed-based approach constructs one CLB at a time. A BLE is first chosen from all the unpacked BLEs by an attraction function, and set as the seed of a new cluster. Then, an unpacked BLE is selected using a second attraction function, and packed with the seed. The second step repeats until the cluster is full. If the cluster needs more distinct inputs than the limit, a hill climbing technique is applied to seek BLEs that do not increase the number of distinct inputs used by the cluster. Packing is done by repeating the two steps above. Different objectives could be achieved by applying different attraction functions on selecting a BLE. Example works of this approach are as follows: VPack, T-VPack [38], RPack [9], iRAC [46], LAP [22], T-RDPack [43], Yang *et al.* [51], and MO-Pack [44].

2. **Depth optimal approach:** This approach attempts to duplicate timing-critical logics during packing, and merge blocks blocks on timing-critical paths

into the same CLB. Although this approach obtains much shorter critical path delay, the process of logic duplication often leads to large increases in area. Besides, the timing estimation made during packing may not be accurate when compared with the final placement result [36]. Representative works of this approach are as follows: TLC [16], MLC [47], and RCP [17].

3. **Partition-based approach:** This approach applies k-way partitioning, and then adjusts the clusters that violate CLB constraints because the constraints cannot be formulated into the hypergraph for k-way hMetis [29]. Although k-way partitioning produces a great initial packing, the mismatch after adjustment increases as the constraints become complex. Representative works of this approach are Marrakchi *et al.* [39] and Feng [18].

4. **Affinity-based approach:** This approach is mainly applied in ASIC design (such as first choice [28] in NTUplace3 [13] and best choice [5] in mPL6 [11]). The affinity-based approach recursively merges two clusters that have the best affinity and satisfy the CLB constraints. Packing is finished when the number of the clusters meets the expectation or no cluster could be merged. An example work of this approach is HD-Pack [12].

As mentioned in HD-Pack, affinity-based algorithms give a great net elimination which is better than seed-based algorithms, but fail to further pack when the sizes of the most clusters are larger than half capacity. Therefore, in this thesis, we develop our general CLB packing based on a hybrid algorithm that applies a best choice algorithm with a capacity threshold to trigger a successive packing procedure.

As FPGAs have evolved, packing also needs to handle complex blocks. Figures 1.6 (a) – (d) illustrate a RAM block, a DSP block, an adder, and a shift register

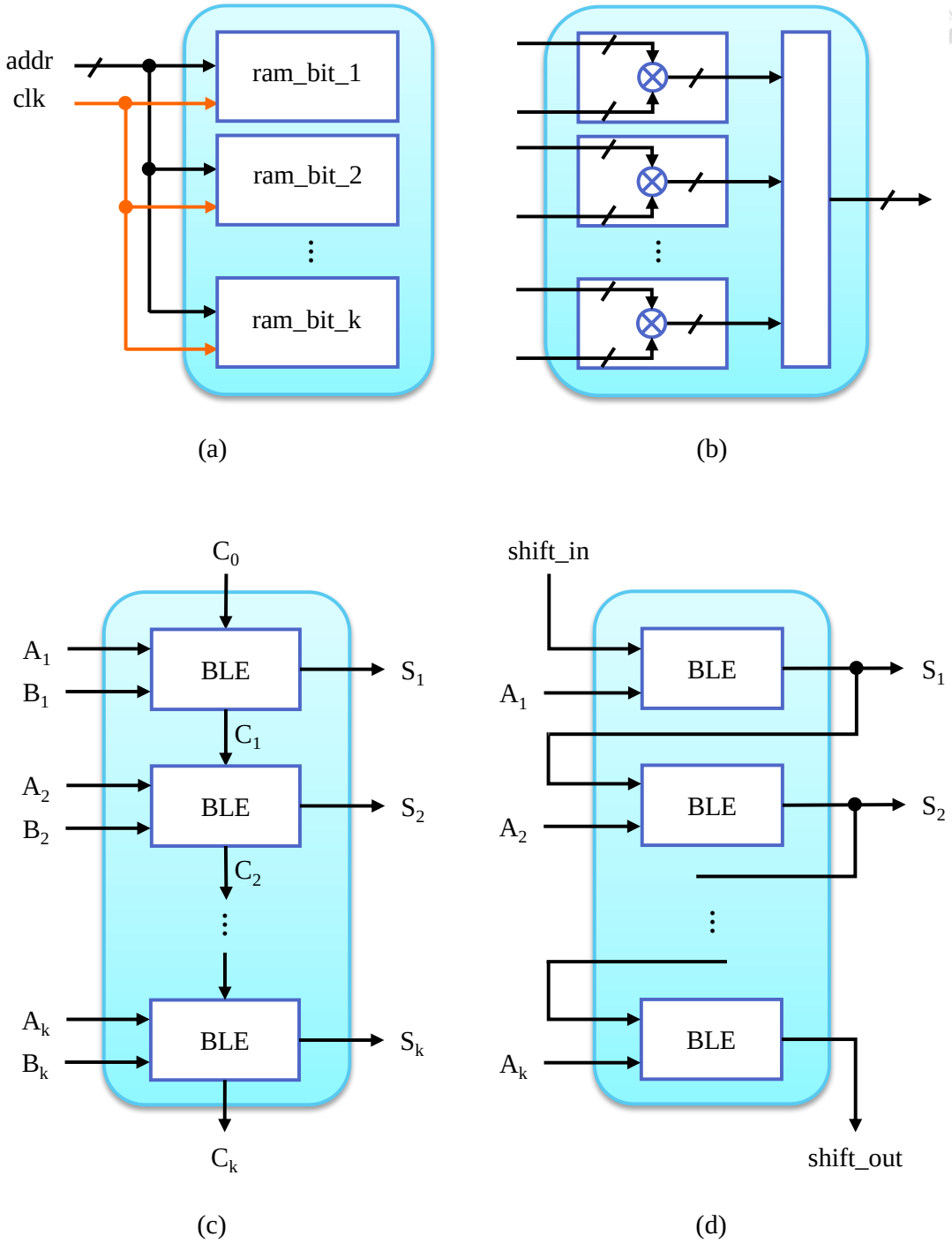


Figure 1.6: Illustrations of complex blocks. (a) A RAM block. (b) A DSP block. (c) A k-bit adder. (d) A k-bit shift-register chain.

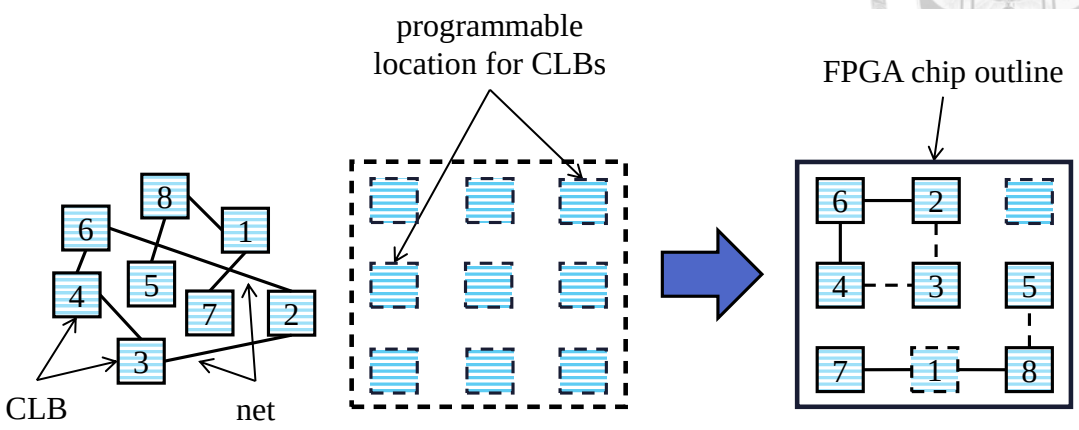


Figure 1.7: Homogeneous FPGA placement.

chain, respectively. The structures of complex blocks are different from that of general CLBs. Ahmed *et al.* [3] proposed an effective memory packing configuration, and AAPack [33] presented a complex block architecture description language for its packing. To utilize complex blocks effectively, in this thesis, we present an identification packing algorithm for complex blocks.

1.4 FPGA Placement

Placement is one of the main stages that falls between technology mapping and routing in the typical FPGA CAD flow. FPGA placement determines non-overlapping positions for *technology-mapped* logic blocks in the netlist with optimized cost metrics (such as the total wirelength or routability). Figure 1.7 shows an example of FPGA placement. Every block is assigned to a unique position of CLBs. The placement problem is computationally difficult. The wirelength minimization of the unit-size block placement with only two-pin nets is proved to be NP-complete [19]. Therefore, placement is considered as one of the most critical and time-consuming stages among the FPGA CAD flow.

Many works on FPGA placement are mainly based on the following three

approaches:

1. **Simulated annealing (SA) approach:** This approach optimizes placement by SA techniques. Given an initial solution, the SA approach obtains solutions by iteratively perturbing the current solution to generate a new solution. The new solution is kept if it is better than the current solution. Otherwise, an acceptance probability function is applied to decide whether to keep the new solution or not. The probability function helps to escape from local optimum solutions. The state-of-the-art academic FPGA CAD tool VPR [7, 8, 34, 37] adapted SA techniques to be its optimization engine. Besides the basic SA techniques, VPR also improved aspects including: (1) incremental net bounding box updating to improve the placement runtime, (2) better temperature updating so that the annealing process takes longer time when perturbations produces more improvements while saving time for perturbations with less improvements. The SA-based method has been dominating for decades because it can achieve very high-quality placement results. Nevertheless, it tends to have long runtime in large circuits.

2. **Partitioning-based approach:** The partitioning-based approach is proposed to achieve better speedups than the SA approach. Example works of this approach are FPR [4] and PPFF [35], and PPFF is the classic of this approach. PPFF applies the famous multilevel partitioner hMetis which recursively partitions the design and places it hierarchically. At each hierarchical level, PPFF employs an alignment cost in the objective function for delay and congestion minimization. Finally, PPFF applies a low-temperature SA flow, which is basically the VPR SA flow with smaller initial starting temperature than VPR. Although the partitioning-based approach achieves much better

speedup, it lacks global view when performing the partitioning and therefore can easily fall into local optimum solutions. As a result, the partitioning-based approach usually suffers from some quality loss.

3. **Analytical-based approach:** In recent years, the analytical-based approach, as a rising star in FPGA placement, shows rather fast and comparable or even higher solution quality compared to the SA approach. The analytical-based approach applies smooth functions to approximate objective functions and solves the problem by efficient numerical methods. Example works of this approach are as follows: QPF [50], CAPRI [20], StarPlace [49], HeAP [21], and Lin *et al.* [32].

   QPF applied the *quadratic wirelength model* widely used in VLSI placement [31, 45]. The quadratic wirelength model computes the wirelength of a net as the summation of the squared *Euclidean distance* over every two fan-outs of the net. By solving the quadratic programming problem with the total quadratic wirelength over all nets, QPF finds the locations of CLBs. Finally, a low-temperature SA flow is applied.

   CAPRI is based on metric geometry and graph embedding. CAPRI constructs the metric space graph according to routing architecture and embeds the netlist graph into this metric space graph with bipartite graph matching. This bipartite graph matching minimizes the distortion between the placement of the netlist graph and the metric space graph. Finally, CAPRI applied a legalization and a low-temperature SA flow to refine its solution.

   StarPlace proposed a *star+* model. This star+ model is modified from the famous *star* model [40] and is near-linear and continuously differentiable. The traditional star model wirelength for each net is estimated by summing up the

Euclidean distance between each block connected to the net and the *center-of-gravity* of the net. Different from the star model, this star+ model adds square root to the wirelength which is claimed to better approximate the real routed wirelength. StarPlace basically minimizes the sum over all net wirelength using the successive over-relaxation (SOR) optimization solver.

Lin *et al.* applied non-linear optimization using log-sum-exponential (LSE) as the wirelength approximation function and bell-shaped overlap function as density function. This work applied a multilevel framework to accelerate the placement algorithm and enhances the scalability. A partitioning-based look-ahead legalization is introduced to have a better forecast of the solution. Finally, Lin *et al.* refined its solution by a window-based bipartite matching and a low-temperature SA.

Along with the dramatically increasing gate count of modern FPGAs, developing and applying analytical placement tools which are both fast and high-quality have become inevitable trends in FPGA design. Therefore, in this thesis, we develop our placer based on the analytical approach.

Most existing works on FPGA placement focus on CLB placement. Nevertheless, with the advances in process technology, IP cores have become indispensable components in modern FPGAs. As a result, in heterogeneous FPGA placement, the legal positions of blocks are further constrained by the type of blocks. The distribution of the configurable locations for these IP cores, however, is limited and scattered on FPGAs. For example, as shown in Figure 1.1, RAMs could only be placed in Column two, and DSPs could only be placed in Column six. Figure 1.8 illustrates heterogeneous FPGA placement. Notice that analytical-based approaches use continuous and differentiable objective functions, which are against the discrete

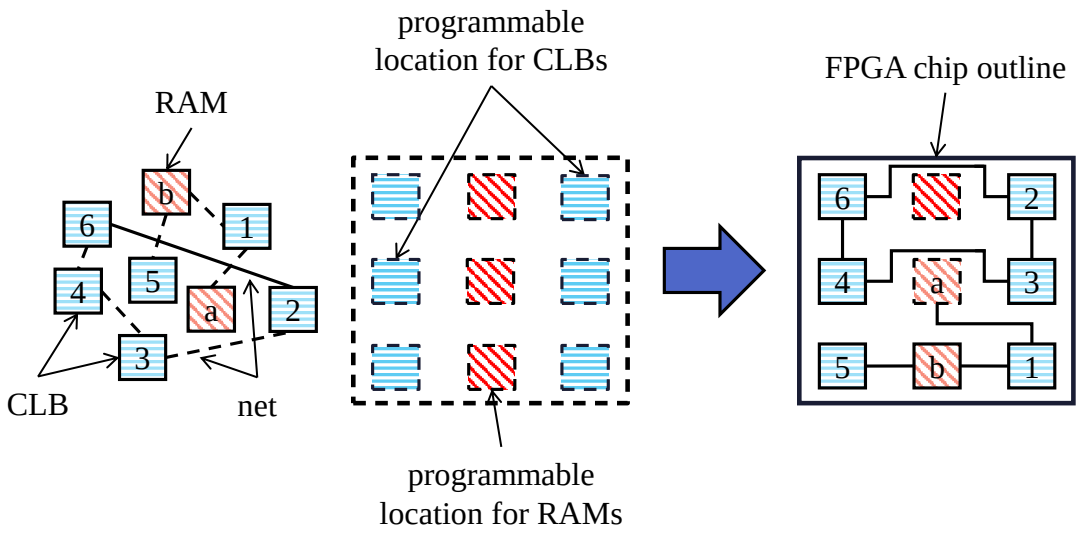


Figure 1.8: Heterogeneous FPGA placement.

