

Wirelength-driven Analytical Placement for FPGA

A THESIS

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA

BY

Satya Prakash Upadhyay

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

Prof. Kiarash Bazargan

August, 2015

© Satya Prakash Upadhyay 2015
ALL RIGHTS RESERVED

Acknowledgements

First of all I would like to express my sincere thanks to my advisor Prof. Kia Bazargan for his immense support and motivation during the work of thesis. He helped me gaining knowledge and deep insight of the topic through the courses I have taken under him. I appreciate his constant guidance and feedback, which helped me keep my interest in the research topic intact throughout my research. I am also grateful to other committee members, Prof. Marc Riedel and Prof. Antonia Zhai, who have given their valuable time for reviewing my work and precious advice wherever needed.

I appreciate the support of developers and support team of VTR tool, which was the platform for my research work. Special thanks to Jason Luu for his selfless guidance and assistance whenever I needed.

I am thankful to my friend Darshak Gandhi, with whom I started this work as a course project in spring 2014 and further extended it as my research work. We discussed many great ideas together before implementing this. My sincere thanks to my fellow lab mate Nimish Agashiwala for his support, valuable ideas and humors.

I express sincere gratitude to my parents Kailash Updhyay and Meena Updhyay, my sisters Chanchal and Nidhi for encouraging me to pursue higher studies and gave moral support throughout my graduate studies.

Last but not the least, I would like to thank all members of ECE department of our University who have assisted me in many ways to accomplish this research successfully.

Dedication

I dedicate this work to my source of inspiration - my grandfather Late Shri Uma Shankar Upadhyay and my entire family.

Abstract

With increasing complexity of modern circuit, FPGA demands to be dealt with good CAD algorithm and suitable architecture to meet the lower non-recurring engineering cost and faster time-to-market. Placement is one of the crucial steps among them, as it decides time for implementing FPGA, routing resources and power consumption by digital circuits. Our research is centered on the placement algorithm for FPGA design. Simulated Annealing (SA) being the most popular among all the placement methods for quality results, takes huge compile time to implement larger circuits with the current new architectures. Researchers try to find a way for getting similar or better results with less run time. Based on the requirement, placement can be wirelength driven, timing driven and path driven. There are alternate ways of placement which are based on min-cut algorithm and analytic placement, and take less time.

We targeted to optimize wirelength while doing placement using Gordian method [25] in multiple iteration to get similar results as that of well-known academic research tool for FPGA – Versatile Place and route (VPR). Each iteration divides a subspace in four partitions and applies linear and bounding constraint to solve for quadratic optimization. We bypassed the placement methodology of VPR with our placement algorithm of analytical placement, implemented in MATLAB, and then fed back the output of our placer to the VPR flow for detailed placement and routing. We compare our results of placement and routing using 20 MCNC benchmarks and homogeneous VTR benchmarks with the VPR flow. Our MATLAB placer is faster by 38% with the expense of wirelength quality. It gets 1% better wirelength with 11% increase in runtime compared to the whole VPR placer after low temperature simulated annealing based final detailed placement .

Contents

Acknowledgements	i
Dedication	ii
Abstract	iii
List of Tables	vii
List of Figures	viii
1 Introduction	1
2 Background and Motivation	5
2.1 FPGA Architecture	5
2.1.1 Programming Technology	7
2.1.2 Configurable Logic Block	8
2.2 FPGA CAD flow	9
2.2.1 Logic Synthesis	10
2.2.2 Technology Map	10
2.2.3 Clustering	11
2.2.4 Placement	11
2.2.5 Routing	11
2.3 Motivation	12
2.4 Previous Work on FPGA placement	12
2.4.1 Simulated Annealing based placement	12

2.4.2	Partition-based placement	15
2.4.3	Analytical Placement	15
2.5	Summary	16
3	Analytical Placer Algorithm	18
3.1	Preliminaries	18
3.1.1	Problem Formation	18
3.1.2	Analytical Placement Steps	19
3.1.3	GORDIAN Placement	20
3.1.4	Quadprog	21
3.2	VTR-VPR	22
3.3	Analytic Placer Setup - Complete experimental flows	23
3.4	MATLAB Engine Flow	24
3.5	Connection Matrix Generation	27
3.6	Decision of number of iteration	30
3.7	Legalization	31
3.8	Low temp. Simulated annealing	33
3.9	Summary	34
4	Experimental Setup and Result Analysis	35
4.1	Architecture Selection	35
4.2	Benchmark Selection	36
4.3	Result and Analysis	38
4.3.1	Environment	38
4.3.2	Results	38
4.3.3	Analysis	43
5	Conclusion and Discussion	44
5.1	Conclusion	44
5.2	Future Work	45
	References	47

Appendix A. Acronyms	50
A.1 Acronyms	50

List of Tables

2.1	Temperature Update Schedule [3]	14
4.1	Major Architecture Files in VTR 7.0 Release [28]	36
4.2	Statistics of Benchmark Circuits for k6_N10_40nm Architecture	37
4.3	Bounding Box Wirelength and Runtime comparison after placement for VPR and MATLAB placer	39
4.4	Bounding Box Wirelength and Runtime comparison after placement for VPR and Analytical Placer with low temp. Simulated Annealing	40
4.5	Comparison: Routed Channel Factor and total wirelength after routing of VPR and Analytical Placer	42
A.1	Acronyms	50

List of Figures

1.1	FPGA Design Flow	2
2.1	Island-style Architecture[10]	6
2.2	Heterogeneous FPGA Architecture[11]	7
2.3	Classical CLB[6]	8
2.4	FPGA CAD flow[10]	9
2.5	Technology Mapping[10]	10
3.1	GORDIAN: Spreading overlapped CLBs with center of mass constraint [26]	21
3.2	VTR and our placer Flow	23
3.3	Illustration of CLB connection to form Connection Matrix	29
3.4	Illustration of no. of iteration for design with a) 15 CLBs and b) 16CLBs	30
3.5	Spiral Legalization	32
4.1	Bounding Box Wirelength comparison	41
4.2	Runtime comparison of VPR and Analytical Placer	41

Chapter 1

Introduction

With Moore's law completing 50 years [1], the electronics industry has witnessed scaling down the transistor size and chip becoming denser. It reduces the chip area and hence the cost. Depending on the performance, cost and time-to-market, designers implement their design on either of Application Specific Integrated Circuit (ASIC) or Programmable Logic Device (PLDs). But when it comes to re-usability of chip, PLDs are the first choice due to its advantage of on-field programmability. Among the types of PLDs- Complex Programmable Logic devices (CPLD) and Field Programmable Gate Array (FPGA), FPGA is widely used as one can implement complex and large digital circuit easily. Its on field programmability gives more flexibility. Once a functional chip is available in the market it does not cost anything for implementing any logical design on it. As soon as we get the mapped bit stream of the design into a specific FPGA chip, it starts functioning. That gives FPGA significant advantage over the ASIC design.

Like ASIC design, FPGA design cycle also follows same kind of cycle as shown, but it has different synthesis step as shown in Figure 1.1. Each step goes through a CAD tool. At first the design is written in high level descriptive language (in Verilog, VHDL etc.). First step of synthesis converts the circuit description into the netlist of basic gates. Then it does technology independent logic optimization followed by technology mapping on look-up tables. In third stage it tries to group optimized logic elements. It aims to pack those LUTs and registers together which are connected so that it reduces the cost of routing in routing stage. Finally it gives the netlist of logic blocks. This netlist is used as input for the placement tool. Placement tool tries to put those blocks

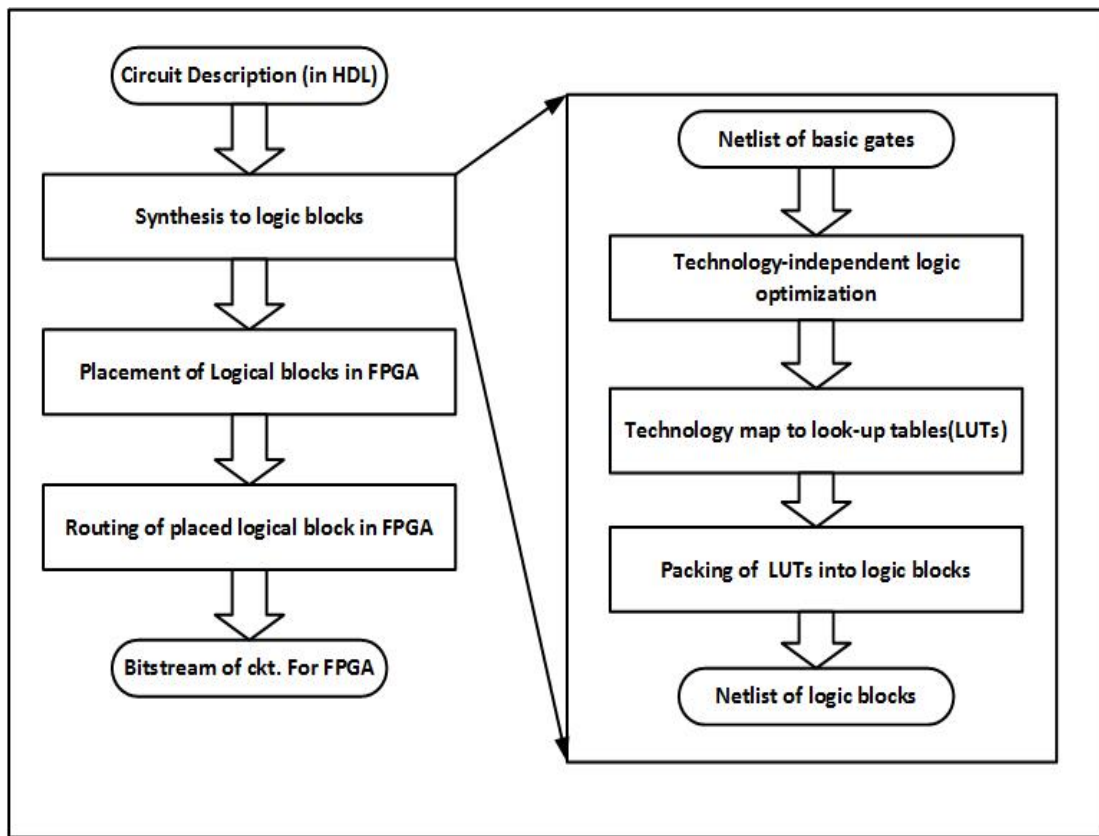


Figure 1.1: FPGA Design Flow

on physical locations in such a way that it consumes less area, less routing resources and meet timing criticality. Next, routing is performed, which enables the switches to connect those logical blocks.

For an FPGA implementation, two key elements are important – Architecture and CAD flow. To realize complex digital circuit into an FPGA, different architectures have been suggested. More on FPGA architecture will be discussed in chapter 2.1. On the other side researchers try to find better algorithms and techniques to make that kind of architecture feasible with the FPGA design and the corresponding CAD flow. Packing, placement and routing are three important pillars of the FPGA CAD flow. Among them placement plays valuable role. It optimizes many factors considering the given routing resources and area. The high number of logic blocks, mixed size block and

area constraints makes FPGA placement more focused. We have to meet all constraints to place them unlike ASIC where we do not bother much about placing a block at a certain fixed place in the specified area. Its optimization goal could be timing driven, which tries to minimize the delay, or wirelength-driven which minimizes wirelength, or it could be path-driven which focuses on putting the blocks on the critical path to optimize wirelength and minimize delay under the given routing resources and area. Our thesis work is focused on wirelength driven placement. We tried to minimize the wirelength of the design during placement.

