2005

Efficient quadratic placement for FPGAs.

Yonghong Xu
University of Windsor

Efficient Quadratic Placement for FPGAs

By

Yonghong Xu

A Thesis

Submitted to the Faculty of Graduate Studies and Research through the

Department of Electrical and Computer Engineering in Partial Fulfillment

of the Requirements for the Degree of Master of Applied Science at

The University of Windsor

Windsor, Ontario, Canada

2005

Your file Votre référence
ISBN: 0-494-11520-3
Our file Notre référence
ISBN: 0-494-11520-3

1027299

Abstract

Field Programmable Gate Arrays (FPGAs) are widely used in industry because they can implement any digital circuit on site simply by specifying programmable logic and their interconnections. However, this rapid prototyping advantage may be adversely affected because of the long compile time, which is dominated by placement and routing. This issue is of great importance, especially as the logic capacities of FPGAs continue to grow.

This thesis focuses on the placement phase of FPGA Computer Aided Design (CAD) flow and presents a fast, high quality, wirelength-driven placement algorithm for FPGAs that is based on the quadratic placement approach.

In this thesis, multiple iterations of equation solving process together with a linear wirelength reduction technique are introduced. The proposed algorithm efficiently handles the main problems with the quadratic placement algorithm and produces a fast and high quality placement. Experimental results, using twenty benchmark circuits, show that this algorithm can achieve comparable total wirelength and, on average, 5X faster run time when compared to an existing, state-of-the-art placement tool.

This thesis also shows that the proposed algorithm delivers promising preliminary results in minimizing the critical path delay while maintaining high placement quality.

Acknowledgements

I would like to express my sincere appreciation to my supervisor Dr. Mohammed A. S. Khalid for his guidance and encouragement. He introduced me to this interesting research area and guided me throughout my thesis with great patience. I would like to thank the members of my thesis committee, Dr. C. Chen and Dr. W. Altenhof for their advice regarding the research process and their assistance in the preparation of this thesis. Here, I would also like to thank Dr. S. Erfani and Dr. K. Tepe for their help during my Masters program at the University of Windsor.

I am grateful to my wife who accompanied me all these years. Her understanding and support made things easier for me. I am indebted to my parents. They gave me the opportunity to pursue my own life. I would not have reached this milestone in my life without their continuous encouragement and support.

The financial support provided by the Ontario Graduate Scholarship Program (OGS) is gratefully acknowledged.

Table of Contents

List of Figures

List of Tables

Number *Page*

Abbreviations

ASIC:	Application-Specific Integrated Circuit
BLE:	Basic Logic Element
CAD:	Computer-Aided Design
CG:	Conjugated Gradient
CLB:	Configurable Logic Block
CPLD:	Complex Programmable Logic Device
FPGA:	Field-Programmable Gate Arrays
HDL:	Hardware Description Language
HPWL:	Half-Perimeter WireLength
LUT:	Look Up Table
MCNC:	Microelectronics Centre of North Carolina
MPGA:	Mask-Programmed Gate Arrays
NP:	Non-deterministic Polynomial-time
PLD:	Programmable Logic Device
QPF	Quadratic Placement for FPGAs
VLSI:	Very Large Scale Integration
VPR:	Versatile Placement and Routing tool for FPGAs

Chapter 1

Introduction

Advances in microelectronics and VLSI (Very Large Scale Integration) circuits have contributed to the tremendous growth and pervasiveness of the global electronics industry in the past few decades. One rapidly growing area of microelectronics is Field Programmable Gate Arrays (FPGAs). FPGAs are widely used for implementing digital circuits because they offer moderately high levels integration and the ability to program and reprogram the chip in the field by the end user. A Computer Aided Design (CAD) tool suite is needed for mapping user designs on to the FPGA chip. This mapping CAD flow consists of a series of steps: logic synthesis, technology mapping, placement and routing. This subject of this thesis is efficient quadratic placement for FPGAs.

1.1 VLSI Design Styles

Usually there are several design styles that can be considered for implementing a digital system. As is illustrated in Figure 1.1, each design style has its own merits and shortcomings [1].

In Full Custom design, major parts of chip layout are designed from scratch without utilizing any cell libraries. This can create compact and power/area efficient chips. However, the development cost of such a design style is becoming prohibitively high and this is usually used only for the design of high-volume products such as memory chips, high- performance microprocessors and FPGA masters.

Standard Cell and Masked Programmed Gate Arrays (MPGA) are semi-custom designs. Users work one some pre-developed cells to implement their circuits. In Standard Cell design, all of the commonly used logic cells are developed, characterized, and stored in a standard cell library. Once the circuit is mapped, these cells are arranged in horizontal rows within the chip boundary. The spaces between these rows are used to implement the interconnections between the cells. An MPGA device consist of an array of uncommitted elements (usually rows of transistors), and most of the mask layers are pre-defined by the manufacturer. Users define the final metal layers to connect the transistors in the array, so as to implement their designs.

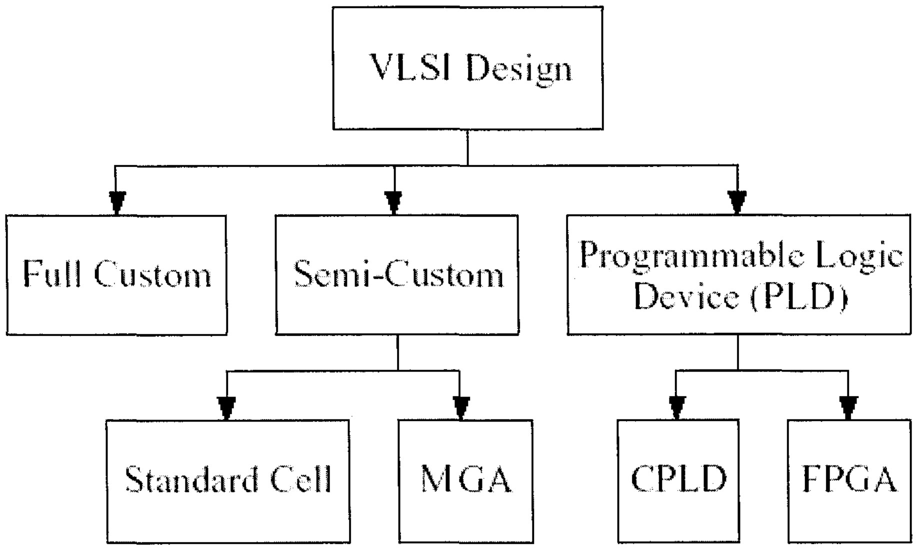


Figure 1.1 VLSI Design Styles

Programmable Logic Device (PLD) design style provides users with pre-fabricated array of programmable logic and interconnections [2]. Users can configure the final logic structure of the device by themselves so that no fabrication step is need in this design style. The difference between Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays (FPGAs) is their structure and logic resources. CPLDs mainly consist of two levels of programmable logic; an AND plane and an OR plane, and a wide number of inputs. While FPGAs have a more general structure that allows high logic capacity and flexibility (details of FPGA architectures are presented in Chapter 2).

Since no physical manufacturing step is necessary for customizing the FPGA chip, a functional sample can be obtained almost as soon as the design is mapped into a specific FPGAs chip. This gives FPGAs significant advantages over the custom designs.

1.2 Motivation

To implement a digital circuit with FPGAs, a set of CAD tools are needed to map a user design into bitstream file that are required to configure the FPGA chip. These tools first transform the circuit description (expressed using hardware description language such as VHDL or Verilog) into a netlist of technology-mapped logic blocks and their connections. Then placement and routing steps are done so that these logic blocks are assigned to physical locations on the FPGA and interconnected correctly. Finally this location and connection information is transformed in to bitstream file that is downloadable for FPGAs. This thesis focuses on the placement part of this process.

Quality and speed are two key metrics for evaluating the goodness of a CAD tool. Now that recently announced FPGAs contain the equivalent of 40-million gates, the compile time is more important than ever. For some large circuits, this CPU compile time, which is dominated by placement and routing, can be in the order of tens of CPU hours.

For a placement tool, high quality means that we can implement more digital logic in an FPGA chip and may also achieve a better circuit performance by utilizing the placement tool. Fast speed means we can get the desired placement within shorter time so that the turn around time of our products will be reduced as well. Unfortunately, the current placement tools that provide high quality solutions require a large amount of CPU time. While other tools with fast speed give poor quality. There is a great need for placement tool that run in a reasonable amount of CPU time while still generating high quality solutions.

1.3 Research Approach

Our research goal was to find a fast algorithm for placement that produces high quality result. We create a fast placement tool based on quadratic technique and integrate it into Versatile Placement and Routing tool for FPGAs (VPR) [3][4], which is a well known, high quality placement and routing tool for FPGAs. We evaluate our placement tool with respect to VPlace (the placement part of VPR) based on both runtime and placement quality by running both algorithms on the same computation platform with the same suite of benchmark circuits and the same FPGA architecture.

VPlace is based on the simulated-annealing based algorithm that is widely used in academia and industry. It starts with a random initial placement and improves it by large number of swaps and moves of nodes. This algorithm can achieve the best placement quality with a large amount of time. It spends a lot of time on examining the poor initial placement, which does not contribute significantly to the final result.

Our efficient placement algorithm uses the quadratic technique to create a reasonably good placement within a very short time, and only uses simulated annealing algorithm to refine the final placement. It combines the fast speed of quadratic based placement algorithm and the high quality of simulated annealing based algorithm.

1.4 Thesis Organization

The thesis is organized as follows:

Chapter 2 contains an introduction to FPGA architectures, generic CAD procedure for FPGA-based designs. Then we give the definition of FPGA placement problem. In this chapter we also describe the previous works on FPGA placement, as well as some related topics of quadratic technique.

In chapter 3, we first describe the common problems with the quadratic placement method. Then we introduce our proposed placement algorithm and explain how we deal with these problems in details. We also give a time complexity analysis of our proposed algorithm.

Chapter 4 presents the results obtained from running our tool on a suite of large benchmark circuits, using a simple and general FPGA architecture. We show the effects of key techniques in our proposed algorithm on placement quality, by presenting and analyzing relevant experimental data. We also provide a comparison between our tool and VPR, which is a well-known, high-quality placement and routing tool for FPGAs.

Chapter 5 highlights the key results from our research, and proposes possible directions for future research in this area.

Chapter 2

Background and Previous Work

This chapter presents the background of placement for FPGAs and the previous work done in this area. The FPGA architecture is described in section 2.1. The general FPGA design flow is discussed in Section 2.2 and the definition of FPGA placement problem is given in Section 2.3. Some placement algorithms developed for FPGAs are summarized in section 2.4. Essentials of quadratic placement and related techniques are discussed in section 2.5. Finally section 2.6 gives an overview of well-known quadratic algorithms used in ASIC placement.

2.1 FPGA Architecture

An FPGA is a completely re-configurable logic chip. Similar to traditional hardwired gate arrays, the chip consists of an array of logic elements. In the traditional gate arrays, these gates are specified and interconnected at the manufacturing stage. The FPGA differs in that it can be programmed, and re-programmed by the users. Although there are a wide variety of architectures for commercial FPGAs, all FPGAs are composed of three fundamental components: logic blocks, I/O blocks and programmable routing resources. A circuit is implemented in an FPGA by programming each of the logic blocks to implement a small part of the logic of the circuit, and the I/O blocks serve as the input and output pads of the circuit. The programmable routing resources are used to connect these logic blocks and I/O blocks together as required by the circuit. Figure 2.1 shows the FPGA architecture model used in this thesis. This FPGA architecture is also used by VPR. And many researchers and CAD tools also employ this model as their prototype [3][5].

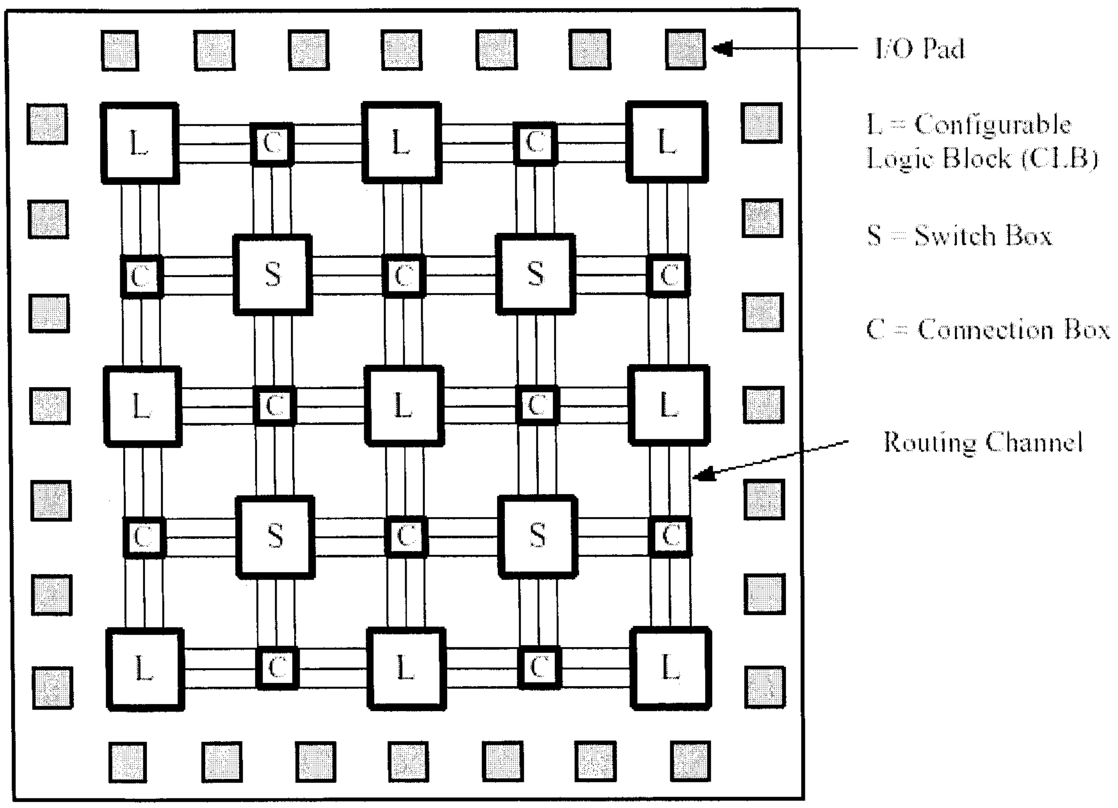


Figure 2.1 FPGA Architecture

As illustrated in Figure 2.1, the I/O blocks are assigned around the chip edges. And the logic blocks are scattered on the chip region like islands. Between the logic blocks, are the routing resources, which include switch box, connection box and routing segments. A connection box can be programmed to connect to a CLB to routing channels. A switch box is a switch matrix that is used to connect wires in one channel to the wire in another channel. Figure 2.2 shows how the connection box and switch box are used to connect the logic blocks as required by the circuit.

Logic blocks are constructed by using Look Up Tables (LUTs) and D flip-flops. Usually, we refer to a LUT and a D flip-flop combination as a Basic Logic Element (BLE) and a logic block can contain one or more such BLEs. Figure 2.3 shows the structure of logic blocks. In this thesis, we will use logic blocks that contain one BLE.