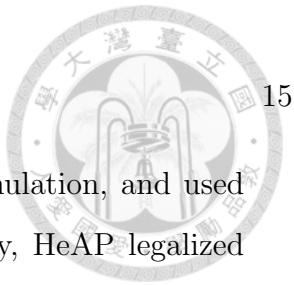
nature of FPGAs. The analytical FPGA placement problem has thus reshaped. Moreover, the increasing design complexity has made the FPGA placement problem even more challenging. Therefore, in this thesis, we propose a guiding function for analytical FPGA placement to cope with the mismatch between discrete and continuous features.

1.5 Related Work

Among many works on FPGA placement, LLP [48], HeAP, and VPR 7.0 [41] are the only three works, to the best of our knowledge, that considered heterogeneous resources.

LLP obtained an initial solution by applying a quadratic method and eliminating the overlap between modules by local spreading. Then, LLP applied low-temperature SA for each type of the modules separately to get the final placement solution.

HeAP adapted an ASIC academic placer, simPL [30], to FPGA placement.



HeAP used a bound2bound net model to form a quadratic formulation, and used legalization to spread the modules. To handle the heterogeneity, HeAP legalized (spread) each type of the modules separately.

VPR, as the state-of-the-art academic open-source FPGA CAD tool, supports both heterogeneous packing and placement. The packing algorithm of VPR 7.0 extended and improved AAPack and [27], which considered complex blocks (such as RAMs and DSPs) and general logic blocks. VPR 7.0 applied SA-based techniques for its placement as well.

The size of FPGA design and the volume of FPGA chips grow dramatically. The sizes of academic benchmarks used in HeAP and LLP, however, are much smaller than the sizes of the latest commercial FPGAs. The largest circuit in HeAP and LLP fills only 3% of the latest commercial FPGAs. Therefore, these benchmarks might not demonstrate the scalability well. Meanwhile, VPR, though runs on a set of modern large-scale FPGA benchmarks, could consume significant runtime in the packing stage and get limited wirelength minimization in the placement stage. The runtime of packing even dominates the runtime of place-and-route. Furthermore, a large set of works on packing, such as VPack, T-VPack, iRAC, and MO-Pack, though targeted on different cost metrics, apply the same procedure as VPR's, which are likely to encounter the same scalability problems.

1.6 Motivation

Placement is a very critical stage in the FPGA CAD flow as it could considerably affect subsequent routing results. Meanwhile, packing is a key step to provide a placement-friendly netlist. In the following, we summarize some key issues of current academic packers and analytical placers:

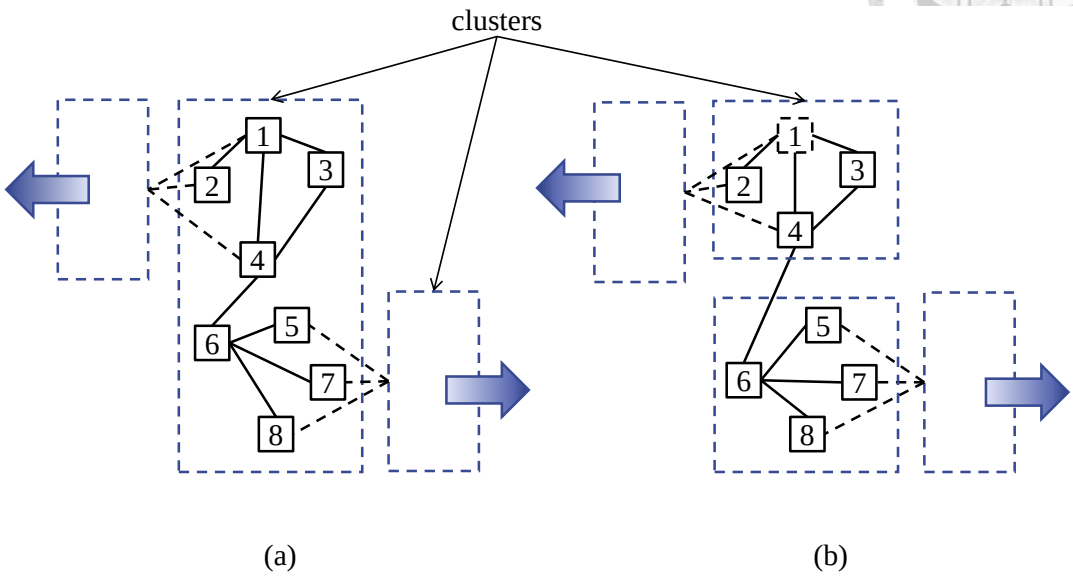


Figure 1.9: Comparison of different packing densities. (a) Dense packing. (b) Sparse packing.

- Modern FPGA design uses IP cores considerably. The IP cores are usually accompanied with datapath circuits. Current academic placers, however, may not handle datapath blocks very well because most of them did not distinguish the datapath blocks from random blocks (blocks without regular structures). Furthermore, placers may fail to extract the datapath blocks if packers break the structure or the regularity of the datapath blocks. These placers may place the datapath blocks irregularly and thus lead to inferior results. Therefore, we need to identify regular structures in the packing stage and place the datapath blocks regularly and compactly for better performance, shorter wirelength, etc.

- The ultimate objective of packing is to provide a placement-friendly netlist while optimizing the certain cost metrics. Hence, reducing the complexity of the given netlist (i.e., minimizing the number of blocks, nets, and pins) is always a key objective. Seed-based packing is extensively applied by academic packers, and meanwhile, seed-based packing is the basis of the VPR packer,

the most widely used packer by academic researchers for FPGA placement. The VPR packer, however, tends to be over-packed as they mentioned in their own work [41]. Take Figure 1.9 as an example. In Figure 1.9 (a), eight blocks are packed in the same CLB and connected with the other two blocks. During placement, if the two connected blocks happened to move to the opposite direction, the CLB gains two opposite forces. Moving the CLB close to either connected block increases the wirelength of nets connected with another block. If we split the CLB into two CLBs (i.e., packing the upper four blocks in a CLB and the lower four blocks in another CLB) as shown in Figure 1.9 (b), it would be much easier for placers to determine the block positions with less wirelength.

- There are three constraints in packing: (1) the cluster size N, (2) the maximum distinct inputs I, and (3) the FF controlling signals. Most packing works have considered the first two constraints. However, none of the packing works, to the best of our knowledge, has addressed constraints on FF controlling signals. This is against the reality of commercial FPGAs.

- Analytical-based methods use continuous and differentiable objective functions to determine block positions. The configurable distribution of complex blocks, however, is discrete and scattered. This difference may cause a significant mismatch between the results obtained from the objective functions and the final legalized results. Therefore, it is desirable to establish a function for analytical placement to handle the discrete feature of heterogeneous FPGAs.

Consequently, to cope with these issues, we develop heterogeneous packing and placement algorithms that can utilize the regular structures of datapth circuits and the state-of-the-art techniques to achieve high quality and scalability.

1.7 Our Contributions

In this thesis, we present efficient and effective packing and analytical placement algorithms for large-scale heterogeneous FPGAs. Our main contributions are summarized as follows:

- This is the first work, to the best of our knowledge, that introduces a datapath extraction technique to handle the special features of heterogeneous FPGAs in both packing and placement stages.

- We propose an efficient and effective packing algorithm for heterogeneous FPGA design, which handle the different structures of blocks. The algorithm consists of (1) identification packing, (2) datapath extraction and packing, and (3) general logic packing with a V-shaped framework. Our packing provides a more placement-friendly netlist than VPR's, which gives great potential for placers to achieve significant quality improvement. Experimental results show that, compared to the state-of-the-art academic FPGA CAD tool VPR, our algorithm leads to significant runtime speedups, while preserving good quality.

- This is the first work, to the best of our knowledge, that has addressed constraints on FF controlling signals. To cope with the constraints of the FF controlling signals in commercial FPGAs and enhance the scalability, we propose a V-shaped framework. This framework is applicable to various cost metrics in the seed-based and the affinity-based packing algorithms, and reduces the runtime of searching packing candidates by eliminating candidates that violate the constraints.

- We propose a novel guiding function for analytical placement algorithms on heterogeneous FPGAs to accurately model the discrete and scattered distribu-

tion of complex blocks (such as RAMs and DSPs). Our analytical placement algorithm consists of global placement and detailed placement. Four stages in global placement are (1) multilevel mixed-size prototyping with complex-block-alignment optimization which gives an approximate placement with the optimized wirelength, (2) look-ahead legalization which provides a quick forecast of the legalized results, (3) datapath placement which places the datapath blocks regularly and compactly, and (4) general logic refinement which further optimizes the total wirelength.

To show the scalability of our approach, the experiments were conducted on a set of modern large-scale FPGA benchmarks, Titan23 benchmark suites [41]. The circuits of Titan23 are composed of LUTs, FFs, RAMs, DSPs, and macros of arithmetic LUTs with carry chains. Experimental results show that our algorithms are scalable on both packing and placement. Compared to VPR, our algorithms can achieve a 199.80× speedup in the packing stage, and a 3.07× speedup with 6% shorter HPWL in the placement stage. The proposed overall flow achieves 18.30 × speedup with 50% shorter HPWL compared to the VPR's packing and placement flow.

1.8 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 gives the preliminaries. Chapter 3 presents our heterogeneous packing and analytical placement algorithm. Chapter 4 shows the experimental results. Finally, Chapter 5 concludes our work.

Chapter 2

Preliminaries

In this chapter, we first give our problem formulation of FPGA packing and placement in Section 2.1. Then, since our placement algorithm is based on analytical approaches, we will introduce the analytical placement framework in Section 2.2.

2.1 Problem Formulation

The FPGA packing problem is to cluster LUTs and FFs of a circuit into groups to minimize the number of blocks, block interconnections, pins on the block interconnections, and average inputs of the blocks such that the cluster size N, the maximum distinct inputs I, and the limitations of FF controlling signals specified in the given architecture are satisfied. The block interconnections are then regarded as nets and the clustered groups are regarded as the CLBs, RAMs, and DSPs in the following placement stage.

The heterogeneous FPGA placement problem can be formulated as a hypergraph $H = (V, E)$ placement problem. The notation used in this problem is listed as follows:

- V^C, V^R, and V^M: a set of the CLBs, RAMs, and DSPs, respectively.
- V: a set of blocks. V is the union of the three disjoint sets V^C, V^R, and V^M.

- E: a set of nets. $E \subseteq V \times V$.
- x_i and y_i: the center coordinate of block v_i, where $v_i \in V$.
- L^C, L^R, and L^M: a set of the legal x coordinates of the CLBs, RAMs, and DSPs in ascending order, respectively.

Given the sets of the legal x coordinates for each type of the blocks L^C, L^R, and L^M, we intend to determine the optimal positions (x_i, y_i) of each block v_i such that (1) the positions are legal (i.e., $x_i \in L^C \iff v_i \in V^C$, $x_i \in L^R \iff v_i \in V^R$, and $x_i \in L^M \iff v_i \in V^M$), and (2) there is no overlap among the blocks. The heterogeneous FPGA placement problem is solved in two major stages: (1) global placement with look-ahead legalization, and (2) detailed placement. Global placement evenly distributes all the blocks and obtains the best position for each block to minimize wirelength. Look-ahead legalization removes all the overlaps and gives a non-overlapping forecast. Finally, detailed placement further refines the solution. Figure 2.1 illustrates a typical analytical placement flow for FPGAs.

