University of Cincinnati

Date: 3/26/2014

I, Tuhin Mukherjee , hereby submit this original work as part of the requirements for the degree of Master of Science in Electrical Engineering.

It is entitled:

Cluster Shaping: A novel optimization technique for large scale VLSI placement

Student's name: **Tuhin Mukherjee**

This work and its defense approved by:

Committee chair: Ranganadha Vemuri, Ph.D.

Committee member: Wen Ben Jone, Ph.D.

Committee member: Carla Purdy, Ph.D.

UNIVERSITY OF Cincinnati

9318

Cluster Shaping: A novel optimization technique for large scale VLSI placement

A thesis submitted to the

Graduate School

of the University of Cincinnati

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Department of Electrical and Computer Engineering

of the College of Engineering and Applied Sciences

March 2014

by

Tuhin Mukherjee

B. Tech (Computer Science and Engineering)

West Bengal University of Technology

April, 2011

Thesis Advisor and Committee Chair: Dr. Ranga Vemuri, Ph.D

ABSTRACT

The process of VLSI placement has been under constant evolution since its early days when the number of cells was ~100 to modern designs containing ~50 million cells with the process technology approaching ~7nm. We have presented a new optimization technique that increases the efficiency of existing clustering methods in placement and can place extremely large designs within a reasonable amount of time without sacrificing on the quality of solution. We have evaluated our technique based on the total wirelength (HPWL) and the timing values such as total negative slack (TNS) and worst negative slack (WNS) produced after completion of placement and routing. We have used real designs for the purpose of evaluation after converting them into the bookshelf format which is the standard format used by most academic placers today. The main concept behind our research was to extend the creation of large clusters in such a way that they could have multiple variations in shape instead of just squares. This was done to allow more flexibility in the placement of cells inside the clusters. After generation of multiple shapes an efficient selection procedure was implemented to get the best shape for a cluster from the several variations. This selection procedure consisted of assigning a shape to all the clusters and placing them in their optimal locations. Following this the cost of the placement solution was evaluated by taking into account the external and internal half perimeter wire length (HPWL) values along with the overlapping area in the design. We have used state-of-art placement tools to place the cells inside a cluster for all its variations. Synopsys DA tools were used for the purpose of evaluating our final performance. We have found from our experimental results that our shaping technique has improved the total wirelength by 8% on an average when compared to LCPlace [2] across all the benchmarks at the cost of a slight increase in runtime.

To my loving parents for inspiring and encouraging me in all my ventures

ACKNOWLEDGEMENTS

With hard work and God's grace we can achieve anything in life. Thanks to the almighty for providing me the strength and surrounding me with so many wonderful people. I would like to express my gratitude and thank Dr. Ranga Vemuri for his guidance and support which helped me to conclude my work efficiently. His advice at every step of this research guided me in the right direction to achieve desired results. It was a privilege to be a part of DDEL and work with you. I would also like to thanks Dr. Wen Ben Jone and Dr. Carla Purdy for being a part of my committee.

I would like to thank my friend and lab mate Nakul Tirumalai for helping me create an initial framework to continue my research on. This thesis is actually a result of extension of his work. His continuous support helped me to debug lot of issues that popped up often during my work. Our numerous discussions also helped me improve my coding skills. I would also like to thanks Ujwal Ramesh for writing scripts that helped to speed up my work.

I would also like to thank my friends Mike, Harsha, Richa, Nikhil, Lava, Mithun, Arun, Ramesh and Shaun for making the lab an exciting place to work in. Special thanks to Suryadip Chakraborty for inspiring and helping me throughout the duration of my work. Mike, thank you for the morale boosters that increased my determination to continue this work.

Finally I would like to thank my parents and my entire family for standing with me through my difficult times and supporting me throughout my life.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

Chapter 1 INTRODUCTION

The VLSI design process is an aggregation of various steps that begin from a concept or idea of an electrical device to perform a range of functions and end with the fabrication of an integrated circuit to do the same. This design cycle includes but is not limited to creation of the RTL for the design, logic synthesis, floorplanning, placement, clock tree synthesis, routing, static timing analysis and finally fabrication of the chip on silicon die. Each of the steps mentioned above is really complex in modern IC design cycles and consists of many sub-steps that aid in optimization and testing of the performance of the integrated circuit at each step. An overview of the entire design cycle is given below in fig 1.1. Detailed descriptions of each step of this design flow can be found in [38].

1.1 Motivation

Since the initial days of VLSI design automation, there has been a lot of effort invested on finding new methods to optimize the various steps in the design cycle. Most of these efforts were rewarded with success and an enormous amount of optimization in the design of integrated circuits was witnessed in the last few decades. Along with the increase in optimization of traditional design flows, problems related to larger designs also increased significantly. Moore's law had predicted the reduction in size of the transistors. With smaller sizes, the number of transistors that could be fabricated on a single chip increased exponentially. This increased the complexity of various stages like placement, routing and so on in the IC design flow.

1.1.1 Placement optimization

Placement of circuit devices or cells has always been one of the most critical steps in the entire design cycle. In modern designs the total number of cells or gate-count is ~50 million. With such an extremely large number of cells, traditional placement techniques fail to perform optimally and result in issues such as routing congestions, thermal hotspot creation and so on. As a result various problem size reduction techniques were implemented to reduce the large problems into smaller chunks and then solving them with existing placement techniques.

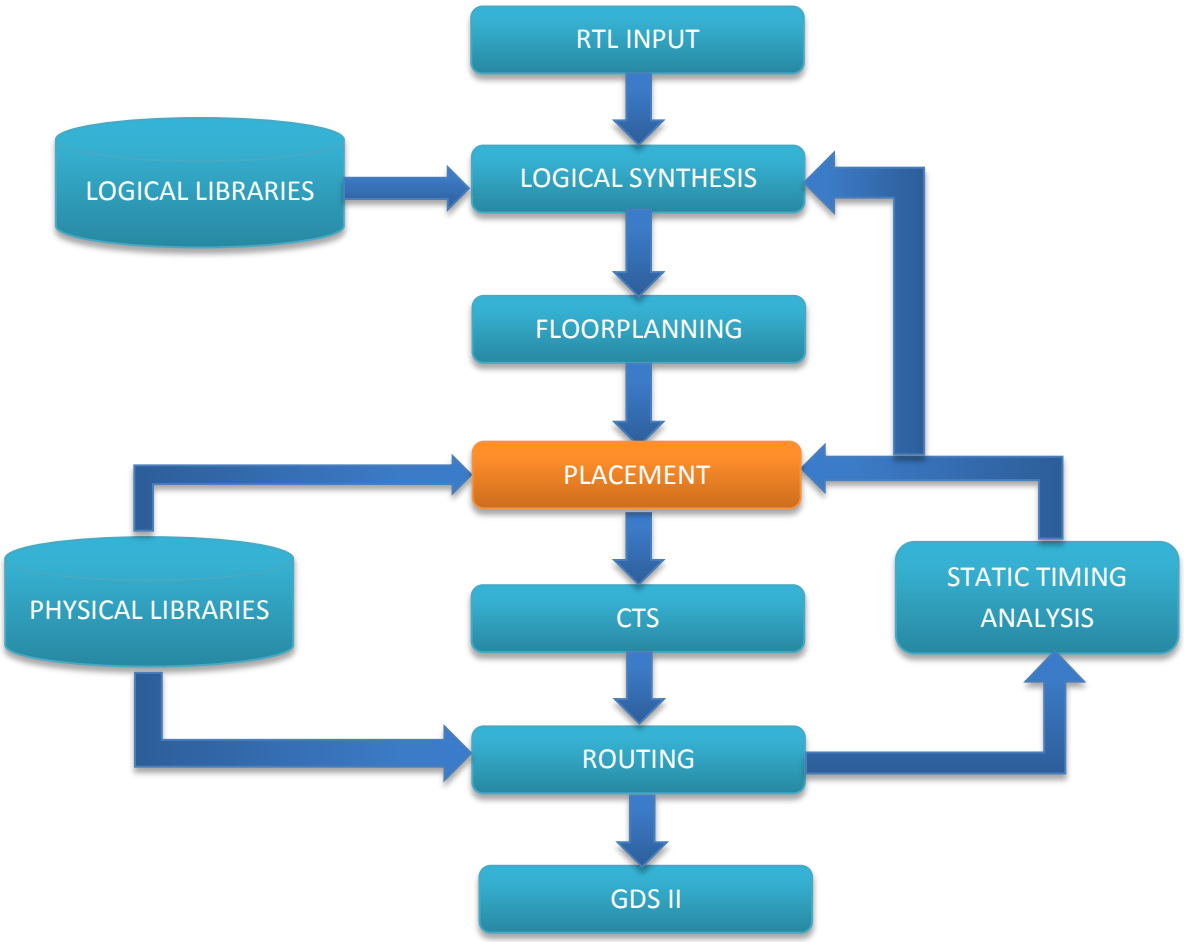


Figure 1.1: Integrated Circuit design flow

Clustering is one such problem reduction technique to handle large designs. Details about reduction techniques and clustering are given in chapter 2 of this thesis. These clustering techniques have some inherent drawbacks which cause degradation in placement quality while decreasing runtime for such techniques.

1.1.2 Major focus areas of our research

In our research we focus mainly on improving the clustering technique that results in the formation of large clusters as mentioned in [2]. We have seen from the experimental results of LCPlace [2] that the improvements in wirelength after global placement are lost to a large extent after the process of unclustering and legalization is executed. In our work we have proposed a novel optimization technique to prevent this degradation in quality of placement by changing the shapes of the clusters and performing mixed size cluster placement. The key contributions to our research are the following:

1. **Creation of 'shape-bank' for all clusters:** Instead of fixing the shape of the large clusters formed using [2] to squares, we created multiple variations of the square shape for each cluster and stored them in the 'shape-bank' for the cluster.
2. **Perform 'in-cluster' placement:** After creation of multiple shapes, the cells belonging to a cluster had to be placed efficiently inside each of the clusters since the number of cells in each cluster was quite large. A flow was created to fix some cells on the boundary of the cluster for all the shape variations and using these cells as anchor points, an external academic placer was used to place the remaining cells inside all the shapes for the clusters.
3. **Selecting the best shape for a cluster:** The best shape had to be selected for a cluster from its 'shape-bank' based on certain objective functions. We first tested our shaping technique using a greedy approach to observe the impact on the internal wirelength of

the clusters and found improvements. Then a simulated annealing based approach was used to decide the best shape for a cluster after physically assigning the shape and moving the clusters to their optimal locations.

4. **Model clusters as movable macros and use mixed sized placer:** In each round of shape selection a set of shapes were assigned to each of the clusters and then they were modelled as movable macros and written out in the bookshelf format. Then a mixed size academic placer was used to place them in their optimal locations.

1.2 Thesis Organization

The rest of this thesis is comprised of five chapters. In chapter 2 we provide some background on VLSI placement in general and clustering techniques that exist in the literature to handle the problems of placement for large designs. In chapter 3 we propose our novel shape generation technique for creating variations from square clusters. Chapter 4 deals with the process of selecting the best shape for the cluster and placement of mixed size clusters. In chapter 5 we present a description of all the experiments performed along with the corresponding results and analysis. Chapter 6 we summarize our work on this thesis and draw conclusions. We also provide insights for future research along these lines.

Chapter 2 VLSI PLACEMENT AND CLUSTERING

In this chapter we define the placement problem and discuss the various approaches to perform VLSI placement. We also provide some background on clustering techniques used to solve the placement problem. Finally this chapter concludes with a review of the clustering technique we have used in this thesis as a precursor to our shaping technique mentioned later.

2.1 The VLSI placement problem

2.1.1 What is placement?

The process of finding the optimal locations of circuit elements, also known as cells, on a die surface is known as VLSI placement in the integrated circuit (IC) design cycle [11]. As we have seen in chapter 1, placement is the step in the design cycle which generally comes after logic synthesis has been performed on the Verilog/VHDL source i.e. the RTL models and before the routing phase. Placement is a very crucial step because the quality of outputs produced in the following steps (i.e. clock tree synthesis, routing, etc.) depends on the quality of the placement solution provided as an input. Placement is one of the key factors that affect interconnect length, routing congestion, thermal hot-spot creation, overall performance of the circuit, and so on.