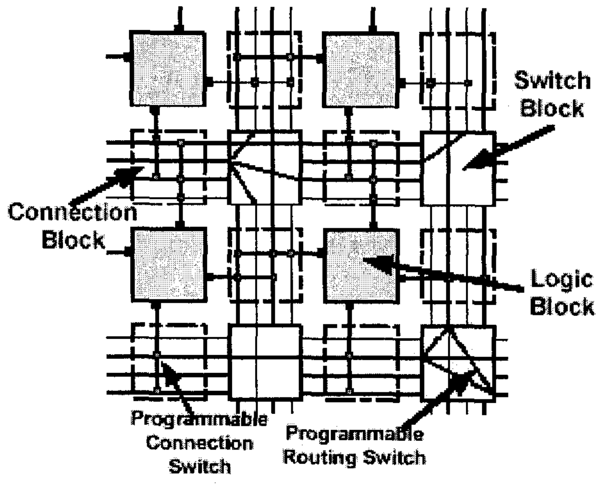


Figure 2.2 Programmable Connection Box and Switch Box

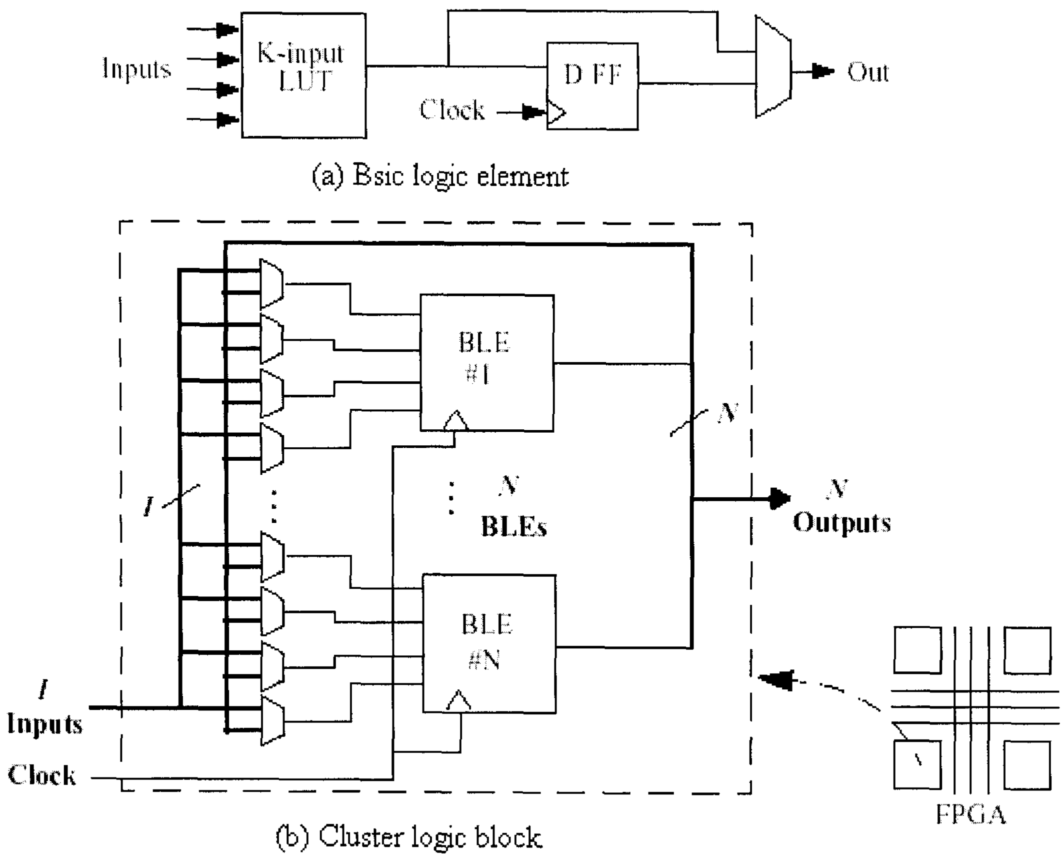


Figure 2.3 Structures of (a) Basic Logic Element and (b) Cluster Logic Block

2.2 FPGA Design Flow

To implement a circuit in a modern FPGA chip, a sequence of mapping steps are needed. Typically users of FPGAs describe the circuit using a hardware description language or schematic input. The CAD tools take this description and compile it into a bitstream file programs the FPGA chip to implement the desired circuit. The FPGA design flow is shown in Figure 2.4.

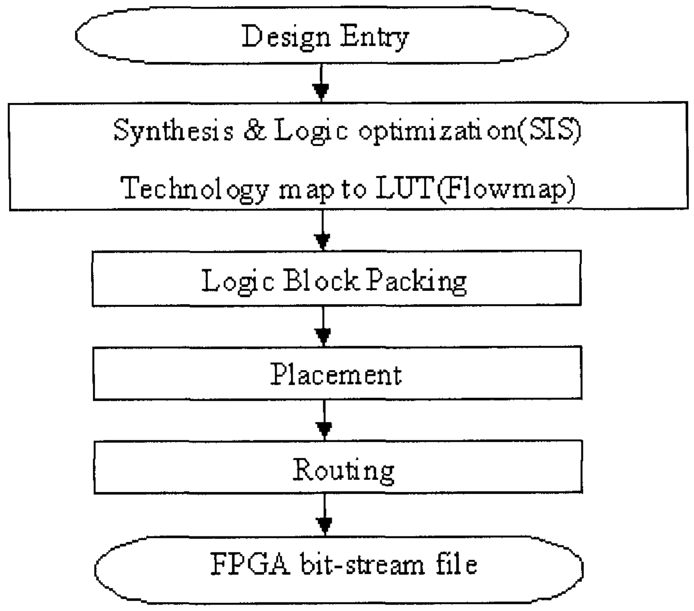


Figure 2.4 FPGA Design Flow

The design entry is the input circuit that the users developed by using a hardware description language (HDL) such as VHDL or Verilog. Other methods are using a state machine description language or a schematic.

The first stage of the design flow first converts the circuit description into a netlist of basic gates. Technology-independent logic optimization removes the redundant logic wherever is possible [6]. Then the optimized netlist of basic gates is mapped to look-up tables.

Logic block packing combines the LUTs and registers into Configurable Logic blocks (CLBs). In this phase, connected LUTs are packed together so that the number of signals to be routed between CLBs is minimized. A CLB might contain one or more BLEs depending on the architecture of the FPGA [3].

The placement process determines the physical location that every logic block of the circuit should be assigned to. The optimization goals of placement are to place the connected logic block close together so that the required wire length is minimized. Sometimes other objectives such as wiring density [7] and circuit delay [8][4] are also considered.

Routing process determines how the routing resources are programmed to connect all the logic blocks as required by the circuit. FPGA routers can be divided into combined global-detailed routers [9], witch determines a complete routing path in one step, and two step routers, which first perform global routing and then detailed routing.

Once the routing process is completed, a CAD tool will create a bitstream file according to the target FPGA architecture. When this file is downloaded to the target FPGA, it configures the logic blocks and routing resources of the target FPGA to implement the desired digital circuit.

2.3 Definition of FPGA Placement Problem

Placement problems can be formulated as optimization and constraint satisfaction problems. The input netlist that represents the nodes and their interconnections in a logic circuit is described by a hyper-graph. The vertices of the hyper-graph represent circuit elements (logic blocks for FPGAs). Placement can be defined as follows [10]:

Given an electrical circuit consisting of modules (logic blocks) with predefined input and output terminals and interconnected in a predefined way, construct a layout (physical location) indicating the position of the modules so that the estimated wire length and layout area are minimized or other constraints are met.

More formally, the placement problem can be expressed as [11]:

- Given: A hyper-graph G = (V, E) representing the circuit, where V is the set of vertices (logic blocks), and E is the set of edges (nets), with edge weight $w(e) \in (0,+\infty)$ for each $e \in E$; an FPGA grid of size $r \times s$, where $r,s \in N$, and $r \times s \geq \sqrt{n}$, n is the number of nodes.
- Find: All placement mappings $p : V \rightarrow [1,r] \times [1,s]$ of blocks to physical locations on the FPGA grid.
- Minimize: A cost function $c(p)$.

The most commonly used cost function for FPGA placement is the total wire length required to complete the routing. Because the cost of the device is proportional to the amount of silicon required to implement it. If we can minimize total wire length used to route the circuit, the area required for the circuit will be minimized. Hence we can use a smaller (and cheaper) FPGA to implement the circuit. A placement that strives to minimize the total wire length is referred to as wirelength-driven placement. There are also other objectives that can be added to the cost function. For example, placement can be done to minimize the length of a critical path to meet some timing constraints, referred to as timing-driven placement [8][4]. Or to balance the wire density across the FPGA device, referred to as routability-driven placement [7].

In this thesis, we use the wirelength minimization as our cost function. This is the first step in FPGA placement research. Timing-driven placement and routability-driven placement algorithm also strive to minimize the total wirelength as well.

2.3.1 Half-Perimeter Bounding Box Wirelength Model

As we described earlier, placement stage only tries to determine the physical location for every logic element of the circuit. We do not have any ideas about how these elements are to be connected in the routing stage. That means that we do not know the actual total wirelength of the current placement. We have to use some approximations to estimate the total wirelength and use this as a metric to evaluate any given placement. There are various approximation techniques available [12][13][14][15] and the Half-Perimeter Wire Length (HPWL) model is the most widely used method [10].

Figure 2.5 shows how the half-perimeter wirelength model works. In Figure 2.5, Net(i) has 8 terminals and is placed on a FPGA chip. The bounding box is defined as the smallest rectangle that covers the net.

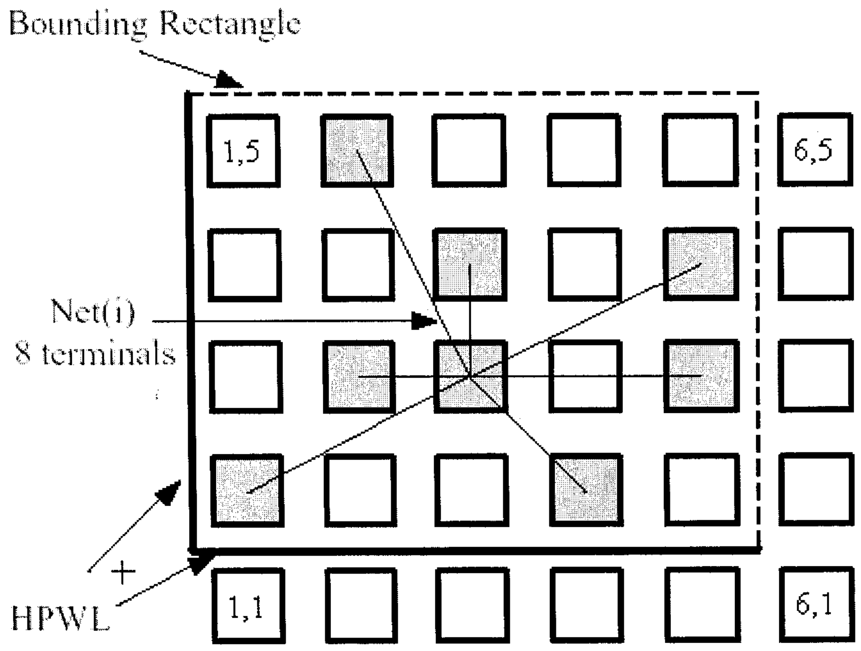


Figure 2.5 Half-Perimeter WireLength Model

The half-perimeter wire length is defined as:

$$HPWL_{net(i)} = \{Max(x_b) - Min(x_b) + 1) + Max(y_b) - Min(y_b) + 1\},\ b \in net(i)$$

In this case, a 5x4 bounding box covers the 8-terminal Net(i), the HPWL=(5+4)=9.

When the number of terminals of the net is less than or equal to three, HPWL is an accurate estimation of the actual wire length. Otherwise, HPWL underestimates the wire length required to connect all terminals of the net. A q(i) factor [16][3] is introduced to compensate for the fact that the HPWL model underestimates wirelength in case number of net terminals is greater than three. For nets with three or fewer terminals, q(i) is 1. It slowly increases to 2.79 for nets with 50 terminals, as shown in table 2.1. For nets that have more than 20 terminals, the value of q(i) linearly increases as:

q(i) = 2.7933 + 0.02616x(NumOfTerminals – 50) [3] (2.1)

Table 2.1 Compensation Factors for Net with Less than 50 Terminals

#Term	q(i)	#Term	q(i)	#Term	q(i)	#Term	q(i)	#Term	q(i)
1	1.0	11	1.4974	21	1.9288	31	2.2646	41	2.5610
2	1.0	12	1.5455	22	1.9652	32	2.2958	42	2.5864
3	1.0	13	1.5937	23	2.0015	33	2.3271	43	2.6117
4	1.0828	14	1.6418	24	2.0379	34	2.3583	44	2.6371
5	1.1536	15	1.6899	25	2.0743	35	2.3895	45	2.6625
6	1.2206	16	1.7304	26	2.1061	36	2.4187	46	2.6887
7	1.2823	17	1.7709	27	2.1379	37	2.4479	47	2.7148
8	1.3385	18	1.8114	28	2.1698	38	2.4772	48	2.7410
9	1.3991	19	1.8519	29	2.2016	39	2.5064	49	2.7671
10	1.4493	20	1.8924	30	2.2334	40	2.5356	50	2.7933

In this thesis, we use HPWL model to estimate the wirelength and the total wirelength is computed as

$$Total\_Wirelength = \sum_{i=1}^{N} q(i) \times HPWL_{net(i)} \qquad (2.2)$$

2.4 Placement Algorithms for FPGAs

Placement is a Non-deterministic Polynomial-time (NP) complete [38] optimization problem. If given the right information, the exact solution of placement problem cannot be verified in polynomial time. The time complexity for obtaining an optimal solution for placement of n modules is O(n!). Except for very small circuits, there is no efficient way to compute an exact solution using a computer program. Approximation methods, also referred to as heuristics, are used to obtain a good solution in reasonable time. Currently popular placement algorithms can be divided into three major classes: partitioning based placement, quadratic placement, and simulated annealing based algorithm.

Partitioning based algorithms [17][18] repeatedly divide the given circuit into densely connected sub circuits by applying partitioning techniques, such as Kernighan-Lin (KL) [19] and Fiduccia-Mattheyses (FM) [20] partitioning algorithms. These algorithms partition the given circuits into sub circuits and minimize the interconnection between the different partitions. Since the interconnection usually means the wiring in the circuit, the partitioning algorithms minimize the total wirelength as well. Recently, more partitioning-based placement algorithms were introduced to solve the placement problem [21][22][23].

In quadratic placement algorithms, we build up linear equations from the interconnectivity of the input circuits and try to minimize the objective function by solving the equations. This technique reduces the placement problem to the solution of a system of linear equations and is widely used for ASIC placement.

Simulated annealing based placement algorithms start with an initial (legal) placement and repeatedly modify it in search of a cost reduction. If a modification results in a reduction in cost, the modification is accepted; otherwise it is accepted or rejected based on a number of factors.

Initially most of these algorithms were developed for ASIC placement. After the emergence of FPGAs in mid-1980s, some of these algorithms were modified for FPGA placement. In the following sections, we will introduce three of these algorithms. They are: the VPR placement algorithm [3][4] which is based on simulated annealing; PPFF placement algorithm [23] which is based on partitioning algorithm; and Ultra-Fast Placement algorithm [5] which is a placement algorithm that uses a combination of simulated annealing and clustering techniques.

2.4.1 VPR Placement Algorithm

As we mentioned above, the VPR placement algorithm [3][4] is based on simulated annealing algorithm. Simulated annealing is a well-developed and widely used algorithm for solving combinatorial optimization problems, including those used in CAD for VLSI physical design automation. As the name suggests, this algorithm mimics the annealing process used to gradually cool molten metal to produce high quality metal structures. An ideally annealed crystal should be in the lowest energy state, which corresponds to an optimum configuration in a combinatorial optimization problem.

```
X = Initial_Random_Placement();
T = Set_Initial_Temperature();  /* T=T0 */
Dlimit = Set_Initial_Range_Limit();       /* Dlimit = whole chip */
while (Exit_Criterion() == false) { /* annealing not done yet */
    while (Inner_Loop_Criterion() == false){/*work per temperature not done yet*/
        Xnew = Generate_Move(X, Dlimit);
        /* return a new configuration generated incrementally from previous one */
        /* by random pairwise exchange or translation within range limit */
        ΔC = Cost(Xnew) - Cost(X); /* calculate change in cost */
        r = Get_Random_Number(0,1);
        /* r = random number uniformly distributed between 0 and 1 */
        if (r < e -ΔC/T)
        X = Xnew; /* update current placement */
        /* always accept move (p=1) if it improves placement (ΔC < 0) */
        /* accept "bad" moves (ΔC > 0) with probability p = e -ΔC/T */
        /* when T is large, all bad moves likely to be accepted, */
        /* when T is small, only bad moves with small ΔC likely to be accepted */
    } /* end inner loop */
  /* exploration at current temperature complete */
  T = Update_Temperature(α, T); /* T = αT */
  Dlimit = Update_Range_Limit(Dlimit);
} /* end outer loop */
/* annealing complete, X = final placement solution */
```

Figure 2.6 Pseudo-codes for Basic Simulated Annealing Placement-based Algorithm

Figure 2.6 shows the pseudo-code for the simulated annealing algorithm. The reference [24] contains a detailed description of the basic algorithm and the various cost functions used for the different types of placement problems.