2.2 Analytical Placement

By dividing a placement region into a uniform non-overlapping bin grid, we can formulate the global placement problem as a constrained minimization problem as follows:

$$\begin{array}{ll} \min & W(\mathbf{x}, \mathbf{y}) \\ \text{s.t.} & D_b(\mathbf{x}, \mathbf{y}) \leq M_b, \text{ for each bin b,} \end{array} \tag{2.1}$$

where $W(\mathbf{x}, \mathbf{y})$ and $D_b(\mathbf{x}, \mathbf{y})$ are the wirelength and density functions, respectively. The density function is the total area of movable blocks in bin b, and M_b is the maximum allowable area of the movable blocks in bin b. Thus, $M_b = w_b h_b$, where w_b and h_b are the width and height of the bin, respectively. Notice that FPGA

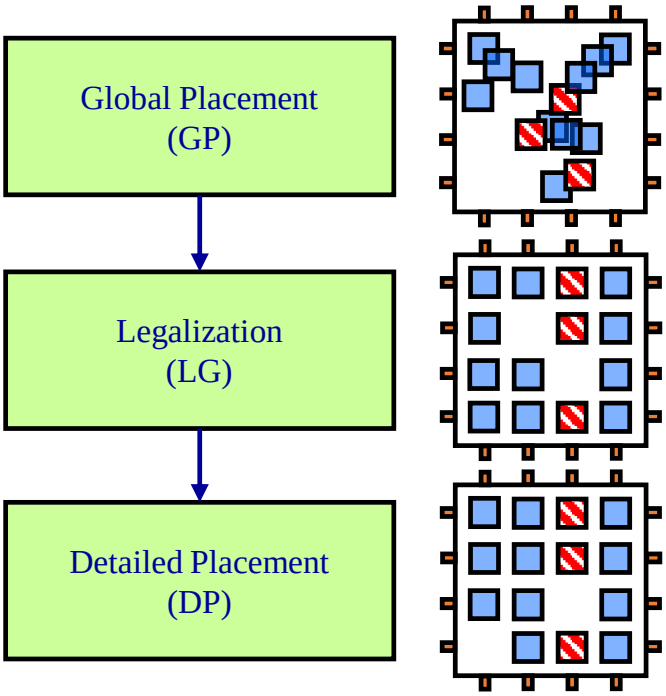


Figure 2.1: The FPGA placement flow.

placement does not have the same block density issue as that in ASIC placement because every block is required to be placed on a unique position. During wirelength minimization, movable blocks tend to concentrate at the center of the chip for smaller wirelength. This may cause a large displacement of the blocks from their original positions obtained by analytical optimization to the legal positions after legalization. Therefore, the density function here acts as a spreading force to the blocks while the blocks shrink to obtain smaller wirelength.

We can convert the above constrained minimization problem into an unconstrained optimization problem by introducing a penalty multiplier λ. As a result, we solve a sequence of unconstrained nonlinear optimization problems of the form

$$\min \quad W(\mathbf{x},\mathbf{y}) + \lambda \sum_{b} \max(D_b(\mathbf{x},\mathbf{y}) - M_b, 0)^2 \tag{2.2}$$

with increasing λ's. This unconstrained problem, which is a nonlinear optimization

problem, then can be solved by the conjugate gradient (CG) method. In a CG solver, the solution of the current iteration is used as the initial solution for the next iteration. This procedure repeats until the solution converges.

To apply the gradient search, the objective function must be smooth and differentiable everywhere. Therefore, we need to find such wirelength and density models to replace the above formulation. We introduce the differentiable wirelength and density models in the following two subsections.

2.2.1 Wirelength Model

During placement, the wirelength $W(\mathbf{x}, \mathbf{y})$ is usually defined as the total half-perimeter wirelength (HPWL). Because HPWL is not smooth and non-convex, several works on smooth wirelength approximation functions have been proposed. Both the log-sum-exp (LSE) wirelength model [42] and the weighted-average (WA) wirelength model [25] are widely used in placement. Although LSE and WA give good approximations to HPWL, HPWL estimation tends to be inaccurate in high-degree nets. The number of high-degree nets in FPGA circuits, however, are usually much more than that in ASIC circuits [14]. Therefore, we apply the weighted stable LSE model proposed in [32] to further enhance the accuracy of the wirelength approximation for FPGA placement. The weighted stable LSE model is defined as:

$$\begin{aligned} W(\mathbf{x}, \mathbf{y}) = \gamma \sum_{e \in E} w_f(e) (& \frac{X_{max_e}}{\gamma} + \ln \sum_{v_i \in e} \exp(\frac{x_i - X_{max_e}}{\gamma}) + \\ & \ln \sum_{v_i \in e} \exp(\frac{X_{min_e} - x_i}{\gamma}) - \frac{X_{min_e}}{\gamma} + \\ & \frac{Y_{max_e}}{\gamma} + \ln \sum_{v_i \in e} \exp(\frac{y_i - Y_{max_e}}{\gamma}) + \\ & \ln \sum_{v_i \in e} \exp(\frac{Y_{min_e} - y_i}{\gamma}) - \frac{Y_{min_e}}{\gamma}), \end{aligned} \tag{2.3}$$

where γ is a parameter to control the smoothness of the model, $w_f(e)$ stands for a compensating factor for the high-degree nets, and X_{max_e}, X_{min_e}, Y_{max_e}, and Y_{min_e} are constants that make sure the exponential terms will never overflow. For low-degree nets, Equation (2.3) approaches the exact HPWL when γ is small.

2.2.2 Density Model

The potential function is expressed as:

$$D_b(\mathbf{x}, \mathbf{y}) = \sum_{v \in V} P_x(b, v) P_y(b, v), \tag{2.4}$$

where $P_x(b, v)$ and $P_y(b, v)$ denote the overlaps between block v and bin b along the x and y directions, respectively. Because the density function $D_b(\mathbf{x}, \mathbf{y})$ is neither smooth nor differentiable, we apply the bell-shaped overlap function p_x proposed in [13] to replace the non-smooth function P_x. The overlap function in the x direction $p_x(b, v)$ is defined as:

$$p_x(b, v) = \begin{cases} 1 - ad_x^2, & 0 \le d_x \le \frac{w_v}{2} + w_b \\ b(d_x - \frac{w_v}{2} - 2w_b)^2, & \frac{w_v}{2} + w_b \le d_x \le \frac{w_v}{2} + 2w_b \\ 0, & \frac{w_v}{2} + 2w_b \le d_x, \end{cases} \tag{2.5}$$

where

$$a = \frac{4}{(w_v + 2w_b)(w_v + 4w_b)}$$
$$b = \frac{2}{w_b(w_v + 4w_b)},$$

w_b is the bin width, w_v is the block width, and d_x is the center-to-center distance of the block v and the bin b in the x direction. The smoothed overlap function $p_y(b, v)$ in the y direction can be defined in a similar way. Figure 2.2 shows the original and the smoothed overlap functions.

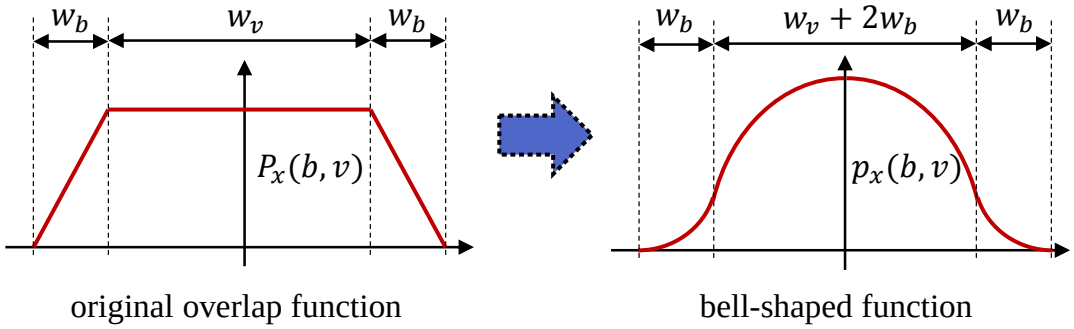


Figure 2.2: An illustration of the bell-shaped function.

Therefore, we obtain a smoothed function $\hat{D}_b$ to replace the non-smooth function $D_b(\mathbf{x}, \mathbf{y})$. $\hat{D}_b$ is defined as follows:

$$\hat{D}_b(\mathbf{x}, \mathbf{y}) = \sum_{v \in V} c_v p_x(b, v) p_y(b, v), \tag{2.6}$$

where c_v is a normalization factor that makes the total smoothed density of a block equal its area.

Finally, to enhance the scalability of our FPGA placement algorithm, we apply the V-cycle multilevel framework proposed in [32] in the global placement stage.

Chapter 3

The Heterogeneous Packing and Placement Algorithms

Packing and placement are two of the most crucial stages in the FPGA design flow because packing can minimize the problem size of placement and preserve the regularity of datapath circuits and placement considerably affects the subsequent routing results. To achieve better placement quality in large-scale heterogeneous FPGA design, we propose two algorithms for packing and analytical placement, respectively. In this chapter, we first give an overview of our proposed algorithms in 3.1. Then, we present our heterogeneous packing algorithm in 3.2. Finally, we introduce our wirelength-driven analytical placement algorithm in 3.3.

3.1 Algorithm Overview

Figure 3.1 shows the overall flow of the proposed algorithms. Our algorithms can be divided into heterogeneous packing and heterogeneous placement. Given device architecture, a technology mapped netlist, and I/O positions, our heterogeneous packing algorithm packs blocks based on different block structures to provide a placement-friendly netlist. In our heterogeneous packing algorithm, three stages are proposed in response to handle the different structures of complex blocks, datapath blocks, and general logic blocks. The three stages are listed as follows: (1) identification packing for the complex blocks, (2) a datapath extraction and packing,

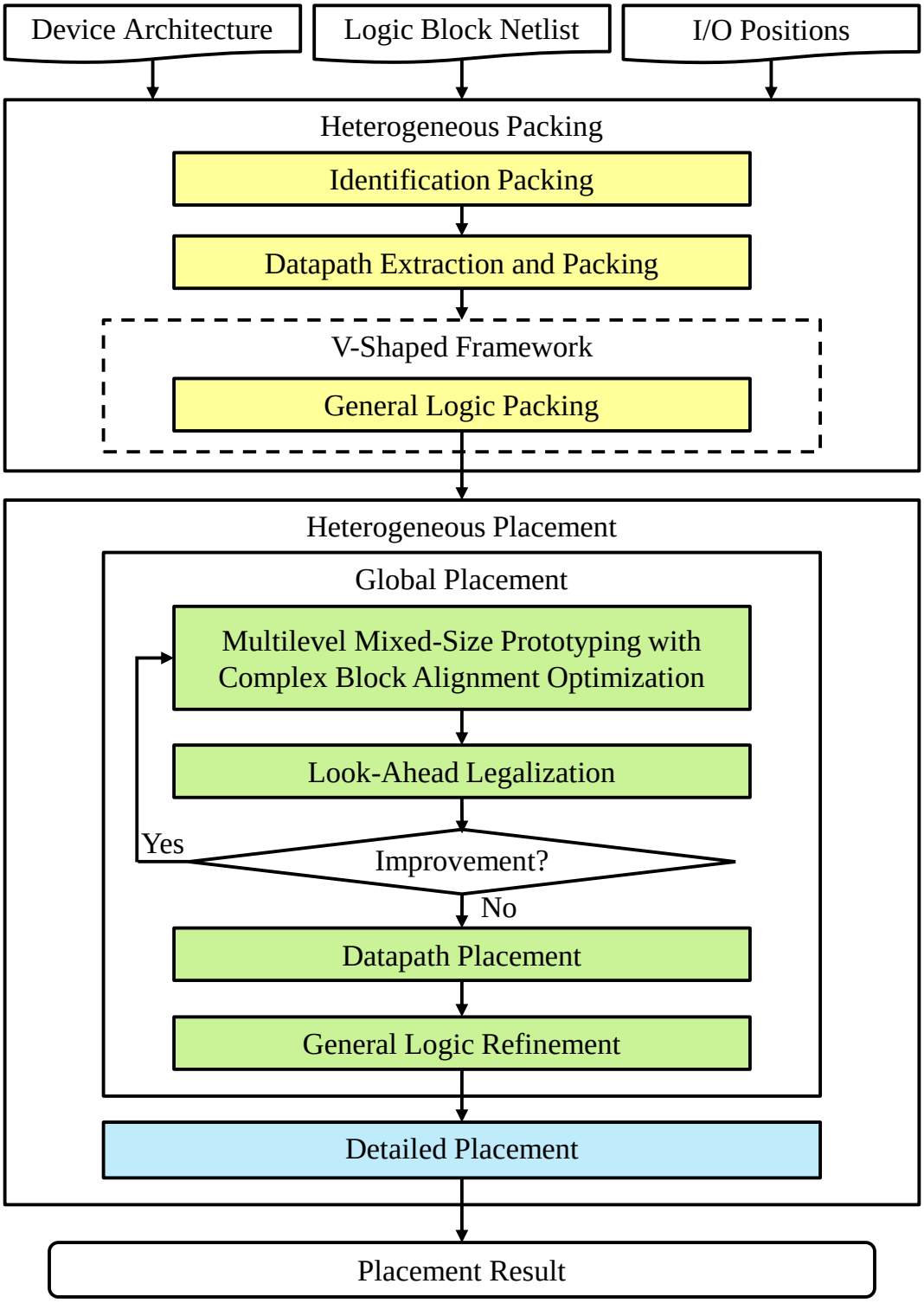
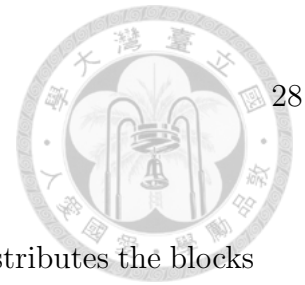


Figure 3.1: The overall flow of the proposed algorithms.



and (3) general logic packing under a V-shaped framework.

After heterogeneous packing, heterogeneous placement distributes the blocks evenly and minimizes the total wirelength in global placement and refines the placement solutions in detailed placement. To achieve scalability and high quality, our analytical placement algorithm applies effective nonlinear optimization techniques and utilizes the regularity of the datapaths to determine the block positions. There are four stages in our global placement: (1) multilevel mixed-size prototyping with complex-block-alignment optimization which gives an approximate placement with the optimized wirelength, (2) look-ahead legalization which provides a quick forecast of the legalized placement results, (3) datapath placement which fixes RAMs and DSPs and then places the datapath blocks regularly and compactly, and (4) general logic refinement which places the general logic blocks and optimizes the total wirelength. Our detailed placement algorithm further refines the placement solutions in both CLB and BLE levels to compensate the insufficient packing and placement qualities.