2.1.2 Formulation of the placement problem

The placement problem can be defined as follows: Given a chip layout or floorplan and a set of circuit elements (cells) containing terminals or pins that are used to establish a connection between two or more cells, determine the positions of those cells in such a way that no two cells overlap with each other and a specific objective function is optimized. The inputs provided are, (1) a set of cells, ‘C’ along with their physical dimensions and positions of pins on the cells, (2) a netlist containing information about how the cells are interconnected among each other, and

(3) the floorplan information specifying the dimensions, spacing and orientation of the feasible locations where cells could be placed on the chip. [12] In this thesis we deal with only two-dimensional placement problems, the output of which are the x and y co-ordinates representing the optimal locations for all cells.

2.1.3 Performance metrics to evaluate placement

Generally the objective function(s) that need to be optimized are the total wirelength and the timing values like TNS, and WNS, which are calculated after performing a static timing analysis (STA) on completion of routing. These two metrics are the most significant parameters that are used to evaluate the quality of our placement solution. Other cost functions such as routing congestion, heat and power distribution, total area, etc. may also be included in the cost function to evaluate the final placed netlist.

2.2 Types of VLSI placement

VLSI placement can be broadly divided into two categories based on the specifications of the circuit devices that need to be placed. The following sections explain the two categories of placement

2.2.1 Standard cell placement

Initial research on placement assumed the height of the cells to be fixed. The only variation allowed was in the width of these cells which are also known as standard cells. This assumption simplified the placement problem to a certain extent and this initiated the practice of creating rows with equal heights in the floorplan. Now the standard cells could only be placed in such a way that they align with these rows. This simplification of the placement problem helped in improving the runtime of placers to a large extent. The method of placing standard cells only on the rows defined in the floorplan is known as row-based placement. Due to the popularity of this

technique, most of the modern placers still use row definitions as a guide to optimizing the placement along with other methods.

2.2.2 Mixed size placement

Due to the increasing complexity of modern designs it not always possible to have all the circuit devices in the form of standard cells. Special components like analog circuits are designed using flows dedicated to the purpose and are used as fixed blocks in the design. These blocks are pre-synthesized before integrating them in the traditional IC design cycle. The increasing number of SoC (System on Chip) in modern designs have spiked up the need to use such pre-designed blocks or macro blocks. These blocks are reused in a lot of designs and are instantiated wherever required. Macro blocks can be movable or fixed. Fixed macros have to be treated as blockages in this category of placement problems whereas movable macros can be moved around freely without changing their shape or orientation. [11]

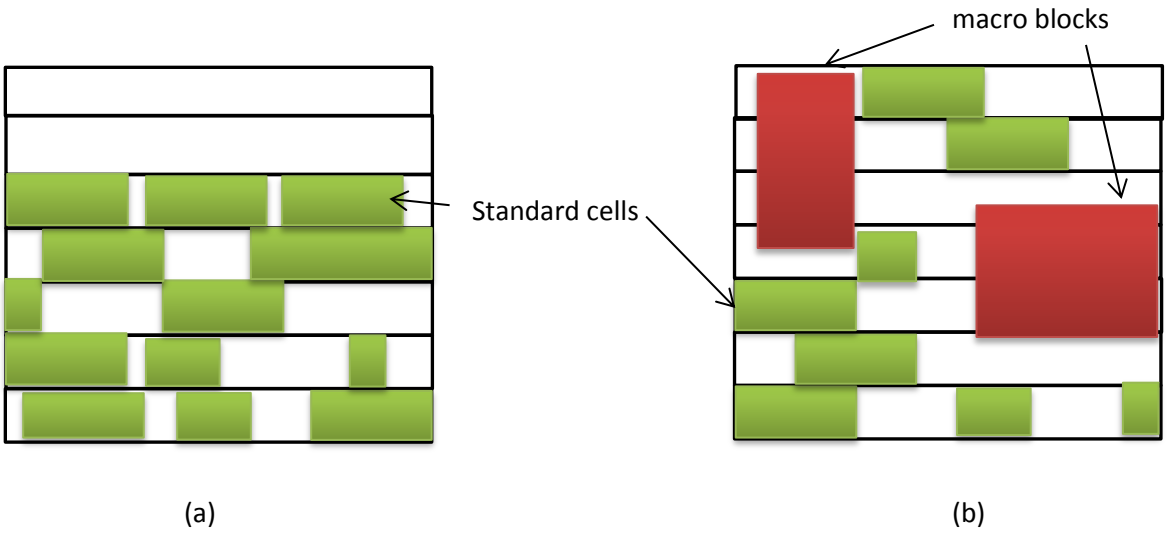


Figure 2.1 (a) Standard cell placement, (b) mixed size placement

2.3 Generic placement flow

Figure 2.2 Generic placement flow

The generic placement flow as shown in fig 2.2 is used by most academic and industrial placers today for both categories of placement mentioned above in section 2.2. Each of the blocks in the flow is elaborately explained in the following sections.

2.3.1 Global placement

In the process of global placement all the cells are placed approximately in their optimal locations without considering any overlap that may have occurred. Global placement includes processing of all the cells together and allocating a location for each of them on the die. This is done by optimizing some objective function which takes into account, a global view of the placement. Standard cells are distributed equally across the entire die to an extent possible during global placement. The execution time for this stage is quite large compared to the other stages in the placement flow. So, any further optimization during this step would help to reduce the total execution time of a placer by a large extent.

2.3.2 Legalization

On completion of the global placement step in the flow, overlaps may exist among the cells. Such overlaps are not desired and would pose a lot of difficulties for the steps that follow placement. So, all overlaps are removed during this step called legalization. A legalizer removes cell-cell, cell-macro, and macro-macro overlap by shifting the movable blocks/cells by the least

amount necessary. Cells should not be moved around a lot as it may result in the degradation of objective function. A lot of research has been done on various legalization techniques. Some of them are found in [13], [14] and [15]

2.3.3 Detailed placement

Detailed placement tries to optimize the placement solution by using local refinement techniques. All detailed placement techniques work with a small group of cells at a time and try to improve the objective function by performing local changes among those cells like cell swapping, flipping of cells about an axis, and so on. The main objective of detailed placement is to regain some placement quality which may have been lost during the legalization step while cells were being moved around. The execution time for each detailed placement step is much less when compared to those of global placement. The reason for this being that detailed placement always work with only a small window of cells.

2.4 Existing global placement techniques

Global placement techniques existing in the literature include partitioning-based [16, 17, 18], simulated annealing based [19, 20], and analytical approach based on finding the minimum force location of cells [3, 6, 21, 22]. A brief description of all these placement techniques is given below.

2.4.1 Simulated annealing based placement

Simulated annealing (SA) is the process of finding a close estimate of the globally optimum solution for any combinatorial optimization problem. SA is used in all situations where a good enough solution is acceptable instead of the best solution possible by decreasing the time taken to solve it. This algorithm was developed by Kirkpatrick, Gelatt and Vecchi [23] and is widely used in a lot of areas of electronic design automation. This algorithm was inspired from the

process of annealing in metallurgy. In VLSI placement the solution space for this process consists of various placement configurations of the cells. Iterating between different placement configurations consist a move in Metropolis procedure. Cost function required to evaluate the solution can be any of those mentioned above in section 2.1.3. In the initial stages of annealing solutions that degrade the cost are also accepted to allow hill climbing. This is required to reduce the probability of getting stuck at local minima/maxima. The probability of accepting inferior solutions decreases as the temperature cools down and eventually a good enough solution is reached.

2.4.2 Partitioning based min-cut placement

In partitioning based approaches the entire process first divides all the cells into two regions [24] or into four regions [25]. Then each of these regions is further divided into two or four regions. This process continues recursively until each of the partitions created contains the minimum number of cells that can be assigned to the smallest partition. Another step that is executed while using a partitioning based min-cut placer is terminal propagation. [18] This affects the quality of the final placement solution. Partitioning based placement techniques have a lot of limitations while modelling the various types of objective functions mentioned earlier. Although the placement solution produced using this method is better routable and can also be scaled up for larger designs. [2]

2.4.3 Analytical placement techniques

In any analytical placement technique, the objective functions for placement and the constraints are modelled in the form of analytical functions of the coordinates of cells [26]. The entire VLSI placement problem is modelled as a set of mathematical equations which when solved using any existing solver will yield the optimal locations for all the cells on the die. One of the necessary conditions for the analytical approach to work efficiently is that the objective function

that has to be optimized should be continuous, convex and smooth. A brief description of some of the most popular analytical placement techniques is given in the following sections.

2.4.3.1 Quadratic placement

Quadratic placement techniques are so named because the objective function that has to be optimized here has a degree of 2. The quadratic wirelength model is the summation of the squares of the net lengths [2]. Equation 2.1 below shows the model of quadratic wirelength used in these placement techniques.

$$f = \sum_{e \in E} \left(\sum_{v_i, v_j \in e} (x_i - x_j)^2 + \sum_{v_i, v_j \in e} (y_i - y_j)^2 \right) \qquad \text{Equation 2.1}$$

In quadratic placement technique, all the multi terminal nets in the design have to be modelled as two terminal nets. This is achieved by using the clique, star or the hybrid model [6]. More details about the net models can be found in [26]. It has been shown that nets which have large fan-outs should be modelled as a star instead of a clique. A quadratic placer is really fast in terms of runtime required because what it essentially has to do is solve a system of linear equations to minimize the objective function (wirelength). Constraints like blockages, chip boundaries and so on increase the execution time. The most significant disadvantage of using a quadratic placer is the large amount of overlapping area that is produced in the final placement. This has to be removed using a legalizer. An example of an academic placer that uses quadratic optimization techniques to perform VLSI placement is FastPlace [6], [28].

2.4.3.2 Non-linear placement

As the name suggests a non-linear placement technique is one in which the method to model the wirelength of nets is non-linear. It is easier to model the various types of constraints into one objective function using a non-linear solver unlike that of a quadratic solver. One existing

academic placer using non-linear solving procedures is NTUPlace 3.0 [3]. The major drawback of this solving technique is the execution time. It is also quite complex from the implementation perspective of the solver. Further details on non-linear solving techniques can be found in [29]

2.5 Clustering in VLSI placement

2.5.1 Reducing problem size

The increasing size of combinatorial optimization problems led to the development of various problem size reduction techniques in the field of VLSI design automation (DA). A problem size reduction technique is defined as the process by which large problems are reduced into relatively smaller problems that can be easily solved by using the solvers existing in the literature. In VLSI DA, these reduction techniques are mainly used in the two major areas of partitioning and placement. In the following sections we present a survey of all the major reduction techniques used for placement of cells for ASIC designs.

2.5.2 Application in VLSI design automation

We can observe from the various placement techniques discussed above that the complexity of these techniques are dependent on either the total number of cells in the design (for e.g. $O(n^2)$ for force directed placement, where n is the total number of cells in the design) [6], or the number of nets or connections present in the design. So reducing either of the two or both can help in effectively placing large designs easily with the existing placement techniques. Almost all modern placers implement some sort of problem size reduction technique to handle the large designs of latest ASICs. Some initial work in literature on problem size reduction for partitioning can be found in [30]. Reduction techniques are used mostly with analytical placement methods

where reducing the number of nets or the number of cells will exponentially decrease the runtime of the solver.

2.5.3 Clustering to reduce problem size

Clustering in VLSI placement is the process of grouping a number of cells together in such way that it minimizes some objective function like the ones mentioned in section 2.5.2 above. Once all the cells have been grouped together into clusters, each of these clusters can be represented like a ‘virtual cell’ and treated similar to regular cells. The process of clustering can be explained in a simpler way using the graph data structures. A graph G consists of a set of nodes V and a set of edges E which connect nodes among themselves. Nodes represent cells in the placement problem and the edges represent the interconnections between cells. Since a net may connect to more than two cells so in the graph world these nets are represented using hyperedges and the resulting graph is known as a hypergraph. On clustering, this hypergraph gets modified to another hypergraph containing a set of clusters ‘C’ as its nodes and a new set of hyperedges that represent the residual nets after some of them got hidden due to clustering, More details on how clustering is applied to hypergraph data structures can be found in [31].