A simulated annealing based placement algorithm initially places logic blocks and I/Os of the circuit randomly on the FPGA chip. The temperature (T) is used to determine the probability of whether configurations that reduce the quality of the placement will be accepted. Parameter D controls the distance that a logic block is to be moved. Initially D is set such that a logic block can be moved to any location in the entire chip area. This indicates that initially all the logic blocks can be selected and moved to any other place even if the moves deteriorate the quality of placement. Given this random initial placement, a source logic block is chosen randomly. Then, a target location is chosen at random for this logic block within the displacement range specified by D. If the target location is occupied, then the target logic block is swapped with the source logic block and the cost of the swap is evaluated. And if the target location is empty, the source logic block is placed in the target location and the new placement is evaluated. If the new cost is less than the cost of the previous placement, the move or swap is accepted. Otherwise, the move or swap is only accepted with probability $e^{-\Delta C/T}$, where ΔC is the change in placement cost duce to the move or swap, and T is the current temperature. As the placement quality improves, the temperature is gradually reduced, make it less likely for bad moves and swaps that degrade the placement will be accepted. Eventually, the value of T will be reduced to a low value such that only the moves and swaps that improve the placement quality will be accepted, making the heuristic greedy at that point. The parameter T is what permits probabilistic hill climbing to take place and helps the placement solution avoid being trapped in local minima.

As we can see from Figure 2.6, besides the cost function that defines the basic way to evaluate a placement, there are also some crucial details that affect the annealing process. They are: the rate at which the temperature is reduced (called the temperature update factor), the number of configurations to explore at each temperature (known as the inner loop criterion, or InnerNum), the exit criterion by which the annealing algorithm terminates, and the behavior of the range limiting mechanism These parameters, together with the cost function, determine the quality of the simulated annealing algorithm. When VPR employs the simulated annealing algorithm for FPGA placement, a new cost function and a dynamic adaptive annealing schedule is introduced [3]. It includes some of the features from the work done on annealing schedules by Huang et al. [25], Lam and Delssme [26], and Swartz and Sechen [27]. It also implements a novel temperature update scheme and stopping criterion, together with a bounding box wirelength cost function.

In VPR placement algorithm, the initial temperature T_0 is set to 20 times the standard deviation in cost after a set of N_{blocks} pair-wise moves have been attempted, where N_{blocks} is the total number of logic blocks and I/O pads in the circuit. The temperature T_0 is high enough to ensure that almost all early moves and swaps are accepted. The number of new configurations evaluated at each temperature T is set to:

$$MovesPerT = InnerNum \times (N_{blocks})^{4/3} \qquad (2.3)$$

where the scaling factor InnerNum, is set to 10 by default, give a best quality at reasonable CPU run time.

VPR placement algorithm uses a dynamic adaptive annealing schedule. Most of the annealing parameters are updated according the acceptance rate of the moves and swaps at the current temperature. Table 2.2 shows the how the temperature update factor, □is automatically determined according to the acceptance rate ate the current temperature stage. The temperature is reduced in a way such that if there is little change in cost, the temperature is reduced by a larger fraction. $T_{new} = \alpha \cdot T_{old}$ Initially, when the temperature is very high and almost all moves are accepted, these moves and swaps do not make much difference. The temperature update factor, α, is set to a very low value, which makes the temperature drops very fast in this stage. When the annealing process goes on, some moves are accepted and some are not. In this case, the temperature update factor, α, is set to a larger value so that the placement can be thoroughly explored at the temperature. In final stage, since most moves are rejected and the placement does not change a lot, the temperature will again reduce quickly.

Table 2.2 VPR Temperature Update Schedule

Fraction of Moves Accepted (R_{accept})	Temperature Update Factor (α)
R_{accept} >0.96	0.5
0.8<R_{accept}<=0.96	0.9
0.15<R_{accept}<=0.08	0.95
R_{accept}<=0.15	0.8

It is shown in [27][26] that the most desirable annealing schedule is one that keeps the acceptance rate of moves near 0.44 as long as possible. VPR placement algorithm also employs this by utilizing the value of the acceptance rate to control the range limiter D_{limit}, as:

$$D_{\lim it}^{new} = D_{\lim it}^{old} \times (1 - 0.44 + R_{accept}) \tag{2.4}$$

and $D_{\lim it} \in [1, \max\_FPGA\_width]$

If the acceptance rate is less than 44%, the range will be shrunk. Otherwise it is expanded. Typically, this range limit covers the whole chip at the beginning, and gradually shrinks to1 at the end of the annealing.

The VPR placement simulated annealing schedule exits when the temperature falls below a certain fraction of the average cost per net.

$$T_f < \frac{0.005 \times Cost}{N_{nets}} \tag{2.5}$$

Even with a good annealing schedule, a large number of potential swaps will be evaluated during the placement. And the most computationally expensive part of evaluating a swap is computing the change of cost ΔC. VPR placement algorithm also developed an incremental bounding box evaluation technique to speed up this computation. For each net, a data structure is designed not only contain the coordinates of the of the four sides of the net bounding box, but also the number of logic blocks that lie on each side. This information is used to determine the new net bounding box after a swap by only examining the logic blocks that moved. The net cost is recomputed only when the terminal moved is the only net terminal on a side of the bounding box and it is moved toward the bounding box center. This technique, on average, yields a five-fold speedup over ten large MCNC benchmark circuits.

2.4.2 PPFF: Partitioning-based Placement for FPGAs

Partitioning-based placement algorithms are based on graph partitioning algorithms such as FM and KL. An FPGA is divided into two or more sub-regions, and a circuit partitioning algorithm is applied to determine which logic block should go to which sub-region to minimize the number of cuts in the nets that connect the blocks between partitions, and leave the highly connected blocks in one partition. This recursive process is repeated until each partition is small enough to be finally placed. The partitioning based placement algorithms are referred to as top-down algorithms. By partitioning the problem into sub regions, a drastic reduction in search space can be achieved so that they can achieve very high speed. However, cut size is an indirect approach to wirelength, the placement quality is not guaranteed. Other techniques are required to further improve the quality.

PPFF [23] is a partitioning based placement algorithm that employs both hMetis [28] partitioning program and VPR simulated annealing techniques to achieve a final placement for FPGAs.

PPFF placement is done by recursively partitioning the circuit by using hMetis while maintaining a tight connection between the circuit graph and the placement. Recursive partitioning is done when each leaf in the hierarchical partition tree contains less than a constant number of logic blocks (e.g. six). If some of the partitions may have more logic blocks than they can accommodate after the partitioning, the overlaps are removed by using a greedy technique that moves the logic blocks to the closest available partition. Finally, simulated annealing is applied to refine the placement.

The key idea of PPFF is that it applies a net terminal alignment technique during the partitioning process. Since partitioning algorithms partitioning the circuit into sub-circuit only according to the interconnection of the circuit and do not care about the actual position of the logic blocks, it will results in a fact that after the placement is done, although the cut size of each level of partitioning is minimized, the total wirelength is not. The net terminal alignment technique is used to solve this problem. When partitioning one part into sub-parts, the logic blocks in other partitions is considered so that one logic block tends to stay in the partition that minimize the wirelength of the nets to which it is connected. Figure 2.7 illustrates this technique.

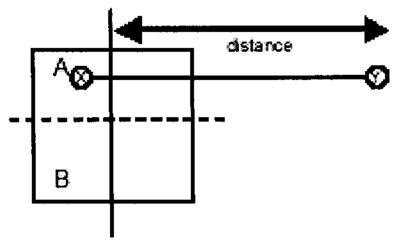


Figure 2.7 Illustration of The Terminal Alignment Technique

In Figure 2.7, the logic block X is in the current partition that is to be further partitioned into parts A and B. Y is another logic block that is connected to X and was previously placed in the upper part (aligned with A). In this case, the partitioning algorithm does not really care about the target position X will go as long as the cut size between A and B is minimized. The net terminal alignment technique will tell the partitioning algorithm to place logic block X in partition A to minimize the wirelength.

2.4.3 Ultra-Fast Placement

Unlike partitioning based algorithm, ultra-fast placement [5] is a bottom up algorithm. It reduces the complexity of placement problem by clustering closely connected logic blocks into multiple levels of clusters.

Figure 2.8 illustrates the abstract view of multi-level clustering. In level 0, the logic blocks in the input circuit are clustered into a according to their interconnections in the circuit. And in the each upper level, clusters at the previous level are grouped together into a larger cluster.

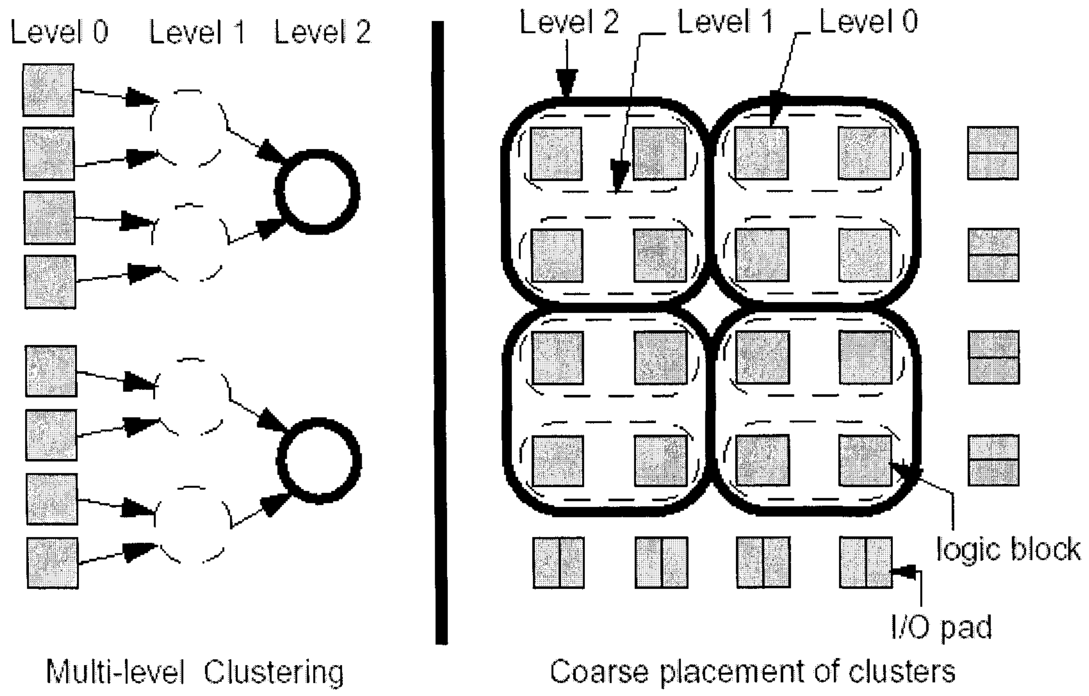


Figure 2.8 Abstract View of Multi Level Clustering

The number of clustering levels, and the size of the cluster at each level can be varied to allow the tradeoff of compile time and quality. As the size of the clusters increases, the placement problems become simpler because more is hidden, but there is lees accurate representation of the netlist and therefore lower quality may result.

The ultra fast placement algorithm is relative simple: It begins with a multi-level, bottom-up clustering of logic blocks based on their connectivity. Once all the required clustering is done, a two-level placement is performed at each level of the hierarchy: an initial constructive placement followed by an iterative improvement step using simulated annealing. After the placement finishes in the current level, the clusters at the current level are decomposed. This process is repeated for the next level until all levels are placed.

2.5 Quadratic Placement Techniques

As we discussed in Section 2.4, quadratic placement is one of the main algorithms used for ASIC placement. Unlike other algorithms, quadratic algorithm has not been investigated for FPGA placement.

Quadratic placement algorithms use squared wire length as the objective function and try to minimize it by solving linear equations. Although quadratic wire length is only an indirect measure of the linear wire length, quadratic placement can minimize the quadratic wire length efficiently. As a result, it is widely used in ASIC placement [29][30][31][32]. The use of quadratic assignment in many applied areas can be traced to the paper [33]. The main concern with quadratic placement is that the positions of the nodes we get from the linear equation solver tends to locate in the center of the placement area with a large amount of overlap among nodes. To deal with this overlap problem, a bisection technique is used in [30] to recursively divide the circuit and adds more linear constraints to pull the nodes into center of corresponding partitions. Repelling forces are added in [34] for nodes sharing a net to maintain a target distance between them, and attractive forces are also introduced to pull nodes from the center to the sparse regions by some fixed dummy nodes. In [35], spreading forces are added to pull the nodes out of the dense regions. Another concern with the quadratic placement is its quality. Because the objective function of the quadratic placement is squared wire length, not linear wire length, its quality is not optimized. To alleviate this problem, [32] adds some linear aspects when building up the quadratic model to improve its quality and [35] uses half perimeter adjustment to improve its result. But since its main concern is to speed up the equation solving process of the quadratic placement, this technique is not well discussed. All the quadratic algorithms discussed above target standard cell ASIC placement.

2.5.1 Essentials of Quadratic Placement

A quadratic placer takes a hyper-graph netlist as its input and produces a placement of nodes on target chip such that the total squared wire length is minimized. Quadratic placement uses the following objective function

$$\Phi(x,y) = \frac{1}{2}\sum_{i,j} W_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2] \qquad (2.6)$$

Where x, y are the coordinates of a logic block of the netlist. W_{ij} is the weight of the edge that connects node (x_i,y_i) and node (x_j,y_j). Since the input of the quadratic placer is usually represented by a hypergraph, and two nodes can be connected by more than one net, we have to convert the hypergraph into a weighted graph first. Two models are used to convert the hypergraph into a graph. Clique model introduces an edge of weight 2/p between every pair of nodes incident to a p-pin net. While Star model created a new node in the center of its net. [35] has shown that the total squared wire length will be the same for any placement under either the star model or the clique model.

The objective function can be rewritten in matrix notation as:

$$\Phi(x,y) = \frac{1}{2}x^T Qx + d_x^T x + \frac{1}{2}y^T Qy + d_y^T y + const \qquad (2.7)$$

Where Q is an n x n symmetric matrix and d_x d_y are n-dimensional vectors.

Since the objective function can be separated into x y dimensions. Only one dimension is considered. Then we get

$$\Phi(x) = \frac{1}{2}x^T Qx + d_x^T x + const \qquad (2.8)$$

To find the minimum value of this objective function, we perform $\nabla\Phi(x) = 0$ and get the matrix equation:

$$Q \cdot x + d_x = 0 \qquad (2.9)$$

This is the quadratic equation that minimizes the total squared wire length of the placement. To obtain the matrix **Q** and vector **d**, let q_{ij} be the entry in row i and column j of matrix **Q**, and d_i be the ith element of vector **d**. For two movable nodes, the cost can be rewritten as $\frac{1}{2}W_{i,j}[x_i^2 + x_j^2 - 2x_i x_j]$. The first and second terms contribute $W_{i,j}$ to q_{ii} and q_{jj} respectively. The third term contributes $W_{i,j}$ to q_{ij} and q_{ji}. For a connection between a movable node and a fixed node the cost is $\frac{1}{2}W_{i,f}[x_i^2 + x_f^2 - 2x_i x_f]$. The first term contributes W_{if} to q_{ii}, the third term contributes to $-W_{if}x_f$ to the vector d_x at row i, and the second term becomes the constant part of expression (2.5) after derivative operation.

The matrix **Q** is positive definite if all movable nodes are connected to fixed nodes, i.e. I/O (Input/Output) pads, either directly or indirectly. This condition holds for all real circuits since each node of the circuit should be accessible from the outside of the circuit. So the matrix equation can be solved by non-stationary iterative methods [36].

2.5.2 Linear Equation Solver

Quadratic placement algorithm produces large sparse linear equations [29][46]. These large linear equations can be solved through the iterative methods, which use successive approximations to obtain more accurate solutions to the linear system at each step. There are two main types of iterative methods: stationary methods and non-stationary methods. The stationary methods can be expressed in the simple form $x^{(k)}=\mathbf{B}x^{(k-1)}+c$, where B and c are constant coefficient, neither of them depend on the iteration count k. The stationary methods are older, simpler to understand and implement, but usually not as effective, like the Jacobi method, the Gauss-Seidel method, the Successive Overrelaxation (SOR) method and the Symmetric Successive Overelaxation (SSOR) method [36].

Non-stationary methods differ from stationary methods in that the computations involve information that changes at each iteration. Typically coefficients are computed by taking inner products of residuals or other vectors arising form the iterative method. Non-stationary methods are a relatively recent development; their analysis is usually harder to understand and implement, but they can be highly effective, like Conjugated Gradient method, Generalized Minimal Residual (GMR) method, Quasi-Minimal Residual (QMR) method, etc [36].

For a real digital circuit, all the nodes of the circuit are connected to, directly or indirectly, I/O pads and matrix Q generated through quadratic model is positive definite [30]. The linear equations can be solved by some non-stationary methods. In our thesis, we apply Conjugated Gradient method, which is also used by many other quadratic based placement tools [30][29][45].