There are mainly three placement methodologies used for physical design. Simulated annealing (SA) - where we try to reduce the cost function by swapping of the logical blocks. It mimics the annealing process used to make high quality metal objects from gradually molten metals [4]. Another approach which is quadratic placement, where squared wirelength is minimized as cost function [8]. The min-cut placement method of FPGA placement is based on partition based optimization [9]. Our research work follows quadratic method of placement. For implementing our idea we took Verilog to Routing VTR [2] tool as our platform. It is widely used open source tool for academic research. It is a complete package for generating bit stream from a circuit description of the design. We focused only on the placer part of the tool – Versatile Placement and routing (VPR). VPR is combined CAD tool for packing, placement and routing [3]. We aimed to bypass the SA based placement method used in VPR with our placer to get effective and optimized result in less time. First we provide netlist file and architecture file to VPR and get the connection matrix from VPR and give this as input to our placer. Our placer gives optimized wirelength for placed logical block, which is legalized later. Then we run low temperature simulated annealing for detailed placement and further optimization. Lastly we analyze our placer using 20 MCNC FPGA benchmarks and VTR’s homogeneous benchmark. Currently our work focuses on homogeneous architecture only. Rest of the thesis is organized as follows:

- Chapter 2 briefly describes the architecture of FPGA, FPGA CAD flow and different placement algorithm for FPGA placement. It also highlights our motivation towards the research and previous work in that field.
- In Chapter 3 the flow is outlined. It describes each of the steps of the flow in

complete detail.

- Chapter 4 hashes out selection criteria of benchmarks and architectures for our research. It contains analysis criteria and the outcomes of our experiments.
- Chapter 5 concludes our experimental results and shows future enhancements.

Chapter 2

Background and Motivation

This chapter covers the background material related to FPGA and the placement methodology of FPGA design. We begin with describing FPGA architecture. Section 2.2 will include general flow of FPGA design. Then we will define the FPGA placement methodology. We have discussed previous work on the different FPGA placement techniques in section 2.4.

2.1 FPGA Architecture

FPGA differs from the ASIC design in re-programmability. It can be re-configured by the end user based on the demanding new circuits. Hence, the name is Field Programmable Gate Array. There exist many FPGA architectures. But all of them has three primary elements- Logic Blocks, Input/Output (IO) blocks and programmable routing channels. The set of logic blocks are used to map the circuit design. IO blocks communicate with the outer world and programmable routing structure is used to connect the logic blocks to implement the logic design over FPGA chip. Now-a-days FPGA has evolved and contains complex blocks, which are collection of small components called primitive. These complex blocks are configured to map the circuit on FPGA. We will use name CLB for them here after.

Due to complex design and concern of consumed time for FGPA design cycle evolved different architecture. FPGA architecture can be categorized as Island or mesh type architecture and Hierarchical FPGA (HPGA) architecture [10]. Different industries use

these kind of architecture based on the requirement and complexity of the design. Our research is based on most widely used architecture- Island based architecture.

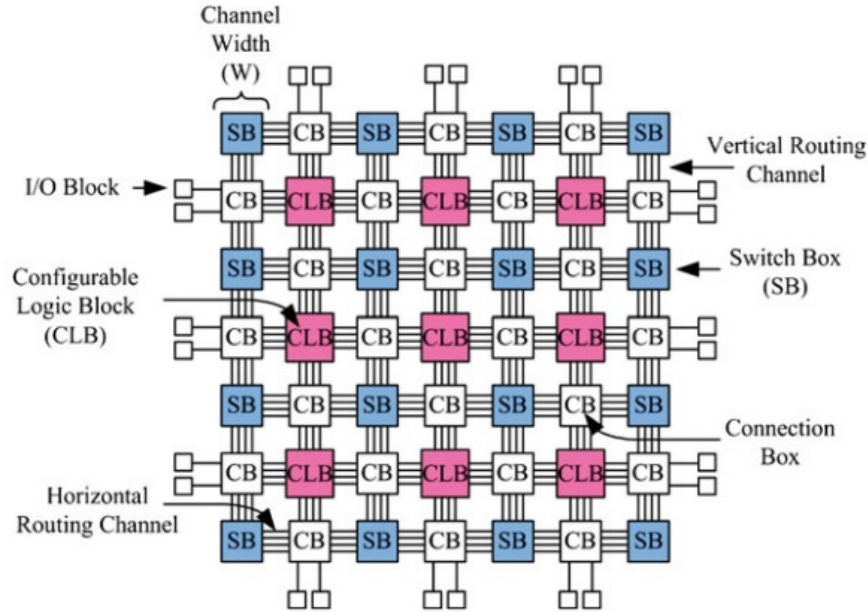


Figure 2.1: Island-style Architecture[10]

Fig 2.1 shows traditional Island-style FPGA architecture. In this architecture the configurable Logic Blocks look like an island in the sea of routing channels. We have IO blocks on the periphery to communicate with the outer world. There are pre-fabricated vertical and horizontal routing wired channel over the entire FPGA chip which is used to transmit data between functionally connected CLBs. These routing tracks are connected with the help of Switch Box (SB) at the crossing point. These SB could be bi-directional or uni-directional. Connection Blocks helps connecting the Input/Output pins of the CLBs to the routing tracks. Based on the type of complex blocks, Island- type architecture can be further categorized as homogeneous architecture and heterogeneous architecture. In homogeneous architecture, we have general-purpose complex block (GPCB) which are flexible enough for implementing any digital logic. While in heterogeneous architecture, we have special-purpose complex blocks (SPCB) (memories, multipliers etc.) along with GPCB, as shown in Figure 2.2. It eases to

implement the circuit with more complexity. Modern digital age demands heterogeneous architecture. But it should be wisely used. If the SPCB is not used for the particular design it will just have an area overhead and can use long routing resources.

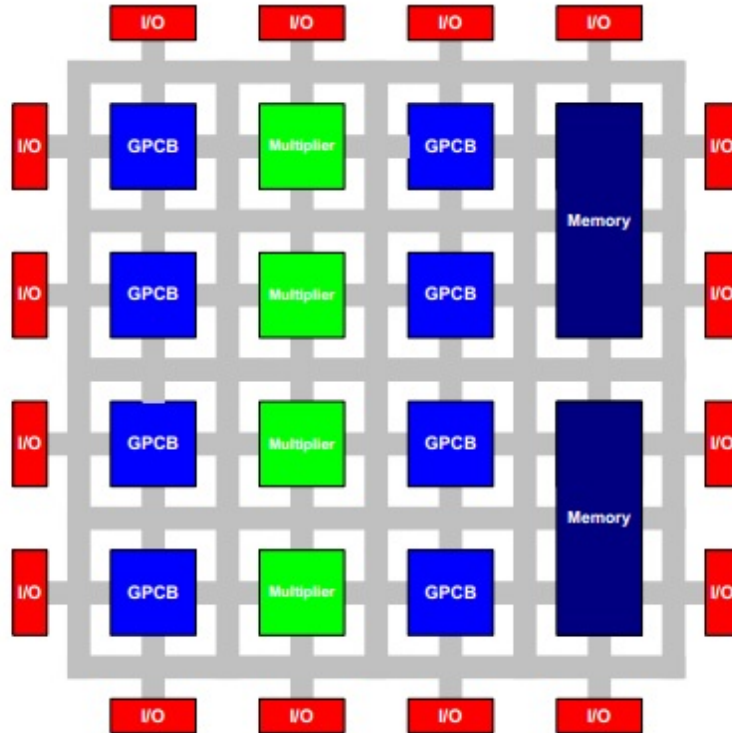


Figure 2.2: Heterogeneous FPGA Architecture[11]

2.1.1 Programming Technology

The re-configurability/programmability term in FPGA means their ability to implement new circuit design on chip after the fabrication is done. Ideally we like to have programming technologies which is re-programmable, non-volatile and easy to integrate. These technologies are used to program routing of interconnects or CLBs, which are used to implement logic function. Many programming technologies are suggested for the reconfigurable architecture. Some of the well-known technologies are static-memory (SRAM – based), flash and anti-fuse [4]. SRAM based technology is widely used as

it uses standard CMOS technologies, which is easy to integrate. Also it has less static power dissipation and high speed. But as basic SRAM contains 6 transistors, this technology uses more area. The volatile nature of SRAM need a permanent external device to intact the configured data, which add extra cost and area overhead. On the other hand flash is non-volatile and takes less area. But we can program it for limited number of time. Anti-fuse technologies uses less area but it could be programmed only once. Hence SRAM -based technology are widely used in spite of area overhead.

2.1.2 Configurable Logic Block

CLBs are the core FPGA design. With the sufficient number of CLBs any logical function can be implemented over FPGA. A basic CLB consists of programmable logic primitives and flip/flop. The most common primitive block is k-input 1-output look-up table (LUT). It can be used to implement any k-input Boolean function. Fig 2.3 illustrates the basic structure of CLB. It contains N Basic Logic Elements (BLEs), and full crossbar for interconnect. This interconnect helps connecting the input of CLBs input to any of the BLEs and it also provides feedback path from the outputs of the CLB to any input of the BLEs. The BLE output directly connect directly to the outputs of CLB.

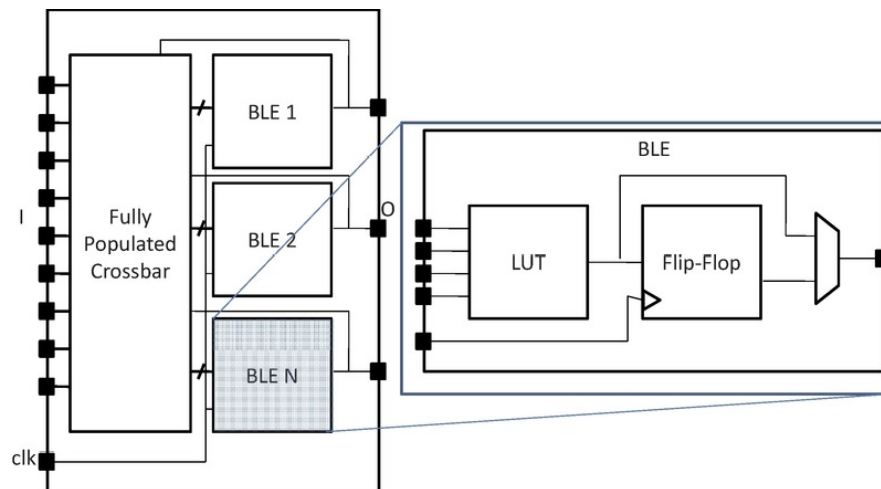


Figure 2.3: Classical CLB[6]

The enlarged view in the right side of Fig 2.3, the internal of the basic BLEs is shown. It is combination of LUT and flip-flop. Multiplexer is used to provide the latched output or direct output from BLE. Modern FPGA has enhanced the basic GPCB for area efficiency and timing efficiency [11]. Our thesis uses basic architectures of the BLEs.

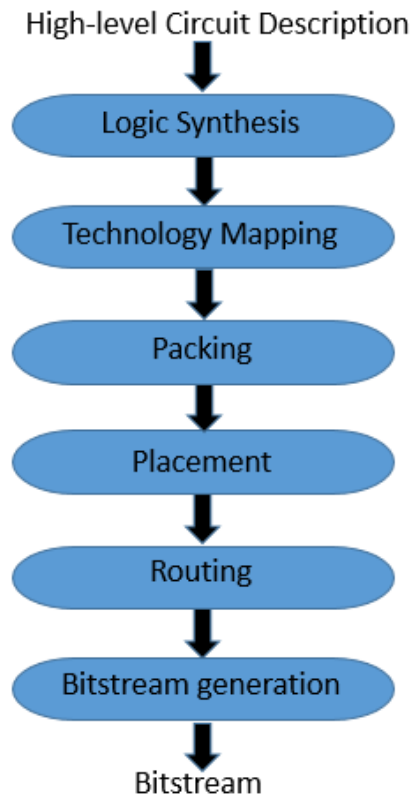


Figure 2.4: FPGA CAD flow[10]

2.2 FPGA CAD flow

To adapt the modern age complex design, many architecture have been introduced. The effectiveness of those architecture depends on the suitable suit of Computer-Aided Design (CAD) tool for FPGA. The FPGA CAD flow takes hardware descriptive language

(HDL) and gives the stream of bits as output. These bitstreams are finally configured to FPGA. The whole process can be categorized as five different steps as shown in Fig 2.4.

2.2.1 Logic Synthesis

In this step we transform given HDL into a set of Boolean gates and Flip-Flops. The register-transfer-level (RTL) description of the design is converted into hierarchical boolean network. This network is further optimized with technology independent techniques.