2.6 Clustering techniques

In this section we discuss the existing clustering techniques in a chronological order that are used for VLSI placement.

2.6.1 Edge-coarsening based clustering

In this technique a group of nodes are joined together to form a single node or vertex. There are a few modifications to the edge coarsening technique like hyperedge coarsening, modified hyperedge coarsening, etc. This technique is reviewed in further details in [31]

2.6.2 First choice clustering

This technique is a modification to the edge coarsening technique mentioned in 2.6.1 above. In first choice clustering technique the order in which the nodes are visited is random. All the vertices that are connected to a single vertex ‘v’ are considered and then the vertex connected to a hyperedge with the largest weight is selected to be clustered with the v. There is a possibility of formation of large clusters using this technique which has to be restricted using some control technique. K-hmetis uses this technique of clustering [1].

2.6.3 Edge-separability based clustering

Global connectivity information is used by the edge-separability based clustering algorithm to guide its process of generating clusters. In this technique the clique net model is used to convert the hyperedges into regular edges connecting two nodes. This is required to compute the flow or separability λ(e) of an edge to determine how the nodes on two ends of the edge will be clustered. Details on this algorithm can be found in [32, 31].

2.6.4 Fine granularity based clustering

Fine granularity based clustering uses the local connectivity information instead of global connectivity unlike the edge-separability based clustering technique mentioned above. In this technique small clusters containing approximately 2 to 3 cells per cluster is formed. This technique also requires the conversion of a hypergraph to a graph by modelling the edges using a clique model. The major disadvantage of these kinds of the clustering techniques which convert a hypergraph into a graph is scalability. The clustering ratio is very low in this technique of clustering based on fine granularity. [31, 33]

2.6.5 Best choice clustering

This method of clustering is different from the previous clustering techniques in the fact that this technique operates directly on the hypergraph without converting it to a graph. Hyperedges are preserved in this technique. The key contributions of best choice clustering are computing a cluster score, usage of a priority queue data structure and lazy update technique [34, 31]. The disadvantages of this technique are that the execution time is large and maintaining a priority queue becomes difficult when designs are really large in size.

2.6.6 Net-cluster

The net-cluster technique aims to capture the natural clusters in a circuit. It focuses more on hiding as many nets as possible instead of hiding cells [35,31]. The execution time of a net-cluster based approach is dependent on the number of pins in the design rather than number of nets or cells. A modified version of the Fiduccia-Mattheyses heuristic is used in this type of clustering [36]. One of the main disadvantages of this technique is that the clustering ratio cannot be controlled.

2.6.7 Safe-choice clustering

In safe-choice clustering, the algorithm proceeds in such a way that the wirelength is never degraded during cluster formation. Physical clustering based on the physical locations of the cells is performed in this clustering technique. More about this algorithm can be found in [37, 31].

2.7 Clustering using k-way partitioning

LCPlace in [31] implements a technique of forming large clusters by using a k-way partitioning algorithm. When the size of the clusters formed is large, it is evident that the number of clusters

formed will be less compared to other clustering techniques. This implies that the runtime of placing the clusters on the die is very less but at the same time since each cluster now contains a large number of standard cells, an efficient placement technique has to be used to place cells inside the cluster for all clusters. [31] The shapes of all the clusters in this technique are approximately squares. So, this restricts any standard cell placer to operate at its optimum conditions inside the cluster and introduces a limitation to this clustering technique. Another major drawback of this technique is the overlapping area that is created after global placement due to the rigid square shapes of the cluster. A legalizer has to be used to remove the overlap and this in turn degrades the quality of the global placement solution in most cases.

In the above clustering technique mentioned in section 2.7, if we remove the restriction on clusters shapes and allow the shapes to change freely, then it would be really interesting to see how the performance of the standard cell placers used for 'in-cluster' placement get affected. Another question that arises in this regard is how these clusters with various shapes would affect the overlaps created after global placement. The focus of our research in this thesis is to study the quality of global placement on changing the shapes of clusters and also coming up with an efficient technique to form the best shape for a cluster.

Chapter 3 ALTERNATIVE SHAPE GENERATION

In this chapter we address some of the existing problems of clustering and describe the solutions to the questions that led to research along these lines.

3.1 Problem Statement

The major problem that this thesis aims to provide a solution for is that arising from degradation of placement quality which occurs during clustering at the cost of run time improvement. Our work in this thesis provides an optimization technique which when combined with a partitioning based clustering technique yields solutions which are much superior in quality compared to only fixed shape clustering during placement. The specific questions answered in this thesis are the following:

1. What role does the shape of clusters, generated by dividing a large design play in deciding the quality of the placement solution?
2. What are the possible techniques by which the shape of a cluster can be varied without degrading the quality of standard cell placement inside the cluster?
3. How to select the best shape of a cluster in order to achieve the best performance in the final placement?
4. How is such a flow with varied cluster shapes different from existing flows that use fixed shape clusters during global placement?
5. What difference does this flow portray in terms of timing values for a specific design?

3.2 Our contributions

Through our research we aim to provide an incremental optimization technique which will improve the quality of solutions provided by the existing clustering methods. This optimization

technique will help to eliminate some of the inherent drawbacks of partition based clustering during VLSI placement. Clustering is used to reduce the problem size of large placement problems into comparatively smaller problems and then solving them independently. Then the smaller solutions are combined to get the final result. This leads to a significant reduction in runtime by trading off quality. Our cluster shaping technique aims to restrict the degradation in quality at the cost of a slight increase in runtime of the placer. Our major contributions in this research are summarized below.

1. Creation of multiple shape variants for a cluster
2. Developing the floorplans for all the various shapes generated
3. Placing cells inside all the shape variants for the cluster
4. Selection of the best shape for a cluster
5. Placement of mixed size clusters
6. Legalization of residual overlaps

We discuss the points 1, 2 and 3 in this chapter. The rest of our contributions are discussed in the next chapter on shape selection and cluster placement

3.3 What is alternative shape generation?

Alternative shape generation or cluster shaping is the process by which the shape i.e. the physical dimensions of a cluster is modified to vary the placement solutions of the standard cells inside a cluster. Also the optimal location of the cluster on the chip may get modified when its shape is changed. Changing the dimensions of a die (i.e. a cluster in this case) causes the same standard placer to produce different solutions for the same group of cells. Amongst all these solutions only one of them will be the optimal one. So in this process, even though the size of solution space increases by a large extent, there is still improvement in the quality of the best placement solution.

3.4 Limitations of fixed shape clusters

Existing clustering techniques that results in the formation of large clusters tend to fix the shape of cluster in terms of aspect ratio. This causes the following limitations in the flow:

1. Aspect ratios of standard cells are not fixed, so trying to place all these cells belonging to a cluster legally in a square shaped die(cluster) may cause the placer to fail.
2. Whitespace inside a cluster is very limited and equally distributed in all clusters. This creates problems during placement as the requirement of white space varies in each cluster.
3. Restricting the aspect ratio of the cluster also restricts the freedom to move cells around inside a cluster. As a result the overall solution quality of any academic placer used for this purpose is adversely affected.

As a result of the limitations mentioned above, the resulting solution contains a lot of overlaps after global placement which has to be removed later using a legalizer. A tradeoff has to be made between the percentage of overlap inside a cluster and the quality of placement in terms of HPWL inside the cluster.

3.5 Expected changes using multiple shapes

A square shaped cluster is optimal when all the standard cells present in the cluster are square in shape. Such a situation is very rare when clustering is done based on partitioning techniques. So, if the cluster is made wider or narrower keeping the cluster area fixed, chances of getting an optimal placement increases when compared to that of fixed shape clusters. Changing the shape of a cluster may also lead to a reduction in overlap between standard cells without sacrificing on quality of placement inside a cluster. When the aspect ratio of a cluster is

changed, the global placement quality (cluster placement in the top level) is also affected along with the local quality of placement (standard cell placement) inside the cluster.

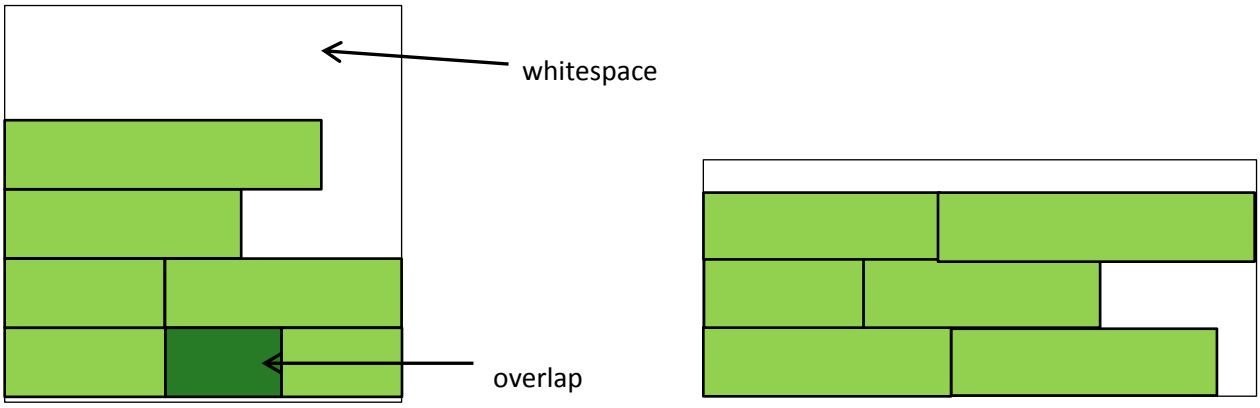


Figure 3.1 Placement of standard cells in square shape versus wide variant

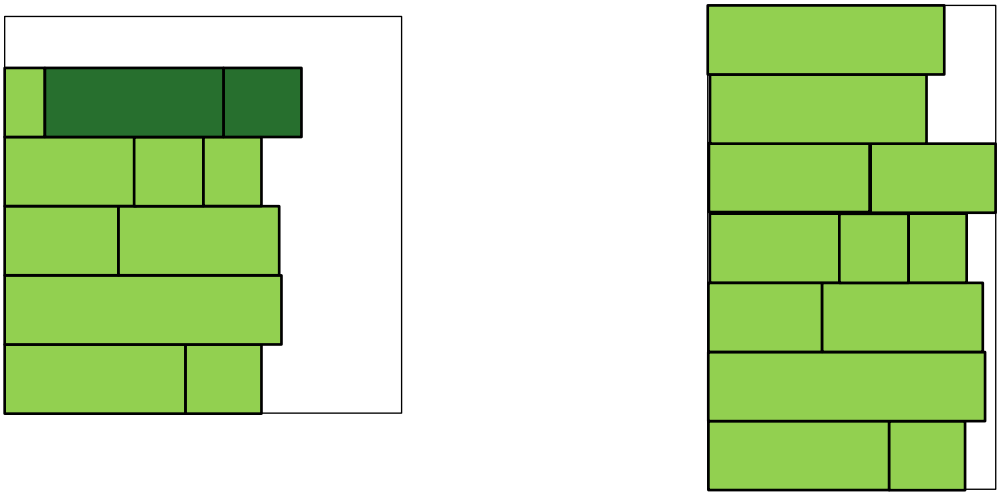


Figure 3.2 Placement of standard cells in square shape versus narrow variant

Figure 3.1 and 3.2 above show the limitations of fixing the shape of a cluster to be a square and the increase in flexibility of placement when the aspect ratio of a cluster is varied based on the requirement of the placer. White space utilization does not happen efficiently if the clusters are

square in shape which leads to the increase in unnecessary white space requirements in the entire design and increasing the overall area requirements of the integrated circuit.

3.6 Proposed placement optimization methodology

In this section we present the entire placement optimization flow that we wish to follow to complete the placement of large designs.