The Conjugate Gradient method is an effective method for symmetric positive definite systems. It is one of the oldest and best known of the non-stationary methods. The method proceeds by generating vector sequences of iterates (*i.e.*, successive approximations to the solution), residuals corresponding to the iterates, and search directions used in updating the iterates and residuals. Although the length of these sequences can become large, only a small number of vectors need to be kept in memory. In every iteration of the method, two inner products are performed in order to compute update scalars that are defined to make the sequences satisfy certain orthogonal conditions. In a symmetric positive definite linear system these conditions imply that the distance to the true solution is minimized in some norm.

The pseudo-code of Conjugated Gradient method is illustrated in Figure 2.9. The iterates $x^{(i)}$ are updated in each iteration by a multiple (α_i) of the search direction vector $p^{(i)}$

$$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)} \tag{2.10}$$

Correspondingly the residuals $r^{(i)} = b - Ax^{(i)}$ is updated as

$$r^{(i)} = r^{(i-1)} - \alpha q^{(i)} \text{ where } q^{(i)} = Ap^{(i)} \tag{2.11}$$

The choice $\alpha = \alpha_i = (r^{(i-1)})^T r^{(i-1)} / (p^{(i)})^T A p^{(i)}$ minimizes $(r^{(i)})^T A^{-1} r^{(i)}$ over all possible choices for α in equation (2.11).

The search directions are updated using the residuals

$$p^{(i)} = r^{(i)} + \beta_{i-1} p^{(i-1)} \tag{2.12}$$

where the choice $\beta_i = (r^{(i)})^T r^{(i)} / (r^{(i-1)})^T r^{i-1)}$ ensures that $p^{(i)}$ and $Ap^{(i-1)}$ or equivalently, $r^{(i)}$ and $r^{(i-1)}$ are orthogonal. In fact, one can show that this choice of β_i makes $p^{(i)}$ and $r^{(i)}$ orthogonal to *all* previous $Ap^{(j)}$ and $r^{(j)}$ respectively.

```
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for i = 1, 2....
    Solve $Mz^{(i-1)} = r^{(i-1)}$
    $\rho_{i-1} = (r^{(i-1)})^T z^{(i-1)}$
    if i = 1
        $p^{(1)} = z^{(0)}$
    else
        $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
        $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
    endif
    $q^{(i)} = Ap^{(i)}$
    $\alpha_i = \rho_{i-1} / (p^{(i)})^T q^{(i)}$
    $x^{(i)} = x^{(i-1)} + \alpha_i q^{(i)}$
    $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
    check convergence: continue if necessary
end
```

Figure 2.9 Pseudo-codes of Preconditioned Conjugated Gradient Method

In Figure 2.9, a preconditioner M is used. Preconditioner is a matrix that transforms the linear equations into one equivalent form but has better convergence rate. Since preconditioner cause some additional computation, it is also a trade off on whether we use a preconditioner and what preconditioner we use. For $M = I$ in Figure 2.9, we can obtain the unpreconditioned version of the Conjugate Gradient Algorithm. In that case, we cam further simplify the algorithm by skipping the "solve" line, and replacing $z^{(i-1)}$ by $r^{(i-1)}$ (and $z^{(0)}$ by $r^{(0)}$). In our thesis, we choose to use Jacobi Preconditioner [36], the simplest one, as our preconditioner since it is not our primitive consideration.

2.6 Quadratic Placement Algorithm Examples

To our knowledge, the quadratic placement algorithm has not been used for FPGAs. The quadratic technique has been used in ASIC placement algorithms. In this section we will present some well-known ASIC placement algorithms based on the quadratic technique.

2.6.1 GORDIAN

Kleinhans et al. [30] describe a placement algorithm named GORDIAN that combines quadratic placement and partitioning to handle the ASIC placement problem. The acronym GORDIAN comes from the two main parts of the method: global optimization and rectangle dissection, which is based on improved partitioning schemes. With GORDIAN, the placement problem is formulated as a sequence of quadratic programming problems derived from the entire connectivity information of the circuit. An increasing number of constraints restricting the freedom of movement of the modules are imposed; reflecting the results of successively refined partitioning.

The GORDIAN algorithm is formed by an iteration of global optimization and partitioning steps. The global optimization starts with an initial region that comprises the whole chip and contains all modules to be placed. One constraint is added to fix all these modules to the center of the chip. In each partitioning step, the modules are divided and the placement regions are dissected into sub-region accordingly. New constraints are established to fix the modules in each sub-region to the center of the corresponding sub-region. This loop of global optimization and partitioning steps is repeated until each region contains less than a predefined number k (e.g. k=10) of modules

GORDIAN introduced a global placement that can refine all modules in the circuit simultaneously, which avoided any dependence on a processing sequence.

2.6.2 GORDIAN-L

GORDIAN-L [32] is an improved version of GORDIAN, which optimizes the linear wirelength objective. Since the linear wirelength objective cannot be addressed directly by the quadratic methods, GORDIAN-L approximates the linear objective by a quadratic objective. It then executes the following loop: first minimize the current objective to yield some approximate solution, and then use this solution to find a better approximation of the linear objective.

The GORDIAN-L is based on the observation that the linear objective can be rewritten as:

$$\Phi_L(x,y) = \frac{1}{2}\sum_{i,j} W_{ij}\left|x_i - x_j\right| = \frac{1}{2}\sum_{i,j} W_{i,j}\frac{(x_i - x_j)^2}{\left|x_i - x_j\right|} = \frac{1}{2}\sum W_{i,j}\frac{(x_i - x_j)^2}{g_{i,j}} \qquad (2.13)$$

If the approximation of $g_{i,j}$ is set, the linear objective can be converted to the squared objective and the problem is solved. If $g_{i,j}$ is constant, GORDIAN-L reduces to GORDIAN. Based on this observation, GORDIAN-L first solves the quadratic problem and gets a reasonable approximation for each $g_{i,j}$. It then performs successive operations to improve it. Each time it uses the coordinates in last iteration to approximate $g_{i,j}$ until no more improvement can be achieved.

GORDIAN-L can achieve up to 20 per cent reduction in area than GORDIAN at the price of significant increase in CPU execution time.

2.6.3 FastPlace

FastPlace [35] is a quadratic based standard cell placement algorithm using cell shifting and iterative refinement and hybrid net model. The basic idea of FastPlace is cell shifting and addition of spreading forces. Chip region is divided into an array of bins structure. The size of these bins is recomputed in both x and y direction so that the utilization of adjacent bins is averaged. By this technique, the cells in the dense area is gradually be pushed to bins around it. And additional spreading forces are added accordingly so that the spread cells do not collapse back to their previous positions during the next step. Half perimeter wire length is also employed to refine the placement based on the bin structure.

The FastPlace algorithm is divided into global placement stage, wirelength improvement stage and detailed placement stage. The first stage mainly deals with cell spreading problem based on quadratic programming. The second stage moves the cells around different bins to reduce the wire length. And the third stage legalizes the placement by assigning cells to pre-defined rows in the placement region and removing overlap among them. It also consists of further reducing the wirelength by a greedy heuristic.

The FastPlace algorithm uses a hybrid net model that combines the clique model and star model when converting the input circuit for quadratic programming, which makes linear sparse system sparser. And it also applies an Incomplete Cholesky Factorization preconditioner to help solve the matrix equations. As a result, this algorithm turns out to be faster than other algorithms.

2.7 Summary

In this chapter, we provided all the background information that is related to our research work. We first introduced the general FPGA design flow together with the FPGA architecture that is used in this thesis. Then we gave a clear definition of the placement problem for FPGAs. Wire length estimation technique was discussed since our work focuses on wire length minimization objective. The previous work done in FPGA placement area was presented next. Finally, we presented the essentials of the quadratic technique and related algorithms in ASIC placement. Our quadratic based FPGA placement algorithm will be described in the next chapter.

Chapter 3

A Quadratic Placement Algorithm for FPGAs

In this chapter, we present the Quadratic Placement algorithm for FPGAs (QPF), which was developed during this thesis work. We start with a discussion of two main problems of the quadratic placement technique in Section 3.1. Then we present our proposed QPF algorithm and explain how we overcome these problems. In subsequent sections, the heuristics used are described in detail, including the pseudo codes and parameter settings.

3.1 Two Main Problems in Quadratic Placement

As we discussed before, squared wirelength objective is used in quadratic placement. This squared wirelength objective is applied only because it allows the one-dimensional placement problem to be reduced to the solution of a system of linear equations. It reduces the computation, but at the same time it also causes some problems. There are two main problems: one is the overlap problem that modules trend to overlap in a dense area, the other is quality problem that squared wirelength objective does not accurately represent the good quality of a placement.

The first problem comes from the lack of legal position information in quadratic placement model. In an FPGA chip (recall from Section 2.1) the legal positions that can accommodate circuit elements are fixed. But in quadratic placement model, it does not consider this information when building up linear equations. This results in an overlapped mathematical solution, as illustrated in Figure 3.1.

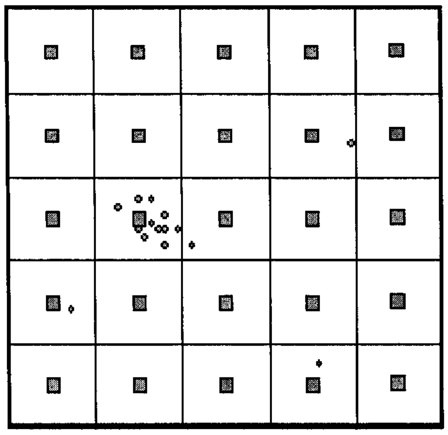


Figure 3.1 Illustration of Overlap Problem for Quadratic Placement

Figure 3.1 shows a 5x5 array of in an FPGA chip. The gray squares are the legal positions in which the logic blocks can be placed. The tiny circles are the positions that a linear equations solution actually produces. A good quadratic based placer must handle this problem efficiently so that the overlap is resolved with a minimum deterioration in placement quality..

The second problem comes from the squared wirelength objective in the quadratic model. To convert the placement problem into a system of linear equations, it uses a squared wirelength objective function, not a linear wirelength function. [31] compared the linear and squared wirelength objectives and concluded that the linear wirelength is superior. Quadratic wire length is an indirect measure of linear wire length. Usually reduction in quadratic wire length leads to reduction in linear wire length. But in some cases the minimization of quadratic wire length does not mean the linear wire length is minimized. Figure 3.2 illustrates the difference between linear and squared wirelength objective. A circuit consists of four nodes A, B, C and D. Nodes A, B, C are fixed, and node D is movable. The distance between A (or B) and C is L. When quadratic wire length model is applied, the optimized position of node D will be in d = (1/3) L. When linear wire length is applied, the position will be d =0, which means node D will be placed as close to A and B as possible.

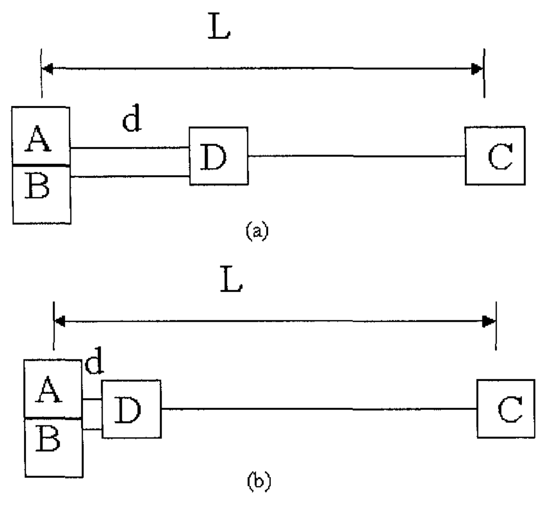


Figure 3.2 Linear vs. Squared Wirelength: a) Squared Objective Model. b) Linear Objective Model

Furthermore, placement with minimum squared wirelength objective has a unique solution that can be found by solving the corresponding linear equations. But placement with linear wirelength objective can have multiple optimal solutions. For example, a single movable logic block connected to two fixed I/O pads by edges of equal weight can be optimally placed anywhere between the two I/O pads. A good quadratic based placer must also handle this problem efficiently so as to achieve the minimum linear wirelength.

3.2 Overview of the QPF Algorithm

An overview of the QPF algorithm (using pseudo code) is given in Figure 3.3. It consists of three stages, where input to the first stage is a technology mapped and packed circuit to be placed [6][3].

The goal of the first stage is to obtain a good initial placement. As we discussed before, the solution of linear equations tends to overlap in a dense area, as shown in Figure 3.1.In this stage, we try to expand the solution to the entire chip area while attempting to minimize the wirelength. This is achieved by repeatedly building up, modifying and solving linear equations to get a progressively better placement. Each iteration contains four steps: First we build up the linear equations and solve them to get the coordinates of every node of the circuit. In the first iteration, we build our equations according to the connectivity of the input circuit. In subsequent iterations, we modify the linear equations according to the new dummy nodes added in the previous iteration. Second, we map the nodes into the entire chip area according to their current coordinates. This mapping process is performed every time we get a new coordinates for the nodes in the circuit. After mapping process, we will have legal placement and can evaluate our placement using half perimeter wire length estimate. Third, we use these new positions as reference and add extra dummy nodes to the circuit. These new nodes are used to modify the coefficient matrix in the next iteration.

```
QPF Algorithm
Stage 1
    - Build and solve linear equations
    - Map the circuit to the FPGA chip
    - Add dummy nodes and expand the placement
    - Refinement for minimizing linear wire length
    - Repeat above steps until there is no significant improvement
Stage 2
    - Refinement for minimizing linear wire length based on legal placement until
    there is no more improvement
    - Re-map the circuit to the FPGA chip
Stage 3
    - Low temperature Simulated Annealing to refine the final placement
```

Figure 3.3 Overview of QPF Algorithm

The first three steps in stage one is used for expansion of the logic blocks. Fourth, during each iteration, we also perform linear adjustment to minimize the linear wire length as well as the squared wire length. This step is relatively independent of the previous three steps. It has nothing to do with the expansion purpose. It is inserted here to ensure that linear wirelength improvement is also considered during the expansion process. This is also achieved by modifying the coefficient matrix of quadratic equation. The difference is that in this step, we modify the coefficient matrix to reduce wirelength, not to expand the nodes in dense area. Stage one is performed until no significant improvement can be achieved.

When we leave stage one, we already have a reasonably good legal placement. In stage two, we try to further improve this placement. In this stage, since the nodes are already evenly distributed among the chip area, we concentrate on improving the linear wirelength. This is achieved by using the similar linear wire length reduction technique described in step four of stage one. The difference is that we do not build and solve the linear equations in this stage. Instead we move the nodes directly (by changing the coordinates of the nodes) to reduce the total linear wire length. Since no equation solver is needed in this stage, this process is much faster than the linear adjustment in stage one and we can perform more iteration and get a better refinement.

Finally, in stage three the placement is refined by low temperature simulated annealing algorithm to further minimize the wire length.

The QPF algorithm uses a heuristic approach that achieves the final placement gradually. Different t techniques are used in different stages. In the following sections, we will discuss these techniques and explain how we get a good placement by utilizing these techniques.

3.3 The Node Mapping Process

The node mapping process maps the nodes of the input circuit to the target FPGA chip area based on the current node coordinates. The solution of quadratic linear equations provides the coordinates of all nodes in the circuit. These coordinates do not give a legal placement; therefore, they are mapped to the physical location of the FPGA chip using mapping process. The mapping process is widely used both in stage one and two. Whenever the coordinates of the nodes in the circuit are updated, we perform this mapping process on the circuit and a new placement is obtained.

The mapping process is done in a partitioning-like way. But unlike typical partitioning algorithms that partition the circuit, we partition the chip area, not the circuit, into four parts. The Figure 3.4 illustrates how the mapping process works. For clarity, we omit the gray squares in Figure 3.1.

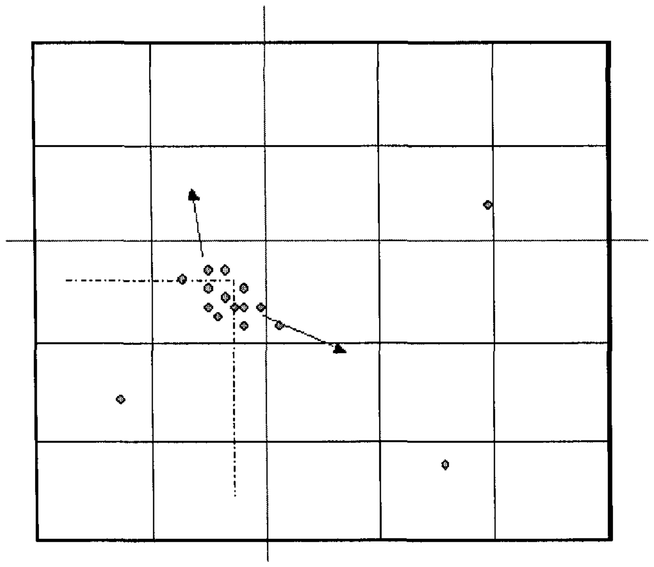


Figure 3.4 Over of Mapping

The basic idea of mapping is as follows: Firstly, we divide the whole chip into four parts and each part has integer number of rows and columns. For example, in Figure 3.4 a 5x5 FPGA is divided into four parts, 2x2 2x3, 3x2 and 3x3 each. Secondly, we check all the four parts for overflow. A part is said to overflow when the number of nodes in that part is greater than the number of physical positions in the part. In Figure 3.4, the left-bottom part is overflowed since the part can only accommodates 6 nodes, far less than number of nodes in this part. All the overflowed nodes must be removed to other parts. Thirdly, we try to balance the chip so that extra nodes on the edges of the overflowed parts are removed from the overflowed parts into other parts. Since the original FPGA chip selected is large enough to hold all the nodes, there can be 1-3 overflowed parts. Depending on the number of overflowed parts, the balance sequence is shown in Figure 3.5.