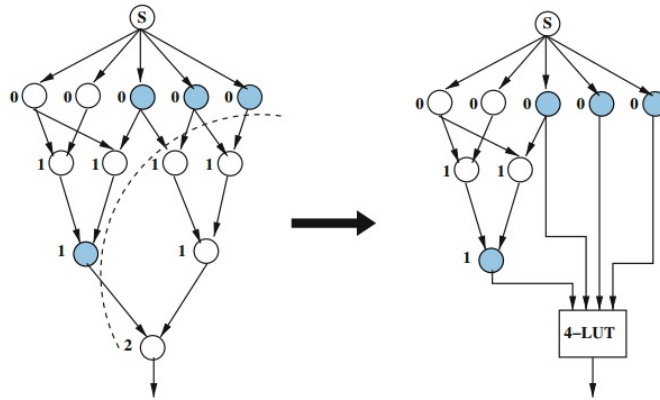


Figure 2.5: Technology Mapping[10]

2.2.2 Technology Map

Output from the above steps contains connected and optimized Boolean logic gates and flip-flops. The connection between those logic gates, flip-flops, primary inputs and output nodes can be represented as Directed Acyclic Graph (DAG) shown in the left side of Fig 2.5. Technology mapping is defined as mapping the boolean network defined as DAG form to the given library of cells. In FPGA the library cells contain k-input LUTs and flip-flops. Different objective like depth, area or power can be optimized while mapping. Final result provides a k-bounded LUTs and flip-flops as shown in right side of Fig 2.5.

2.2.3 Clustering

This is an important step for FPGA design, where we try to cluster the basic logic blocks which are k-input LUT and flip-flops. These clusters are directly mapped to the FPGA CLB locations. Clustering algorithm tries to minimize different objective function like local wirelength in a cluster or timing delay. Different algorithms of clustering have been developed, which can be categorized as top-down, depth optimal and bottom-up. Bottom-up methodology are preferred for FPGA CAD due to less runtime and reasonable time delay [10]. The well-known academic FPGA CAD tool– VTR (discussed well in chapter 3.2) uses this approach named VPack [12]. T-VPack [4] is another timing driven version of VPack.

2.2.4 Placement

In FPGA, placement means assigning the packed logic blocks on real physical locations available on the given FPGA architecture to employ the corresponding circuit. The goal of placement is to minimize objective function like wirelength (wirelength-driven), critical path delay (timing-driven placement) etc. Sometimes it aims to place logic blocks in such a way that it balances the wiring density across the FPGA (routability-driven placement). The placement methodologies can be broadly categorized as three types – Simulated Annealing (SA) based placement, partition-based method and analytic placement. We will go through these placement methodologies proposed for FPGA in next section 2.4.

2.2.5 Routing

Routing problem focuses enabling switch boxes and connection block of the pre-fabricated routing track of FPGA architecture to make a connection between configurable logic blocks and IO blocks at the peripheral. This step is divided into global routing and detail routing.

After completion of routing process, CAD tool generates bit stream of the final output from the routing step based on the target FPGA architecture. The desired circuit is configured once this bit stream file is loaded in the given FPGA chip.

2.3 Motivation

With the advancement of technologies, FPGA design has been enhanced and become more complex. It consists about 2 million of logic cell [13] in CLB, RAM blocks, Multipliers, DSP blocks and other components. It is now highly time expensive process to map the complex circuit onto FPGA for the given architecture. Placement of the technology mapped netlist consumes huge amount of time and shares a great chunk of time of the entire FPGA design cycle. Reducing runtime of the placement step is one of the big target of FPGA CAD researchers for maintaining the property of speedy reconfigurability of FPGA. Our research aims to achieve this.

2.4 Previous Work on FPGA placement

There are three major categories of placement algorithm are available for FPGA CAD. We will discuss all of them in the subsequent subsection. After discussing their pros and cons, we will explain our approach of the proposed placer.

2.4.1 Simulated Annealing based placement

Simulated annealing based placement is most popular among them due to high quality. Versatile place and route (VPR) [3] is the one of such kind of placer, which is now state-of-the-art academic tool for FPGA CAD. There are many advantage of this placer. It has open cost function which can be either of wire-length-driven, time-driven or path-driven. These cost function can be combined. It does both timing-driven (minimizing critical path delay) and wirelength-driven (minimizing Half-perimeter wirelength with tradeoff of 0.5 default. In spite of having great quality of the objective cost function, it takes huge time.

As VPR is SA based placer and easily available for research, we studied it extensively for knowing factors affecting the placement of FPGA. In this sub-section we will discuss the simulated annealing technique and the parameters accounted in VPR tool. Algorithm 2 shows pseudo code of SA based placer implemented in VPR. It start with initial random placement of packed netlist (logic blocks) and IOs on the FPGA chip.

Algorithm 1 Simulated Annealing Based VPR Placer Algorithm

```

1: procedure SA_PLACER
2:    $S \leftarrow \text{RandomPlacement}()$ 
3:    $T \leftarrow \text{InitialTemperature}()$ 
4:    $D_{limit} \leftarrow \text{InitialD}_{limit}$ 
5:   while  $\text{ExitCriterion}() == \text{false}$  do
6:     while  $\text{InnerLoopCriterio}() == \text{false}$  do
7:        $S_{new} \leftarrow \text{GenerateViaMove}(S, D_{limit})$ 
8:        $\Delta C \leftarrow \text{Cost}(S_{new}) - \text{Cost}(S)$ 
9:        $r \leftarrow \text{random}(0, 1)$ 
10:      if  $r \leq e^{-\frac{\Delta C}{T}}$  then
11:         $S \leftarrow S_{new}$ 
12:      end if
13:    end while
14:     $T \leftarrow \text{UpdateTemp}()$ 
15:     $D_{limit} \leftarrow \text{UpdateD}_{limit}()$ 
16:  end while
17: end procedure

```

Random moves are performed to find out objective cost function at certain temperature. If the cost reduces that move is accepted. However, if the cost increases, still there is probability of acceptance of that move. That probability is given by $e^{-\frac{\Delta C}{T}}$. This hill climbing ability allows SA not to converge at local minima and provide global optimization.

- Cost Function

The cost function is defined as equation 2.1, it is the parameter to define the quality of the placement. This linear congestion cost function provides the best result in reasonable computation time.

$$Cost = \sum_{n=1}^{N_{nets}} q(n) \left[\frac{bb_x(n)}{C_{av,x}(n)} + \frac{bb_y(n)}{C_{av,y}(n)} \right] \quad (2.1)$$

Where summation is over all the nets in the circuit. BBx and BBy are the horizontal and vertical span of bounding box of each nets. q(n) is the factor which helps in compensating the underestimation of wire length for net with more than three terminal. As suggested in [14]. $C_{av,x}(n)$ and $C_{av,y}(n)$ are the average

Table 2.1: Temperature Update Schedule [3]

Fraction of Moves Accepted (R_{accept})	α
$R_{accept} > 0.96$	0.5
$0.8 < R_{accept} \leq 0.96$	0.9
$0.15 < R_{accept} \leq 0.8$	0.95
$R_{accept} < 0.15$	0.8

channel capacities (in tracks) in x and y direction respectively, over bounding box of net n.

- Initial Temperature

As the SA starts with random placement, it targets to avoid local minima by hill climbing. It needs good high initial temperature. VPR follows [17] to get the initial high temperature. After creating initial random placement it does N_{blocks} moves (pairwise swaps) of logic block and IOs. The initial temperature is assigned to 20 times of the standard deviation of costs of each N moves.

- Number of moves

Number of moves at each temperature is defined as $10 * (N_{blocks})^{1.33}$. R_{accept} is the rate of acceptance of the move at each temperature. [3] proposed a new temperature update scheme $T = \alpha * T_{old}$, where α depends on the value of R_{accept} as shown in the Table 2.1. It ensures that at high temperature almost every move is accepted avoiding local minima and spent enough time at temperature where significant fraction of, but not all, moves are being accepted.

$$Number\ of\ moves = inner_num * (N_{blocks}^{1.33}) \quad (2.2)$$

- Dlimit

[15][16] suggests to keep R_{accept} near to 0.44 as long as possible. Which can be achieved if the blocks are interchanged in the range of D_{limit} . Initially this limit

is set to the FPGA dimension. It varies as per below equation.

$$D_{limit}^{new} = D_{limit}^{old} * (1 - 0.44 + R_{accept}^{old}) \quad (2.3)$$

- Exit Criteria

Finally annealing is terminated when $T \leq 0.005 * cost/N_{nets}$

2.4.2 Partition-based placement

Another version of FPGA placement has been proposed in [9]. It uses partitioning based approach for delay optimization using alignment cost as the objective function. A circuit is bi-partitioned in breadth first manner recursively keeping criticality of the nets across the partitions and tries to minimize the cut numbers. It is followed by low-temperature simulated annealing to final refinement. The advantage of this method is that it minimizes the delay in placement stage. It also takes less time compared to VPR. But it suffers some quality loss as it depends on the performance of partitioning [18].

2.4.3 Analytical Placement

Third approach of placement is analytical placement. It has basic idea to express the cost function and constraint as analytical function of the coordinate of the modules (nodes). Then the placement problem is transferred to mathematical program. This method is widely used in current generation of ASIC physical design due to less time consumption and efficient performance. FPGA also adapted this technique to deal with complex design and reduce physical design cycle time. This motivated us to search for new algorithm of placer which can be faster, efficient and effective. The base of our research is analytical placement.

We did immense literature survey over the analytical placement of FPGA. In the last couple of decades, analytical placement have been favorite for FPGA CAD researchers to cope with the new introduced version of FGPA and architecture.

[8] presented first quadratic based analytical placer, which tries to minimize quadratic wirelength cost by solving linear equation with conjugate gradient methods. In each iteration they map the circuit on the chip and add dummy node to expand placement.

They do linear adjustment to minimize linear and squared wirelength. It follows low temp simulated annealing for final placement. QPF claims on average, 5.8 times faster than well-known FPGA placement tool VPR. SCplace [19] presented an algorithm which performs simultaneous clustering and placement to minimize both total wirelength and longest path delay.

StarPlace [20] is based on near-linear model net model called star+. This model tries to minimize the over-estimated squared wirelength with the help of modified star based net model. It helps to create efficient solution for the resulting non-linear equation systems, which is solved with conjugate gradient solver and successive over-relaxation. Compared to VPR it has 8-9% of reduction in critical path delay with 5 times faster.

HeAP [21] has proposed analytical placement for heterogeneous FPGAs, which outperforms industry standard Altera's both timing and non-timing driven placement. It adapted bound2bound net-model of SimPL[22] which gives high quality solution as it directly model HPWL. SimPL legalizes placement by spreading out the block across the chip. It adds artificial pseudo connection between each block and its target location in legalized overlap-free placement.

[23] presented another efficient and effective analytical placer based on multilevel frame work, which is superior to VPR with respect to criticality and wirelength. They proposed multilevel strategy for both timing and wirelength driven placement with block alignment consideration to decrease delay profile. They have taken weighted and stable log-sum-exp model of the wirelength model and have done look-ahead legalization which improved quality of placement in faster convergence. For wirelength driven detailed placement it uses windows based bipartite matching techniques. It takes help of VPR based SA to get further refinement at low temp SA.

2.5 Summary

In this section we have discussed architecture for FPGA, our motivation towards the thesis and different work done in that field. As discussed above currently FPGA CAD researchers are looking for a placer which are more efficient to follow the advanced FPGA design. Few works have been done in the area of analytic placement which takes lesser time and provide quality results. It motivated us to look into new methods of

analytical placement. In the next section we will discuss our approach of research work.

Chapter 3

Analytical Placer Algorithm

In this chapter we will propose our idea of analytic placer. In first section we will do problem formulation of analytical placement which is used as base of our placer. Next section will show the placement frame work. Later we will discuss our proposed algorithm in complete detail. That section will discuss about our MATLAB placer engine flow. We will show how the connection matrix has been created, which is the main input of the analytical placer. Then we will look into the process of deciding the number of iteration used in our algorithm. It has great importance in deciding the final placement. Further we will see our proposed algorithm of legalization for the final placed CLBs. These placed CLB location will be fed back to the VPR flow again for low temperature simulated annealing for further refinement. In the last section, we will focus on the process how we have found the low temperature for SA.