3.2 Heterogeneous Packing

As mentioned in Chapter 1, the structures of complex blocks and datapath blocks are different from general logic blocks and thus should be extracted and packed separately. In our heterogeneous packing algorithm, identification packing first picks out the complex blocks and packs them together. The datapath blocks are then extracted from the forward and backward stages of the complex blocks. Finally, the general logic blocks are packed to reduce the area and the problem size of place-and-route.

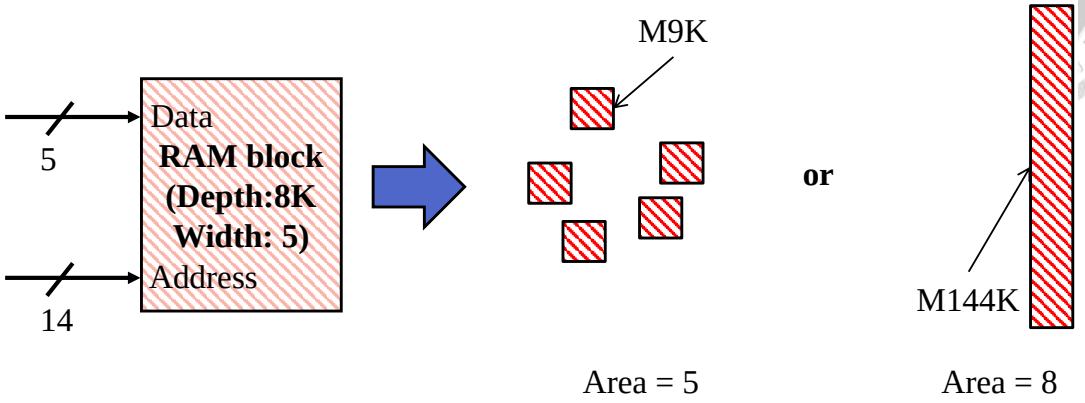
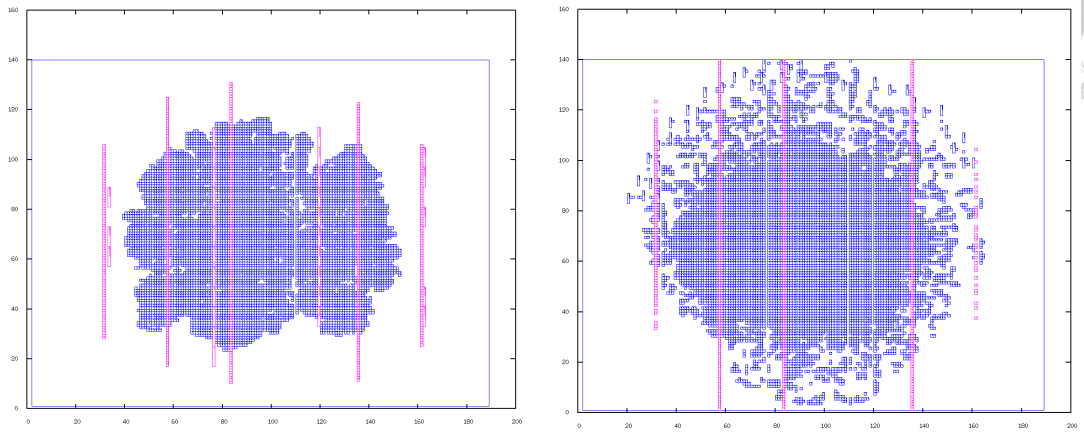


Figure 3.2: An example of RAM configuration.

3.2.1 Identification Packing

To enhance the flexibility of mapping blocks on different resources or devices of FPGAs, unencrypted IP cores are declared as several unit blocks in the netlist generated from the technology mapping stage. For example, as shown in Figure 3.2, a RAM block with depth 8K and width 5 could be instantiated on five M9Ks or one M144K of the Altera Stratix IV family [1]. Therefore, we need to identify and reconstruct the IP cores into complex blocks. The way to identify a RAM block is to seek memory blocks with the same address bits and the same controlling signals. The way to group multi-bit adders is to follow carry-in and carry-out signals. Similarly, we can group shift registers by shift-in and shift-out signals. For RAM blocks, we need to further determine their configurations (e.g., M9K or M144K). Basically, we determine the configurations of the RAM blocks based on their area. In Figure 3.2, the RAM block mapped into five M9Ks takes five unit areas while the RAM block mapped into a M144K takes eight unit areas. Therefore, we map the RAM block into five M9Ks. Nevertheless, the ratio of instantiated M9Ks and M144Ks should also be taken into consideration. Figures 3.3 (a) and (b) show a comparison that if all the RAM blocks in the circuits are instantiated into M9Ks, the placement region



(a) (b)

Figure 3.3: Comparison of the placement results. (a) Placement with both M9K and M144K blocks. (b) Placement with only M9K blocks.

increases due to the limited and scattered memory resources.

3.2.2 Datapath Extraction and Packing

To achieve physically compact placement results in FPGAs, we extract and preserve the regularity of the datapaths during packing. We apply a regularity extraction similar to that in [15]. Given an initial reference stage that contains blocks of the same type, our regularity extraction algorithm sequentially groups other blocks to grow functional stages of a datapath. Different from [15] which uses blocks connected by high-degree nets as the initial reference stages, in this thesis, we use the packed complex blocks in Section 3.2.1 as our initial reference stages.

To grow the functional stages, we perform forward and backward searches. Figure 3.4 illustrates the flow of our datapath extraction algorithm. In each datapath extraction, an initial reference stage is picked out and set as a reference stage. Here we first define an *attribute code* for each net as a sequence of types of all the blocks connected to the net, as shown in Figure 3.5. A more detailed and

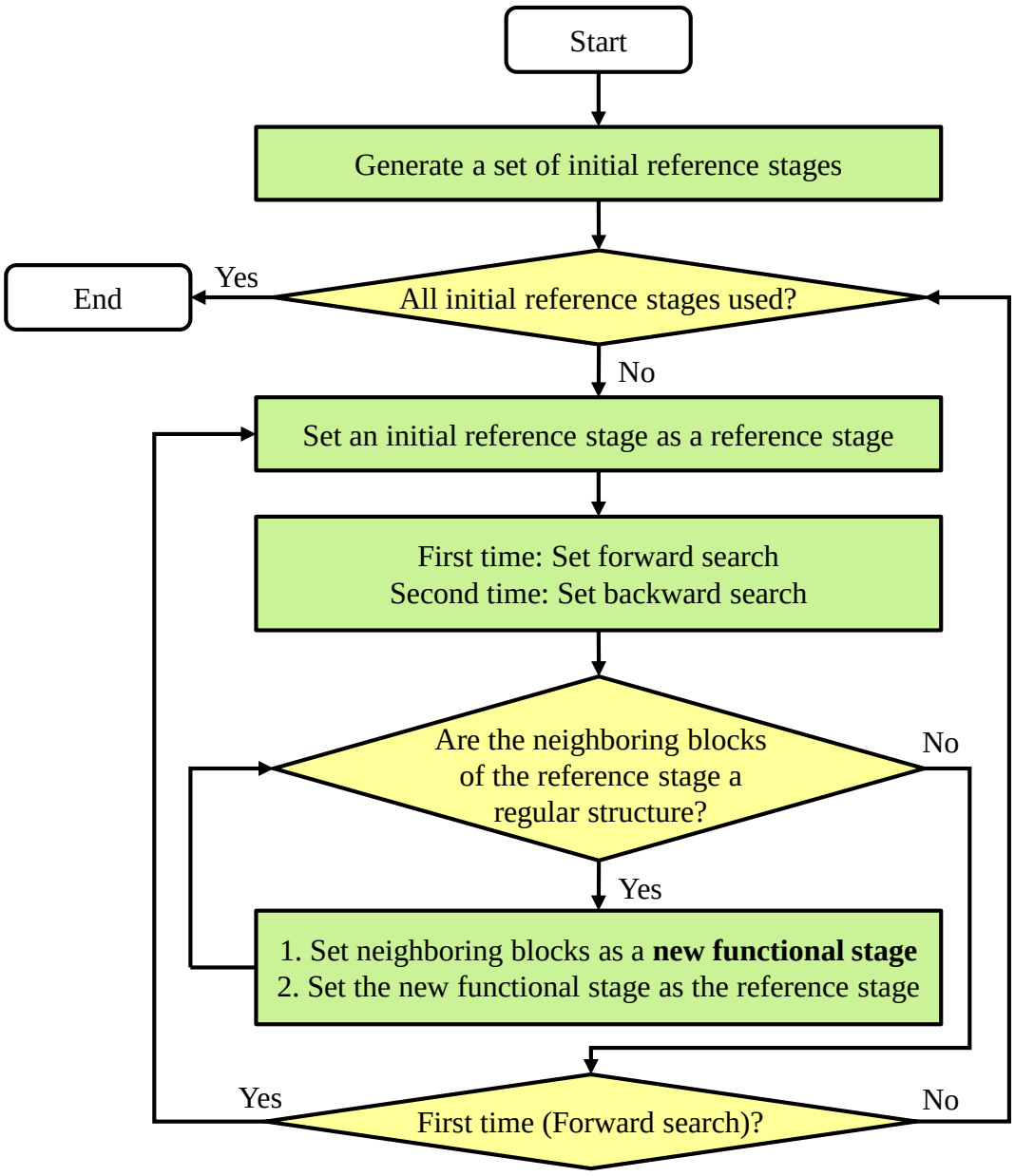


Figure 3.4: The flow of the proposed datapath extraction algorithm.

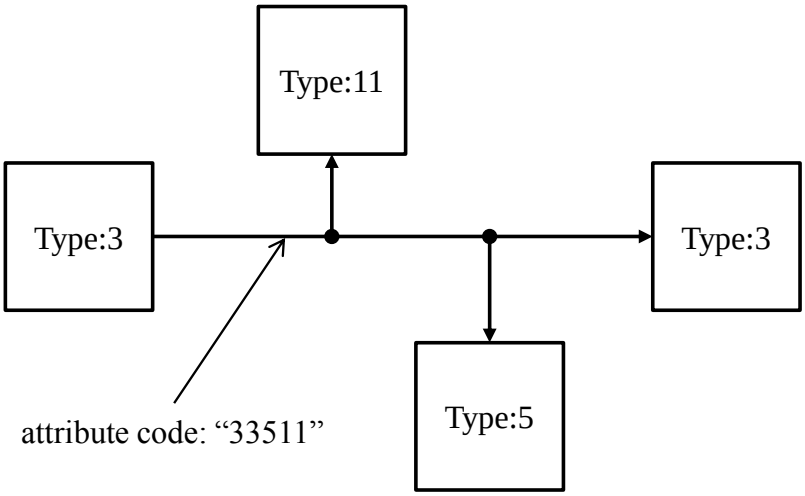


Figure 3.5: The attribution code of a net.

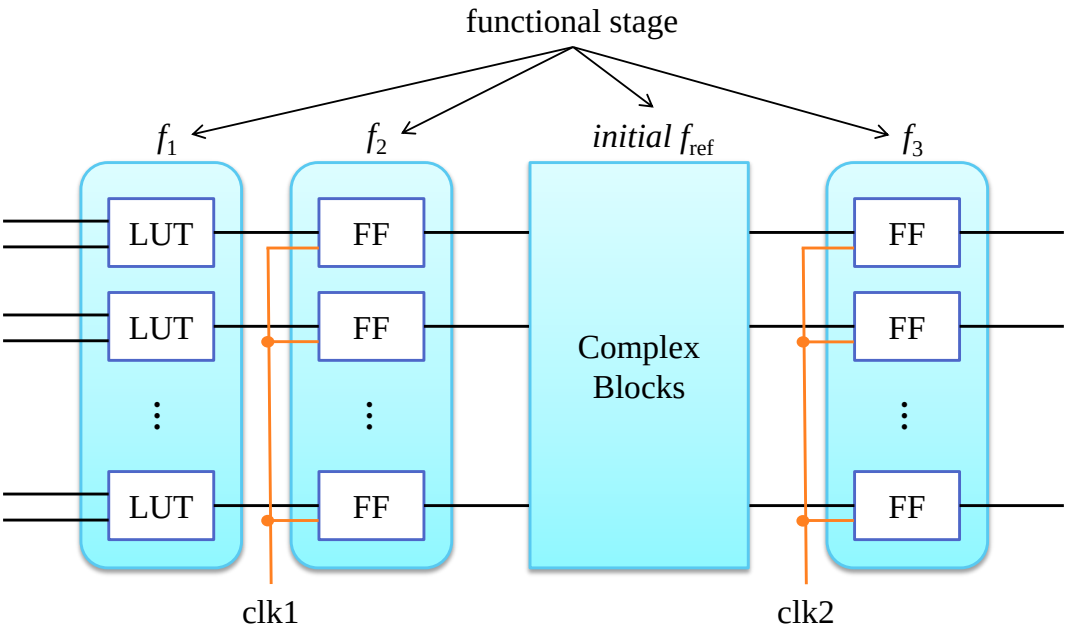


Figure 3.6: An example of a datapath extraction and packing.

longer attribute code represents a stricter regularity identification. The regularity identification is to identify nets with the identical attribute code. For each block in the reference stage, we generate the attribute codes for all its fan-in nets in the beginning of the forward search. Then, we group blocks into the same functional stage if the blocks drive nets with the identical attribute code. To relax the regularity identification, we can also pick nets with partially matched attribute codes. Once a new functional stage is identified, the functional stage is set to be the new reference stage, and we continue the forward search until there is no more new functional stage. The backward search performs the same operations (regularity identification and grouping functional stages) on all the output nets of the reference stage. Figure 3.6 shows an example of identified functional stages in a datapath after a datapath extraction and packing.