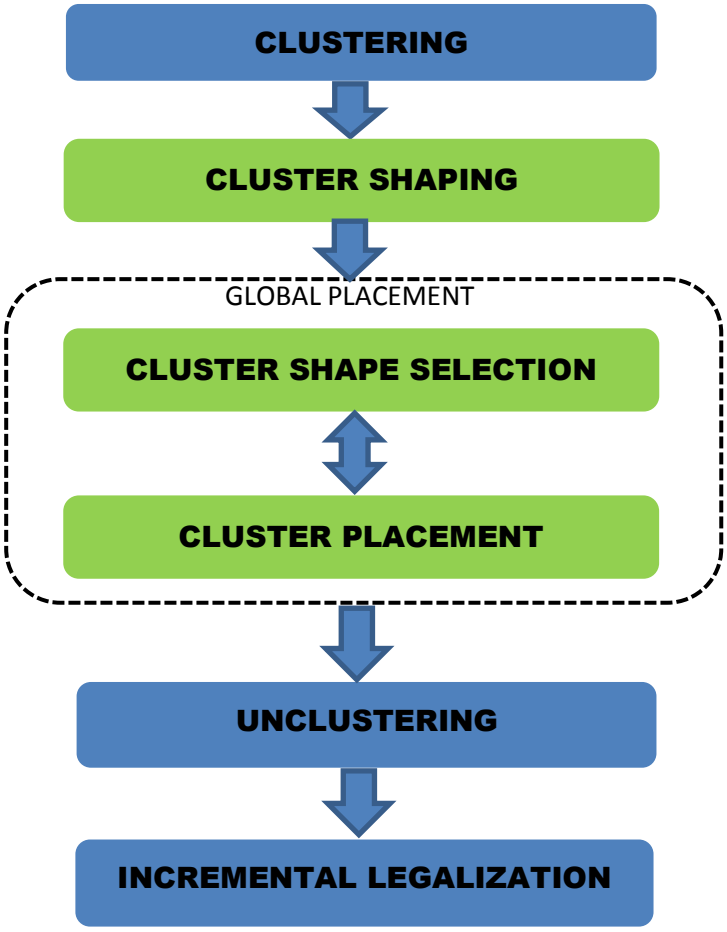


Figure 3.3 Overview of the complete placement flow with shaping

In fig 3.3 above the blocks in green represent our major contributions in this research. All the various stages of the placement flow are explained in detail in the following sections. In this

chapter we discuss the flow till the cluster shaping step. The steps following shape generation are discussed in the next chapter.

3.7 Clustering

Clustering is the technique of reducing a very large placement problem into smaller chunks and solving them independently. Next these independent solutions are recombined together to provide the final placement solution. After performing clustering on a flat netlist, a group of cells together in one cluster represent one “virtual” cell. If the number of standard cells in each cluster is large enough then the number of such “virtual” cells formed is a very low quantity. As a result even for very large designs the placement problem gets reduced to the placement of a few hundreds of “virtual” cells instead of ~1 million standard cells. For the purpose of creating large clusters i.e. clusters containing a large number of cells, we chose to use a k-way partitioning based clustering technique. More details about this partitioning technique can be found in [2]. This clustering technique not only helps to reduce the number of cells at the top level but also helps to hide a large number of nets, thus reducing the complexity of processing.

3.8 Cluster shaping

Cluster shaping is the process by which various shapes are generated for each of the clusters and all the standard cells that are identified to be a part of this cluster are placed using an academic placer in all of these configurations. Each of the steps involved in achieving this is elaborated below. Fig 3.4 shows the components of the flow involved in generating alternate shape variants for a single cluster.

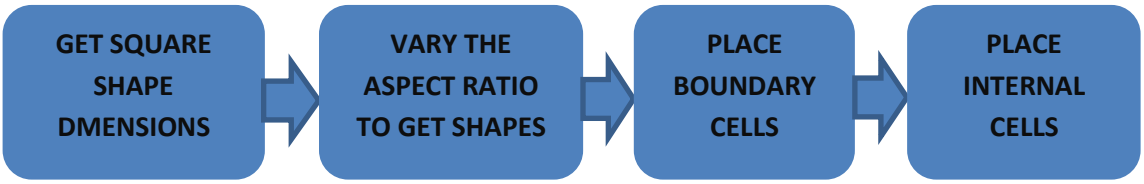


Figure 3.4 Overview of the complete shape generation flow

3.8.1 Methods of changing cluster shapes

In this section we present the various methods that can be used to change the shape of a cluster and also discuss the optimal method in our context of VLSI placement. The metric that has to remain constant while the shape of a cluster is changed, is its area. The area of the cluster is expected to be the sum of the total cell area belonging to a cluster and the white space allocated to the cluster during the process of clustering. There are countless geometric shapes that can be formed with a fixed value for area of that shape. The major factors that decide how to change the shape of a cluster are the following:

1. Ease of floorplanning

We are using a standard academic placer to perform placement inside the clusters. Almost all such placers use a row based placement methodology to optimize cell locations. So, consistent row definitions which are either all horizontal or all vertical are required to perform ‘in-cluster’ placement.

2. White space utilization

During the process of clustering mentioned in earlier sections the total white space available in the entire chip is divided equally among all the clusters. So, the amount of whitespace allocated to each cluster is very less. Shapes should be created in such a way that there is no further requirement of white space in the cluster.

3. Shapes of standard cells

The shapes of standard cells that will be placed inside the clusters play a significant role in deciding the shaping strategy of a cluster. To the best of our knowledge, generally all standard cells are rectangular in shape with a constant height and variable widths. So, cluster shapes which are not rectangular may create problems during placement inside a cluster.

4. Probability of having overlap-free global placement

In our cluster placement technique mentioned below, we do not allow clusters to be rotated on any axis. Due to this condition in our flow, we should generate those shapes for a cluster which are more likely to be placed without overlaps from the perspective of geometry. They should not require any rotation to remove overlapping areas.

Initially only square shapes are produced for all the clusters similar to that in [2]. On changing the number of edges of a square shaped cluster while keeping its area constant, polygonal shapes like those shown below in fig 3.6 are produced. For such polygonal shapes with more than four edges, it is extremely difficult to define rows those are either all horizontal or all vertical. Even if it is possible to define a few rows, a lot of whitespace gets wasted which has to be carved out from the global whitespace, thus reducing 'placeable' area of the chip. This is shown in fig 3.5 and 3.6 below. Hence, we chose to create variations in the shape of a cluster by keeping its rectangular property and changing only its height and width. The area of a cluster needs to remain constant for all possible variations, so, only one of the parameters i.e. height or width could be varied to get rectangles with varied aspect ratios. It was arbitrarily decided that the height of the cluster would be varied and the width would automatically be calculated from the area of the cluster.

A different method was tested out as well to create shapes from a square. This involved combining two rectangles and creating a cluster with an “L-shape” or inverted “L-shape” as shown below in fig 3.7. Definition of rows during floorplanning was not a problem for these shapes but in the later stages of global placement these shapes were very difficult to place without causing any overlaps. Also these shapes tend to create a lot of problems during routing of global nets like power rails, clock, etc. As a result this approach to changing cluster shapes was abandoned in our work.

Figure 3.5 Rectangular shapes with horizontal rows showing maximum area utilization

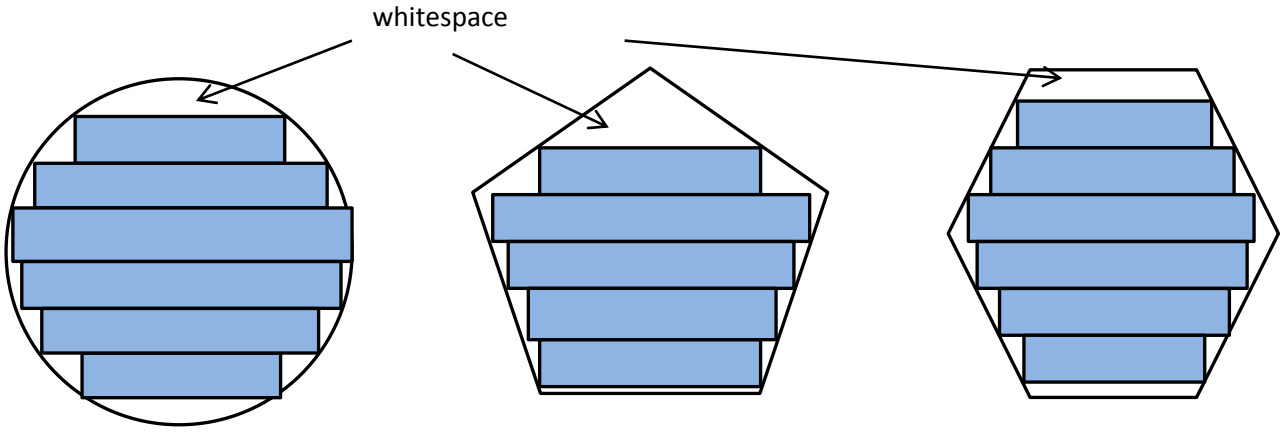


Figure 3.6 Non-rectangular shapes with horizontal rows

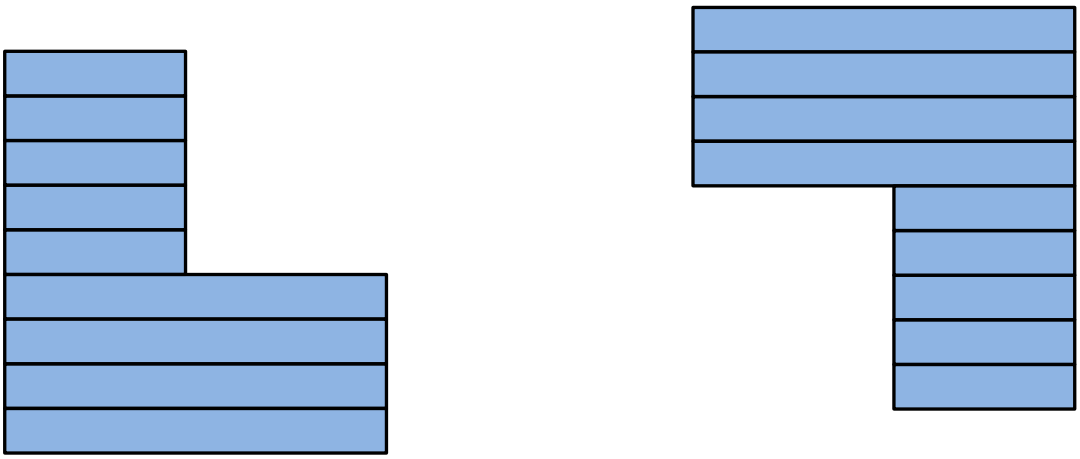


Figure 3.7 ‘L’ and inverted ‘L’ shapes with horizontal rows

3.8.2 Get square shape dimensions of clusters

Once it was decided that the shapes of cluster will be created only by varying the aspect ratios of its square counterpart, the next step was to actually create the square shapes for each of the clusters. The discrete width and height of a cluster is deduced for the square shape of the cluster. These metrics are calculated from the allocated total area of cluster which in turn is derived from two things, the total cell area inside the cluster and the white space allocated to the cluster. The following algorithm 3.1 describes the steps to deduce the height and width of this square shaped cluster. In this technique the total height of the cluster is made to be an integral multiple of a single row height and the width of the cluster is calculated accordingly. This technique is based on the assumption that the height of all the standard cells is same and they only vary in width. This method of calculating height and width is done in a way similar to that mentioned in [2].