3.1 Preliminaries

3.1.1 Problem Formation

FPGA placement problem can be defined as problem of hypergraph $H = (V, E)$. $V = \{v_1, v_2, v_3, \dots, v_n\}$ represents n CLBs and $E = \{e_1, e_2, e_3 \dots e_k\}$ represents k hyperedges connecting those CLBs. Let x_i and y_i be the x and y coordinate of the CLB v_i for the given architecture of FPGA. The placement problem tries to find optimal location (x_i, y_i) of v_i in 2D array of placement region, minimizing the total wirelength or the

critical path delay. Our research aims to minimize the wirelength. For our placer we have taken quadratic wirelength as our objective cost to minimize. The total quadratic wirelength of two pin nets can be defined as

$$\Phi(\vec{x}_i, \vec{y}_i) = \sum_{i,j} w_{i,j} [(x_i - x_j)^2 + (y_i - y_j)^2] \quad (3.1)$$

Where $w_{i,j}$ is the weight of the hyperedge connecting blocks i and j . These objective function can be solved separately for x and y components and cast in the matrix form. For x dimension Q_x represents connection between movable blocks and vector \vec{c}_x represents connection between movable blocks and fixed IO locations.

$$\phi(\vec{x}) = \frac{1}{2} \vec{x}^T Q_x \vec{x} + \vec{c}_x^T \vec{x} + const \quad (3.2)$$

Equation 3.2 is degree –two polynomial, minimizing them involves taking partial derivation with respect to each variable and setting resulting system to zero. The problem will reduced to

$$Q_x \vec{x} = -\vec{c}_x \quad (3.3)$$

This equation can be solved with simple linear solver. After solving this equation it gives global optimized solution. This concept can be understand with respect to spring connected blocks where IOs are at fixed position and CLBs are connected to IOs and other CLBs with a spring. Solving 3.3 tries to position each CLBs in such a way that total energy of the system will be minimized.

3.1.2 Analytical Placement Steps

There are three main analytical placement steps [24], namely

- global placement
- legalization
- detailed placement

The global placement gives high quality placement by minimizing our objective cost function (in our case quadratic wirelength). But this placement cannot be accepted as the equation is solved to minimize total potential energy of the spring systems as

discussed in previous section which results overlaps of blocks. In quadratic placement of analytical framework, to avoid overlap we use two ways to spread out blocks evenly over the chip. First we can add center of mass constraint to prevent clustering together. Other way is to add forces to pull blocks from dense region to sparse region. In both ways we need to add constraint/force in iterative manner to spread out blocks gradually. We have adopted the first method based on GORDIAN [25] in slightly modified manner. We will discuss this concept in next subsection. But still in the final steps of iterative solution we do not get fully non-overlap solution. So we need extra effort to remove those overlap. In FPGA it has some extra constraint to get all blocks at certain physical location, unlike ASIC where removing all overlap can work as the final legal placement. Hence in FPGA placement assigning blocks on physical location is also the part of legalization apart from removing overlap. The final step of analytical placement is detailed placement. During legalization we change the optimal solution found from the global placement. In global placement we have the best optimized solution for the given constraint, but the objective cost increases as we legalize. Detailed placement helps to moves blocks such a way that it tries to reach nearby the optimized result of global placement. Low temperature simulated annealing is one of the way we can do for this refinement. We have chosen low temperature simulated annealing for this.

3.1.3 GORDIAN Placement

We used this techniques with some modification for our global placement. This technique adds center of mass constraint while doing even distribution of blocks over the chip. Given uneven quadratic solution obtained after solving 3.2, the block distribution can be improved as following procedure. First a vertical cutline is used to partition all blocks into two subcircuit and the region into two subregion. Then, for each subcircuit, constraint in the x-direction is added. It forces the center of the mass of all its module towards the center of corresponding subregion. Now placement problem is solved again.

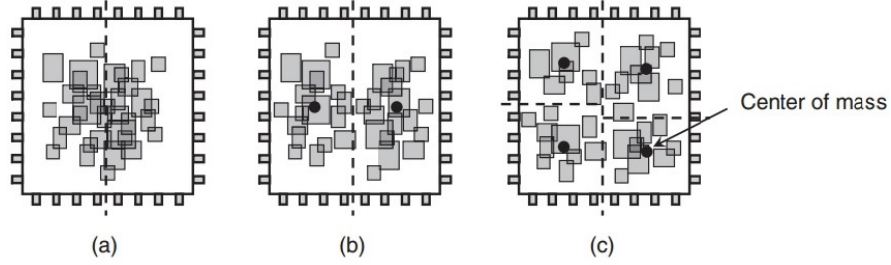


Figure 3.1: GORDIAN: Spreading overlapped CLBs with center of mass constraint [26]

The center-of-mass pulls two sets of blocks of both subregion away from each other as shown in Figure 3.1. The process is applied hierarchically to improve the distribution of CLBs in each subregion. This happens until we reach a point where each subregion contains a certain number of blocks only. In each iteration of this process the placement of all blocks is considered together as a single global optimization. The coordinates of the center of mass are the area weighted mean values of block's co-ordinate. This is the linear equality constraint. Hence, solving for the global optimization at each iteration is a convex quadratic program, which is equivalent to solving a systems of linear equation. For solving this convex quadratic problem we have taken help of MATLAB's quadprog function, hence we implemented our idea in MATLAB for global placement. Next subsection will elaborate quadprog function in brief.

3.1.4 Quadprog

For minimizing quadratic equation like 3.2 with linear constraint, MATLAB offers a function- quadprog. It helps minimizing the problem specified by

$$\min \frac{1}{2} x^T A x + B_x^T x$$

$$\text{such that } A_{lin} * x \leq B_{lin}$$

$$B_{xlb} \leq x \leq B_{xub}$$

Where A is the nxn matrix, which is based on the connectivity Matrix and resulted as equation 3.3, n is number of total movable blocks. B_x is nx1 matrix which says connectivity of CLBs with IOs based on its location. A_{lin} and B_{lin} are used to set linear

constraint of center-of-mass while solving iteratively. B_{xlb} and B_{xub} are the matrices used to set the boundary condition in each subregion. There is similar kind of equation, which is used to solve y-coordinate.

3.2 VTR-VPR

As our placement methodology was analytical placement, we were searching for a platform where we can replace our placer and test with the existing analytical based placement. [21] was one of them. We approached that research group working on analytical placement of FPGA, but could not get required details. So we decided to go with well-known FPGA CAD tool-VTR, which is considered as state-of-the-art for FPGA CAD researchers. We used VTR version 7.0 for our research work. We downloaded it from <https://github.com/verilog-to-routing/vtr-verilog-to-routing>. While installing it we made sure that `ENABLE_GRAPHICS` variable is true, so that we can see the graphical view of the placed logic blocks. It helped us a lot in the initial phase when we needed to understand the connectivity of the packed logic block. After exploring the source code and manual of VTR, we figured out that VTR is wrapper over VPR and other synthesis CAD tool named as Verilog-To-Routing. It takes Verilog HDL as input which passes through the synthesis tool ODIN and results flatten netlist and blackboxes. The output of this tool goes to the logic synthesis tool named ABC which performs technology independent logic optimization of the circuit. The output from the ABC is in .blif format of LUTs, blackboxes and flip-flops. This .blif format is the input for VPR. VPR is combined package of packing, placement and routing, which packs the circuit in more coarse-grained logic blocks, place them and route respectively. VPR dumps out files at all different stages. Hence we aimed to focus completely on the VPR tool rather than the entire VTR flow. The source code of the whole VTR is well organized, but huge, that one can work independently on any of the flow for research. We decided to bypass the placement with our idea of analytical placer. We developed our idea in MATLAB and integrated with VPR which detoured the CAD flow from VPR-based SA to MATLAB based analytical placer. Next section will discuss the complete set up of our flow.

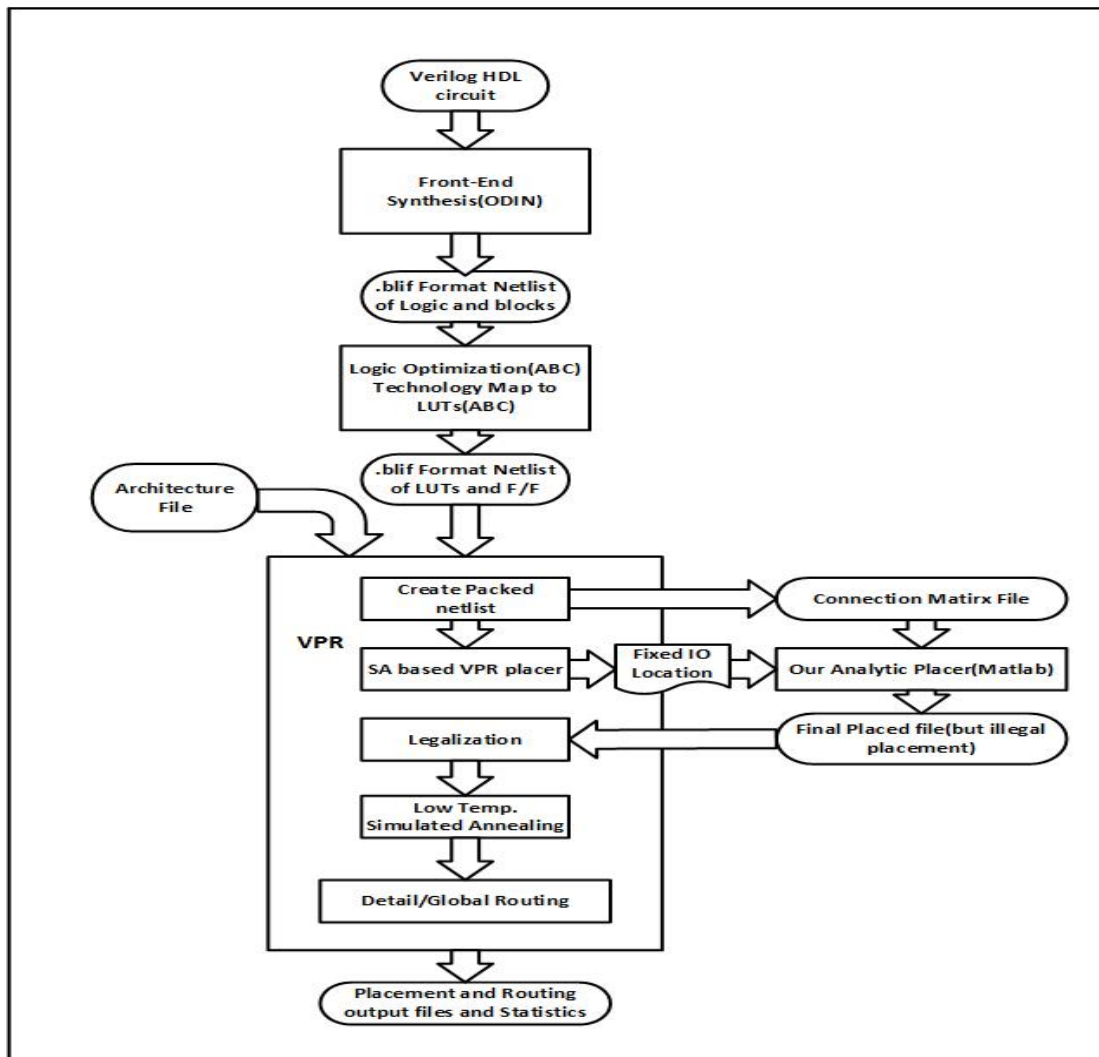


Figure 3.2: VTR and our placer Flow

3.3 Analytic Placer Setup - Complete experimental flows

Our whole flow is illustrated in Figure 3.2. For making the complete flow of the placer we needed to know the connection Matrix of the packed netlist which is the output of the packing step of the VPR tool. VPR does the best packing of the connected gates through LUTs [3]. Packing step is optimized to take care of the best routing resources

available for the given architecture. So we decided to take the packed netlist from the packer of VPR. In the beginning we have decided to break the flow of VPR in its source code only and implement our algorithm. But then we left this idea behind as we wanted to use some available quadratic solver for wirelength optimization and test our idea. MATLAB has such quadratic solver – quadprog. In future one can write the quadratic solver methods in C and make it the part of VPR flow itself.