3.2.3 General Logic Packing

To reduce the problem size of the placement stage and provide a placement-friendly netlist, our general logic packing algorithm minimizes the number of blocks, block interconnections, pins on the block interconnections, and average inputs of the blocks such that the cluster size N, the maximum distinct inputs I, and the limitations of FF controlling signals specified in the given architecture are satisfied.

There are two major steps in our general logic packing algorithm: calculating affinity and determining the packing order. We will detail them in the following.

3.2.3.1 Affinity Calculation

The affinity of two clusters, S_i and S_j, is defined as follows:

$$A(S_i, S_j) = |N_{S_i} \cap N_{S_j}| + T(N_{S_i} \cap N_{S_j}), \tag{3.1}$$

where N_{S_i} denotes the net set of the cluster S_i and $T(N)$ denotes the number of two-pin nets in the given net set N. The first term is the common nets of S_i and S_j, and it also represents the amount of pin elimination. Meanwhile, the second term gives the common two-pin nets of S_i and S_j, and it also indicates the amount of interconnection elimination after packing S_i and S_j together. Every cluster S_i records the best affinity $A(S_i, S_j)$ (largest value) and the corresponding candidate S_j where S_j is connected with S_i and merging S_i and S_j will not violate the packing constraints. For example, a cluster S_0 is connected with clusters S_1, S_2, and S_3. After calculating the affinities $A(S_0, S_1) = 2$, $A(S_0, S_2) = 1$, and $A(S_0, S_3) = 4$, S_0 records its best affinity 4 and best candidate S_3.

3.2.3.2 Packing Order

Because the affinity-based approach gives great pin and interconnection elimination than the seed-based approach, we first apply best choice clustering for LUTs and FFs. The best choice clustering, however, fails to further cluster when the sizes of most clusters are larger than half capacity. Therefore, we present a hybrid algorithm that applies best choice clustering with a capacity threshold to trigger a successive packing procedure.

At the beginning of our general logic packing algorithm, the best affinity and the best candidate of each cluster are calculated and recorded. The best choice clustering algorithm chooses a cluster with the largest best affinity among all clusters, and packs the cluster with its best candidate. Then, the best affinities and best candidates of clusters connected to the packed cluster are updated. If the packed cluster exceed the half cluster size, a successive packing procedure is triggered to merge the packed cluster and its updated best candidate recursively until there is no more candidate. Otherwise, the best choice clustering algorithm repeats to select

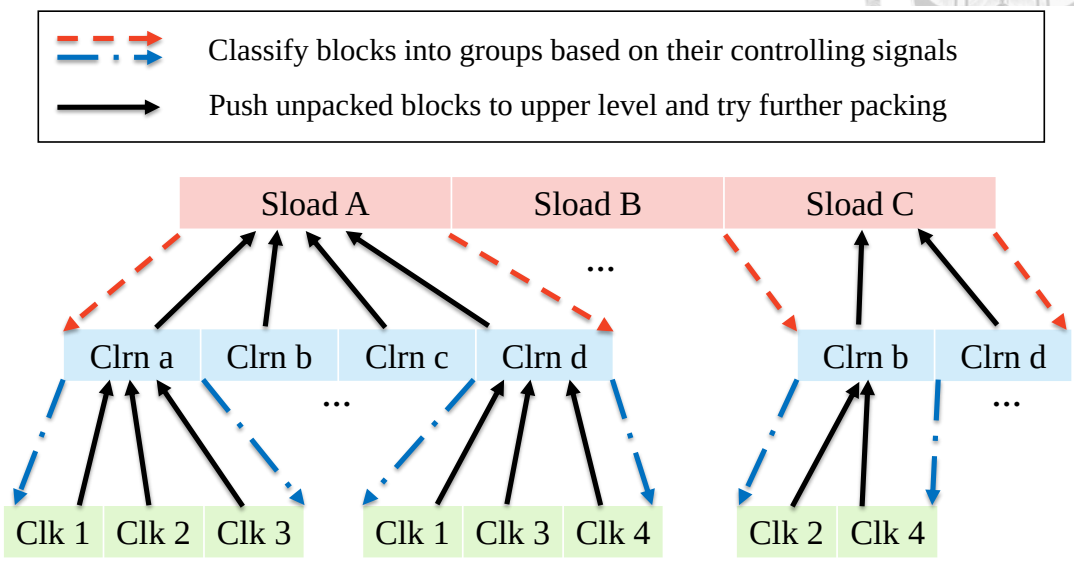


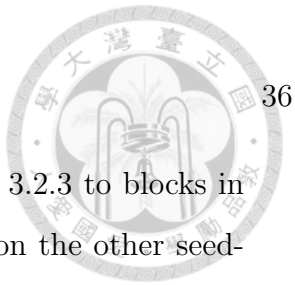
Figure 3.7: The flow of the proposed V-shaped framework.

a new cluster with the largest best affinity and then pack with its best candidate. The general logic packing is done while no clusters can be further packed together.

3.2.4 V-Shaped Framework

The runtime of the affinity and legality calculation mentioned in 3.2.3.1 has dominated the runtime of general logic packing. When considering miscellaneous FF controlling signal constraints in each legality check, the runtime of the affinity and legality calculation may significantly raise. Therefore, we propose a V-shaped framework to reduce the solution space of the affinity calculation and speed up the packing algorithm while considering the FF controlling signal constraints. This framework is applicable to various cost metrics in the seed-based and the affinity-based packing.

We describe the V-shaped framework in the following. First, sort the given controlling signal constraints by their critical levels. Then, classify all general logic blocks from the most critical constraint to the least critical constraint, level-by-level.



We then perform the general logic packing mentioned in Section 3.2.3 to blocks in the small groups in the bottom level. To apply this framework on the other seed-based or the affinity-based packing, we simply use them to replace the general logic packing. Blocks in a small group in the bottom level tend to be more related than blocks in other groups because the blocks share the common controlling signals. Those blocks which are not related or failed to be packed in the bottom level would be push to the upper level, and thus to have a second chance to be packed. Figure 3.7 gives an example. Given controlling signal constraints that FFs within a CLB could have (1) one distinct sload, (2) two distinct clrns, and (3) three distinct clks, all the blocks are first separated into groups based on their sload signals. Next, blocks with the same sload are divided into groups by their clrn signals, and then clk signals. Blocks in the same group are packed into clusters, and clusters which are under the cluster size N and the maximum input pins I would be pushed back to the upper level to seek for further packing. Notice that only one sload is tolerated with a CLB. As a result, there is no need to pack between groups in the top level.

3.3 Heterogeneous Analytical Placement

To cope with the increasing design complexity of heterogeneous FPGA circuits, we apply a heterogeneous analytical placement algorithm for better scalability. Our heterogeneous analytical placement consists of two stages: global placement and detailed placement. Our global placement algorithm exploits regularity of the datapaths and effective nonlinear optimization techniques to obtain a physically compact placement result in shorter runtime. Our global placement algorithm consists of four stages: (1) multilevel mixed-size prototyping with complex-block-alignment optimization which gives an approximate placement result with optimized wirelength, (2) look-ahead legalization which provides a quick forecast of the legalized placement

solutions, (3) datapath placement which fixes RAMs and DSPs and then places the datapath blocks regularly and compactly, and (4) general logic refinement which further optimizes the total wirelength. Our detailed placement algorithm refines the placement solutions in both CLB and BLE levels to compensate the insufficient packing and placement qualities.

3.3.1 Multilevel Mixed-Size Prototyping

At the beginning of our heterogeneous analytical placement algorithm, multilevel mixed-size prototyping efficiently gives an approximate placement result with optimized wirelength by solving the nonlinear optimization problems. We apply the multilevel analytical technique addressed in Section 2.2.

Because the legal locations of RAMs and DSPs are discrete and scattered on FPGAs, there exists a gap between the continuous solution of analytical placement and the legalized solution. As shown in Figure 3.8 (a), all blocks are evenly distributed while minimizing wirelength. In Figure 3.8 (b), the blocks are moved to the closest legal positions. It is obvious that a significant mismatch is generated during legalization. The objective to place the blocks in global placement is not accurate enough. Therefore, we propose a complex-block-alignment function to guide RAMs and DSPs in the multilevel mixed-size prototyping stage.

Our complex-block-alignment function $G(\mathbf{x})$ is proposed to minimize the gap between global placement and legalization. $G(\mathbf{x})$ is defined as follows:

$$G(\mathbf{x}) = \sum_{v_i \in V^R} \min_{x_j \in L^R} |x_i - x_j| + \sum_{v_i \in V^M} \min_{x_j \in L^M} |x_i - x_j|, \tag{3.2}$$

where x_i is the x coordinate of v_i and x_j is the x coordinate of the jth legal column from the left boundary. The rest of the notation is predefined in Section 2.1. The first summation is to sum up the cost of the RAMs v_i in V^R while the second

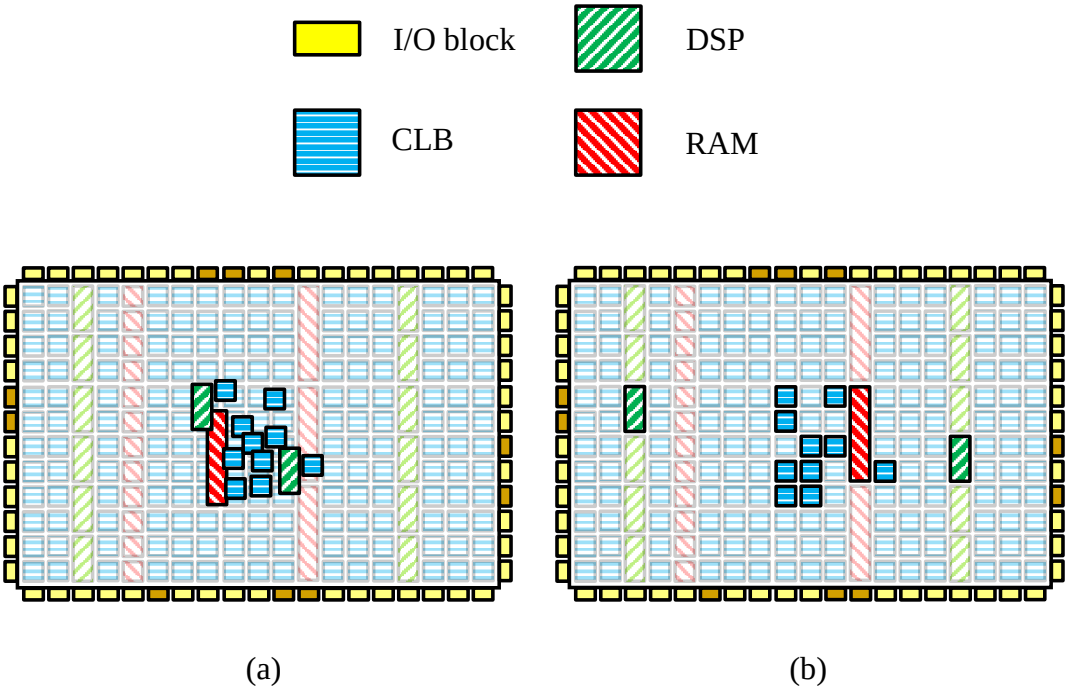


Figure 3.8: The mismatch between global placement and legalization. (a) The result of global placement. (b) The result after legalization.

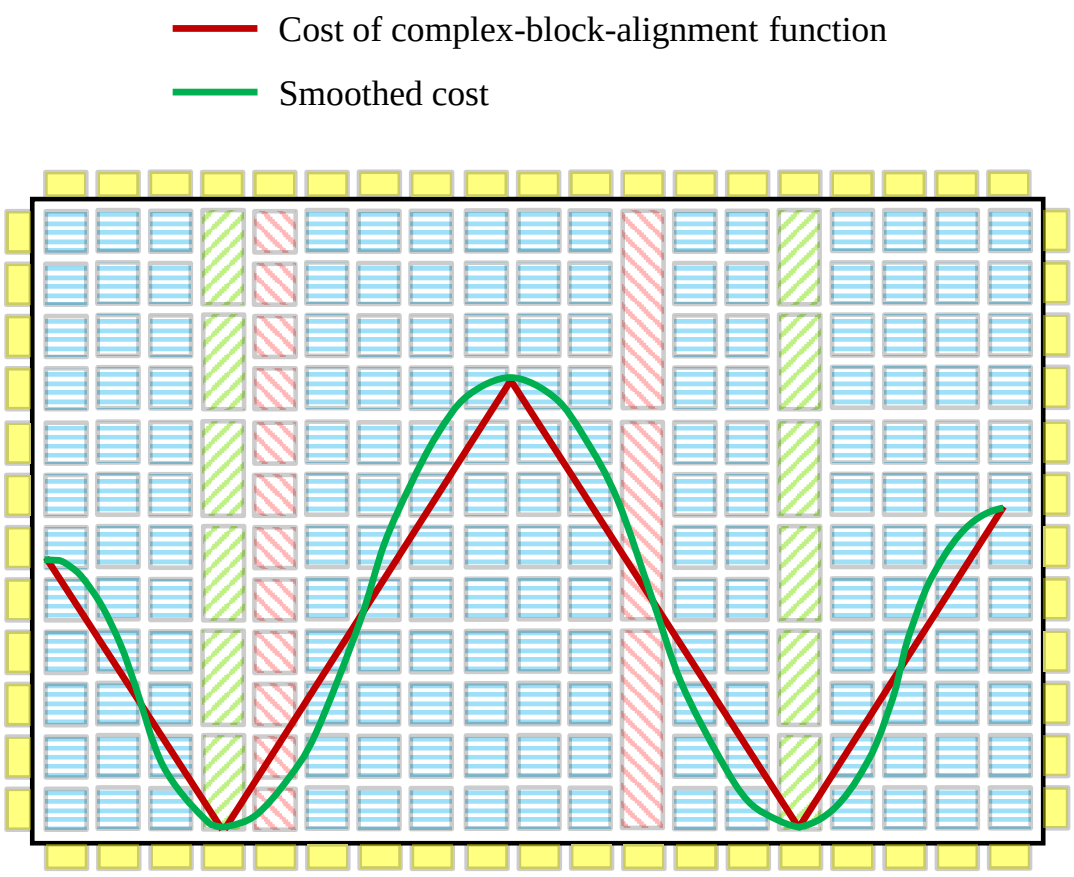


Figure 3.9: An illustration of our guiding function for DSPs.

summation is for the DSPs. The *min* term of the absolute value, which is the cost of a block, indicates the distance between a block v_i and its closest legal column. To minimize $G(\mathbf{x})$ is to minimize the distance between the best positions and the legal positions of the RAMs and DSPs. Figure 3.9 illustrates a curve of the cost function for the DSPs. The red, straight line represents the cost for the DSPs. The cost of a DSP block is minimized to 0 when the DSP block is placed on a legal position. On the contrary, if the DSP block moves away from the legal position, the cost increases. The cost decreases again when the DSP block approaches the next legal position.