Algorithm 3.1 *CalculateHeightAndWidth*

Inputs:

LC: Collection of all standard cells
ws: The minimum amount of white space required to be present in the cluster, represented as a percentage of total area
heightVar: Maximum percentage to which the height can be varied over the square shape height, represented as a percentage
rowHeight: Height of a single row
siteWidth: Width of a single site present in rows

Outputs:

w: final cluster width
h: final cluster height
area: calculated cluster area

```
totalArea ← 0;
For each std cell sc in LC
      totalArea ← totalArea + area(sc);
EndFor
clusterArea ← totalArea + (ws%) * totalArea;
sideOfSquare ← sqrt(totalArea);
integralMult ← ceil(sideOfSquare / rowHeight);
integralHeight ← integralMult * rowHeight;
integralMult ← ceil(sideOfSquare / siteWidth);
integralWidth ← integralMult * siteWidth;
If heightVar != 0 then
  integralHeight ← integralHeight + (heightVar%) * integralHeight;
  integralMult ← ceil(sideOfSquare / rowHeight);
  integralHeight ← integralMult * rowHeight;
  integralMult ← ceil(sideOfSquare / siteWidth);
  integralWidth ← integralMult * siteWidth;
EndIf
h ← integralHeight;
w ← integralWidth;
area ← h * w;
End
```

Algorithm 3.1 Calculate height and width of square shape

3.8.3 Vary the dimensions to form 'shape-banks'

For each cluster, both wider and narrower variants of the square shape are formed. The number of shapes to be generated for a cluster is decided based on two parameters that are taken as inputs from the user. These parameters are height variation percentage and the number of steps in which these variations need to happen. All these variations are done with respect to a square shape (aspect ratio: 1). If the height of a cluster for its square shape is 'h' units and the width of the cluster is 'w' units, the area of the cluster which is a constant is h * w = A square units. If the height variation percentage provided is 'v' percent, then the maximum height h_{max} and minimum height h_{min} are calculated from equations 3.1 and 3.2 below.

Maximum allowed height of the cluster, $h_{max} = h + (v\ \%)\ \times h$ Equation 3.1

Minimum allowed height of the cluster, $h_{min} = h - (v\ \%)\ \times h$ Equation 3.2

This maximum and minimum height of the cluster represents the height of the narrowest shape variant and of the widest shape variant of each cluster respectively. The corresponding minimum and maximum width (w_{min} and w_{max}) of the cluster are calculated from algorithm 3.1 mentioned above. Once the minimum and maximum dimensions of a cluster are found out then the number of shapes for each cluster is controlled using the value of number of variation steps provided by the user. The number of narrow variants formed from the square shape will always be equal to the number of wider variants formed. If the number of variation steps is denoted by 'x' then the height h of the cluster will be increased to h_{max} and decreased to h_{min} in x steps each. The step size 's' is the percentage by which the height is varied at each step of shape generation till both the high and low extremes are reached. Step size is calculated from eqn 3.3 shown below. The total number of shapes that will be generated for a cluster including its

Step size percentage, $s = v/x$ Equation 3.3

square shape is 2x +1. Algorithm 3.2 describes the steps of getting the height variations and creating wider and narrow shapes from the square variant.

Algorithm 3.2 PopulateShapeVariations

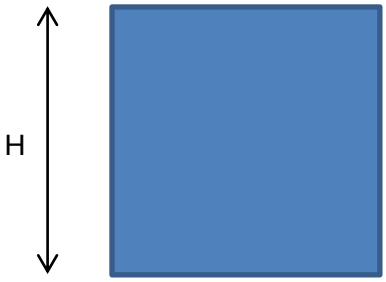
Inputs:
H: Maximum allowed variation in the height (0% to 100%)
S: Number of steps required to get maximum variation
Outputs:
L: Computed list of percentage variations in height at each step

```
stepPercent ← H / S
index ← S
For index > 0 do
   percentageChange ← stepPercent * index
   H ← H.push_back(-percentageChange)
   index--
End For
index  ←  S
For index > 0 do
   percentageChange ←  stepPercent * index
   H ← H.push_back(+percentageChange)
   index--
End For
```

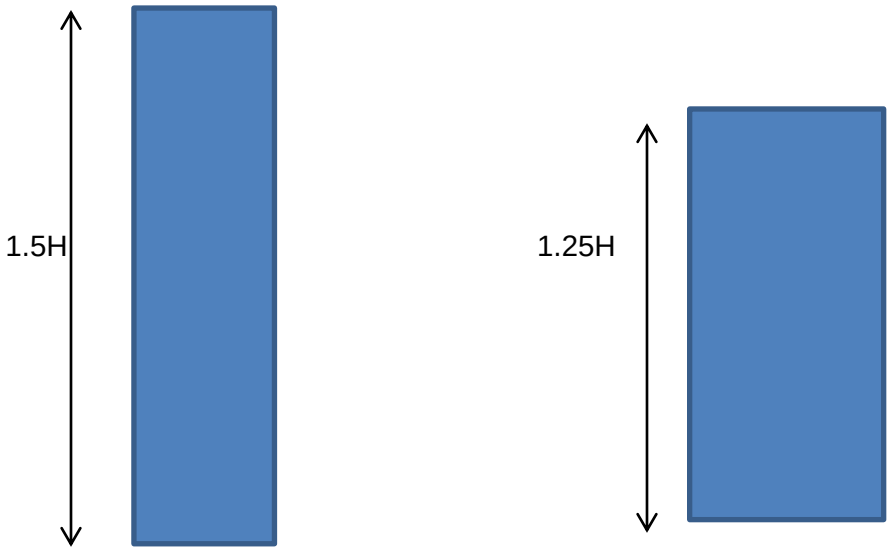
Algorithm 3.2 Get steps of percentage variation

The above algorithm when preceded by algorithm 3.1 gives a list of heights and widths for each cluster. Fig 3.7 depicts an example of how this algorithm works and how shape variants are generated for each cluster. For example, if the height variation percentage ‘v’ is provided to be 50. This means that the maximum and minimum height possible for this cluster is 1.5 * H and 0.5 * H respectively, where H is the height calculated using algorithm 3.1 assuming the shape of

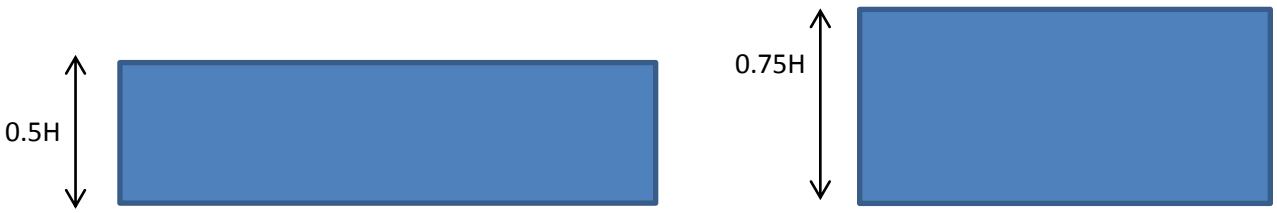
the cluster to be a square. Now, we know the height of the narrowest and widest variant of the cluster. If the number of variation steps ‘x’ is provided to be 2, this signifies that both the maximum and minimum height variant should be reached in two steps each. Step size, s = 50 / 2 = 25. So, at each stage of shape generation for the cluster, the height increases or decreases by 25% depending on which variant of shape is generated.



(a)



(b)



(c)

Figure 3.8 (a) depicts the square variant. (b) and (c) show the narrow and wider variants for the same cluster

So, to summarize a total of four shapes i.e. two wider and two narrower variants are generated apart from the square shape, when the number of variation steps is given as 2. The shapes generated will have their heights to be 1.5H, 1.25H, H, 0.75H and 0.5H. The widths of these shapes are calculated using the algorithm 3.1 mentioned above. From fig 3.7 above we notice a very interesting aspect of shape generation. It can be seen that instead of four shape variants for a square shape, if only two of them were generated and then rotated clockwise or counter-clockwise by 90 degrees then the other two shape configurations would have been created automatically. But the problem with this approach is that once the rows are created for a cluster in the floorplan, the rows are already defined as horizontal or vertical. Changing the alignment of rows will require the standard cells present in the cluster to be rotated. Rotation of standard cells can cause some serious issues during the routing phase of the ASIC design cycle which is why it not a preferred step during placement. This is also a reason why we chose to perform cluster shaping instead of rotating clusters as a placement optimization technique for clustering based placers.

3.8.4 Floorplanning for the multiple shapes

During the process of floorplanning, horizontal rows are defined and stored in the .scl file which is the bookshelf format for storing floorplan specifications. These row definitions contain the row index, height of the row, number of sites in the row, width of each site in the row, sub-site

boundaries if present, orientation of the row (N, S, FN, FS), etc. These floorplan files are written out for each shape of the cluster. This step is essential in the flow for generation of various shapes because generally any academic placer that is used to place the standard cells inside a cluster takes only the files in bookshelf format as an input to run the placer. One important aspect of floorplanning here is the number of rows and the number of sites in each row defined for all the shapes should be an integer. This is necessary to ensure maximum area utilization and legality in the final solution after global placement.

3.8.5 Placement of cells inside the clusters

After floorplanning is completed for all the shapes generated for each cluster, the clusters have to be 'filled'. In other words the cells that were allocated to each cluster during clustering now have to be actually placed within each cluster in their optimal locations. This step is required in our case unlike [2] because the placement of cells inside a cluster plays a significant role in deciding the shape of the cluster as well as the optimal location of the cluster after global placement. The placement of standard cells inside a cluster can be divided in two steps:

1. Standard cells with external connections are placed along the boundary of the cluster using a deterministic technique
2. Remaining internal cells are placed inside the cluster boundary by using an external academic placer that treats the cells placed in the previous step as pivot or anchor points

The block diagram of the flow for placing cells inside a cluster is shown below in Fig 3.8. The blocks in green represent our contribution in the flow for placing cells inside a cluster

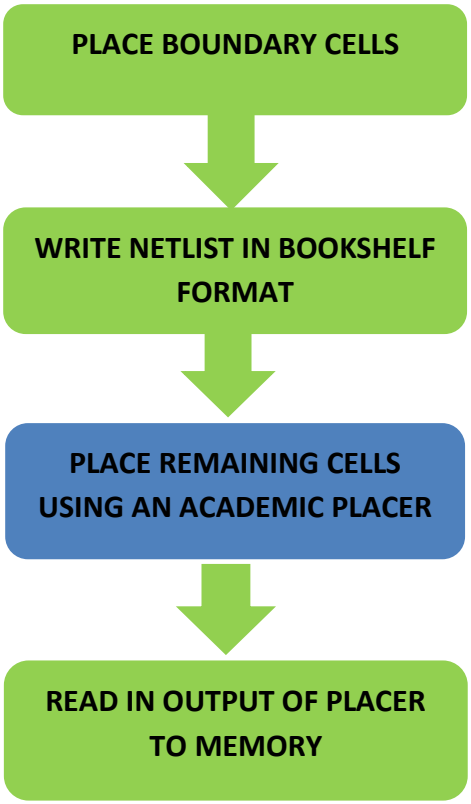


Figure 3.9 Steps showing placement of cells inside all shapes

3.8.5.1 Place boundary cells inside all the shapes for a cluster

Cells which were allocated to the cluster can be divided into two categories such as boundary cells and internal cells. To understand this difference, let us first define internal nets and external nets of a cluster. This is done in the manner similar to [2].

1. **Internal Nets**: Nets which are connected only to cells inside a cluster and have no external connections with cells in other clusters. These nets get hidden during clustering.
2. **External Nets**: These nets are connected to cells that are associated with at least two different clusters

From the definition of internal and external nets we can categorize cells inside a cluster into internal and boundary cells:

1. **Internal Cells**: Internal cells are those that are only connected to internal nets and do not have any external connections
2. **Boundary Cells**: Boundary cells are the cells that are connected to at least one external net and they may or may not be connected to internal nets

The major reason for dividing all the cells inside a cluster into internal cells and boundary cells was to place the cells connected to external nets close to the edge of the cluster. This helped in reduction of overall cost in terms of wirelength and finally a better placement solution. These boundary cells are placed according to the deterministic method described in LCPlace. [2]

3.8.5.2 Place the remaining internal cells in the cluster

We decided to use an analytical placer to place the internal cells inside the cluster. For any analytical placer to perform efficient placement, some pivot or anchor cells are required to optimize the placement of all the movable cells with respect to these anchor cells. So, the other reason for categorizing the cells in a cluster as internal and boundary cells was to allow the boundary cells to behave as fixed points for reference to the analytical placer. The placer would optimize the placement of the remaining movable/internal cells with respect to the fixed cells which were placed using a deterministic technique in the previous step. The academic placer used for this purpose was NTUPlace 3.0 [3]. The cells of the cluster, their connections, and the floorplan of the cluster that included its row definitions were written out in the bookshelf format. The boundary cells were marked as fixed cells. After the execution of the analytical placer, it writes the output of placement in the bookshelf format for each of the shape variant in a cluster. This step is executed for all the clusters in the design. Output of placement is not read back into the memory at this step. But the HPWL (Half Perimeter Wire length) values for each of the

shapes are stored in the memory and are referred to as the internal HPWL. The reason for not reading back the placement solution is the fact that we would not know at this point in the placement process which shape would be selected for a cluster and which placement solution needs to be read back in. The solution in bookshelf format is only read back in the next step of execution known as cluster shape selection.