After the placer gets an optimized placement of the packed logic blocks, we send it back to the VPR tool where we spread the overlapped blocks. We take each overlapped location one-by-one and make a spiral move for the blocks which are overlapped. Further for the detailed placement, we do low temperature simulated annealing which improved our wirelength cost function. Whole placement follows three steps in our placer -

- Stage 1:
 - Build connection matrix from VPR
 - Find IO locations from VPR based SA placement
- Stage 2:
 - Build and solve quadratic equation with linear constraint of boundary condition and center-of-gravity. -
 - Legalize the final output of the placer to remove the overlap of CLBs
- Stage 3:
 - Low temperature simulated annealing to refine the final placement

3.4 MATLAB Engine Flow

Our analytical placer is developed in MATLAB. We tried to minimize global quadratic wirelength cost of the Complex Logic Blocks (CLBs) connections. Connection matrix of the movable block and IOs are given to the placer. The algorithm to find connection matrix is discussed in 3.5. This placer solves the equation 3.2 using MATLAB quadratic solver, where we have given different linear constraints, like in Gordian Method, in each iteration.

First we have created Matrix A and B from our connection matrix as per equation 3.2. Pseudo code of the analytical placer is shown in Algorithm 2. The side size of the FPGA placement is taken as the square root of the movable gate counts. In quadratic placement we keep our IOs fixed at the boundary and moves the movable block to minimize the cost function. For fixing the IOs we had two options - 1. Put IOs in random function like VPR does before the simulated annealing. 2. Run the VPR placer first till the placement stage and take the IOs location from the final placed location to our initial stage of the placer. We prefer to follow second method, as it gives us good way to compare our result with the placement result of the VPR. Hence we give two files to our placer as input. One is the final placed output of the VPR placement stage and other is the connection matrix file generated form the VPR after packing. We place the IO blocks first at fixed location. It helps us to make the B matrix equation.

Initially, we solve the equation with quadratic solver using just the boundary condition of the FPGA chip, which gives us optimized wirelength cost. But this placement cannot be accepted, as most of the CLBs cells are overlapped and clustered. We need to move cells apart so that it will come to the legal position without deteriorating the obtained minimized cost from the quadratic solver much. Our aim is to move CLBs apart from the clustered place.

We have adopted the partition based quadratic solver where we assign linear constraint - center-of-mass for each partition and lower and upper boundary of each partition and solve the quadratic placer to minimize the global cost. This is repeated for a certain number of times until we get very less overlap of CLBs. Decision of number of the iteration is discussed in next section 3.6. Each iteration divides all sub-region in four partition and tries to make a uniform density of movable CLBs over the FPGA placeable area.

Algorithm 2 Analytical Placer Algorithm

```

1: procedure ANALYTICAL_PLACER(Connection Matrix, Fixed IO location)
2:   Create  $A$ ,  $B_x$ ,  $B_y$  Matrix
3:    $sidesize \leftarrow \text{ceil}(\text{square\_root}(\text{movable\_blocks}))$ 
4:    $B_{mov_i} \leftarrow \text{total\_movable\_blocks}$ 
5:   Set boundary condition of X and Y
6:   Set centre-of-mass at the center of chip ▷ Global placement
7:   Get_placement(linear constraint) ▷ solve quadprog with linear constraint
8:    $n_{v_s} \leftarrow 0$  ▷ number of vertical stripes in a iteration
9:    $n_{h_{ps}} \leftarrow 0$  ▷ number of horizontal partitions in a stripe
10:   $n_i \leftarrow \text{floor}(\log_2(B_{mov_i})/2)$  ▷ number of iteration
11:  while  $n_i \neq \text{NULL}$  do
12:     $n_{v_s} \leftarrow n_{h_{ps}} \leftarrow 2^{n_i}$ 
13:     $B_{sort_x} \leftarrow \text{sort\_blocks\_in\_X}(B_{mov_i})$  ▷ Sort block w.r.t X coordinate
14:     $n_{B_v} \leftarrow B_{mov}/n_{v_s}$  ▷ number of blocks in each vertical stripes
15:     $\text{set\_lower\_bound\_for\_X}()$ 
16:     $\text{set\_upper\_bound\_for\_X}()$ 
17:    while  $n_{v_s} \neq \text{NULL}$  do
18:       $B_s \leftarrow \text{assign\_blocks}(n_{B_v}, B_{sort_x})$  ▷ blocks in a stripe
19:       $B_{sort_y} \leftarrow \text{sort\_block\_in\_Y}(B_s)$  ▷ sorted blocks in a stripe
20:       $n_{B_{h_{ps}}} \leftarrow n_{B_v}/n_{h_{ps}}$  ▷ number of block in each partition of a stripe
21:      while  $n_{h_{ps}} \neq \text{NULL}$  do
22:         $B_{ps} \leftarrow \text{assign\_blocks}(n_{B_{h_{ps}}}, B_{sort_y})$  ▷ Blocks in each partition of
        stripe
23:         $\text{set\_lower\_bound\_for\_Y}()$ 
24:         $\text{set\_upper\_bound\_for\_Y}()$ 
25:         $\text{set\_centre\_of\_gravity}()$ 
26:      end while
27:    end while
28:     $\text{Global\_placed\_CLB\_location} \leftarrow \text{Get\_placement}(\text{linear constraint})$  ▷ solve
    quadprog with linear constraint
29:  end while
30:   $\text{Final\_placed\_CLB} \leftarrow \text{locate\_grid}(\text{global\_placed\_CLB\_location})$ 
31: end procedure

```

In first iteration, it first divides a sub-region in two partitions vertically. We call it vertical stripes. We sort the movable blocks with respect x- co-ordinate and assign half of them in each of the vertical stripes. Movable blocks of each stripes are further sorted with respect to y-direction. We divide each stripes horizontally in two parts and assign half of the sorted block in each divided part of the vertical stripes. It tries to do equal distribution of CLBs in each partition of a sub-region, which helps to have uniform distribution of the CLBs over the FPGA chip area. The same process repeats for the calculated number of times. CLBs locations of final iteration are not at the actual location of the CLBs of FPGA fabric. We snap the X and Y location of the final location to the nearest integer to find valid physical location of the CLBs. But still few of the CLBs overlap. In next step we perform legalization, which is discussed in section 3.7, to remove those overlap.

After finishing the analytic placement, we perform low temperature simulated annealing which refines our result further and improves our wirelength cost. The steps and strategy for simulated annealing is described in Section 3.8.

3.5 Connection Matrix Generation

Connection Matrix is the soul of the quadratic solver based placement. It says how the blocks are connected to each other and with IOs. In this FPGA architecture we aim to place CLBs (movable blocks) at pre specified CLB location (grid location) of FPGA. VPR gives packed logical block (CLB), according to the architecture. We need to find the CLB connectivity and make a connectivity matrix of that. Further, this matrix is used to get the A , B_x , B_y matrix of linear equation of our problem statement.

We found that VPR itself dumps an intermediate .net file after packing, which could be used to find the connectivity matrix. This file is input to the classical VPR placer. We could parse this .net file and get the connectivity matrix but it would have created extra overhead. Further exploration of VPR code guided us to use the in-built data structure to create connectivity matrix. All the CLBs are connected to other blocks via pins which are organized hierarchically. The connectivity can be traced from data structure as [27] –

- The name of the input pin in a CLB is same as that of the net using that pin.

- The name of output pin of a primitive (leaf level) is same as that of the net using that pin.

We created our own data structure named `s_clb_info` which helped us to dump the connectivity matrix. Our procedure for getting connectivity is as below-

- Select each CLB from Data structure of post pack VPR
- Go to each CLB's input pin and check if that is open or not
- If OPEN, then this pin is not connected to any pin
- If not then find the `pin_index` and get the net name from `vpack_net` data structure of VPR
- Get the CLB index of the obtained net name, as this net name is the name of the primitive level block (LUT or Flip-Flop) of CLB.

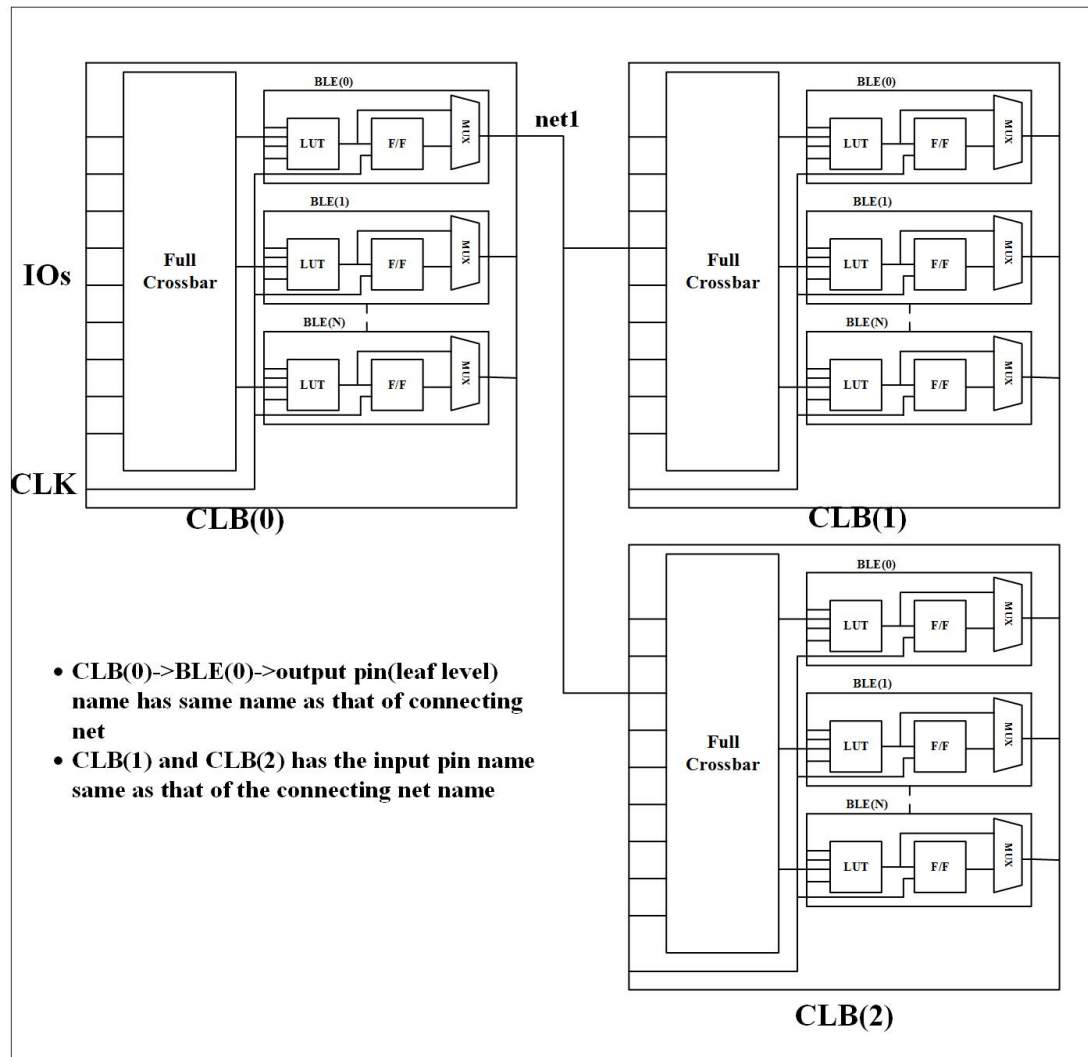


Figure 3.3: Illustration of CLB connection to form Connection Matrix

This way CLB under observation is connected to the other CLB via a net name. As shown in Fig 3.3 The net name is having the same name of primitive level block of another or same CLB to which the first CLB is connected. VPR data structure logical_block contains all information with clb_index. We make a matrix of $N \times N$, where N is the total sum of number of IOs and CLBs. We mark corresponding entries of the matrix as 1 which are found connected from the above technique. Unconnected CLBs,

IO indexes are marked 0.