Therefore, our problem here can be formulated as a constrained optimization problem as follows:

$$\begin{array}{ll} \min & W(\mathbf{x},\mathbf{y}) + \mu G(\mathbf{x}) \\ \text{s.t.} & D_b(\mathbf{x},\mathbf{y}) \leq M_b, \text{ for each bin b,} \end{array} \tag{3.3}$$

where μ is the weight for the complex-block-alignment function. Notice that $G(\mathbf{x})$ is neither smooth nor differentiable, and thus the relaxation of Equation (3.3) cannot be solved by the CG method. Therefore, we propose a smooth and differentiable complex-block-alignment function $\hat{G}(\mathbf{x})$ based on the quadratic sigmoid function [26] $p(t)$ as follows:

$$p(t) = \begin{cases} 1, & 0.5 \leq \alpha t \\ 1 - 2(\alpha t - 0.5)^2, & 0 \leq \alpha t \leq 0.5 \\ 2(\alpha t + 0.5)^2, & -0.5 \leq \alpha t \leq 0 \\ 0, & \alpha t \leq -0.5, \end{cases} \tag{3.4}$$

where α is the parameter for controlling the smoothness. Compared with the bell-shaped function, the sigmoid function gives a more accurate approximation to the original alignment function. Besides, by adjusting the parameter α, the smoothness can be easily controlled. In this thesis, α is set to 1.

$\hat{G}(\mathbf{x})$ is then formulated as follows:

$$\hat{G}(\mathbf{x}) = \sum_{v_i \in V^R} h_j \cdot p(\frac{d_j}{h_j} - 0.5) + \sum_{v_i \in V^M} h_j \cdot p(\frac{d_j}{h_j} - 0.5), \tag{3.5}$$

where

$$\begin{aligned} h_j &= \tfrac{x_{j+1}-x_j}{2}, \\ d_j &= \begin{cases} x_i - x_j, & x_i < \frac{x_{j+1}+x_j}{2} \\ x_{j+1} - x_i, & x_i \geq \frac{x_{j+1}+x_j}{2}, \end{cases} \end{aligned} \tag{3.6}$$

x_j and x_{j+1} are two successive elements in the ascending-ordered set L (L is either L^R or L^M), and $x_j \leq x_i < x_{j+1}$ is satisfied. The boundary conditions for x_0 and $x_{|L|+1}$ are set as the left and the right boundaries of placement region, respectively. $\hat{G}(\mathbf{x})$ is an approximation of $G(\mathbf{x})$. The smoothed cost function is represented by the green curve in Figure 3.9. Therefore, we can solve the Equation (3.3) with the smoothed complex-block-alignment function by transforming it into an unconstrained problem as follows:

$$\min \quad \hat{W}(\mathbf{x}, \mathbf{y}) + \mu \hat{G}(\mathbf{x}) + \lambda \sum_b \max(\hat{D}_b(\mathbf{x}, \mathbf{y}) - M_b, 0)^2. \tag{3.7}$$

By solving the Equation (3.7), we can obtain a placement prototyping with less block displacement between global placement and legalization.

3.3.2 Look-Ahead Legalization

Look-ahead legalization gives a quick forecast of legalized placement, which could achieve more accurate wirelength estimation and speed up the convergence with improved quality of placement solutions. In FPGA placement, blocks are mapped onto a two-dimensional array of grids. The prefabricated location of complex blocks such as RAMs and DSPs are distributed in a few columns or rows, and two legal columns/rows may have a great distance. Blocks after global placement, however, are not guaranteed to locate in such positions. This in turn may cause significant displacement and generate large differences on wirelength estimation. Thus,

we apply a fast legalization algorithm during each iteration in the finest level of the global placement stage.

To quickly find suitable positions for RAMs and DSPs, we propose a two-way relocation method for each dimension as our look-ahead legalization algorithm. In column-based FPGAs, blocks are first gathered to the nearest legal column. Next, we calculate the available capacity of each column. The available capacity of a column is defined as the difference between the total area of blocks on the column and the total area of the column. A negative available capacity of a column means that there are too many blocks on the column. If there exists negative available capacity, we then calculate the accumulation of the available capacity in two directions from the left most column to the right most column and from the right most to the left most. By doing so, we could know trends of loose regions. Then, we push the blocks in the overcrowded column to the loose column. After allocate all the complex blocks to each column, the x coordinate of the blocks is determined. Then, we apply the two-way relocation method again to each column and determine the y coordinate of the blocks.

We legalize the general logic blocks in FPGAs with a fast greedy method, called Tetris [24]. This approach sorts the blocks according to their x coordinates first, then legalizes each block at a time from left to right until all the blocks are placed. For a given block, the legalization is performed by scanning through the rows which are near the block and selecting the left-most vacant integer coordinates in the rows. For these vacant integer coordinates, the block is then legalized to the nearest integer coordinate from its original position. Alternatively, the general logic blocks could be legalized from right to left, up to down or down to up for flexibility.

3.3.3 Datapath Placement

Datapath placement is to place the datapath blocks regularly around complex blocks. After mixed-size prototyping and look-ahead legalization, blocks are placed at a legal position. The displacement during legalization, however, may destroy the regularity of the datapath blocks, as shown in Figure 3.10(a). To alleviate this effect, we place the datapath blocks stage-by-stage in the vertical direction and then bit-by-bit in the horizontal direction. By aligning blocks in the same functional stage in both the horizontal and vertical directions, we could further improve the wirelength and routability. Figure 3.10(b) shows a placement result after datapath placement. Besides, because blocks in some functional stages occupy only half of CLBs, we could further merge these blocks and get a more compact result and shorter wirelength, as shown in Figure 3.10(c). After all the positions of the complex blocks and datapath blocks are determined, we fix them to guide the remaining general logic blocks in the following stage.

3.3.4 General Logic Refinement

In the general logic refinement stage, all the complex blocks and datapath blocks are already fixed. These fixed blocks could become a guideline for the remaining movable general logic blocks to further improve placement solutions. Because the positions of the complex blocks are already determined, the smoothed complex-block-alignment function $\hat{G}(\mathbf{x})$ can be released from Equation (3.7). Therefore, the problem can be now transformed to Equation (2.2), which can also be solved by a CG solver. After solving this problem, we can refine the positions of the general logic blocks.

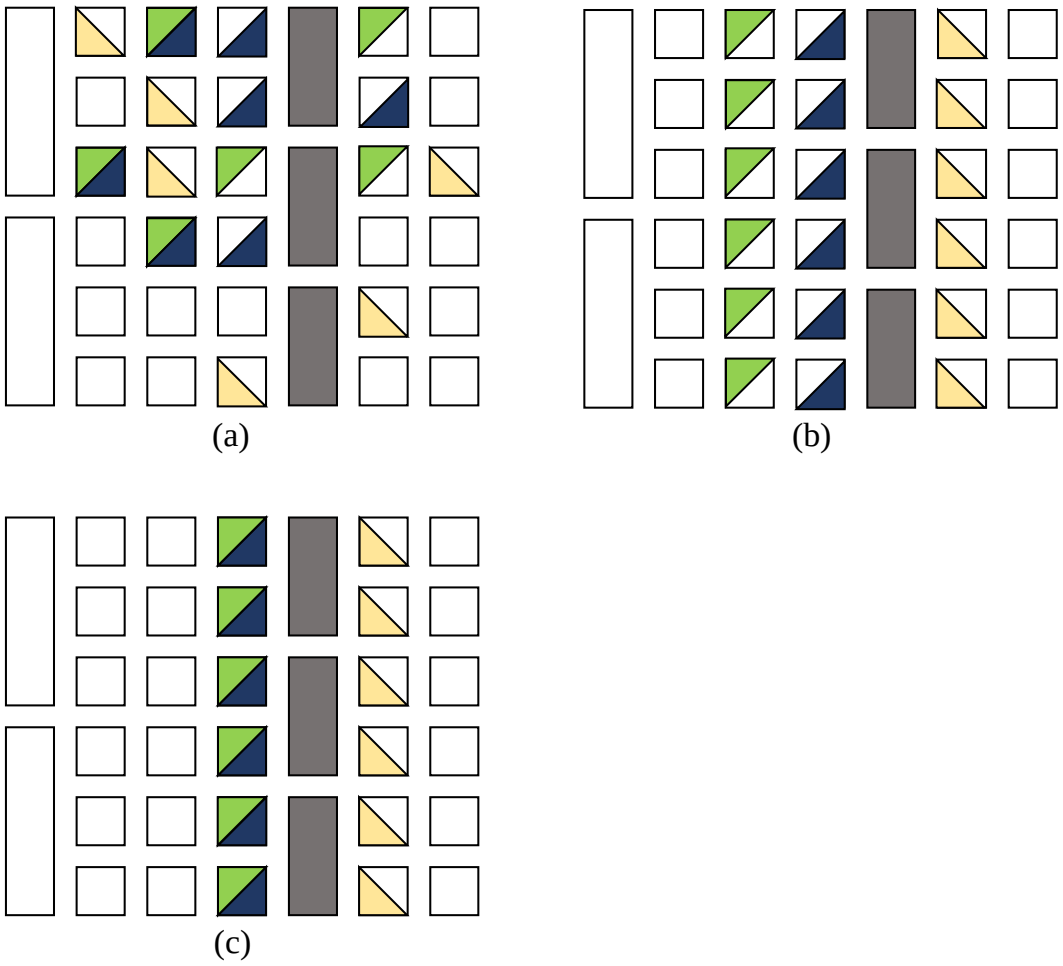


Figure 3.10: An example of conducting a more regular and compact datapath placement. (a) Placement without considering regularity. (b) Placement considering regularity. (c) Placement considering regularity and further refinement.

3.3.5 Detailed Placement

After global placement, detailed placement techniques are often used to achieve better placement solutions. Cell swapping and cell matching [13] are two of the most popular and effective detailed placement techniques. Both algorithms first set a user-defined region as a window and relocate cells inside the window. Cell matching models the cell relocation problem into a bipartite matching problem while cell swapping uses a branch-and-bound method to determine cell locations. We iteratively apply these two algorithms until the placement solutions can not be further improved.

Besides, some unrelated BLEs are clustered together to minimize the number of CLBs in the packing stage. This unrelated packing, on the contrary, may increase wirelength. Thus, in addition to relocate the CLBs using the above two algorithms, we also move BLEs between different CLBs in the detailed placement stage to make up the inferior clustering during the packing stage.

Chapter 4

Experimental Results

To evaluate the scalability of the proposed algorithms, we conducted experiments on the Titan23 benchmark suites. Table 4.1 shows the statistics of the benchmarks. We implement our algorithms in C++ programming language, and all the experiments were performed on a Linux workstation with Intel Xeon 2.93 GHz CPU and 48 GB memory. We compared the quality and runtime of our packing and placement results with the latest version of the VPR packer and placer (VPR 7.0) [41]. VPR is chosen because it has become a golden state-of-the-art benchmark for comparison on FPGA packing and placement, and is the only existing work that has run on modern large-scale circuits.

We chose to follow the architecture of the commercial device family, Altera's Stratix IV [1]. The device aspect ratio and average spacing between blocks was determined according to the EP4SE820 device. Because the VPR packer performs very poorly in the detailed architecture of Stratix IV, for fair comparisons, we use a simplified architecture of Stratix IV to fit the VPR packer. All the placers were executed on this simplified FPGA architecture and the same pre-specified locations of input/output blocks. We set VPR to bounding box mode.

Table 4.2 lists the packing results and runtime of the VPR packer and our packer. N/A states non-available. The normalized average does not take the non-

Table 4.1: Statistics of the Titan23 benchmarks.