The overflowed parts are shown in gray. The balance sequence in a) and c) of Figure 3.5 is obvious.

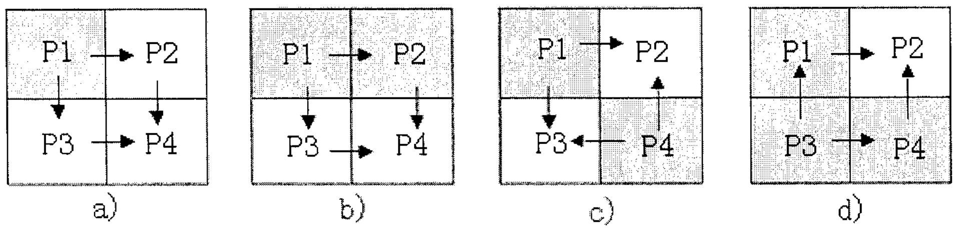


Figure 3.5 Balance Sequence

We remove the extra nodes in the overflowed parts to its adjacent parts. For example the balance sequence for a) is: 1. P1→P2&P3, 2. P2→P4, 3. P3→P4. For b), we start from the denser one in the two overflowed parts, and for d) we always start from the overflowed part that is in the middle and push the extra nodes to the other parts, as the arrows show in Figure 3.5. When nodes in one dense part are to be moved to both adjacent parts (e.g., the overflowed nodes in P1 are to be moved to both P2 and P3 in Fig3.5 a), the number of nodes moved to each adjacent part should be carefully chosen so that they make the two adjacent parts equally overflowed.

```
P = n x n FPGA CLB array;
Repeat:
    Partition P into four parts. P1, P2, P3, P4;
    Check overflow for all parts P1~P4;
    If (overflow){
        Find all overflowed parts;
        Find balance sequence;
        Remove all overflowed nodes to appropriate parts;
    }
    mapping(P1);
    mapping(P2);
    mapping(P3);
    mapping(P4);
Until all parts are small enough
```

Figure 3.6 Pseudo-codes for Node Mapping Process(P)

The pseudo code of the mapping process is shown is Figure 3.6. By recursively partitioning and mapping, all the nodes will be fixed to a small local area. And within that small area we can find the exact physical location for them with ease. Mapping process is the basic operation is our placement algorithm, both the expansion and linear adjustment use mapping process to legalize and evaluate their placement.

3.4 The Expansion Process

As discussed in Section 3.1, one of the main problems with quadratic placement is that the nodes of the input circuit tend to overlap in some dense area of the chip. One of the challenges for quadratic placement is how to solve this problem efficiently. In the QPF algorithm, we developed an iterative expansion technique that expands the nodes gradually, while satisfying the wirelength minimization objective as well.

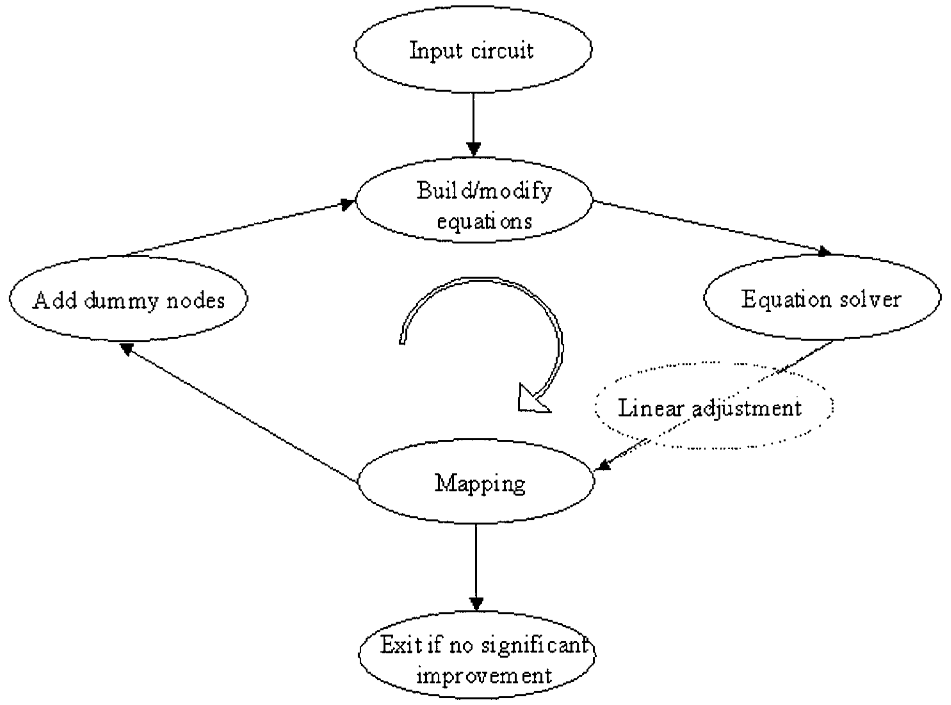


Figure 3.7 Overview of Expansion Process

The overview of our expansion process is shown in Figure 3.7. The input to the expansion process is the circuit to be placed, which is used to build up the original linear equations. The equation solver solves the linear equation by Conjugated Gradient method as discussed in Section 2.6. The mapping process maps the nodes to the entire chip area according to the current node coordinates as discussed in the previous section. Then one dummy node is added to every node in the circuit according to the mapping result. These dummy nodes help to modify the coefficient of linear equations and get a better result in the next iteration. The linear adjustment stage here is used to reduce the linear wirelength during the expansion process, which will be discussed in the next section. In every iteration, we check the placement quality after the mapping process. We exit the loop when no significant reduction in placement cost is obtained in that loop.

The basic intuition behind this expansion process is as follows: according to the input circuit, we can build up a matrix equation, in x dimension for example, $Ax = b$. And suppose we have already got the ideal placement $x = [x_1, x_2 \ldots x_i \ldots x_n]$, this ideal placement vector can be a solution of a similar matrix equation $A'x' = b'$. So if we can modify our origin matrix A and vector b to make them approach the ideal matrix A' and vector b', the solution will approach the ideal solution as well. In short, $\begin{cases} A \to A' \\ b \to b' \end{cases} \Rightarrow x \to x'$.

The main challenge of this expansion process is how we set the dummy nodes in Figure 3.7. The dummy-node-setting process must accomplish two goals. One is that the dummy nodes must be set so that they help to pull the nodes in the dense area apart, which is the main goal of expansion process. The other is that these dummy nodes must not greatly change the relative position of the nodes obtained by solving linear equations. In our algorithm, we use the mapping result in the current iteration as a reference placement to set the dummy nodes. It is shown in Figure 3.8.

Figure 3.8 a) shows the original node position that we get from previous iteration in expansion process. Figure 3.8 b) shows the node position after mapping process. One node in the circuit is marked as a square, and we call it node A. This node A is in the center near position (3,3) in Figure 3.8 a) and is mapped to the right side (4, 2) as in Figure 3.8 b). This tells us that based on the current information node A should be placed in (4, 2). So we add a dummy node at this place and connect node A (and only node A) to the dummy node. The weight between nodes A and the dummy node is set according to the average weight connected to node A.

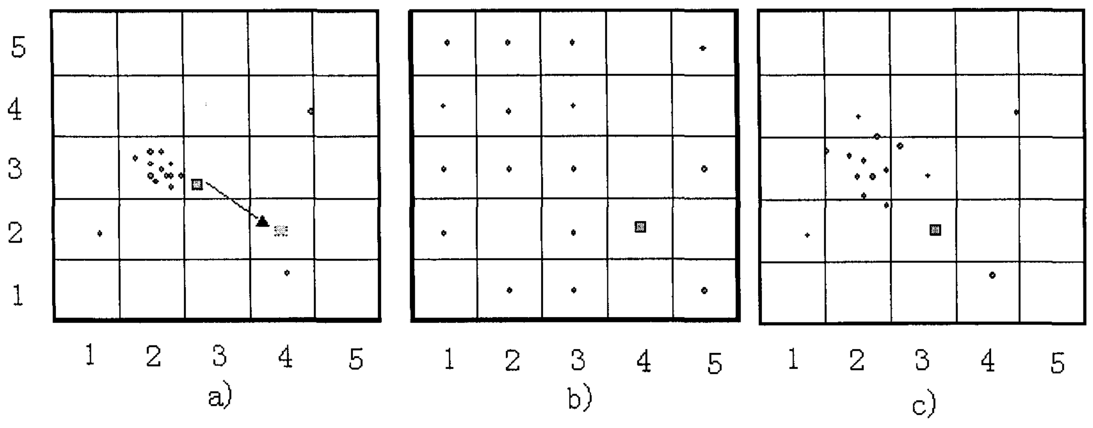


Figure 3.8 Add a Dummy Node a) Origin Node Position & Dummy Node b) Reference Node Position After Mapping c) Result of One Iteration of Expansion

The information about the dummy node added for node A will be used to modify the coefficient of the linear equations and will pull node A to itself in the next iteration of expansion process. It will also affect the other nodes in the circuit, which are connected to node A directly or indirectly, through node A. After we add a dummy node for every node in the circuit and modify the linear equation accordingly, we will get a better placement by solving the equations in the next iteration. As we can see in Figure 3.8 c), the nodes in dense area in Figure 3.8 a) are expanded to a larger space. Note that the dense area in Figure 3.8 c) is not just a direct expansion of that Figure 3.8 a). The relative position among the circuit nodes has been changed during the convergence of equation solver. The equation solver also guarantees the minimization of squared wirelength.

The weights between circuit nodes and the dummy nodes are set by their average weight. Based on the fact that the weight needed for expansion increase significantly as the expansion process goes on, we start with a small portion of average weight, like 1/10 of the average weight of each node, and gradually increase the weight of dummy nodes in later expansion process.

Although dummy nodes are necessary for expansion, they also have side effects. Dummy nodes with small weights work well without causing much interference to the original circuit. But when the expansion process goes on, the effect of those dummy nodes accumulates. With enough number of iterations the weight added by dummy nodes will become dominant. At that time, although we can still expand our circuit, the squared wirelength minimization objective of quadratic model will not hold any more. Usually, we terminate this expansion process after several iterations, when the improvement rate of one iteration is not significant (e.g. less than 10%).

3.5 Linear Adjustment

Beside the overlap problem we discussed above, another big concern about quadratic placement algorithm is the quality. As we mentioned in Section 3.1, quadratic placement algorithm is based on the squared wirelength model. Squared wirelength is only an indirect approach and not always proportional to linear wirelength. Further reduction of linear wirelength is a key problem for all algorithms based on the quadratic model.

Recall from Section 2.3.1, Half-Perimeter Wire Length (HPWL) is an efficient way to estimate the wirelength needed to connect the circuit nodes. Our linear adjustment technique is based on the difference between linear wirelength and squared wirelength under Half-Perimeter Wire Length (HPWL) model.

Figure 3.2 shows the basic difference between linear and squared wirelength. In real circuits, it is more complicated. A general case is presented in Figure 3.9. It comes from the result we get from quadratic placement model. Node A is connected to net 1, net 2, and net3. The nodes in these three nets are represented by rectangle, circle, and triangle respectively. In this example node A is on the right edge of net 2, bottom edge of net 3 and is in the middle of net 1.

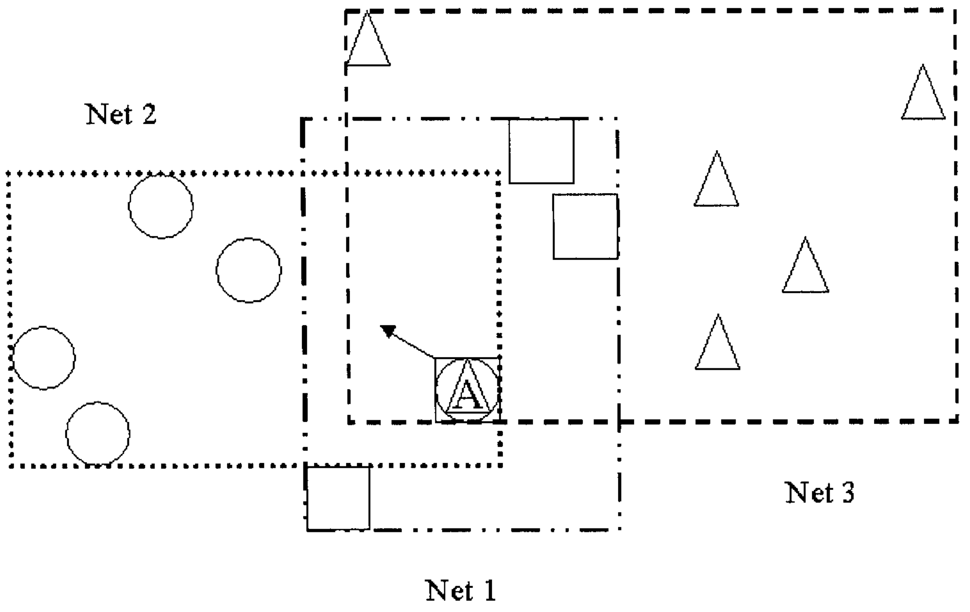
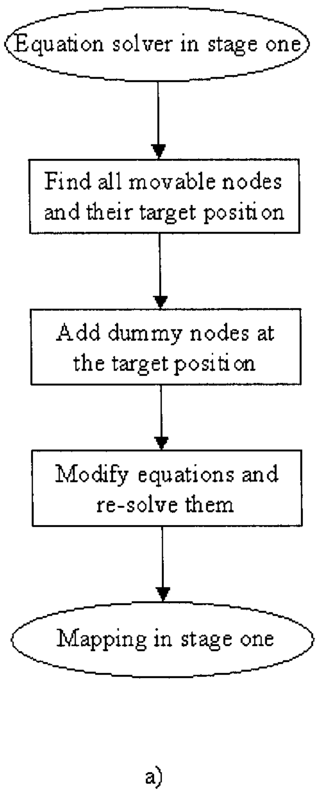


Figure 3.9 Wirelength Contribution of One Node Connected to Three Nets

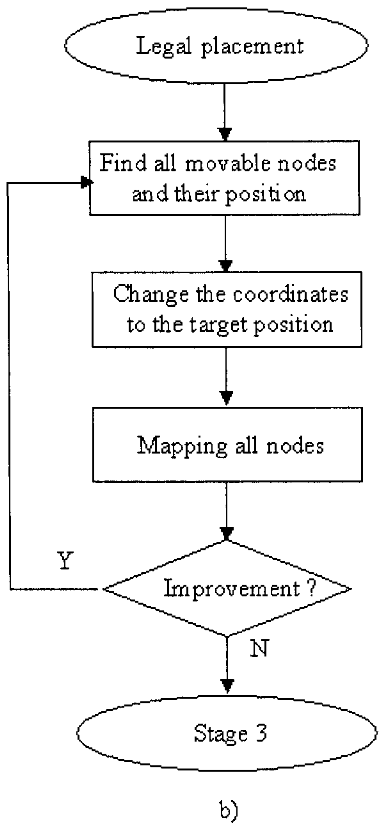
Under Half-Perimeter Wire Length model, only the nodes on the edges of each net contribute to the total wirelength. As in Figure 3.9, node A contributes to the total wirelength through net 2 and net3. So if we move node A to the left, it will reduce the wirelength of net 2 while not increasing the wirelength of other two nets before it reaches the edge of net 3. The same scenario results if it is moved upwards.

In real circuits, a node might be on the different edges of two or more nets, e.g. a node can be on the left edge of a net and the right edge of another net at the same time. In these cases, the weight of all involved nets is considered so that the node is moved to the direction that reduces the total wirelength.