3.6 Decision of number of iteration

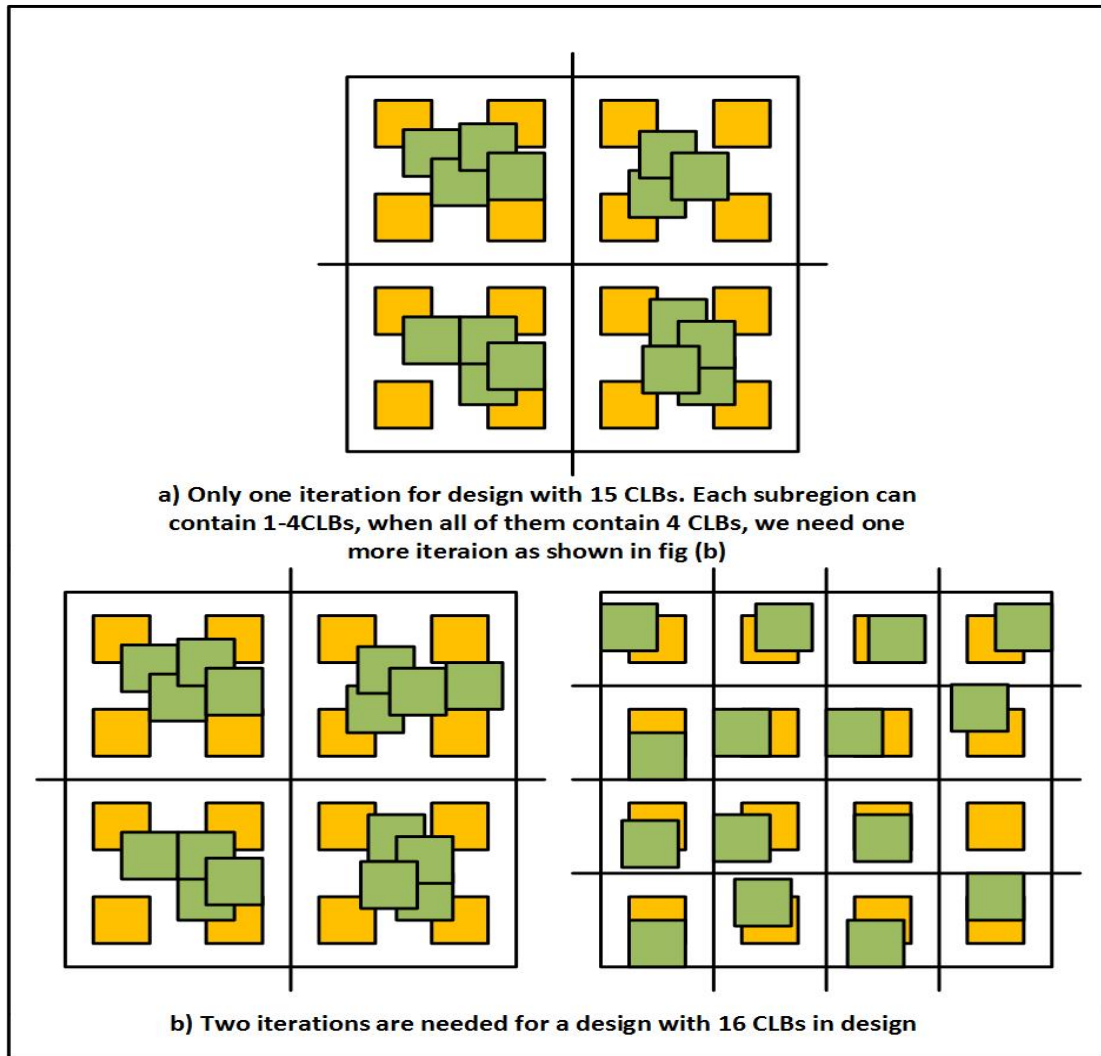


Figure 3.4: Illustration of no. of iteration for design with a) 15 CLBs and b) 16CLBs

Number of iteration after the first global placement is important parameter to spread out the clustered CLBs. Our aim is to make uniform distribution of CLBs across the

FPGA chip in each iteration.

$$\text{Number of sub-region in a iteration} = 2^{2*I}, \text{ where } I \text{ is the current iteration count} \quad (3.4)$$

Equation 3.4 shows the total number of subregion in current iteration. For instance, in first iteration total number of sub-region is 4, in next iteration it is 16 then 64 and so on. We targeted to find the number of iteration in such a way that at the end of the final iteration every sub-region contains at least one CLB or max 4 CLBs. This makes our task easy while doing legalization in the next step. This way the total number of iteration depends on the total number of movable blocks. Which we formulated as-

$$\text{Number of iteration} = \text{floor}\left(\frac{\log_2(\text{movable block count})}{2}\right) \quad (3.5)$$

We have explained iteration count number for two cases as shown in Figure 3.4. Fig (a) shows the case where we need only one iteration for a design with 15 CLBs. Here number of iteration is one as per Equation 3.5. For the benchmark with 4 – 15 CLB will have max iteration number of 1. So each sub-region can contain minimum of 1 and maximum of 4 blocks in a region. If the number of CLBs for a benchmark lies in between 16-63 the total number of iteration will be set to 2. Fig(b) is one example with 16 CLBs. For this case, there will 2 number of iterations based on Equation 3.5. In each iteration of our algorithm we spread movable block in each sub-region to have uniform density over the FPGA chip as mentioned in section 3.4. It enable us to get less overlap at the end of our final iteration. Later we snap all CLBs to the nearest valid the location of FPGA.

3.7 Legalization

In FPGA, quadratic placement has two main problem. First, it gives the overlapped CLBs for the optimized cost and second, most of those placed CLB's locations are not at the legal position as per the FPGA architecture. In our placer, we spread out CLBs in each iteration as described in section 3.4. Then each of those CLBs were snapped to the nearest logical place of the FPGA CLB location. But many of those CLBs are still overlapped. Legalization is the final step to get the CLBs at the correct position.

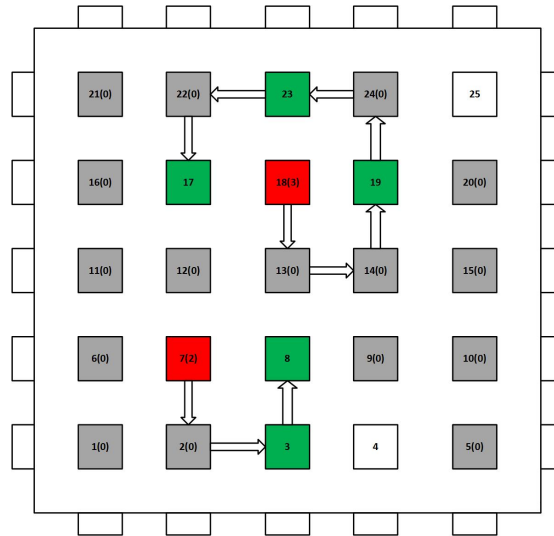


Figure 3.5: Spiral Legalization

We thought many ways to legalize them. First we thought to sort every CLBs with respect to sum of its X and Y co-ordinate and place them in wave fashion all over the FPGA. But this way we lose the optimized wirelength based placement obtained from our MATLAB placer. Other idea was to start from the center of the FPGA and move in a spiral wave fashion clockwise or counter clockwise. If we find the overlap position we take each overlapped CLB block and place them to the next available place. But this also does not serve our purpose to save the placement obtained from the analytical placer, as few overlapped CLB can be placed far apart from the region where it was assigned at the end of the final iteration of our placer. Then we decided to do the spiral based legalization for each overlapped grid of FPGA.

Steps of spiral based legalization -

- create a list of overlapped grid index.
- create list of overlapped CLB index
- iterate through each overlapped grid index
 - take start point as the overlapped grid position obtained from the list of overlapped grid index.

- move overlapped CLB block one by one to the nearest empty grid location found after spiral in counter-clock wise fashion.

Figure 3.5 illustrates the spiral legalization of results obtained from MATLAB engine. The number of overlapped CLBs are in the parenthesis of grid location. If a grid location has no parenthesis, then that is empty location. We picked the overlapped location and searched empty location anti-clockwise to place overlapped blocks. In figure 3.5 location 2 and 18 have 2 and 3 CLBs overlapped respectively. The overlapped CLBs of grid location 2 have been placed at location 3 and 8. Similarly overlapped CLBs from grid location 18 moved to 19, 23 and 17 grid location.

3.8 Low temp. Simulated annealing

For detailed placement we have used low temperature simulated annealing. In legalization we have pulled blocks obtained from global placement apart and put them at the legal physical location. Due to this the cost function increases from the global optimal solution. Low temperature simulated annealing tries to achieve the cost function near to the global optimal cost function. We have used the same concept of SA as described in subsection 2.4.1. But we need to set an initial temperature which works better in our case. This temperature should not be so high that it will deteriorate our analytical placer output and if the starting temperature is too low then the optimization will not be sufficient and placement might get trapped at local minima.

For deciding this crucial temperature value, we have followed same method what VPR did for deciding its initial high temperature. But in this case we do not have to accept all bad move like VPR does as discussed in the case where the objective cost value increases after a swap. According to the simulated annealing, a bad move will be accepted with probability $e^{-\Delta C/T} > r$, where r is uniform random value generated in the range $[0,1]$, ΔC is the cost difference due to the swap.

$$e^{-\Delta C/T} \geq r$$

$$T = \frac{-\Delta C}{\ln(r)} = C * \frac{-\Delta C}{\ln(r)}$$

First we have set a temperature based on above equation which has only 10% probability of accepting ($r = 0.1$) 5% degradation ($\Delta C/C = 0.05$) in the cost due to move.

We put value of C as 1 because VPR uses normalized cost function while assessing swap. At that temperature we do swap of blocks for certain number of time decided by VPR for starting_t function. We take average of each accepted move and put them in the above equation to find the initial temperature and then divide by 60. It gives us a temperature where acceptance rate is approximately 30-35 percentage. After getting initial temperature we reset all moves to get the placement obtained from our analytical placer. We have used this as our initial temperature for low temperature simulated annealing.

3.9 Summary

Current section explained our proposed methodology for analytical placer and its integration with the VPR tool. We have discussed about the algorithm used for global placement which was developed in MATLAB. Spiral legalization was done to remove overlap obtained from final step of the global placement. Low temperature assignment was one of the important thing which helped in final refinement after legalization. In the next section we will explain our experimental setup done for the research.

Chapter 4

Experimental Setup and Result Analysis

This chapter explains the experimental setup for our research work. We have used Verilog-To-outing version 7.0, the well-known academic research tool for FPGA CAD. Architecture is the most important input for the FPGA design. In first section, we will discuss about the architecture we chose for our experiment. The next section shows our criteria of selecting benchmark and then we analyze our result.

4.1 Architecture Selection

Architecture of FPGA says which resources are available for mapping a digital circuit on FPGA. For checking our algorithm we needed suitable architecture. New FPGA architectures are suggested to deal with the complex circuits like fracturable LUTs, carry chain, multipliers and configurable memories. VTR has provided few such heterogeneous benchmarks. It also provides some classical architecture, which has simpler version of FPGA with LUTs, flip-flop and IOs only. Table 4.1 listed few important architecture used for research in VTR. As research is in nascent stage, it was suggested to test our flow with classical architecture. So we have chosen one of the classical architecture - k6_N10_40nm. It is homogeneous architecture with no hard blocks or memories. In future the algorithm can be updated for heterogeneous benchmark.

These homogeneous architecture consists of N basic logic element (BLEs), where

Table 4.1: Major Architecture Files in VTR 7.0 Release [28]

File name	Description
k6_frac_N10_frac_chain_mem32K_40nm	Comprehensive Arch: ten fracturable 6-LUTs with carry chains, 32kb RAM and hard mutipliers
k6_frac_N10_frac_chain_depop50_mem32K_40nm	Comprehensive Arch with depopulated crossbar
k6_frac_N10_mem32K_40nm	Comprehensive Arch without carry chains
k6_frac_N10_40nm	Comprehensive Arch without carry chains or hard logic
k4_N4_90nm	Classical Arch: four 4-LUTs per logic cluster and no hard blocks
k6_N10_40nm	Classical Arch: ten 6-LUTs per logic cluster and no hard blocks
hard_fpu_arch_timing	Classical Arch with hardened floating point block

each BLE is a LUT with an optimally registered output. There are two flavors of this architecture. First has 10 General input and four BLEs per cluster ($N=4$) and each LUT has 4 input. They have routing wire of length, which is single-driver, with $Fc_{in} = 0.15$ and $Fc_{out} = 0.25$ and $F_s = 3$. It has 3 IO pins per IO pad. The second variant has 40 inputs and 10 BLEs per cluster ($N=10$). Each LUT has 6 inputs. All the routing wire are of length 4, with $Fc_{in} = 0.15$ and $Fc_{out} = 0.1$ and $F_s = 3$. These architecture are based on the flagship k6_frac_N10_mem32k_40nm architecture without any hard blocks. There are 8 IO pins per IO blocks. k6_N10_40nm is such kind of architecture which we have used whit our flow.