Circuit	# Blocks	# Nets	# LUTs	# FFs	# Multipliers	RAM Bits
bitcoin_miner	1062373	1419988	455263	546597	0	297664
sparcT1_chip2	818362	825267	371910	430976	24	1585435
LU230	565120	636089	206675	293177	924	10112704
mes_noc	553021	583507	278300	248988	0	399872
gsm_switch	491473	511918	158878	296681	0	6254592
denoise	331337	407763	308992	9668	192	1134892
sparcT2_core	292615	295358	173297	110075	0	371416
cholesky_bdti	252425	305930	74030	173385	1030	4280448
stap_qrd	224960	267858	60447	161822	582	2548957
openCV	212207	258030	108042	86529	932	9412224
dart	201708	223411	103069	87386	0	955072
bitonic_mesh	188767	210367	107277	49570	676	1078272
segmentation	167001	201709	153487	7477	104	3166061
SLAM_spheric	127872	149334	108849	18342	296	0
des90	108791	125831	62070	30244	352	560640
cholesky_mc	105998	127399	27209	74051	456	4444096
stereo_vision	93617	124670	38206	51258	152	201544
sparcT1_core	92060	92673	42257	45272	8	337184
neuron	86619	105541	24413	59822	608	640442

available case into calculation. The proposed heterogeneous packing algorithm achieves a 199.80× speedup on average compared to VPR's. In the largest circuit of the benchmarks, the runtime of the VPR packer exceeded 48 hours while our packing ran within two minutes. Besides, although VPR achieves high packing density, packing density is not always directly proportional to the quality of packing [41]. In Table 4.3, we list the numbers of CLBs and the average input and output pins of a CLB after packing. Notice that although our packer have more CLBs than the VPR packer, our packer have much less average input and output pins. To see the effectiveness of our packer, we use both packing results for the same placer, NTUFPGA. Table 4.4 shows that our packer achieves 41% shorter wirelength, that is, our packer provides much better initial netlist to placers. This again shows that high packing density is not always directly proportional to the quality of packing. The ratio of the placement runtime for both netlist is proportional to the ratio of the total blocks.

We further examined the quality of our placer by using the same packing netlist generated by VPR 7.0. Because the VPR router is not scalable enough, routing failed on most of the large circuits in the benchmarks. Therefore, we report the total HPWL after placement for both placers in stead of routed wirelength. Table 4.5 shows that compared to the VPR placer, our placer achieves a 3.07× speedup with 6% shorter wirelength. Finally, the total HPWL and runtime of the overall flow (packing and placement) are listed in Table 4.6. The proposed heterogeneous packing and placement algorithms achieve a 18.30× speedup with 50% shorter wirelength compared to the VPR's packing and placement flow.

Figure 4.1, Figure 4.2, and Figure 4.3 show the results of the circuit *mes_noc* obtained after our global placement, legalization, and detailed placement. Figure 4.4 gives the placement result of the circuit *mes_noc* obtained from VPR. The blue

Table 4.2: Comparison of packing results.

Circuit	VPR 7.0				Ours			
	# Blocks	# Nets	# Pins	CPU (sec)	# Blocks	# Nets	# Pins	CPU (sec)
bitcoin_miner	N/A	N/A	N/A	N/A	78094	817375	2194305	78
sparcT1_chip2	34499	325186	1315257	23310	46806	308204	1505336	296
LU230	21386	350336	953360	29842	39260	409298	1062226	77
mes_noc	24958	293056	1103861	23655	27205	264415	1095071	98
gsm_switch	21368	196852	717126	14852	37480	277561	922996	26
denoise	17450	244144	852333	4062	20317	263417	905786	139
sparcT2_core	13683	159226	688997	5853	18898	143982	698072	69
cholesky_bdti	10222	12504	361976	10718	18509	150958	418946	57
stap_qrd	9419	102793	304859	4382	16019	127797	355781	67
openCV	8073	135591	382290	6497	16041	138396	418848	18
dart	7392	102275	340222	3430	11961	94152	324445	18
bitonic_mesh	8484	123708	479905	7078	15811	160458	496261	29
segmentation	8998	122257	437521	2411	9931	129541	451952	50
SLAM_spheric	6549	81305	312368	2097	8993	98516	337654	57
des90	4779	73205	268665	3302	6910	92961	279640	11
cholesky_mc	4736	54081	158144	1535	6856	62967	172042	14
stereo_vision	3418	56488	133869	925	6176	57366	143873	4
sparcT1_core	4168	44492	185744	1180	8381	46772	217494	9
neuron	3490	47008	119117	2675	7273	48822	127489	9
Average	0.65	0.92	0.92	199.80	1.00	1.00	1.00	1.00

N/A states that packing exceeded 48 hours run time.

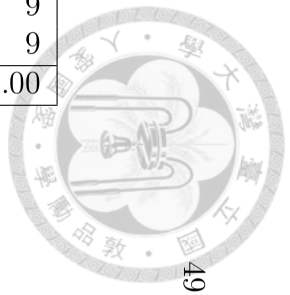


Table 4.3: Comparison of block number and average nets of a block after packing.

Circuits	VPR 7.0		Ours	
	# CLBs	Average Nets/CLB	# CLBs	Average Nets/CLB
bitcoin_miner	N/A	N/A	86758	24.29
sparcT1_chip2	32878	38.77	45370	12.89
LU230	15810	48.46	37057	23.12
mes_noc	24167	39.48	26785	21.04
gsm_switch	19482	25.88	35876	13.10
denoise	16060	54.55	22378	42.22
sparcT2_core	12954	53.93	18218	16.06
cholesky_bdti	9202	34.54	17915	19.73
stap_qrd	8562	32.23	15539	19.92
openCV	6587	51.72	15142	24.54
dart	6793	46.41	11482	18.34
bitonic_mesh	6516	56.36	14688	27.57
segmentation	7974	56.23	10683	43.91
SLAM_spheric	5868	55.52	8611	40.25
des90	3698	56.93	6533	35.20
cholesky_mc	3894	33.73	6263	22.68
stereo_vision	2840	41.43	5755	22.51
sparcT1_core	3714	45.86	7944	10.56
neuron	3072	34.01	7017	15.76
Average	0.59	2.10	1.00	1.00

N/A states that packing exceeded 48 hours run time.

Table 4.4: Comparison of packing quality using the same placer, NTUFPGA.

Packer	VPR Packer		Our Packer	
Placer	NTUFPGA			
	HPWL	CPU	HPWL	CPU
Average	1.41	0.66	1.00	1.00

Table 4.5: Comparison of placement quality using the same packer, VPR 7.0.

Packer	VPR 7.0			
Placer	VPR Placer		Our Placer	
	HPWL	CPU	HPWL	CPU
Average	1.06	3.07	1.00	1.00

Table 4.6: Comparison of packing and placement results.

Circuit	VPR 7.0		Ours	
	HPWL	runtime (sec)	HPWL	runtime (sec)
sparcT1_chip2	5730770	26525	5233292	5587
LU230	19018000	33064	10641276	1427
mes_noc	4784260	25957	3797526	954
gsm_switch	4386890	16322	4141500	1276
denoise	2324530	5345	2133100	520
sparcT2_core	2780370	7044	3493585	562
cholesky_bdti	3989500	11246	2313692	444
stap_qrd	2090230	5044	1845716	516
openCV	5198820	6959	1751970	244
dart	1747150	3899	1162140	187
bitonic_mesh	4640700	7654	2860898	270
segmentation	1308880	2879	1124166	205
SLAM_spheric	1736060	2372	1616844	183
des90	2350420	3503	1513372	115
cholesky_mc	1167250	1703	887830	107
stereo_vision	798480	1018	387684	58
sparcT1_core	635008	1310	728458	125
neuron	1359090	2776	476198	114
Average	1.50	18.30	1.00	1.00

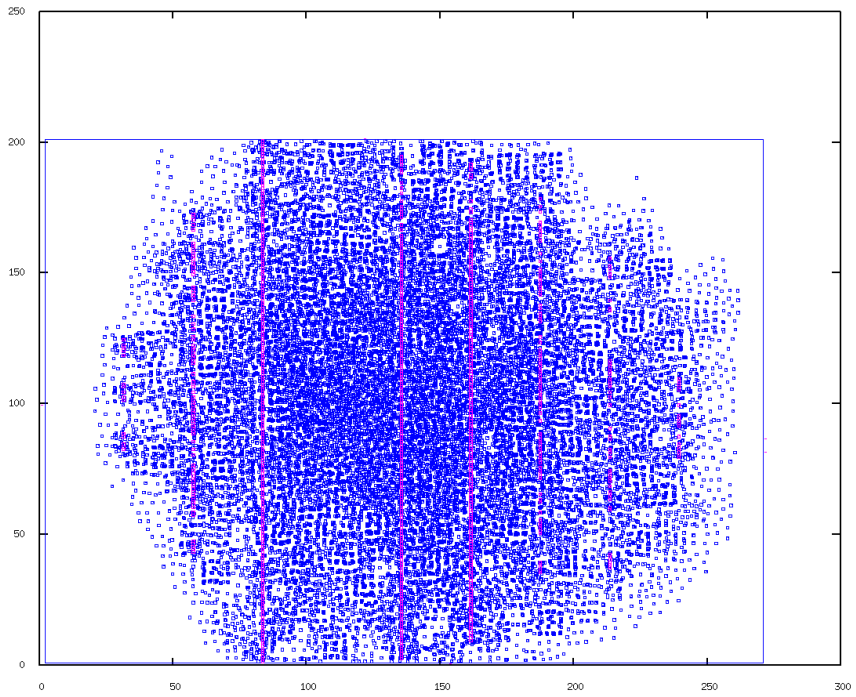


Figure 4.1: A result of our global placement.

blocks are CLBs and the pink blocks are RAMs and DSPs.

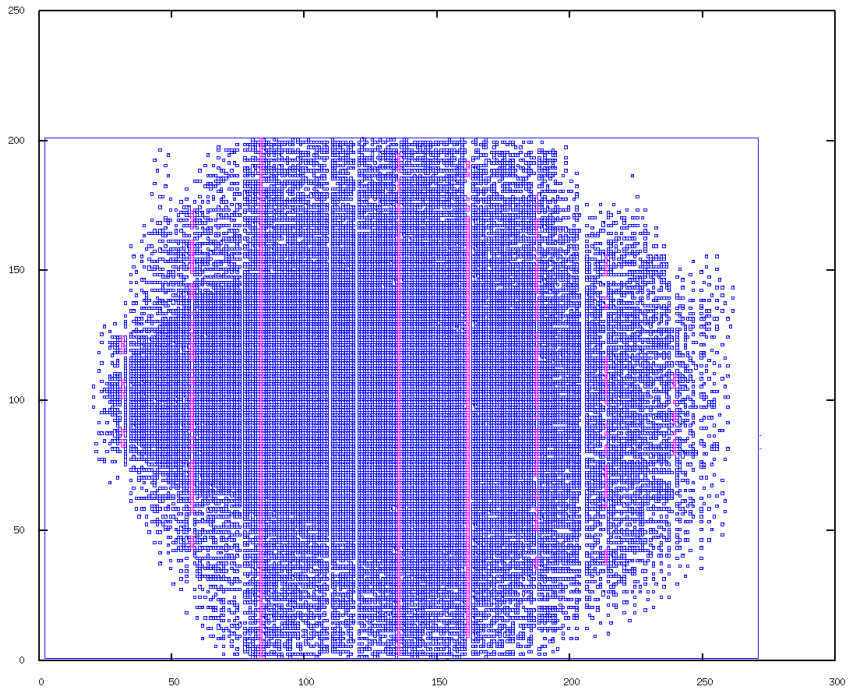


Figure 4.2: A result of our legalization.

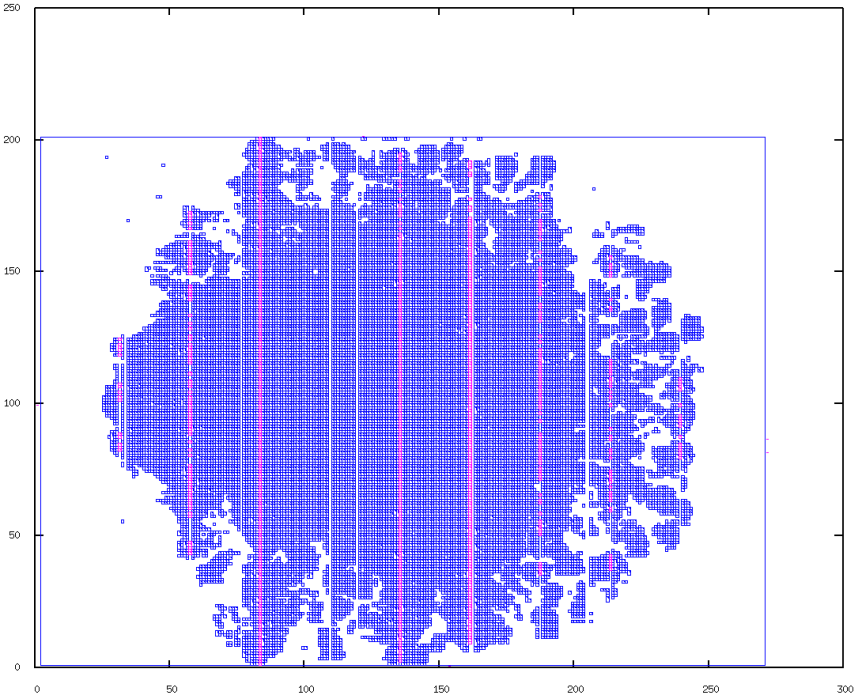


Figure 4.3: A result of our detailed placement.