Every time we perform linear adjustment, we first check the circuit to find all such movable nodes and their target positions. Then if this technique is used in stage one of our algorithm, we add extra dummy nodes at the target positions and this technique will take effect by modifying the coefficient matrix and re-solving the equations. When it is used in stage two, we modify the coordinates of the node directly so that the relative position of the nodes is changed, and it will take effect in the next mapping process, as shown in Figure 3.10.



Equation solver in stage one
Find all movable nodes and their target position
Add dummy nodes at the target position
Modify equations and re-solve them
Mapping in stage one
a)



Legal placement
Find all movable nodes and their position
Change the coordinates to the target position
Mapping all nodes
Improvement ?
Y
N
Stage 3
b)

Figure 3.10 Linear Adjustment Used in a) Stage 1, b) Stage 2

3.6 Low Temperature Simulated Annealing

In stage three, we finally improve the placement quality using low temperature simulated annealing-based [37][24] iterative improvement. Refer to Chapter 2 for a description of the basic simulated annealing method as it is applied to placement. We have adapted the annealing implementation in VPR described in [3].

The key parameters that control the quality and time for simulated annealing are: the starting temperature T_0, the rate at which the temperature is reduced represented as "α", the number of configurations to explore at each temperature called "InnerNum", the behavior of the range limiting mechanism as D_{limit}, and the exit criterion by which the annealing algorithm terminates.

1. The starting temperature T_0

The starting temperature T_0 is a crucial parameter for our refinement because it determines the initial probability that a bad move or swap is accepted. If the starting temperature is set too high, the subsequent annealing will destroy the placement we have developed in the previous stages. Otherwise, if it is set too low, then insufficient optimization will be performed, the placement might get trapped in some local minimum and the quality will not be good enough.

In our program we set the starting temperature in a way that a bad move or swap that causes 0.1% degradation in quality still has 0.1% probability to be accepted. According to the simulated annealing schedule, a bad move or swap is only accepted with probability $r < e^{-\Delta C/T}$, where r is a random value in range [0,1], ΔC is the quality difference caused by the move. We get,

$$T \geq -\frac{\Delta C}{\ln(r)} = C \cdot \frac{-\Delta C / C}{\ln(r)} = C \cdot \frac{0.001}{\ln(0.001)} = \frac{C}{6907}$$

where C is the cost of the current placement.

2. The number of moves per temperature

   The basic annealing algorithm of VPR makes $InnerNum \cdot N_{blocks}^{4/3}$ moves at each temperature, where N_{blocks} is the total number of logic blocks and I/O pads and the InnerNum is set to 10 by default. Since we have already had a good placement from the first two stages of our algorithm, we find by experiments that it does not make much difference for InnerNum greater than 3. In our program we set it to 4.

3. The move limit

   In VPR, the move limit D_{limit} is the distance that a node is free to move during the annealing process. It always starts with the largest number, Dlimit covers whole chip. In our algorithm, the placement we get from the previous stages is good enough so that it is unlikely that a node should be moved such a long distance to improve the quality.

By observing the annealing procedure, we set this value to one-fifths of the whole chip range.

4. Other parameters

   The modification on other parameters such as exit criterion, temperature update factor also slightly affects the annealing process. We did not change these parameters.

3.7 Computational Complexity

To evaluate the computational complexity of our QPF algorithm, we have to consider all the based operations used by our algorithm. They are: the mapping process, the linear adjustment process, the linear equations building and solving operation and the low temperature simulated annealing process.

The time complexity of simulated annealing is quite obvious. Since the number of moves per temperature is $InnerNum \cdot N_{blocks}^{\frac{4}{3}}$, the complexity of stage three is $O(N_{blocks}^{\frac{4}{3}})$ [3].

In the mapping process, we map the circuit by recursively partitioning the circuit into four parts. So the computational complexity of mapping process is $O(N_{CLB})$ + $4 \times O(\frac{N_{CLB}}{4})$ + $4^2 \times O(\frac{N_{CLB}}{4^2})$ + ... $+4^n \times O(\frac{N_{CLB}}{4^n})$. And the total level of partitioning is n= $\log_4^{\sqrt{N_{CLB}}}$, where $\sqrt{N_{CLB}}$ is the width of CLB array of the FPGA. So the computational complexity of mapping process becomes

$$O(N_{CLB}) + 4 \times O(\frac{N_{CLB}}{4}) + 4^2 O(\frac{N_{CLB}}{4^2})... = O(N_{CLB} \cdot \log_4^{\sqrt{N_{CLB}}}) = O(N_{CLB} \cdot \ln N_{CLB})$$

For linear adjustment, we check every node for better position. Since in real circuit, the average fanin and fanout of a node are limited, we only need to search limited number of other nodes to determine the new position of the current node. This operation is about $O(k \cdot N_{CLB}) = O(N_{CLB})$

To determine the computational complexity of the quadratic placement, we must consider both the computational work of building and solving the linear equations.

Building the linear equation is simple. We check all nodes in the circuit and compute the weight for the nodes connected to them. The complexity is $O(nnz)$. Where nnz means the Number of Non Zero element in the matrix. When we modify the coefficient matrix, we only add one dummy node to each node, the complexity is N_{CLB}.

The complexity of the equation solver is more complicated.. Basically, one iteration of the Conjugated Gradients method contains 3 N_{CLB}-dimension vector additions, 2 N_{CLB}-dimension vector multiplications and 1 Matrix vector multiplication. The complexity of one iteration is $O(N_{CLB})+O(N_{CLB})+O(nnz)=O(nnz)$. The problem is the convergence rate of conjugated Gradients method is not well studied. In [36] the convergence rate of Conjugated Gradients method without preconditioning is $O(h^{-1})$, where h is the discretization mesh width [39] and depending on the spectrum of the coefficient matrix involved [40]. So the total complexity of the quadratic equation solver is $O(nnz)\cdot O(h^{-1})$. For a real circuit, $O(N_{CLB})\cong O(N_{blocks})\cong O(nnz)\cong O(N)$ since the number of I/O pads and the average fanout are usually limited and independent on the circuit size. In [41] it is said that the computational work of unpreconditioned Conjugate Gradients is from $O(N^{\frac{4}{3}})$ to $O(N^{\frac{3}{2}})$, depending on the uniform grid used. With the help of a preconditioner, an additional matrix to transform the original system to an equivalent one with an improved spectrum, the convergence rate of the Conjugated Gradients method can be accelerated. So we believe that the complexity of our algorithm is in a way less than the range $[O(N^{\frac{4}{3}}), O(N^{\frac{3}{2}})]$. It is comparable to VPR, whose complexity is $O(N^{\frac{4}{3}})$.

Based on the discussion above, the linear equation solver is the most computationally expensive operation. The overall computational complexity of our placement algorithm is same as the equation solver, whose computational complexity is comparable to VPR.

Since the computational complexity only shows the potential performance of the algorithm when the scale of input circuit increases, it does not directly determine the speed of the algorithm. In this case, VPR performs swaps and moves at a large number of temperatures, each temperature is an $O(N^{\frac{4}{3}})$ operation, while the proposed algorithm only needs several iterations of such operation. So the proposed algorithm can run much faster than VPR does although both algorithms have similar computational complexity.

3.8 Summary

In this chapter, we first examined the problems with the general quadratic placement algorithm. Then we presented QPF, our proposed placement algorithm for FPGAs and explained how we handled these problems. The QPF placement algorithm consists three stages. It begins in stage 1 with a normal quadratic placement method and gradually expands the initial placement by a loop of equation modifying and solving process. During the expansion process, special cares is taken to ensure that linear wirelength is minimized while satisfying squared wirelength objective as well. Then linear wirelength reduction technique is applied in stage 2 to improve the quality of the previous result. Low temperature Simulated Annealing is used to finally refine the placement in stage 3. Details about all three stages are discussed in sections 3.3 to 3.6. Finally the computational complexity of the QPF algorithm is discussed in section 3.7. The QPF algorithm was evaluated and compared to the state-of-the-art VPR placement tool using 20 benchmark circuits. The experimental results are presented in the next chapter.

Chapter 4

Experimental Results and Analysis

In this chapter, we will present the experiment results that demonstrate the quality and efficiency of the QPF placement algorithm. We first present experimental results showing the effects of different techniques employed in our algorithm on the quality of placement obtained. These techniques were discussed in detail in Chapter 3. Then a comparison is made between QPF and VPR, a well-known high-quality FPGA placement tool. The quality versus run time trade off for QPF is presented later in this chapter. We also describe our attempt at enhancing QPF to make it a timing-driven placement tool and present encouraging preliminary results. We first describe the experiment evaluation environment that was used to obtain the results.

4.1 Experimental Evaluation Environment

All the experiments were performed under identical hardware environment. We tested both our QPF placement tool and VPR on a Pentium 2.5GHz based computer with no other application program running.

The benchmark circuits we used in our experiments were obtained from two sources: fifteen of the circuits originate from the Microelectronics Centre of North Carolina (MCNC) suite [42], five circuits were generated by GEN [43], which is a synthetic benchmark circuit generation tool that has been used in FPGA research. All circuits were originally in the ***blif*** format. They were optimized by *SIS* [44] and technology mapped into

4-LUTs using *Flowmap* [6]. Then we used VPACK [3] to pack the netlist of 4-LUTs and flip-flops into basic logic blocks. The sizes of benchmark circuits range from 2000 to 20,000 logic blocks.

We have implemented our QPF placement tool within the framework of VPR using the most recent version 4.30 of VPR code. We also use exactly the same function to estimate the bounding box wirelength. Time consumption is measured by using the CPU clock and then converted it to seconds. The run time of both tools is measured from the very beginning to the end of placement, including the initial input file reading time. Since our algorithm is written within the framework of VPR, extra time is used to convert the data structure from VPR's to ours. If only the time for placement is considered, QPF placement tool can run slightly faster than what has been reported in this chapter.

4.2 Effects of Key Parameters and Techniques on Placement Quality

Recall that in Chapter 3, we explained the techniques used in QPF to handle the overlap problem and also how the wirelength minimization was achieved by using expansion and linear adjustment techniques. In this section, we present experimental results obtained using these techniques and show how they affect the placement quality.

4.2.1 Compensation Factor

As we discussed in Chapter 2, compensation factor q(i) is used to compensate the under-estimated linear wirelength of half perimeter wirelength model for large nets. Table 4.1 shows the improvement in wirelength when q(i) is applied in the quadratic model over the original quadratic placement. To prevent the extreme situation that most nodes are overlapped together in a very small region, we do not increase the compensation factor for nets larger than 50 terminals when building linear equations. This is done by setting all q(i) = q(49) for all i >= 50.

The first column shows the circuit name. The second and third column shows the number of blocks and number of CLBs in the circuit respectively. The bounding box wirelength is shown in column 5,and 6. Finally the wirelength reduction is shown in column 6.

We can observe from Table 4.1 that after the compensation factor is applied to the quadratic model, we have a better estimation for the wirelength for all nets and we can get about 20% reduction in total linear wirelength.

4.2.2 Expansion

In the QPF algorithm, we deal with the overlap problem of quadratic placement by performing expansion process in stage 1, as shown in Figure 3.3. We applied an iterative process discussed in section 3.4 so that we spread all the nodes out gradually, while at the same time achieving the wirelength minimization objective as well.

To show the exact effect of the expansion technique, we perform one iteration of expansion process based on the basic quadratic placement with modification of compensation factor. The result is shown in table 4.2. We can achieve about 21% improvement of quality out of the first iteration of expansion process. When we perform more such iterations, the rate of improvement falls off.

4.2.3 Linear Adjustment

Similarly, we show the effect of the linear adjustment by employing one linear adjustment step to the previous result.

The experimental results are shown in Table 4.3. We can see that the first linear adjustment step can achieve about 15% improvement based on the previous result. Since this linear adjustment is performed on a legal placement and does not need to modify or solve the matrix equation, it runs very fast. We can try this as long as more improvement is possible.

4.2.4 Starting Temperature

We perform low temperature simulated annealing to finally refine the placement in QPF algorithm. As we discussed in Chapter 3, the starting temperature should be carefully selected. We only want to accept reasonable number of bad moves during the annealing process so that enough optimization is tried while making sure that we do not destroy the placement we get in previous stages.

A high starting temperature can produce a high quality placement. But at the same time it will take a long time to finish the annealing process. Because at high temperature, the annealing process accepts many bad moves so that it almost breaks the existing placement and re-places it again. At extreme cases, the high starting temperature destroys everything and treats the placement we get from the previous stages as a random one. In this section we compare our adaptive starting temperature with a fixed but lower starting temperature. We run our placement tool with T0=C/6907 as discussed in Chapter 3 and T0=0.01, a fixed and lower starting temperature. The results are presented in Table 4.4 and show that the fixed starting temperature works well for some circuits. On the whole the adaptive starting temperature produces about 7% wirelength reduction over all 20 benchmark circuits.

Table 4.1. The Wirelength Comparison of Original Quadratic Placement Before and After the Compensation Factor is Applied

Circuit Name	Number of blocks	Number of CLBs	Bounding box wirelength		Wirelength Reduction
			Without compensation factor	With compensation factor	
alu4	1544	1522	405.86	278.00	46.0%
Apex2	1919	1878	557.42	504.11	10.6%
Apex4	1290	1262	340.98	312.06	9.3%
Bigkey	2133	1707	458.95	370.18	24.0%
Clma	8527	8383	3560.04	3183.64	11.8%
Des	2092	1591	562.82	563.25	-0.1%
Elliptic	3849	3735	1106.72	875.60	26.4%
Ex1010	4618	4598	2201.81	1420.31	55.0%
Ex5p	1135	1064	280.77	259.44	8.2%
Frisc	3692	3556	1235.92	1138.92	8.5%
Misex3	1425	1397	375.26	321.53	16.7%
Pdc	4631	4575	1833.91	1646.49	11.4%
S38417	6541	6406	2127.30	1258.72	69.0%
S38584	6789	6447	2155.01	1449.24	48.7%
Spla	3752	3690	1376.16	1219.26	12.9%
Fcom1	10307	9986	6445.90	6020.18	7.1%
Fcom2	11426	10984	6479.99	5396.21	20.1%
Fsm5	12104	11852	6793.37	6790.22	0%
Fcom3	13063	12962	5065.75	4446.27	13.9%
Fsm6	20051	19903	14779.78	13117.56	12.7%
				Average	**20.6%**

Table 4.2 The Effect of One Iteration of Expansion Process on Placement Quality

Circuit Name	Number of blocks	Number of CLBs	Bounding box wirelength		Wirelength reduction
			Original Quadratic placement	After one iteration of expansion	
Alu4	1544	1522	278.24	249.43	11.6%
Apex2	1919	1878	504.22	445.82	13.1%
Apex4	1290	1262	312.06	268.90	16.1%
Bigkey	2133	1707	370.18	368.67	0.4%
Clma	8527	8383	3183.64	2374.73	34.1%
Des	2092	1591	563.25	543.83	3.6%
Elliptic	3849	3735	875.60	819.62	6.8%
Ex1010	4618	4598	1420.31	976.48	45.5%
Ex5p	1135	1064	259.44	238.80	8.6%
Frisc	3692	3556	1138.92	1004.54	13.4%
Misex3	1425	1397	321.53	283.83	13.3%
Pdc	4631	4575	1646.49	1425.79	15.5%
S38417	6541	6406	1258.72	993.94	26.6%
S38584	6789	6447	1449.24	1166.55	24.2%
Spla	3752	3690	1219.26	1021.25	19.4%
Fcom1	10307	9986	6020.18	4288.48	40.4%
Fcom2	11426	10984	5396.21	4023.05	34.1%
Fsm5	12104	11852	6790.22	4707.32	44.2%
Fcom3	13063	12962	4446.27	3498.13	27.1%
Fsm6	20051	19903	13117.56	10285.94	27.5%
				Average	**21.3%**

Table 4.3 Effect of One Linear Adjustment Step on Placement Quality

Circuit Name	Number of blocks	Number of CLBs	Bounding box wirelength		Wirelength reduction
			Without linear wirelength adjustment	With linear wirelength adjustment	
Alu4	1544	1522	249.43	227.64	9.6%
Apex2	1919	1878	445.82	397.57	12.1%
Apex4	1290	1262	268.90	238.01	13.0%
Bigkey	2133	1707	368.67	351.91	4.8%
Clma	8527	8383	2374.73	2027.23	17.1%
Des	2092	1591	543.83	466.62	16.5%
Elliptic	3849	3735	819.62	717.72	14.2%
Ex1010	4618	4598	976.48	855.84	14.1%
Ex5p	1135	1064	238.80	212.38	12.4%
Frisc	3692	3556	1004.54	815.03	23.3%
Misex3	1425	1397	283.83	248.22	14.3%
Pdc	4631	4575	1425.79	1212.01	17.6%
S38417	6541	6406	993.94	889.13	11.8%
S38584	6789	6447	1166.55	1018.39	14.5%
Spla	3752	3690	1021.25	837.14	22.0%
Fcom1	10307	9986	4288.48	3499.59	22.5%
Fcom2	11426	10984	4023.05	3371.54	19.3%
Fsm5	12104	11852	4707.32	3921.53	20.0%
Fcom3	13063	12962	3498.13	2940.22	19.0%
Fsm6	20051	19903	10285.94	8837.39	16.4%
				Average	**15.7%**

Table 4.4 Effect of Different Starting Temperatures on Placement Quality

Circuit Name	Number of blocks	Number of CLBs	Bounding box wirelength		Wirelength reduction
			T0=0.01	T0=C/6907	
Alu4	1544	1522	198.16	195.72	1.2%
Apex2	1919	1878	315.43	277.22	13.8%
Apex4	1290	1262	195.84	186.96	4.7%
Bigkey	2133	1707	324.09	317.75	2.0%
Clma	8527	8383	1629.29	1491.26	9.3%
Des	2092	1591	401.43	390.33	2.8%
Elliptic	3849	3735	596.40	551.65	8.1%
Ex1010	4618	4598	702.65	665.03	5.7%
Ex5p	1135	1064	178.31	173.87	2.6%
Frisc	3692	3556	628.39	564.03	11.4%
Misex3	1425	1397	206.51	196.72	5.0%
Pdc	4631	4575	975.73	914.00	6.8%
S38417	6541	6406	755.07	699.87	7.9%
S38584	6789	6447	867.37	823.58	5.3%
Spla	3752	3690	698.25	641.88	8.8%
Fcom1	10307	9986	2770.53	2491.26	11.2%
Fcom2	11426	10984	2671.69	2440.23	9.5%
Fsm5	12104	11852	3171.18	2850.71	11.2%
Fcom3	13063	12962	2392.17	2113.42	13.2%
Fsm6	20051	19903	7317.3	6799.44	7.6%
				Average	**7.4%**

4.3 Comparison Between QPF and VPR

In this section we present the placement results obtained using QPF and compare them to placement results obtained using VPR. Since quadratic placement uses fixed I/O pads when building the linear equations, we randomly place the I/O pads of the benchmark circuits first and then feed these I/O pads physical locations to both QPF and VPR.