Chapter 4 SHAPE SELECTION AND CLUSTER PLACEMENT

In this chapter we discuss the shape selection techniques that we have developed to assign optimal shapes to all the clusters. We also provide details about how this mixed size clusters are placed using a macro placer. Finally this chapter concludes with the execution of unclustering and legalization steps.

4.1 Cluster shape selection

Once all the shape variants for all the clusters are generated and cells are placed inside each of these clusters, the step that remains in this optimization technique is selection of the best shape for a cluster. This process of selecting the best shape of the cluster can be executed in various ways. A few of these techniques were implemented and based on the observations from these experiments the most efficient technique was chosen from the list. Explanation of these techniques and the respective observations are given below.

4.1.1 Global wirelength based shape selection

Initially we decided to use the global placement solution generated using the methods mentioned in LCPlace [2]. In this case the global placement solution contained all square shapes for the clusters and this was the starting point for our shaping algorithm. We had modified the algorithm to include the creation of the ‘shape-banks’ for all the clusters before using the modified FD placement technique[2] to place clusters at the top level. The cost function used in this flow consisted of only the global or external HPWL values. Global HPWL is the wirelength that is calculated by taking into account only the external nets in the current design. Algorithm 4.1 describes the shape selection procedure. The process consisted of the following steps of execution:

1. **Square shape placement**: Initially for all the clusters in the design, the default square shape was selected as the initial shape variant and the clusters were placed using a modified force directed placement algorithm [2].
2. **Select, evaluate and accept**: Once all the square clusters were placed using the top level placer, a cluster was selected and a modified shape (except the square shape) was assigned to it. Then the global HPWL was re-calculated based on this new shape for the cluster. If there was any improvement in the cost function, then this change was accepted, else it was rejected for changes causing degradation or no improvement in HPWL. This process was executed for each shape variant for a cluster and when the execution completed for a cluster, the best shape in terms of global HPWL was assigned to the cluster. This step was carried out for all the clusters in the design and the shapes that produced the minimum global HPWL remained in the final design.

Algorithm 4.1 AssignBestShapeToCluster

```
Inputs:
      L ← List of Clusters
      I ← Initial HPWL with all square shapes
      N ← Number of shapes for each cluster

Outputs:
      Clusters with changed shapes
      F ← Final HPWL with modified cluster shapes
```

```
currentHPWL ← I
previousHPWL ← I
shapeIndex ← 0
For each cluster cl in L
      While (shapeIndex < N)
            Assign shape to cl
            currentHPWL ← CalculateHPWL()
            if (currentHPWL < previousHPWL) then
                  Accept shape for cl
```

```
                previousHPWL ← currentHPWL
            End if
        End While
End For
F ← previousHPWL

End
```

Algorithm 4.1 Assign best shape to a cluster: Greedy approach

The only significant advantage of this process was this ensured that the final global HPWL never degraded beyond that of the placement with square shapes only. In other words the final global HPWL using this process can only improve and can never degrade due to changing of cluster shapes. This technique had a lot more limitations and drawbacks shown below, as a result of which it was not chosen as the shape selection technique in our flow.

a. Greedy nature of the algorithm increased the probability of getting stuck in a local minimum instead of the global minimum for HPWL
b. Changing the shape of a cluster without moving it from its original location after square cluster placement created large overlaps. These overlaps were removed during the unclustering stage of execution when a standard cell legalizer was used to remove the overlaps. During this process the cells had to be moved around a lot resulting in degradation of HPWL.
c. Fixing the cluster positions even after changing their shapes did not guarantee the best placement solution because changing the shape of a cluster might have changed its ideal location in the entire chip.

The cost function that was used to evaluate if a shape is to be accepted or rejected comprised only of the global wirelength. This was not the most efficient evaluation system as a result of which the final wirelength values after unclustering did not improve beyond that of the reference wirelength calculated using only square shapes.

4.2 Cluster shape selection including cluster placement

4.2.1 Simulated Annealing based shape selection and force-directed cluster placement

The greedy approach to shape selection was abandoned due to its evident drawbacks and we decided to proceed with a simulated annealing (SA) based shape selection technique. Methods to predict the optimal shape of a cluster before actually assigning the shape and placing it at an optimal position, is extremely complicated to model and is also not present in the existing literature to the best of our knowledge. So, to determine the best shape of a cluster, a shape has to be actually assigned to a cluster by modifying its physical dimensions in memory and then a modified force directed placer had to be used to place all the clusters in the design. The next step is to evaluate the cost for this particular placement solution and validate if the shape variation has actually improved the cost function or not.

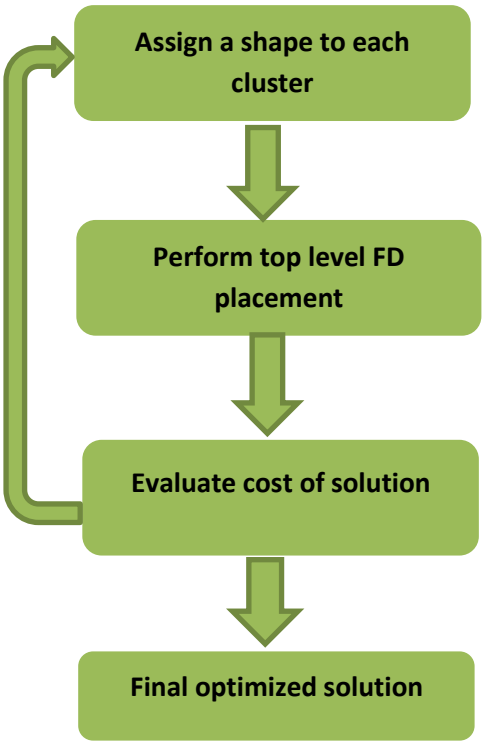


Figure 4.1 Flow for SA based shape selection and FD cluster placement

After taking all these observations into consideration, a non-deterministic algorithm such as SA was chosen for the purpose of deciding the best shape for a cluster. Fig 4.1 shown above describes the flow using the SA selection procedure and FD placer. Detailed explanation of all the steps involved in the flow is given below.

4.2.1.1 Assign a shape to each cluster

Firstly an array of size equal to the numbers of clusters is created in memory. Also a shape index, which is a number between 0 to n - 1 where n, is the number of shapes for a cluster, is associated with each shape variant for a cluster. Then the following steps are executed for all clusters in order to assign a shape to each cluster:

1. A random shape is selected from the 'shape-bank' for each cluster and the shape index corresponding to that shape is stored in the array defined above.
2. The dimensions of the shape are retrieved using its index and these values for the cluster are updated in the memory
3. The standard cell placement solution corresponding to the selected shape for a cluster is read back in memory

Algorithm 4.1 shown below describes the steps of changing the shapes for all the clusters and updating the locations of all the standard cells inside a cluster from the 'shape-bank'.

4.2.1.2 Top level placement of the clusters

Once shapes have been assigned to all the clusters, then these clusters have to be moved into their optimal locations in the chip. A modified force directed solver was used for this purpose. The entire chip area was divided into bins and a bin was allocated to a cluster based on its vacant, occupied or locked state after locating the position of the optimal bin for the cluster. This method of force directed top level cluster placement was done in a manner similar to that mentioned in [2]. In our force directed placement technique a cluster (cell) could belong to two

bins at the same time and depending on the utilization of a bin, it was assigned vacant, occupied or locked status. The assignment of shapes to all the clusters and then placing them using the above mentioned technique together comprised the move function for the annealer.

4.2.1.3 Evaluation of the cost function

Once again the cost function used in this annealing procedure comprised of only the change in global HPWL for all the clusters. The global HPWL for one cycle of execution is compared against the previous value of global HPWL and the difference gives the cost of executing the current cycle of annealing procedure. Based on this annealing procedure the placement solution with the modified shapes are either accepted or rejected. If the solution is accepted then the current cost is retained to be compared against the new cost after the next round in annealing.

Algorithm 4.2 ChangeClusterShape

Inputs:

LC: List of clusters
X: Number of shape variants generated for each cluster
N: Total number of clusters generated

Outputs:

Updated dimensions of the clusters LC, positions of cells inside clusters

---

```
For each cell Ci in LC
     Assign a shape index (except square) to Ci
     shapeIndices[i] ← shapeIndex(Ci)
End For
For each cell Ci in LC
     CHeight ←   GetHeight (Ci)
     Cwidth  ←   GetWidth (Ci)
     Std_cells  ← Get all standard cells inside Ci
```

```
    Cell_postions ← Get the coordinates of all the cells in
 Std_cells
    For each cell sc in Std_cells
       c_x ← X-coordinate (sc)
       c_y ← Y-coordinate (sc)
    End For
    Reset the pin offsets of the cluster C_i based on the locations of
the boundary cells in the clusters
End For

End
```

Algorithm 4.2 Change the shape of a cluster physically

We have used a general simulated annealing package to perform the annealing procedure for shape selection. More details about this package can be found in [4].

The limitations and drawbacks of this technique comprises mainly of the difficulty in placing clusters which were not square shaped, using a FD placement technique. A force directed placement technique requires a grid to be defined for assigning clusters to their optimal locations on the grid. The basic problem here was to come up with an efficient technique of creating the grid by dividing the entire chip area into equal sized bins. It was not possible to define a grid where each location in the grid would have varying aspect ratios. As a result, clusters that were not square in shape were not able to fit completely in one location on the grid and had to be assigned to multiple grid locations. This increased the complexity of processing to a large extent and also created very large amounts of overlapping area in the design. Also, the process of modelling the conditions that decide the optimal site for a cluster becomes more complicated with increasing number of shapes, so these processes failed to complete placement if the number of clusters formed was quite large. As a result the main benefit of creating shape variants for a cluster was not prominent enough and did not reflect in the results of the quality of placement. So we had to approach this problem of mixed size placement from a

different direction. We decided not to use deterministic algorithms to place mixed size clusters as it extremely difficult and complicated to model the constraints correctly in the design. Instead we looked for non-deterministic placers like SA based.

4.2.2 Simulated annealing based shape selection and simulated annealing based macro placement

In this flow we have used the same technique of selecting shapes using a SA based algorithm as mentioned above in section 4.2.1, but the placement of the clusters with various shapes is now done using a SA based macro placer, MetaPlacer-Capo [5, 39] instead of a deterministic approach. Another major change we have incorporated in this flow is that of changing the cost function to include the percentage of overlap as well. Also the HPWL now contains the sum of the global and the internal HPWL. This helps in taking into account the internal wirelength which changes when the shape of a cluster is changed. Details of this change in the cost function are given below. The entire flow for SA based shape selection and macro placement of clusters is shown in fig 4.2 below. The following sections describe the details about each step of this flow.