4.2 Benchmark Selection

Once we decided our architecture, we looked for suitable benchmarks. As currently we have developed our algorithm for homogeneous architecture, we wanted such benchmarks which do not have any hard blocks or memories. Also we wanted to compare

Table 4.2: Statistics of Benchmark Circuits for k6_N10_40nm Architecture

Circuit	# Blocks	# IOs	# CLBs	# Nets	Chip Area
alu4	175	22	153	697	13 * 13
apex2	229	41	188	969	14 * 14
apex4	155	28	127	699	12 * 12
bigkey	596	426	170	1024	14 * 14
blob_merge	739	136	603	3113	25 * 25
clma	982	144	838	4815	29 * 29
des	661	501	160	997	13 * 13
diffeq	253	103	150	943	13 * 13
dsip	563	426	137	691	12 * 12
elliptic	606	245	361	1907	19 * 19
ex1010	480	20	460	2572	22 * 22
ex5p	179	71	108	669	11 * 11
frisc	492	136	356	1748	19 * 19
misex3	168	28	140	716	12 * 12
pdc	514	56	458	2292	22 * 22
s298	204	10	194	722	14 * 14
s38417	771	135	636	3567	26 * 26
s38584.1	977	342	635	3641	26 * 26
seq	251	76	175	879	14 * 14
sha	303	74	229	1322	16 * 16
spla	431	62	369	1808	20 * 20
stereovision3	61	41	20	125	5 * 5
tseng	279	174	105	588	11 * 11

our work with legacy work. After going through previous research we found Microelectronics Center of North Carolina (MCNC) benchmark suit. VTR provides logical optimized and synthesized netlist of those benchmarks in blif format. We have used 20 of those largest benchmarks. Table 4.2 provides the statistics of those benchmark circuits based on the architecture k6_N10_40nm. We also tested our flow with VPR specific homogeneous benchmark -blob_merge, sha and stereovision3.

4.3 Result and Analysis

4.3.1 Environment

All experiments were performed in identical condition. Both VPR and our placer had been ran on the same machine Intel(R) Core(TM)2 Duo CPU E840 @ 3.00GHz. All the benchmark were taken from the VTR 7.0 tool package. It had all synthesized and optimized benchmark in .blif format. We have taken 20 MCNC benchmarks from there and 3 of the VTR specific homogeneous benchmarks.

We developed our placer in MATLAB version R2013a and intergraded to VPR framework. We have used the same method to calculate bounding box wirelength as VPR does. The runtime of both the placer has been calculated from the very beginning to the end of placement, including all the intermediate steps of reading connection file and file of IO placement obtained from VPR. First, we have ran VPR to get initial location of IO and used that location to the input of our placer. We calculated the time consumed in classical VPR and then started our placer taking fixed IO location from VPR placer. We have given trade off factor of 0.5 which was default to find the cost during simulated annealing. Our MATLAB placer gives good initial placement to start so we do not want to swap CLBs all over the chip in detailed placement stage. Hence, we have assigned D_limit to the half of the size of FPGA chip size in our low temperature simulated annealing stage.

4.3.2 Results

Here we will show our results compared to VPR. We ran VPR in default mode, which are assigned to give best result out of it.

Placement Result : Table 4.3 presents comparison between VPR placer's runtime (RT) and wirelength (WL) with that of our MATLAB based Analytical Placer. First column have benchmark. Next two columns are for wirelength and runtime for VPR. Forth and fifth columns contain data of our MATLAB placer. In the last two columns we have taken ratio of MATLAB results and VPR results. The average of those results shows our MATLAB placer is 38 % faster but wirelength quality is poor. To improve wirelength quality we have ran low temp. simulated annealing over our MATLAB based placer. We have used VPR's simulated annealing engine with trade off factor

of 0.25 to get wirelength-driven results. Table 4.4 shows our result with respect to the VPR tool. The first column contains benchmarks. Second and third column have wirelength (WL) and runtime (RT) of VPR placer. Column fourth and fifth contains our analytical placer’s bounding box wirelength and runtime after low temperature simulated annealing. Last two column have ratio of Analytical Placer’s and VPR’s wirelength and runtime. Then we took average of this average of ratio obtained in last two columns. For wirelength it is 0.99 and for runtime it is 1.11.

Table 4.3: Bounding Box Wirelength and Runtime comparison after placement for VPR and MATLAB placer

Benchmark	VPR WL	VPR RT	MAT WL	MAT RT	WL (MAT/VPR)	RT (MAT/VPR)
alu4	7309.54	1.66	9736.20	1.05	1.33	0.63
apex2	11362.37	2.69	15240.85	1.28	1.34	0.48
apex4	7750.66	1.51	9859.67	0.99	1.27	0.65
bigkey	6776.50	3.90	11495.14	1.64	1.70	0.42
blob_merge	46980.02	19.98	82797.33	7.23	1.76	0.36
clma	70331.59	24.74	121879.60	14.39	1.73	0.58
des	8614.38	4.15	13855.22	1.82	1.61	0.44
diffeq	7291.86	2.20	10608.66	1.16	1.45	0.53
dsip	5845.72	3.09	8744.98	1.58	1.50	0.51
elliptic	21764.80	7.59	32836.37	2.78	1.51	0.37
ex1010	31907.66	9.20	59762.87	4.81	1.87	0.52
ex5p	7176.37	1.54	8592.26	1.02	1.20	0.66
frisc	25481.05	9.15	36157.44	3.25	1.42	0.36
misex3	7759.50	1.72	10323.72	1.08	1.33	0.63
pdc	39331.05	9.67	53868.06	4.97	1.37	0.51
s298	7427.17	2.14	9809.32	1.17	1.32	0.55
s38417	33393.49	14.56	64846.44	6.36	1.94	0.44
s38584.1	35105.41	16.47	68248.90	7.43	1.94	0.45
seq	10557.81	2.59	13840.20	1.21	1.31	0.47
sha	12173.20	4.73	19028.13	1.34	1.56	0.28
spla	26802.82	6.98	38282.61	3.38	1.43	0.49
stereovision3	500.53	0.26	655.57	0.83	1.31	3.21
tseng	3978.25	1.49	6098.06	1.10	1.53	0.74
Average					1.51	0.62

Table 4.4: Bounding Box Wirelength and Runtime comparison after placement for VPR and Analytical Placer with low temp. Simulated Annealing

Benchmark	VPR WL	VPR RT	AP WL	AP RT	WL (AP/VPR)	RT (AP/VPR)
alu4	7309.54	1.66	7217.51	1.67	0.99	1.01
apex2	11362.37	2.69	11249.04	2.47	0.99	0.92
apex4	7750.66	1.51	7584.53	1.57	0.98	1.04
bigkey	6776.50	3.90	6670.23	3.51	0.98	0.90
blob_merge	46980.02	19.98	47227.05	19.69	1.01	0.99
clma	70331.59	24.74	67824.41	30.95	0.96	1.25
des	8614.38	4.15	8594.74	4.81	1.00	1.16
diffeq	7291.86	2.20	7498.32	1.97	1.03	0.89
dsip	5845.72	3.09	5760.89	3.32	0.99	1.08
elliptic	21764.80	7.59	21254.04	7.34	0.98	0.97
ex1010	31907.66	9.20	31482.92	10.08	0.99	1.10
ex5p	7176.37	1.54	7043.68	1.64	0.98	1.06
frisc	25481.05	9.15	25141.94	7.46	0.99	0.82
misex3	7759.50	1.72	7666.87	1.73	0.99	1.00
pdc	39331.05	9.67	38100.55	11.12	0.97	1.15
s298	7427.17	2.14	7115.36	1.97	0.96	0.92
s38417	33393.49	14.56	32635.48	14.17	0.98	0.97
s38584.1	35105.41	16.47	35058.98	17.99	1.00	1.09
seq	10557.81	2.59	10393.91	2.39	0.98	0.93
sha	12173.20	4.73	12401.36	3.25	1.02	0.69
spla	26802.82	6.98	26260.26	7.31	0.98	1.05
stereovision3	500.53	0.26	524.05	0.89	1.05	3.43
tseng	3978.25	1.49	4201.57	1.70	1.06	1.14
Average					0.99	1.11

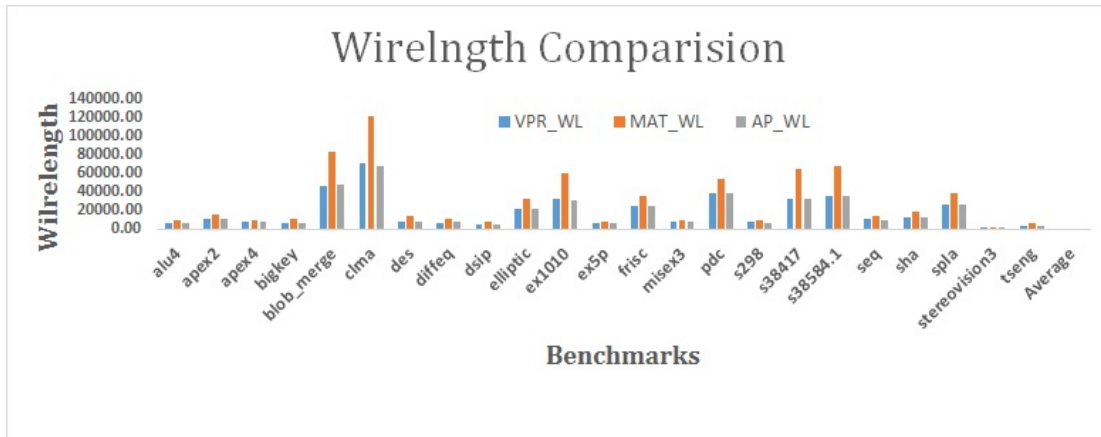


Figure 4.1: Bounding Box Wirelength comparison

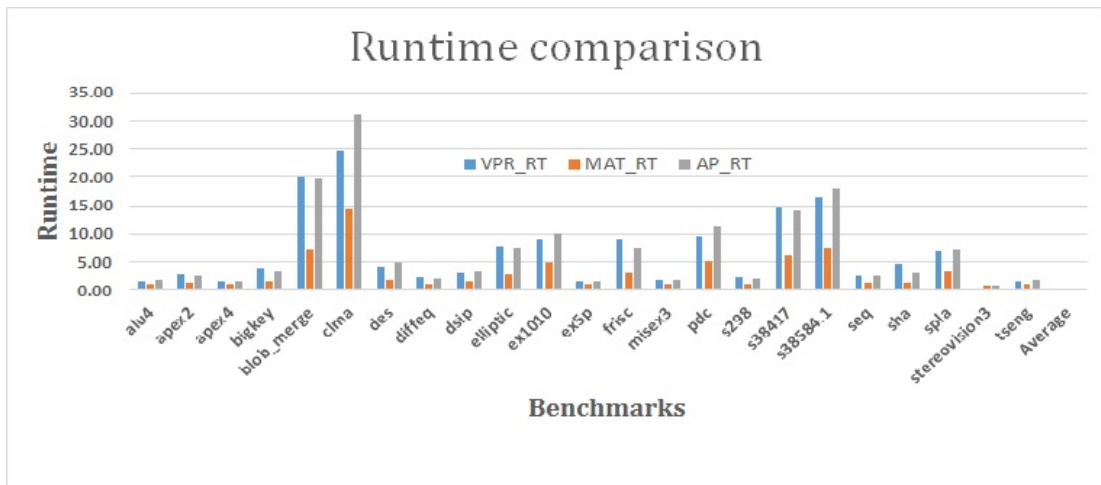


Figure 4.2: Runtime comparison of VPR and Analytical Placer

Post-Routed result: Table 4.5 presents post routed total wirelength and Channel factor. First columns names all benchmarks. Second and third column have data VPR routed Channel Factor(CF) and total wirelength. Fourth and fifth column have information of post routed analytical placer's channel factor and wirelength. We have divided the result of the Analytical placer by VPR's result. These result were averaged