We ran the VPR placer using the default mode, which is well tuned to give the best result. Since we only compare the wirelength, the VPR placer was set to run in wirelength driven-mode so that it achieves better wirelength quality. We ran QPF with default parameter settings that were discussed in Chapter 3. The placement wirelength and the CPU runtime are compared between these two tools. A comparison of results obtained using QPF and VPR is shown in Table 4.5.

The first column is shows the circuit name. The second column shows the wirelength obtained for each circuit by QPF and VPR. And similarly the column 3 and 6 present the wirelength and CPU time for VPR placer. The wirelength ratio of QPF and VPR is shown in the third column. Similarly columns four and five show the run time (given in seconds) and the run time ratio of QPF and VPR.

From Table 4.5 we can observe that, for each circuit, the wire length obtained by QPF is very close to that obtained by VPR. On average, across 20 benchmark circuits, the wirelength obtained by QPF is only 0.8% longer than VPR. But QPF gives better runtime as evidenced by Table 4.5. On average, across 20 benchmark circuits QPF runs about 5X faster than VPR. Thus we were able to achieve the goal that motivated our work, i.e. to achieve equal or better placement quality while improving the placement run time.

Table 4.5 Comparison of Placement Results Obtained by QPF and VPR

Circuit Name	Bounding box wirelength		Ratio QPF/VPR	Time Consumption(s)		Ratio VPR /QPF
	QPF	VPR		QPF	VPR	
alu4	195.72	193.41	1.012	6.53	33.32	5.10
Apex2	277.22	275.99	1.004	7.87	50.57	6.43
Apex4	186.96	183.47	1.019	4.12	27.01	6.56
Bigkey	317.75	311.14	1.021	9.90	49.35	4.98
Clma	1491.26	1465.18	1.018	255.45	1113.07	4.36
Des	390.33	385.31	1.013	7.96	49. 12	6.17
Elliptic	551.65	547.41	1.008	63.40	164.89	2.60
Ex1010	665.03	661.88	1.005	55.67	281.21	5.05
Ex5p	173.87	173.96	0.999	3.21	23.32	7.26
Frisc	564.03	552.89	1.020	37.53	162.78	4.34
Misex3	196.72	193.13	1.019	4.50	29.53	6.56
Pdc	914.00	905.91	1.009	60.65	292.85	4.83
S38417	699.87	722.71	0.968	158.92	680.84	4.28
S38584	823.58	802.43	1.026	218.29	716.60	3.28
Spla	641.88	635.32	1.010	33.25	167.42	5.04
Fcom1	2491.26	2481.15	1.004	433.34	1946.9	4.49
Fcom2	2440.23	2425.86	1.006	481.81	2209.25	4.59
Fsm5	2850.71	2871.94	0.993	629.90	2736.96	4.35
Fcom3	2113.42	2111.55	1.001	642.23	3101.96	4.83
Fsm6	6799.44	6763.13	1.005	1959.39	6922.34	3.53
Average			**1.0081**			**4.932**

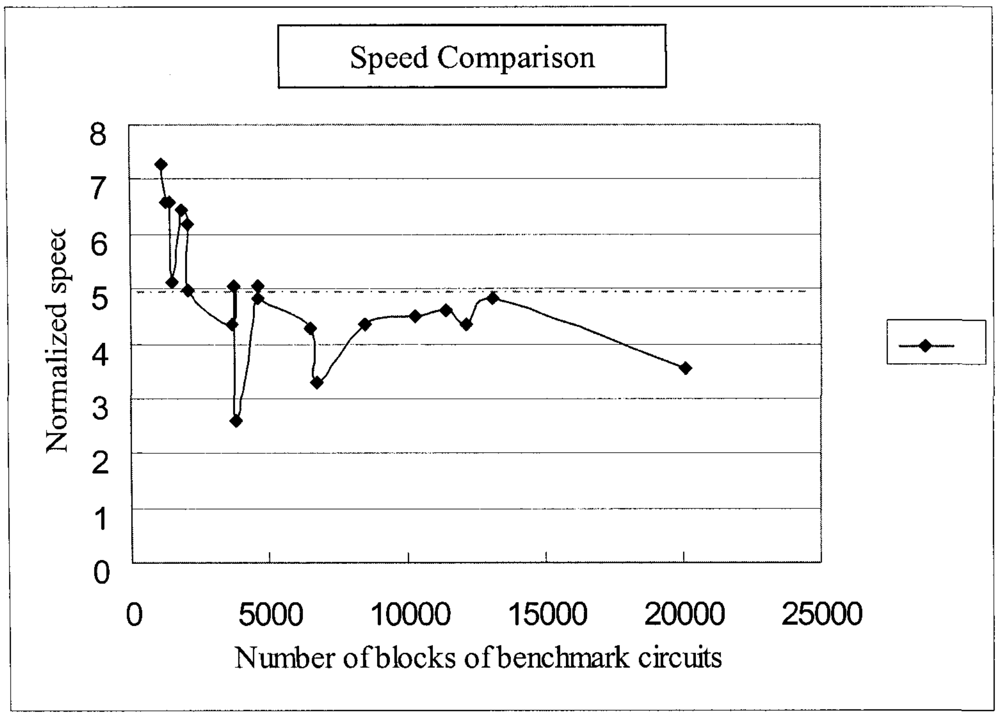


Figure 4.1 CPU Time Comparison vs. Number of Blocks

To further analyze the performance of our placement tool, we plotted the normalized (with respect to VPR) CPU time against circuit size, given by the number of blocks in the circuit. The plot is shown Figure 4.1. We notice that speed decreases as the size of the input circuit increases. Nevertheless, we get significant speedup even for the largest circuit (more than 3X). In order to know what exactly affects speed, we first analyze the time and quality relationship of the three stages in our QPF placement tool. The comparison of placement quality versus run time of QPF is shown in Figure 4.2. The values for this plot were obtained by averaging across 20 benchmark circuits. We select four typical points in our algorithm, they are: The first legal placement we get by the quadratic model, the output of stage 1 in our algorithm, the output placement of stage 2 and the final placement. In Figure 4.1, we also show the random placement for comparison. The random placement almost takes no time, but the wirelength is about 300% worse than the final placement. It takes about 8% of the total time to finish the initial placement based on the modified quadratic model and the wirelength is reduced to about 65%. Then the expansion technique improves the quality to about 45% with another 9% of total time. The linear adjustment of stage 2 in our algorithm spends another 11% of total time to further improve the wirelength to about 18% longer than the final placement. Finally, rest of the 72% of total time is used to get rid of the last 18% wirelength by the low temperature simulated annealing process.

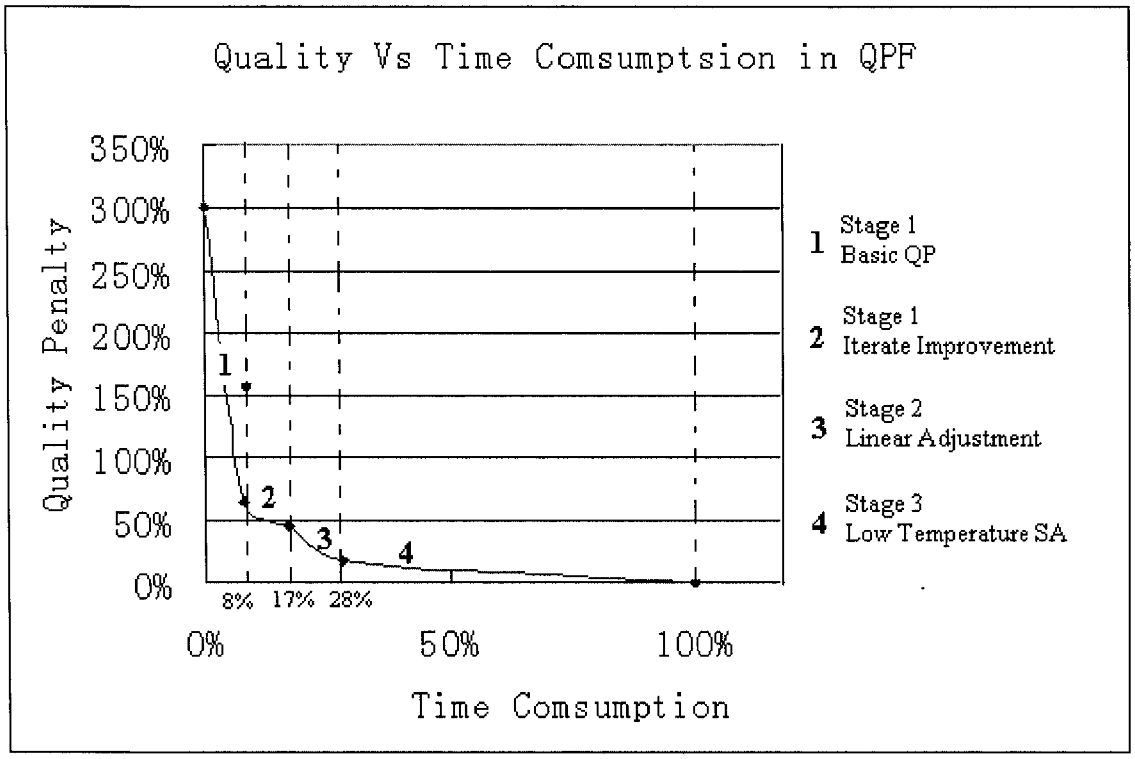


Figure 4.2. The Relationship between Normalized Wirelength Penalty and Time Consumption in QPF Algorithm

From Figure 4.2, we find that the most of the run time of our placement tool is used by the last refinement stage, which is the low temperature simulated annealing process. We believe the last refinement step is the reason that our placement tool becomes a little slower for larger circuits in Figure 4.1. As discussed earlier, the last refinement step uses simulated annealing technique, and the most important parameter is starting temperature T_0. Recall that we set it as $T_0 = \frac{C}{6907}$ in Chapter 3. For larger circuits, the cost C here usually is larger and the starting temperature is larger for these circuits. This results in more time in the last step. Although, by experiments, we can add some factor in our algorithm to make it work better for larger circuits, this factor should be carefully selected by testing large amount of circuits. In our algorithm we still use $T_0 = \frac{C}{6907}$ as the starting temperature and all the experimental data is based it.

4.4 Quality and Time Tradeoff

In this section, we present the quality versus run time tradeoff of for QPF and compare it with the VPR placement tool.

It is straightforward to perform the tradeoff test for VPR because both the time and quality are determined by InnerNum, which is the number of moves per temperature. For our placement tool, it is different. As discussed in Section 4.3, the performance of our placement tool is determined by three sages, and each stage uses a different technique. It is not likely to have an exact quality and time tradeoff over all these stages. Since the last stage of our algorithm also requires most of the run time, as shown in Figure 4.2, we test the quality versus time tradeoff based on the last stage. By setting different InnerNum for the simulated annealing part of our placement algorithm, we obtained a similar quality versus run time tradeoff for QPF and compared it with VPR, as shown in Figure 4.3.

Both algorithms can achieve faster speed at the some percentages of quality penalties. But VPR has a better tradeoff curve than QPF. When we focus on the better quality, QPF can achieve the same quality with much faster speed than VPR. When both algorithms are about 10 times fast, these two tools almost have the same performance. If we want to run faster, VPR is a better choice.

The reason is that we are doing the tradeoff only in the last stage of our algorithm and the maximum speed we can achieve is about 15 times faster, which means we do not use stage 3 at all. And the deeper reason behind this is in the quadratic model. For any placement tools based on quadratic technique, the basic operation is the equation solver. We have to modify and solve the linear equations to get a better placement. This basic operation takes a lot of time. While for simulated annealing based algorithms, the basic operation is one move of a node, and it can be done in negligible time.

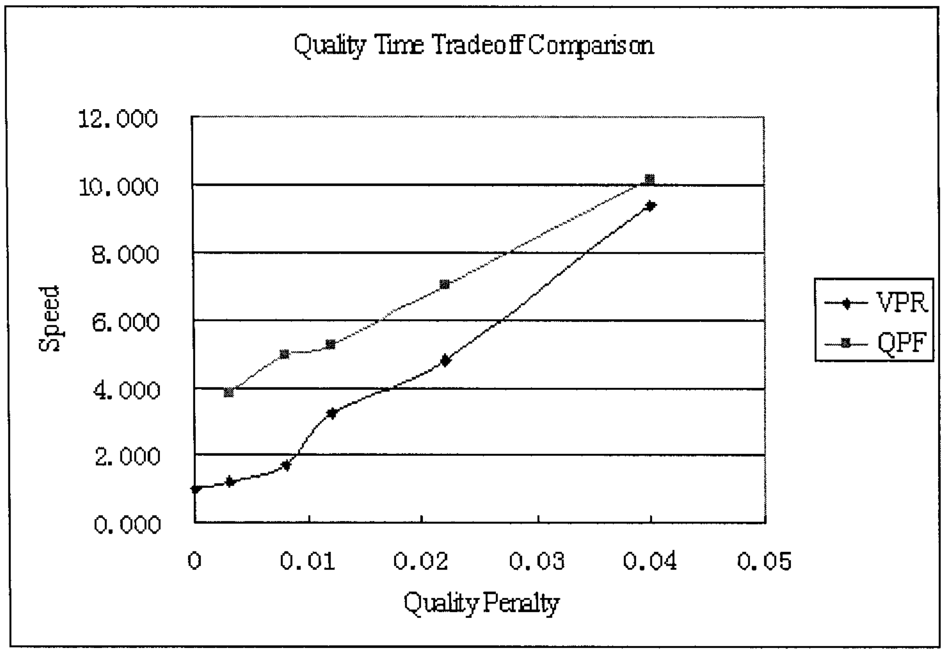


Figure 4.3 Quality vs. Time Tradeoff Comparisons between QPF and VPR

4.5 Critical Path Delay Comparison with VPR

Although we are mainly focusing on the wirelength minimization objective in our thesis, we also wanted to have an idea about the timing performance of our placement tool.

Through experiments, we found that the maximum critical path delay obtained by QPF is almost same as VPR in wirelength driven mode. Both are about 25% longer than the timing driven VPR (TVPR). We used 20 MCNC circuits for comparison.