4.2.2.1 Assignment of shapes to all the clusters

This step of assigning shapes to all the clusters is executed in the same manner as section 4.2.1 above. A random shape is selected from the list of available shapes for the cluster and its dimensions are updated in the memory. Following this, the standard cell locations for the cluster are updated as well considering the bottom left corner of the cluster to be the origin. Algorithm 4.2 above describes the steps for assigning shapes to all clusters

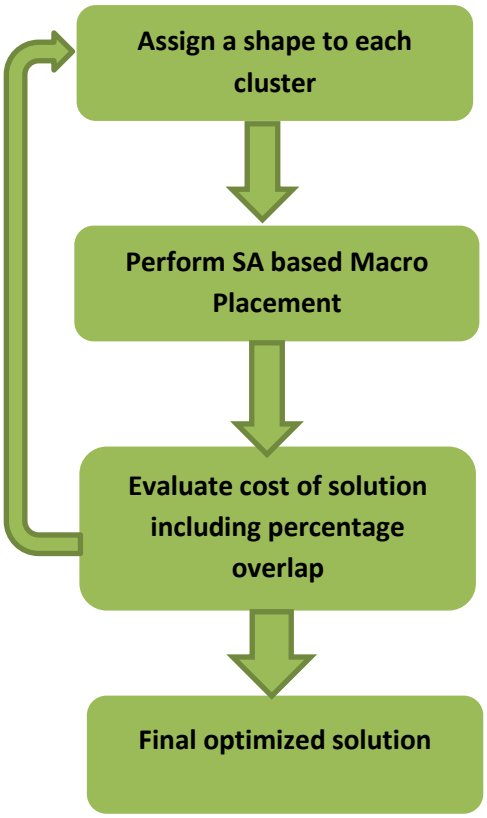


Figure 4.2 Flow for SA based shape selection and non-deterministic cluster placement

4.2.2.2 Non-deterministic approach to cluster placement

In the previous section 4.1.2 we found that using a force-directed solver for placing the clusters at the top level was not the most efficient technique for cluster placement. So, we decided to use a non-deterministic algorithm such as SA to solve the problem of modelling constraints in top level placement. MetaPlacer-Capo [5, 39] was used for this purpose. The steps involved in top level placement are the following:

1. All the clusters are initially placed at the origin of the final chip at [0, 0]. The terminals/ports of the cluster are placed at various positions on the chip boundary. These ports will be used as fixed anchor/pivot points with respect to which the rest of the clusters will be placed in the entire design.

2. The variedly shaped clusters are modelled as movable macros in the design and written out in the bookshelf format. So basically what is provided as an input to MetaPlacer-Capo[5, 39] is a netlist containing only macro blocks which represent all the clusters in the design. The other files like .scl, .nodes, .nets, etc files are written out as well based on which the macro placer will determine the optimal placement of the clusters.
3. The macro placer is executed with its most optimal switches so that it reduces the percentage overlap among all the clusters to almost null. The result of the macro placer is written out in a bookshelf format. This is read back in memory and cluster locations are updated. Another important aspect to note here is we do not have to update the locations of cells inside the clusters after macro placement. The reason for this being at this point in the flow the locations of the cells inside a cluster represent only relative positions with respect to the cluster left bottom corner as the origin. So, standard cell locations have to be updated only when the shape of a cluster is changed and not when the clusters are moved around.

4.2.2.3 Evaluation of the cost function

The cost function used for evaluation of the solution generated above is a weighted combination of both the total HPWL (internal and global) and the percentage of overlap. We have included the internal HPWL along with the global HPWL instead of only the global HPWL because when the shape of a cluster is changed, its internal HPWL i.e. the wirelength of the internal nets do not remain fixed throughout the placement. In fact the internal HPWL plays a major role in selecting the optimal shape of a cluster. Also the percentage overlap plays the deciding factor to either accept or reject a solution i.e. a move in the simulated annealing procedure. The overlapping area between two clusters is calculated using algorithm 4.3 shown below.

Algorithm 4.3 FindOverlap

Inputs:

LC: List of all the cluster objects

totalArea: Total area of cells in the design

Outputs:

percentOverlap: Percentage of overlap in the placement solution

```
i ← 0
j ← 0
n ← GetNumberOfClusters(LC)
totalOverlap ← 0.0
overlap ← 0.0
visitedNodes ← NULL
For each cluster Ci in LC
      R1 ← RightX (Ci)
      L1 ← LeftX (Ci)
      For each cluster Cj in LC
            if (exists(Cj) in visited nodes || i == j)
                  continue
            End if
            R2 ← RightX (Cj)
            L2 ← LeftX (Cj)
            right ← R2
            if (R1 < R2)
                  right ← R1
            End if
            left ← L2
            if (L1 > L2)
                  left ← L1
            End if
            overlap ← right – left
            if (overlap < 0)
                  overlap ← 0.0
            End if
            totalOverlap ← totalOverlap + overlap
      End For
```

```
      visitedNodes ← visitedNodes + C_i
End For
percentOverlap ← totalOverlap / totalArea * 100
End
```

Algorithm 4.3 Find percentage of overlap

A decrease in percentage of overlap is desired because that will eliminate the need for using a standard cell legalizer after the unclustering process. This in turn will preserve the quality of placement which is generally lost in clustering based techniques. Equation 4.1 and 4.2 given below describes the modified cost function used along with the macro placer.

$$Total\ HPWL, T = Global\ HPWL + internal\ HPWL \qquad \text{Equation 4.1}$$

$$cost = (\alpha \times T) + (\beta \times P)$$

α and β are constants set according to the design size. T is total HPWL and P is the percentage overlap — Equation 4.2

4.3 Unclustering

All this while the positions of the standard cells saved in memory were the relative positions considering the left bottom corner of the cluster they belong to, as the origin of the chip. So these cells need to be given absolute positions before finalizing the placement solution. Once the above SA based shaping and macro placement technique converges, then this process of unclustering is executed. During this process all the standard cells are assigned their absolute positions in the chip and the cluster boundaries are resolved. The assignment of absolute positions proceeds using the equations 4.3 and 4.4 below

$$cell_x = clust_x + cell_{relativeX}$$ Equation 4.3

$$cell_y = clust_y + cell_{relativeY}$$ Equation 4.4

Here $cell_x$ and $cell_y$ are the absolute 'x' and 'y' positions of the standard cells in the final chip. $clust_x$ and $clust_y$ are the absolute positions of the clusters on the chip. $cell_{relativex}$ and $cell_{relativey}$ are the relative positions of the clusters assigned to by the external academic placer. After the process of unclustering is completed the global placement for the design has been finalized.

4.4 Legalization

The major purpose of changing the cluster shapes for placement optimization is to eliminate the need for post processing of the global placement by performing steps such as legalization, detailed placement, etc. In standard global placement techniques using clustering, legalization is a necessary step which is required to eliminate the overlaps caused by restricting the shape boundaries of clusters. By using our shaping technique we have tried to eliminate overlap as much as possible from the design even before unclustering. The macro placer used here takes into account overlap reduction while performing placement with varied shapes. Also since the percentage overlap is included in the cost function for selecting the best shape for a cluster, not only the placement for such shapes but also the selection of the shape itself is based on the reduction in the percentage of overlap. In all the cases using our technique and the macro placer we have been able to successfully reduce the overlapping area after global placement to negligible quantities, if not null. For all such cases the incremental legalization is required. In our flow, we use the FastPlace legalizer [6]. It works incrementally by first aligning all the cells to rows and then legalizing them within each row. The objective function for the legalizer is to

minimize a combination of HPWL and cell movement. This step is optional and may be executed if the percentage of overlap after global placement is non-zero.

Chapter 5 EXPERIMENTAL RESULTS AND ANALYSIS

This chapter begins with a description of our experimental setup used to evaluate our methodology. Observations from all our experiments and detailed analysis of the results are also articulated in this chapter.

5.1 Experimental flow setup

The evaluation of our proposed flow was done by setting up a complete ASIC design cycle that began with synthesizing the RTL to create benchmarks in the bookshelf format. This step was followed by placement, routing, etc. Synthesis was done from RTL of real designs found in OpenCores[7]. The three major metrics for evaluating our results are 1) total HPWL (Half Perimeter Wire Length), 2) TNS (Total negative slack) and 3) WNS (Worst negative slack). Our results of these experiments are compared with a placement technique that uses fixed square shapes for clustering as mentioned in [2]. Fig 5.1 below gives an overview of the entire experimentation flow. Brief description of each step of experimentation is given below.

5.1.1 Benchmark generation

The process of benchmark generation consists of reading in the verilog/VHDL files of any real design and using Synopsys Design Compiler to perform logic synthesis resulting in the generation of a gate-level netlist. Following this step, the gate-level netlist along with other input parameters is written out in the bookshelf format using a combination of perl scripts. Throughout this thesis we have been using the bookshelf format for various stages in the flow because all existing academic placers and utility tools associated with them are developed using the bookshelf format as a standard input format.

All the RTL used for the purpose of experimentation was collected from OpenCores[7]. The designs were arbitrarily chosen from a range of processor cores, video Encoder/Decoders, etc. Designs containing memory blocks were avoided due to the fact that the timing values for these blocks were not of primary importance compared to the memory access times. The other reason for not including memory blocks in our design was after running synthesis on these designs the memory blocks were reduced to very few blocks decreasing the cell count in the process.

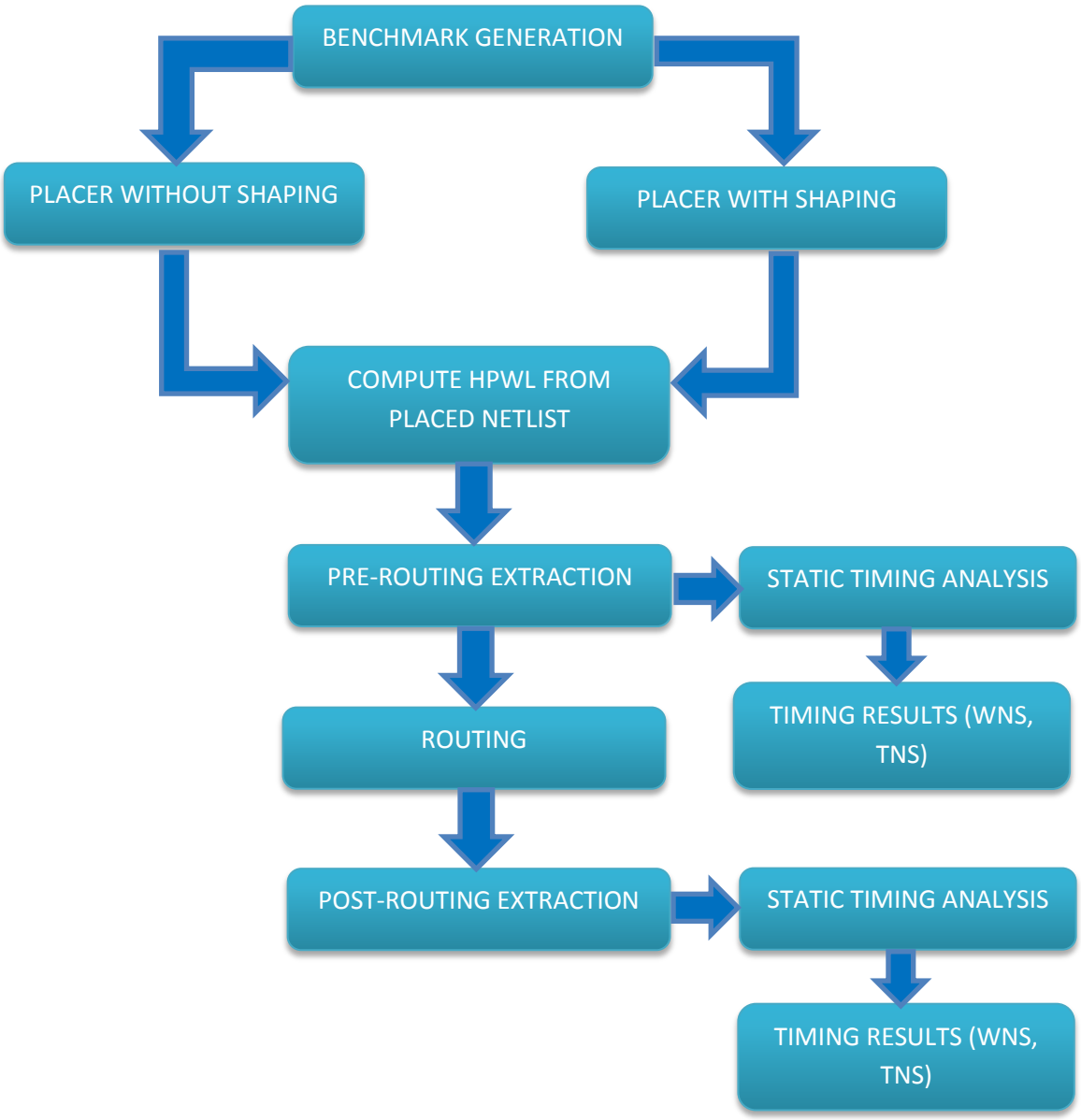


Figure 5.1 Overview of complete experimental setup

Library cell data is required to perform the steps of extraction, routing, static timing analysis,etc. This was the reason why designs containing macros were not included as a part of our experiments. Also none of the benchmarks from ISPD'05[8], ISPD'06[9] or the IBM benchmark suite [10] were used in our experiments due to the absence of library cell data and presence of macros in them. Descriptions of the benchmarks used in our experiments are given below in Table 5.1.