Table 4.5: Comparison: Routed Channel Factor and total wirelength after routing of VPR and Analytical Placer

Benchmark	VPR Routed Chan F	VPR Routed WL	AP Routed CF	AP Routed WL	CF (AP/VPR)	WL (AP/VPR)
alu4	34.00	10060.00	36.00	9800.00	1.06	0.97
apex2	46.00	16176.00	48.00	15934.00	1.04	0.99
apex4	54.00	10745.00	46.00	12067.00	0.85	1.12
bigkey	46.00	9691.00	40.00	9958.00	0.87	1.03
blob_merge	72.00	71409.00	72.00	73612.00	1.00	1.03
clma	72.00	92565.00	68.00	95124.00	0.94	1.03
des	44.00	12740.00	40.00	12925.00	0.91	1.01
diffeq	36.00	10616.00	36.00	10231.00	1.00	0.96
dsip	42.00	9047.00	38.00	9035.00	0.90	1.00
elliptic	52.00	30721.00	52.00	30846.00	1.00	1.00
ex1010	58.00	45814.00	56.00	44235.00	0.97	0.97
ex5p	50.00	10743.00	50.00	10979.00	1.00	1.02
frisc	60.00	37084.00	62.00	36910.00	1.03	1.00
misex3	44.00	11138.00	44.00	11319.00	1.00	1.02
pdc	72.00	57490.00	72.00	56993.00	1.00	0.99
s298	30.00	9730.00	30.00	10009.00	1.00	1.03
s38417	44.00	44068.00	44.00	44608.00	1.00	1.01
s38584.1	46.00	45434.00	48.00	44568.00	1.04	0.98
seq	46.00	15569.00	46.00	15741.00	1.00	1.01
sha	44.00	18378.00	50.00	18285.00	1.14	0.99
spla	60.00	40068.00	60.00	38388.00	1.00	0.96
stereovision3	22.00	743.00	22.00	774.00	1.00	1.04
tseng	34.00	5926.00	34.00	5838.00	1.00	0.99
Average					0.99	1.01

over the total number of benchmarks.

4.3.3 Analysis

To make a flawless analysis all the experiments were ran for ten number of seed. The presented result in Table 4.3, Table 4.4 and Table 4.5 are average of those result. We have taken wirelength as total bounding box wirelength using the function `comp_bb_cost` of VPR. These wirelength are calculated at three stage - i) the end of VPR placer ii) after getting the legalized placement obtained from MATLAB engine iii) after final detailed placement of the output from MATLAB engine. The results are depicted in graph 4.1. Similarly runtime are compared in 4.2. We can see the wire length obtained from VPR and Analytical placer are comparable after low temperature simulated annealing. Table 4.3 shows our MATLAB placer ran 38% faster than classical simulated annealing based placer, but wirelength are poor. Results are improved with low temperature simulated annealing as shown in Table 4.4. The total runtime can be improved if we get faster detailed placement (low temperature simulated annealing) for refinement of our result of MATLAB engine.

Next we performed both post-placement routing by VPR router for both of the placer. Data in Table 4.5 shows that we have 1% less channel factor in the routed result of Analytical placer with respect to the VPR placed routed result. Although there is no improvement in total wirelength.

Chapter 5

Conclusion and Discussion

5.1 Conclusion

In this thesis we targeted to develop a new analytical placer for FPGA design. Starting from basic idea of GORDIAN placement in analytical placement we aimed to develop a complete flow of FPGA placement. The basic idea of the placer was developed in MATLAB and tested over some ISCAS circuits which were provided during our class assignment in VLSI Automation course. We needed a FPGA CAD tool to test our ideas. We started with an academic research VPR tool, as suggested by our Professor, to get better understanding of the entire FPGA CAD flow. We spent couple of months to explore the flow and the source code. We enhanced our placer to make it work in FPGA architecture based environment and integrated with VPR tool. We bypassed simulated annealing based placer of VPR with our analytical placer.

Currently FPGA CAD focuses on heterogeneous design. We wanted to check feasibility and robustness of our ideas in FPGA CAD environments first, so our placer has developed for homogeneous architecture for now. This placer can be scaled for heterogeneous architecture by assigning some constraints while iteratively doing partition based spreading of overlapped blocks. Our focus was to achieve the efficient and fast placer. We have compared our results with VPR placer. Our MATLAB placer is 38% faster than the simulated annealing based VPR placer, but with poor wirelength quality. After doing further refinement with low temperature simulated annealing its wirelength quality increases by 1% but at the cost of 11% more run time. The runtime quality

can be improved if we get better outcomes from MATLAB placer. As low temperature simulated annealing takes around 45-50% runtime of our complete Analytical Placer (MATLAB + Low Temp Simulated annealing). On the other hand we sacrifice wirelength quality then SA can be started at lower temperature to get better runtime.

5.2 Future Work

The current results of our analytical placer shows positive sign of feasibility of the idea in the current FPGA domain. This can be enhanced further to achieve a remarkable result for the wirelength-driven placer. Below are some of the ideas which can be implemented to provide this placer a significant standing among the FPGA placers of the market.

- As we have already checked the feasibility and robustness of the MATLAB based placer, one can map this algorithm to the C-based environment and integrate with the VPR tool just after the packing stage. This way the data structure of VPR can be used directly and we can overcome the extra overhead of creating new data structure while using our placer. This also reduces the runtime of the placer.
- Our current placer works for homogeneous architecture only. One can take care of hard blocks, carry chain and memory blocks by assigning some constraints while partitioning and updating connection matrix in each iteration to make it suitable for heterogeneous architecture. Once it is updated for heterogeneous, we can easily compare our tool with industry standard tool like Altera's Quartus with the help of Quartus II University Interface Program (QUIP). It will also open our way to compare with works like [21].
- In this placer, we have not used a net model converted multi-pin net model into two pin nets. If we convert them to clique and star based model, we can speed up placement.
- While solving iteratively, if we assign some weight over the net by predicting future result, we can get more efficient placement result.
- We need to have fix IOs at the boundaries for analytical placement. In the proposed placer we have taken fixed IO location obtained from VPR placer. The new

algorithm can be explored to have better IO locations based on the connectivity of CLBs. It can give more efficient result compared to VPR.

- While doing legalization we have followed spiral way to remove overlaps. It is a heuristic way of removing overlaps by placing the overlapped block to the nearest location. But it destroys the optimal result of our MATLAB based analytical placer significantly for denser circuit. It causes increment in wirelength cost. So, the runtime of low temperature simulated annealing also increases. This is because the initial temperature is a function of wirelength cost in our placer. One can include some cost function based decision while moving overlapped CLBs to the empty locations. The network flow concept can be adopted to keep the relative position intact during legalization

In conclusion, our thesis work presents a new idea for analytical placement. It can be made more efficient and effective placer for current FPGA architecture with couple of enhancements mentioned above.

References

- [1] Moore, Gordon E. Cramming more components onto integrated circuits. Proceedings of the IEEE 86.1 (1998): 82-85.
- [2] Luu, Jason, et al. VTR 7.0: Next generation architecture and CAD system for FPGAs. ACM Transactions on Reconfigurable Technology and Systems (TRETs) 7.2 (2014): 6.
- [3] Betz, Vaughn, and Jonathan Rose. VPR: A new packing, placement and routing tool for FPGA research. Field-Programmable Logic and Applications. Springer Berlin Heidelberg, 1997.
- [4] Betz, Vaughn, Jonathan Rose, and Alexander Marquardt. Architecture and CAD for deep-submicron FPGAs. Vol. 497. Springer Science & Business Media, 2012.
- [5] Kirkpatrick, Scott, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. science 220.4598 (1983): 671-680.
- [6] VPR website-http://www.eecg.utoronto.ca/vpr/utfal_ex1.html
- [7] ROMEO, F., VINCENNELLI AK SANGIOVANNI, and MD HUANG. An efficient general cooling schedule for simulated annealing. PROCEEDING OF IEEE INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN, 1986.
- [8] Xu, Yonghong, and Mohammed AS Khalid. QPF: efficient quadratic placement for FPGAs. Field Programmable Logic and Applications, 2005. International Conference on. IEEE, 2005.

- [9] P.Maidee C.Ababei and K.Bazargan. Time-driven partitioning-based placement for island style fpgas.IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 24:395–406,2005
- [10] Farooq, Umer, Zied Marrakchi, and Habib Mehrez. FPGA architectures: An overview. Tree-based Heterogeneous FPGA Architectures. Springer New York, 2012. 7-48.
- [11] Luu, Jason. A Hierarchical Description Language and Packing Algorithm for Heterogenous FPGAs. Diss. University of Toronto, 2010.
- [12] Marquardt, Alexander Sandy, Vaughn Betz, and Jonathan Rose. Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density. Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays. ACM, 1999.
- [13] Bacon, David F., Rodric Rabbah, and Sunil Shukla. FPGA programming for the masses. Communications of the ACM 56.4 (2013): 56-63.
- [14] Cheng, Chih-Liang Eric. RISA: accurate and efficient placement routability modeling. Proceedings of the 1994 IEEE/ACM international conference on Computer-aided design. IEEE Computer Society Press, 1994.
- [15] W. Swartz and C. Sechen, New Algorithms for the Placement and Routing of Macro Cells, ICCAD, 1990, pp. 336 - 339.
- [16] J. Lam and J. Delosme, Performance of a New Annealing Schedule, DAC, 1988, pp. 306 - 311.
- [17] ROMEO, F., VINCENNELLI AK SANGIOVANNI, and MD HUANG. An efficient general cooling schedule for simulated annealing. PROCEEDING OF IEEE INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN, 1986.
- [18] Shi, Xiaoyu. FPGA Placement Methodologies: A Survey. Dept. of Computing Science, University of Alberta (2009).

- [19] Chen, Gang, and Jason Cong. Simultaneous timing driven clustering and placement for FPGAs. *Field Programmable Logic and Application*. Springer Berlin Heidelberg, 2004. 158-167.
- [20] Xu, M., Gary Gréwal, and Shawki Areibi. StarPlace: A new analytic method for FPGA placement. *INTEGRATION, the VLSI journal* 44.3 (2011): 192-204.
- [21] Gort, Marcel, and Jason H. Anderson. Analytical placement for heterogeneous FPGAs. *Field Programmable Logic and Applications (FPL)*, 2012 22nd International Conference on. IEEE, 2012.
- [22] Kim, Myung-Chul, Dong-Jin Lee, and Igor L. Markov. SimPL: An effective placement algorithm. *Computer-Aided Design of Integrated Circuits and Systems*, *IEEE Transactions on* 31.1 (2012): 50-60.
- [23] Lin, Tzu-Hen, Pritha Banerjee, and Yao-Wen Chang. An efficient and effective analytical placer for FPGAs. *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013.
- [24] Paul, Sudipta, Rana Das, and Pritha Banerjee. A study on detailed placement for FPGAs. *VLSI Systems, Architecture, Technology and Applications (VLSI-SATA)*, 2015 International Conference on. IEEE, 2015.
- [25] Kleinhans, Jürgen M., et al. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *Computer-Aided Design of Integrated Circuits and Systems*, *IEEE Transactions on* 10.3 (1991): 356-365.
- [26] Chu, Chris. "Placement." *Electronic Design Automation: Synthesis, Verification, and Testing* (2007): 635-684.
- [27] VPR User's Manual 7.0 -<https://github.com/verilog-to-routing/vtr-verilog-to-routing>
- [28] Luu, Jason, et al. VTR 7.0: Next generation architecture and CAD system for FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 7.2 (2014): 6.

Appendix A

Acronyms

Care has been taken in this thesis to minimize the use of acronyms, but this cannot always be achieved. This appendix contains a table of acronyms and their meaning.

A.1 Acronyms

Table A.1: Acronyms

Acronym	Meaning
CLB	Configurable Logic Block
BLE	Basic Logic Element
LUT	Look-Up Table
SA	Simulated Annealing
AP	Analytical Placement