We enhanced QPF by apply a timing driven refinement in stage 3 of our placement tool (we call it TQPF) and compared the critical path delay with timing driven VPR. The results for wirelength, critical path delay and runtime are shown in table 4.6. We find that with a timing-driven refinement in stage 3, our placement tool can achieve about 5% improvement in critical path delay than TVPR at the penalty of only 1.4% longer wirelength. TQPF still gives significant run time speedup over VPR (about 3X).

Table 4.6 Comparisons between QPF with Timing-driven Refinement and Timing-driven VPR

Circuit	Critical Path Delay (ns)		Ratio	Total wire length		Ratio	Time		Ratio
	TQPF	TVPR	TQPF/ TVPR	TQPF	TVPR	TQPF/ TVPR	T QPF	T VPR	TQPF/TVPR
Alu4	101.67	94.75	1.073	202.00	201.41	1.003	24	68	2.83
Apex2	105.98	127.00	0.835	285.52	281.13	1.016	39	125	3.21
Apex4	100.24	105.70	0.948	197.51	194.13	1.017	16	47	2.94
Bigkey	82.17	83.88	0.980	317.58	318.31	0.998	48	132	2.75
Clma	199.75	238.04	0.839	1563.27	1544.2	1.012	952	2697	2.83
Des	119.45	115.35	1.035	400.01	395.66	1.011	44	124	2.82
Diffeq	62.44	70.40	0.887	180.83	180.40	1.002	29	82	2.83
Dsip	70.98	81.56	0.870	312.25	310.61	1.005	33	89	2.70
Elliptic	116.57	116.21	1.003	617.49	614.85	1.004	215	555	2.58
Ex1010	199.21	201.08	0.991	681.21	678.30	1.004	288	868	3.01
Ex5p	103.68	100.31	1.034	189.24	186.66	1.014	13	39	3.00
Frisc	134.85	136.89	0.985	654.05	627.49	1.042	205	585	2.85
Misex3	98.08	100.96	0.971	201.91	199.47	1.012	19	56	2.95
Pdc	183.36	217.89	0.842	986.73	955.37	1.033	323	975	3.02
S298	132.51	142.73	0.928	233.32	229.07	1.019	43	107	2.49
S38417	100.63	99.46	1.012	763.44	725.79	1.052	591	1540	2.61
S38584.1	183.92	193.19	0.952	856.33	852.81	1.004	607	1716	2.83
Seq	105.19	116.54	0.903	281.19	274.22	1.025	37	103	2.78
Spla	158.65	179.39	0.884	671.72	681.63	0.985	187	615	3.29
Tseng	59.09	56.73	1.042	126.95	124.94	1.016	13	40	3.08
	Average		**0.9506**			**1.014**			**2.87**

The reason for this also comes from the quadratic model. As we discussed in Chapter 3, quadratic model tends to suppress very long and very short wires as in Figure 3.2. This property gives a good timing feature to the coarse placement that is obtained in stage 1 of QPF. When we use the timing driven simulated annealing refinement in stage 3, this characteristic is preserved. More discussion about timing driven quadratic placement is presented in the next chapter.

4.6 Summary

In this chapter, we presented the experimental results obtained from our QPF placement algorithm. We first described the experimental evaluation environment in Section 4.1. Then the experimental data was presented and analyzed that showed the effects of different techniques used in QPF on placement quality. A comparison of QPF with VPR and related analysis was presented in Section 4.3. The tradeoff of quality versus run time was discussed and compared with VPR in Section 4.4. This was followed by the critical circuit path delay comparison between TQPF and TVPR in Section 4.5. The experimental results show that QPF runs faster for wirelength driven placement while providing comparable quality and TQPF also gives better critical path delay and faster run time compared to TVPR.

In the next chapter, we will summarize the main contributions of this thesis work and discuss open problems that need to be explored in future research on this topic.

Chapter 5

Conclusions and Future Work

5.1 Conclusions and Contributions

In this thesis we presented QPF, an efficient placement algorithm for FPGAs. It combines the quadratic placement algorithm and the simulated annealing placement algorithm to achieve both high quality and fast speed. We incorporated multiple iterations of equation solving process together with linear wirelength reduction techniques in our algorithm, which help to alleviate the problems caused by quadratic model. We refine our placement with low temperature simulated annealing technique, which help to produce a high quality result. By applying these techniques intelligently, our placement algorithm takes the advantages of quadratic and simulated annealing placement algorithms and produces a fast and high quality placement for FPGAs.

The first contribution of this work is the exploration of quadratic technique in the FPGA placement area. We know that quadratic placement technique has been used in ASIC area for a long time. But because of the different architectures, ASIC placement tool cannot be directly used in FPGA placement. To our knowledge, our work is the first investigation of quadratic placement in the area of FPGAs.

The second contribution of this work is the use of compensation factor in quadratic model. Half perimeter wire length is an efficient model to estimate the wirelength of the placement, but it usually underestimates the large nets. The old quadratic model does not consider this situation. By applying the compensation factor when building the linear equations, the solution of the linear equations carries the information of bounding box wirelength and is closer to the ideal placement.

The third contribution of this work is the iterative expansion technique. One of the main problems with quadratic placement is the node overlap problem of linear equations solution. We handle this problem with an iterative approach. Each iteration removes the overlap a little bit according to the reference placement we got in the previous iteration. This technique avoids adding artificial information to the circuit and preserves original nodes relationship. In each iteration, since we get new coordinates information by re-building and re-solving the linear equations, the original squared wirelength objective is always minimized as well.

We also give a linear adjustment technique to reduce the wirelength based on the half perimeter wire length model. The minimization of total linear wirelength is the final objective for wirelength driven placement algorithm. Even for timing driven placement, linear wirelength is also an important factor. We studied the node distribution in practical placements and proposed the linear wirelength reduction technique. It is very fast, and since it only uses the position information of the circuit nodes it can be applied to any existing placement.

Finally, an attempt was made to extend our wirelength driven placement algorithm to a timing driven placement algorithm. Preliminary results are very promising.

5.2 Future Work

Our work is the first attempt to apply quadratic placement technique in FPGA placement area. There are a lot of things within this topic that need to be explored thoroughly.

The most interesting topic that should be studied is the timing driven placement based on quadratic technique because the circuit delay is another important objective for placement algorithm, and much work remains to be done in this area. A direct approach that can be done based our work is to add the timing information in our linear adjustment technique. In our work, we further reduce the wirelength by searching all nodes of the input circuit and trying to find the movable nodes and move them to better position. In current algorithm we do this only focusing on the wirelength reduction. If we perform timing analysis before this and take the timing information into account, it is possible to reduce the wirelength as well as the circuit delay during this process. More research is needed to find out how to incorporate the timing information into the quadratic model.

Another interesting area to pursue is to further improve quality of quadratic placement technique. Since quadratic model use indirect approach to minimize the wirelength objective, how to improve the quality of the placement is always the main problem. Although researchers have proposed different techniques, when compared with simulated annealing based algorithm, the quality obtained is not as good. If we can achieve similar quality without simulated annealing process, the runtime can be further reduced.

Finally, since quadratic placement technique involves linear equation solver, mathematical techniques [29][36] can be employed to further reduce the program runtime.

References

[1] Steven M. Rubin, "Computer Aids for VLSI Design", Addison-Wesley Publishing Company, 1994.

[2] Daniel Mlynek, "Design of VLSI System", Integrated Systems Center, Swiss Federal Institute of Technology, 1998.

[3] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGAs", FPL 1997, pp. 213 - 222.

[4] A.Marquardt, V. Betz, and J. Jose, "Timing-Driven Placement for FPGAs", Intl. Symposium on FPGA February 2000, pp. 203 – 213.

[5] Y. Sankar and J. Rose, "Trading Quality for Compile Time: Ultra-Fast Placement for FPGAs", Proc. of 7th ACM/SIGDA Intl. Symposium on FPGAs, 1999, pp. 157-166.

[6] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Based FPGA Designs", IEEE Trans on CAD, January 1994, pp. 1-13.

[7] G. Parthasarathy, M. Marek-Sadowska, A. Mukherjee, and A. Singh, "Interconnect Complexity-aware FPGA Placement Using Rent's Rule", Proc. of System Level Interconnect Prediction, March 2001.

[8] W. Swartz and C. Sechen, "Timing-Driven Placement for Large Standard Cell Circuits", DAC 1995, pp. 211-215.

[9] C. Ebeling, L. McMurchie, S. A. Hauck and S. Burns, "Placement and Routing Tools for the Triptych FPGA," IEEE Trans. on VLSI, 1995, pp. 473-482.

[10] K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques," ACM Computing Surveys, vol. 23, no. 2, 1991, pp. 143-220.

[11] T. Lengauer, "Combinatorial Algorithms for Integrated Circuit Layout", Chichester: John Wiley & Sons, 1990.

[12] S. M. Sait and H. Youssef, "VLSI Physical Design Automation", IEEE Press, New Jersey, 1995.

[13] S. Areibi, M. Xie, and A. Vannelli, "An Efficient Rectilinear Steiner Tree Algorithm for VLSI Global Routing", Canadian Conference on Electrical and Computer Engineering, May 2001.

[14] J. B. Kruskal, "On the Shortest Spanning Tree of a Graph and The Traveling Salesman Problem", Proc. of The American Mathematical Society 7, 1956, pp. 48-50.

[15] R. Prim, "Shortest connection networks and some generalizations", Bell System Technical Journal, pp. 36:1389-1401, 1957.

[16] C. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling", Proc. of ICCAD, 1994, pp. 690-695

[17] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuit", IEEE Trans. on CAD, vol.4, no.1 1985, pp. 92-98

[18] A. Caldwell, A. Kahng and I. Markov, " Can Recursive Bisection Produce Routable Placement?", Proc. of DAC' 00, 2000. pp. 477-482.

[19] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," Bell System Technical Journal, Vol. 49, Feb 1970. pp. 291-307.

[20] C. M. Fiducia, R. M. Mattheyses, "A Linear Time Heuristic for Imporving network Partitions", Proc. of 19th IEEE/DAC, pp. 175-181.

[21] D. J-H. Huang and A.B. Kahng, "Partitioning-based Standard-cell Global Placement with an Exact Objective", Proc. of ACM/IEEE ISPD, 1997, pp. 18-25

[22] M. Wang, X. Yang and M. Sarrafzadeh, " DRAGON2000: Standard-Cell Placement Tool for Large Industry Circuits", Proc. of ACM/IEEE ICCAD, 2000, pp. 260-263.

[23] P. Maidee, C. Ababei and K Bazargan, "Fast Timing-driven Partitioning-based Placement for Island Style FPGAs", Proc. of DAC 2003, pp. 598-603.

[24] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," IEEE Journal of Solid-State Circuits, vol. 20, no. 2, Apr. 1985, pp. 510-522.

[25] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An Efficient General Cooling Schedule for Simulated Annealing," Proc. of ICCAD, 1986, pp. 381-384.

[26] J. Lam and J. Delosme, "Performance of a New Annealing Schedule," Proc. ACM/IEEE DAC, 1988, pp. 306-311.

[27] W. Swartz and C. Sechen, "New Algorithms for Placement and Routing of Macro Cells," Proc. Intl. Conference on Computer-Aided Design, 1990, pp. 336-339.

[28] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel Hypergraph Partitioning: Application in VLSI domain", Proc. of ACM/IEEE DAC, 1997, pp. 526-529.

[29] C. J. Alpert, T. Chan, D. J.-H. Huang, I. Markov, and K. Yan, "Quadratic Placement Revisited", Proc. of ACM/IEEE DAC 1997, pp. 752-757.

[30] J. M. Kleinhans, G. Sigl, F.M. Johannes, "Gordian: A New Global Optimization/ Rectangle Dissection Method for Cell Placement", Proc. of ICCAD,1988, pp. 506-509, 1988

[31] I.I. Mahmoud, K. Asakura, T. Nishibu and T. Ohtsuki, "Experimental Appraisal of Linear and Quadratic Objective Functions Effect on Force Directed Method for Analog Placement", IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, E77-A(4). April 1994, pp. 719-725

[32] G. Sigl, K. Doll, F. M. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?", ACM/IEEE DAC 1991, pp. 427-432.

[33] K.M. Hall, "An r-dimensional Quadratic Placement Algorithm," Management Science 17 (1970), pp. 219-229.

[34] H. Etawil, S. Arebi, and A. Vannelli. "Attrctor-repeller Approach for Global Placement" In Proc. IEEE/ACM ICCAD, 1999. pp. 20-24.

[35] N. Viswanathan, C. Chu, "FastPlace: Efficient Analytical Placement Using Cell Shifting, Iterative Local Refinement and a Hybrid Net Mode", Proc. of ISPD 2004. pp. 26-33.

[36] R. Barrett, M. Berry, and et al. "Templates for The Solution of Linear Systems: Building Blocks for Iterative Methods", SIAM, 1994

[37] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing", Science, vol. 220, no.4589, May 13, 1983, pp. 671-680

[38] Garey, M.R. and Johnson, D.S. "Computers and Intractability: A Guide to the Theory of NP-Completeness". New York: W.H. Freeman, 1983.

[39] O. Axelsson, A. Barker, "Finite Element Solution of Boundary Value Problems. Theory and Computation", Academic Press, Orlando, FL. 1984

[40] W. Hackbush, "Iterative Solution of Large Sparse Systems", Springer Verlag, 1994.

[41] A. Ramage, "An Introduction to Iterative Solvers and Preconditioning Techniques", PIMS Workshop 2003.

[42] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0", Tech. Report, Microelectronics Centre of North Carolina, 1991.

[43] M. Hutton, J. Rose, and D. Corneil, "Generation of Synthetic Sequential Benchmark Circuits", Proc. of 5th ACM/SIGDA Intl. Symposium on FPGAs, 1997, pp. 149-155.

[44] E. M. Sentovich et al., "SIS: A System for Sequential Circuit Analysis", Tech. Report No. UCB/ERL M92/41, University of California, Berkeley, 1992.

[45] C. J. Alpert, T. F. Chan, D. J.-H. Huang, A. B. Kahng, I. L. Markov, P. Mulet and K. Yan, "Faster Minimization of Linear Wirelength for Global Placement", Proc. of ISPD 1997, pp. 4-11.

[46] R. Tsay, E, Kur and C. Hsu, "Module Placement for Large Chips Based on Sparse Linear Equations", Int. J. Circuit Theory Appl 16, pp. 411-423

Appendix A

Basic Data Structures

Structure s_net and s_block are the main data structures used by VPR. They contain all the connectivity information of the input circuit. VPR parses the input circuit and saves the connectivity information in these two data structures by using hash table.

```
struct s_net {char *name; int num_pins; int *blocks; int *blk_pin; };

/* name:   ASCII net name for informative annotations in the output.          */
/* num_pins:   Number of pins on this net.                                    */
/* blocks: [0..num_pins-1]. Contains the blocks to which the pins of this     */
/*          net connect.   Output in pins[0], inputs in other entries.        */
/* blk_pin: [0..num_pins-1]. Contains the number of the pin (on a block) to   */
/*           which each net terminal connects.   Since I/O pads have only one */
/*           pin, I set blk_pin to OPEN for them (it should only be used for  */
/*           clb pins).   For clbs, it is the block pin number, as expected.  */

struct s_block {char *name; enum e_block_types type; int *nets; int x; int y;};

/* name:   Taken from the net which it drives.                                */
/* type:   CLB, INPAD or OUTPAD                                               */
/* nets[]:   List of nets connected to this block.   If nets[i] = OPEN       */
/*             no net is connected to pin i.                                  */
/* x,y:   physical location of the placed block.                              */
```

Structure q_block is the main data structure used by QPF to store the connectivity information of the circuit. We initialize this data structure by searching through the two data structures of VPR and converting the hyper-graph into graph by clique net model.

```
struct q_block {char* name; int orig_blk_num; int x, y; int edge_num; int clb_edge_num;
                int  io_edge_num;     int* q_blk_list; float* q_blk_weight; float av_weight;
                };

/* name: Block nam.                                                          */
/* orig_blk_num: the origin block number in VPR                              */
/* x,y:   physical location of the placed block.                             */
/* edge_num: number of edge of this node                                     */
/* clb_edge_num: number of edges connected to CLBs                           */
/* io_edge_num: number of edges connected to IO pads                         */
/* q_blk_list[]: nodes connected to this node                                */
/* q_blk_weight[]: weight of the nodes connected to this node                */
/* av_weight: average weight of this node                                    */
```

VITA AUCTORIS

Yonghong Xu graduated from Zhejiang University in P.R.China, where he obtained B.Sc in June 1995 and M.Sc in March 1998 in Electrical Engineering. And afterward he worked for Alctel Shanghai Bell till 2002. He is currently a candidate for the Master's degree in Electrical and Computer Engineering department at the University of Windsor and hopes to graduate in June 2005.