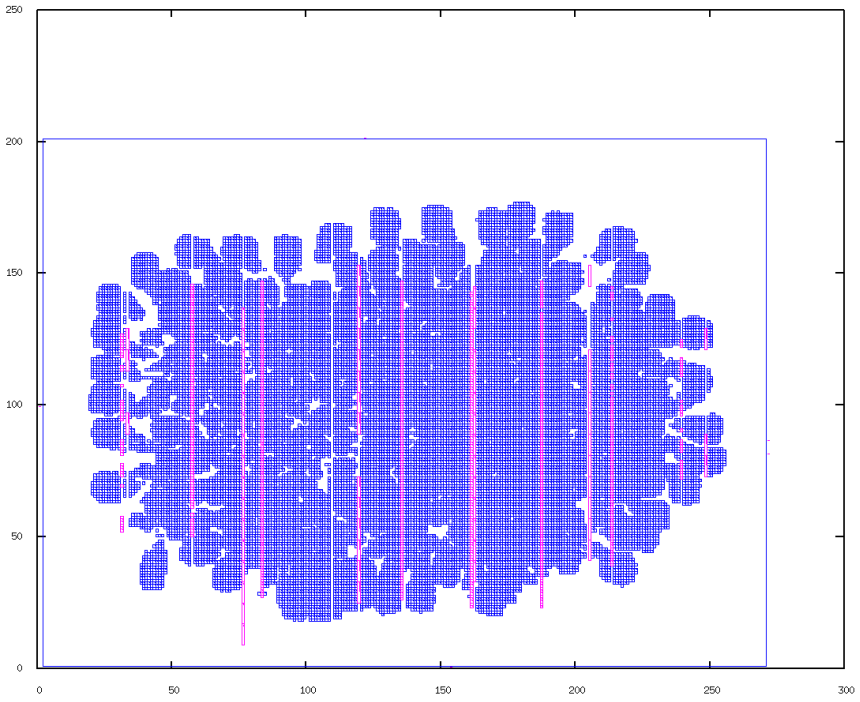


Figure 4.4: A result of VPR placement.

Chapter 5

Conclusions and Future Work

This paper has presented efficient and effective packing and analytical placement algorithms for large-scale heterogeneous FPGA design.

Our packing algorithm is composed of three stages to handle different structures of heterogeneous components and datapath blocks. Our packing provides a more placement-friendly netlist than VPR's, which gives a great potential for placers to achieve significant quality improvement. A V-shaped framework is proposed to enhance the scalability while considering more exact design constraints than existing works. Moreover, our wirelength-driven analytical placement algorithm applies effective nonlinear optimization techniques and utilizes the regularity of the datapaths to achieve scalability and high quality. A complex-block-alignment function is proposed to better handle the heterogeneity, and a multilevel framework is applied to further enhance the scalability of our placement algorithm.

We compared our packer and placer with the state-of-the-art FPGA CAD tool, VPR. The experiments were conducted on a set of modern large-scale FPGA benchmarks, Titan23 benchmark suites. The experimental results have shown that our approach achieves a 199.80× speedup, compared to the VPR's latest packing flow (VPR 7.0). Meanwhile, our placer achieves a 3.07× speedup with 6% shorter wirelength, compared to the VPR's latest placement flow. Furthermore, the overall

flow (packing and placement) achieves a 18.30× speedup with 6% shorter wirelength, compared to the VPR's latest packing and placement flow.

There are three future research directions: the analytical FPGA placement problem with *segmented-length routing architecture*, the analytical FPGA placement problem considering timing delay, and FPGA routing. We detail the future research directions below.

First of all, the pre-fabricated programmable routing architecture of modern FPGAs is segmented and there are several types of length. Hence, the way to predict routing and estimate routability in FPGA placement could be very different from that in ASIC placement. Furthermore, the current wirelength model might not be accurate enough for the routing estimation. As a result, how to utilize the various routing resources effectively and combine the estimation with an analytical placement framework have become a new challenge.

Secondly, in addition to wirelength and routability, timing is another critical issue in FPGA placement. Timing has been well explored in ASIC placement. The timing issue in FPGA design, however, still has lots more to work on because multiple routing resources could lead to different delays, which makes timing harder to predict.

Finally is the FPGA routing. Because the existing open-source FPGA router demonstrated poor scalability when facing large-scale circuits, FPGA routing is also a research direction on the scalability issue. Furthermore, understanding the FPGA routing could also help to build a quick routing forecast during placement.

Bibliography

[1] *Altera Corp.* `http://www.altera.com/`.

[2] *Xilinx Inc.* `http://www.xilinx.com/`.

[3] T. Ahmed, P. D. Kundarewich, J. H. Anderson, B. L. Taylor, and R. Aggarwal. Architecture-specific packing for virtex-5 FPGAs. In *Proceedings of Symposium on Field Programmable Gate Arrays*, pages 5–13, 2008.

[4] M. J. Alexander, J. P. Cohoon, J. L. Ganley, and G. Robins. Placement and routing for performance-oriented FPGA layout. *VLSI Design*, 7(1):97–110, Jan. 1998.

[5] C. Alpert, A. Kahng, G.-J. Nam, S. Reda, and P. Villarrubia. A semi-persistent clustering technique for VLSI circuit placement. In *Proceedings of ACM International Symposium on Physical Design*, pages 200–207, San Francisco, USA, Apr. 2005.

[6] V. Betz and J. Rose. Cluster-based logic blocks for FPGAs: area-efficiency vs. input sharing and size. In *Proceedings of Custom Integrated Circuits Conference*, pages 551–554, Santa Clara, USA, May. 1997.

[7] V. Betz and J. Rose. VPR: a new packing, placement and routing tool for FPGA research. In *Proceedings of Field-Programmable Logic and Applications*, pages 213–222, London, UK, Sep. 1997.

[8] V. Betz and J. Rose. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer, 1999.

[9] E. Bozorgzadeh, S. O. Memik, X. Yang, and M. Sarrafzadeh. RPack: routability-driven packing for cluster-based FPGAs. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 629–634, Yokohama, Japan, Feb. 2001.

[10] T. J. Callahan, P. Chong, A. DeHon, and T. Wawrzynek. Fast module mapping and placement for datapaths in FPGAs. In *Proceedings of Symposium on Field Programmable Gate Arrays*, pages 123–132, Monterey, USA, Feb. 1998.

[11] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie. mPL6: enhanced multilevel mixed-size placement. In *Proceedings of ACM International Symposium on Physical Design*, pages 212–214, San Jose, USA, Apr. 2006.

[12] D. T. Chen, K. Vorwerk, and A. Kennings. Improving timing-driven FPGA packing with physical information. In *Proceedings of Field-Programmable Logic and Applications*, pages 117–123, Amsterdam, Dutch, Aug. 2007.

[13] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. NTUplace3: an analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1228–1240, Jul. 2008.

[14] C.-L. E. Cheng. RISA: accurate and efficient placement routability modeling. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 690–695, San Jose, USA, Nov. 1994.

[15] S. Chou, M.-K. Hsu, and Y.-W. Chang. Structure-aware placement for datapath-intensive circuit designs. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 762–767, San Francisco, USA, Jun. 2012.

[16] J. Cong and M. Romesis. Performance-driven multi-level clustering with application to hierarchical FPGA mapping. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 389–394, Las Vegas, USA, Jun. 2001.

[17] M. E. Dehkordi and S. D. Brown. Performance-driven recursive multi-level clustering. In *Proceedings of IEEE International Conference on Field-Programmable Technology*, pages 262–269, Tokyo, Japan, Dec. 2003.

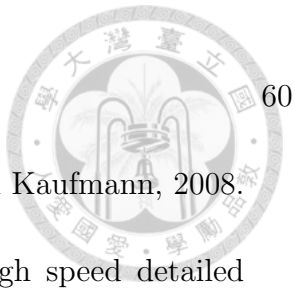
[18] W. Feng. K-way partitioning based packing for FPGA logic blocks without input bandwidth constraint. In *Proceedings of IEEE International Conference on Field-Programmable Technology*, pages 8–15, Seoul, South Korea, Dec. 2012.

[19] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, Feb. 1976.

[20] P. Gopalakrishnan, X. Li, and L. Pileggi. Architecture-aware FPGA placement using metric embedding. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 460–465, San Francisco, USA, Jul. 2006.

[21] M. Gort and J. H. Anderson. Analytical placement for heterogeneous FPGAs. In *Proceedings of Field-Programmable Logic and Applications*, pages 143–150, Oslo, Norway, Aug. 2012.

[22] H. Hassan, M. Anis, and M. Elmasry. LAP: a logic activity packing methodology for leakage power-tolerant FPGAs. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 257–262, San Diego, USA, Aug. 2005.



[23] S. Hauck and A. Dehon. *Reconfigurable Computing*. Morgan Kaufmann, 2008.

[24] D. Hill. US patent 6,370,673: Method and system for high speed detailed placement of cells within an integrated circuit design. 2002.

[25] M.-K. Hsu, Y.-W. Chang, and V. Balabanov. TSV-aware analytical placement for 3D IC designs. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 664–669, San Diego, USA, Jun. 2011.

[26] M.-K. Hsu, S. Chou, T.-H. Lin, and Y.-W. Chang. Routability-driven analytical placement for mixed-size circuit designs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 80–84, San Jose, USA, Nov. 2010.

[27] E. Hung, F. Eslami, and S. J. Wilton. Escaping the academic sandbox: realizing VPR circuits on xilinx devices. In *Proceedings of Field-Programmable Custom Computing Machines*, pages 45–52, Seattle, USA, Apr. 2013.

[28] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: applications in VLSI domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(1):69–79, Mar. 1999.

[29] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 343–348, New Orleans, USA, Jun. 1999.

[30] M.-C. Kim, D.-J. Lee, and I. L. Markov. SimPL: an effective placement algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(1):50–60, Jan. 2012.

[31] M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Transac-*

tions on Computer-Aided Design of Integrated Circuits and Systems, 10(3):356–365, Mar. 1991.

[32] T.-H. Lin, P. Banerjee, and Y.-W. Chang. An efficient and effective analytical placer for FPGAs. In *Proceedings of ACM/IEEE Design Automation Conference*, page 10, Austin, USA, Jun. 2013.

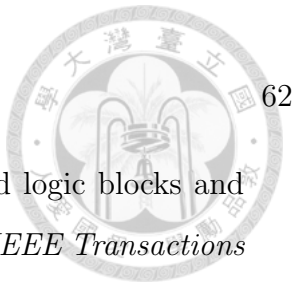
[33] J. Luu, J. H. Anderson, and J. S. Rose. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. In *Proceedings of Symposium on Field Programmable Gate Arrays*, pages 227–236, Monterey, USA, Feb. 2011.

[34] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, K. Kent, and J. Rose. VPR 5.0: FPGA cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling. *ACM Transactions on Reconfigurable Technology and Systems*, 4(4):32, Dec. 2011.

[35] P. Maidee, C. Ababei, and K. Bazargan. Timing-driven partitioning-based placement for island style FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3):395–406, Mar. 2005.

[36] V. Manohararajah, G. R. Chiu, D. P. Singh, and S. D. Brown. Difficulty of predicting interconnect delay in a timing driven FPGA CAD flow. In *Proceedings of International Workshop on System Level Interconnect Prediction*, pages 3–8, Munich, Germany, Mar. 2006.

[37] A. Marquardt, V. Betz, and J. Rose. Timing-driven placement for FPGAs. In *Proceedings of Symposium on Field Programmable Gate Arrays*, pages 203–213, Monterey, USA, Feb. 2000.



[38] A. S. Marquardt, V. Betz, and J. Rose. Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(3):288–298, Apr. 1999.

[39] Z. Marrakchi, H. Mrabet, and H. Mehrez. Hierarchical FPGA clustering based on multilevel partitioning approach to improve routability and reduce power dissipation. In *Proceedings of International Conference on Reconfigurable Computing and FPGAs*, pages 4–25, Puebla, Mexico, Sep. 2005.

[40] F. Mo, A. Tabbara, and R. K. Brayton. A force-directed macro-cell placer. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 177–181, San Jose, USA, Nov. 2000.

[41] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz. Titan: enabling large and complex benchmarks in academic CAD. In *Proceedings of Field-Programmable Logic and Applications*, pages 1–8, Porto, Portugal, Sep. 2013.

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

[43] A. Pandit, L. Easwaran, and A. Akoglu. Concurrent timing based and routability driven depopulation technique for FPGA packing. In *Proceedings of IEEE International Conference on Field-Programmable Technology*, pages 325–328, Taipei, Taiwan, Dec. 2008.

[44] S. T. Rajavel and A. Akoglu. Mo-pack: many-objective clustering for FPGA CAD. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 818–823, San Diego, USA, Jun. 2011.

[45] G. Sigl, K. Doll, and F. M. Johannes. Analytical placement: A linear or a quadratic objective function? In *Proceedings of ACM/IEEE Design Automation Conference*, pages 427–432, San Francisco, USA, Jun. 1991.

[46] A. Singh, G. Parthasarathy, and M. Marek-Sadowska. Efficient circuit clustering for area and power reduction in FPGAs. *ACM Transactions on Design Automation of Electronic Systems*, 7(4):643–663, Feb. 2002.

[47] C. N. Sze, T.-C. Wang, and L.-C. Wang. Multilevel circuit clustering for delay minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(7):1073–1085, Jun. 2004.

[48] D. Xie, J. Xu, and J. Lai. A new FPGA placement algorithm for heterogeneous resources. In *Proceedings of ACIS International Conference on Computer and Information Science*, pages 742–746, Changsha, China, Oct. 2009.

[49] M. Xu, G. Gréwal, and S. Areibi. StarPlace: A new analytic method for FPGA placement. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 44(3):192–204, Feb. 2011.

[50] Y. Xu and M. Khalid. QPF: efficient quadratic placement for FPGAs. In *Proceedings of Field-Programmable Logic and Applications*, pages 555–558, Tampere, Finland, Aug. 2005.

[51] M. Yang, H. Xu, and A. Almaini. Power-aware FPGA packing algorithm. In *Proceedings of ACIS International Conference on Computer and Information Science*, pages 817–819, Changsha, China, Oct. 2009.

[52] A. G. Ye and J. Rose. Using multi-bit logic blocks and automated packing to improve field-programmable gate array density for implementing datapath cir-

cuits. In *Proceedings of IEEE International Conference on Field-Programmable Technology*, pages 129–136, Brisbane, Australia, Dec. 2004.