Benchmark Name	Number of Cells	Number of Nets	Description
cordic	20,577	25,767	Cordic core
reedsoldec	31,179	32,964	Reed-solomon decoder
seq_align	44,098	46,143	Seqential alignment of DNA strands
Pairing	288,622	290,430	No description available
jpegenc	384,646	417,794	JPEG encoder

Table 5.1 List of all benchmarks used during experimentation

5.1.2 Placer without shaping

The reference placer used here is LCPlace [2] which is a clustering based placement technique that generates large clusters using a k-way partitioning flow. All the clusters formed during placement with LCPlace are square in shape and the sizes of all the clusters are approximately equal. This makes it an ideal placer to be used as reference to evaluate our shaping based optimization technique.

5.1.3 Placer with shaping

In our shaping based flow, we have incorporated the same clustering technique as used in LCPlace mentioned above [2]. Formation of large clusters with square shapes is an integral part of our flow which helps in generating the initial shapes that are varied to optimize the placement

of the final solution. We have also used MetaPlacer-Capo [5] to place all the variably sized clusters optimally in the chip after specific shapes have been selected for all the clusters. A flow diagram giving an overview of the global placement flow with shaping is shown in Fig 5.2 below.

Figure 5.2 Flow diagram of global placement with shaping

5.1.3.1 Form large clusters

Large clusters are formed using a k-way partitioner similar to the technique used to generate clusters in LCPlace [2]. The partitioner used here divides the design into large clusters in such a way that the number of hyperedges cut by each partition is minimized. This method of clustering is preferred because using this method the total number of clusters can be restricted to a relatively less quantity, thus simplifying the problem of mixed size placement of clusters at the top level. It is evident that for a given design the larger the size of each cluster, the lesser the number of clusters generated by partitioning. Once all the cells in the design are allocated to some cluster then the process of multiple shape creation is started. During this process, the square shape height of all the clusters are varied to generate various aspect ratios which represent the shape variants for a cluster. The cells belonging to a cluster are placed for all its shape variants using a standard cell placer NTUPlace 3.0 [3].

5.1.3.2 Change cluster shapes and run macro placer

The process of selecting the best shape for a cluster begins once all shapes are generated and 'filled' with standard cells. During this stage, according to our proposed approach a SA based shape selection technique is used to assign a shape from the 'shape bank' of each cluster and an attempt is made to place them without overlaps using a macro placer such as MetaPlacer-Capo [4]. The placement quality of this solution is evaluated using a weighted combination of HPWL and overlap and based on the annealing procedure this solution is either accepted or rejected.

5.1.3.3 Unclustering

In this step all the cluster boundaries are resolved and standard cells are assigned their absolute locations on the chip. After execution of this step is complete, clusters will no longer be present in memory.

5.1.3.4 Legalization

In some cases the macro placer is not able to completely remove all overlaps from the design after the variably shaped clusters are placed. For such situations we need to execute a standard cell legalizer after unclustering. This will ensure complete legality in the design. This step is optional and is required only in a very few cases.

5.1.4 Routing

Routing of the final placed netlist is completed using Synopsys IC compiler. This step is required in order to get the timing values (TNS, WNS) from the parasitics after the extraction step.

5.1.5 Extraction

Extraction is done at two stages in our experimental flow. Once before routing and then once again after routing is completed. StarRC in Synopsys IC Compiler is used to perform extraction of the parasitics.

5.1.6 Static timing analysis (STA)

Once the parasitics are extracted, Synopsys PrimeTime uses the netlist and the extracted values to accurately estimate the net delays for all the paths. From this information, it gives the WNS and the TNS values required to evaluate the quality of placement.

5.2 Experimental Results

The platform used to execute all the experiments for evaluating our shaping optimization technique was a personal computer with an Intel® Core™ i3 2.5 GHz processing unit and 4GB of RAM. The operating system used was a Linux Mint with a 64-bit kernel.

The majority of the experiments were performed to compare the results of large clustering based placement with and without including the shaping technique. Another set of experiments were performed to vary the parameters involved in shaping and observe the optimization trends with those variations.

5.2.1 Results with and without shaping

5.2.1.1 Comparison of performance metric values

Table 5.2 below shows the final results of the entire experimental flow which includes placement, routing, extraction, etc. The TNS and WNS values present in the table above are calculated from the post-routed netlist.

Benchmark Name	Number of shapes	LCPlace			Placer with shaping		
		HPWL (x 10^8 nm)	$WNS_{postroute}$ (ns)	$TNS_{postroute}$ (ns)	HPWL (x 10^8 nm)	$WNS_{postroute}$ (ns)	$TNS_{postroute}$ (ns)
reedsoldec	5	6.03	-1.28	-3348.91	5.83	-1.25	-3184.6
seq_align	5	7.89	-1.50	-1593.03	7.15	-1.55	-1928.63
cordic	5	5.05	-1.03	-1130.05	4.79	-1.06	-711.23
pairing	5	98.41	-3.15	-22905.61	87.23	-3.13	-23005.12
jpegenc	5	133.69	-7.84	-39345.80	122.34	-3.71	-39283.15

Table 5.2 Compares the results of placement with and without shaping

5.2.1.2 Percentage of improvement/degradation between LCPlace and our placer with shaping

Benchmark Name	Improvement in HPWL (%)	Improvement in WNS (%)	Improvement in TNS (%)
reedsoldec	3.31	2.30	4.9
seq_align	9.30	-3.34	-21.06
cordic	5.10	-2.90	37.06
pairing	11.36	0.60	-0.43
jpegenc	8.49	52.6	0.15

Table 5.3 List of percentage improvement or degradation of the performance metrics

Table 5.3 above shows the percentage of improvement or degradation in the performance metrics such as HPWL, TNS and WNS that we chose to evaluate the quality impact of our optimization technique. The improvement or degradation shown above depicts the difference in the values of total HPWL, TNS, WNS when compared with placement using only square shaped clusters versus placement using variably shaped clusters. Negative values in the table represent the degradation when compared to the execution of placer without shaping.

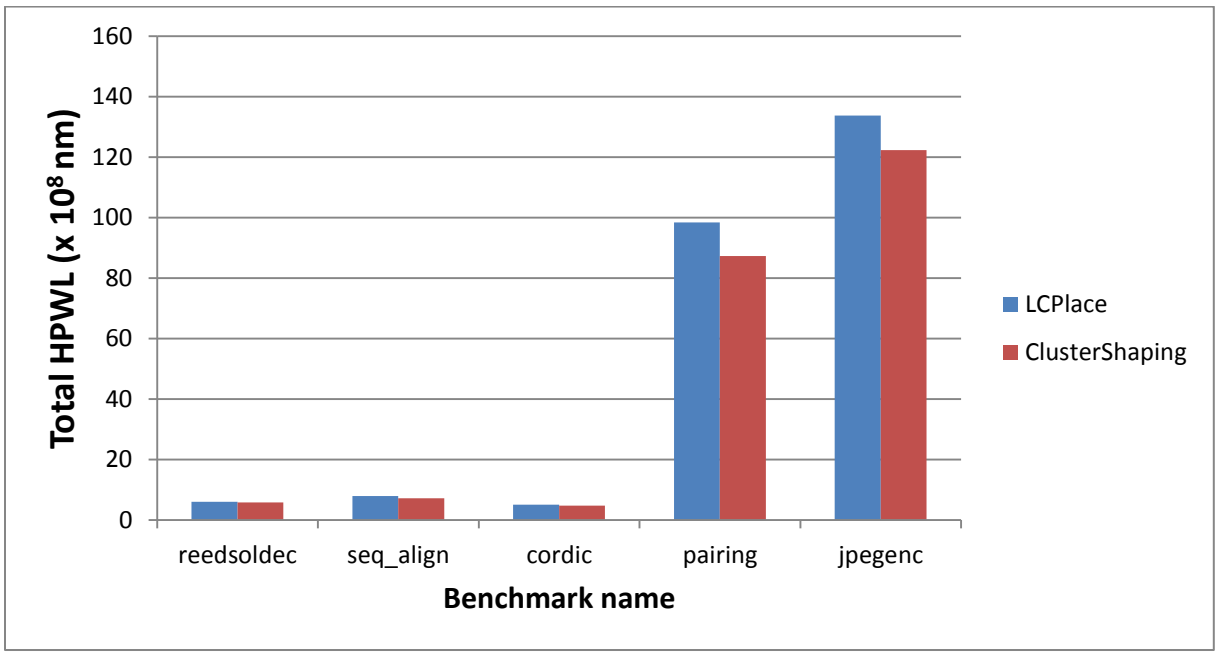


Figure 5.3 Comparison of HPWL values in LCPlace with ClusterShaping

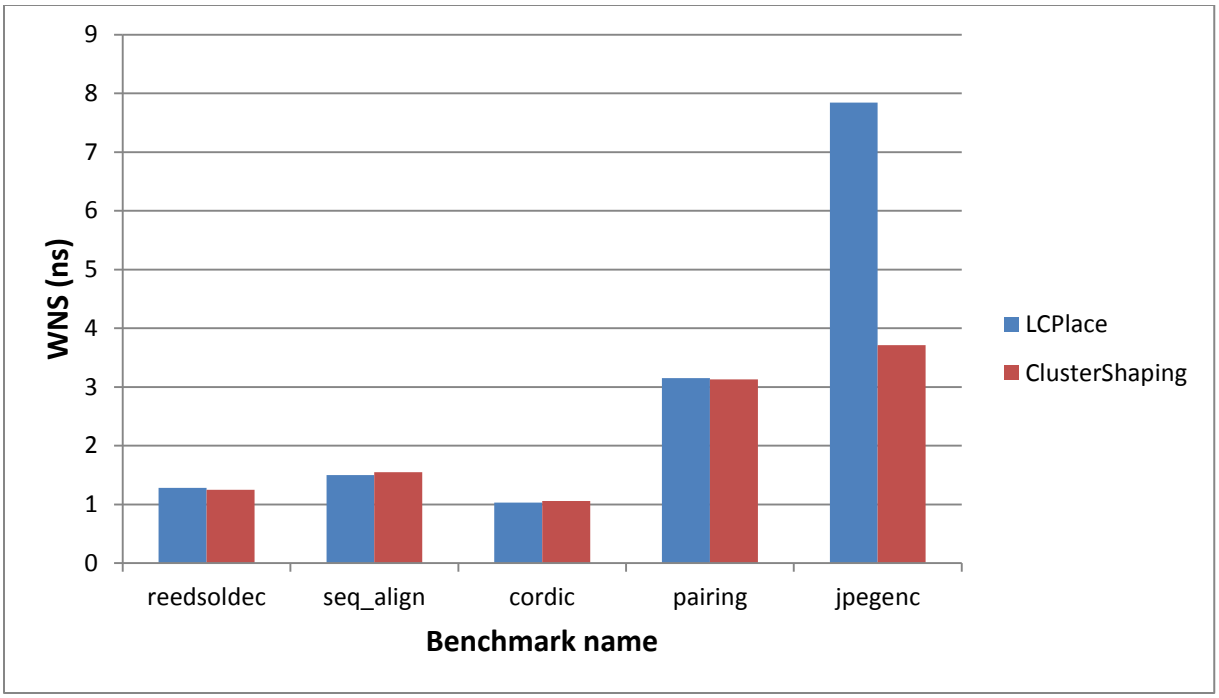


Figure 5.4 Comparison of WNS values for LCPlace with ClusterShaping

5.2.2 Analysis of the results

From table 5.2 and 5.3 above, we can clearly identify some major changes in the quality of placement when the cluster shaping technique is used as compared to that of fixed shape based placement. We notice from the results above that the HPWL values have improved (i.e. decreased) for all the benchmarks using our approach. The major reason for this is changing the shape of the clusters and reducing the overlapping area of the clusters have ensured that the standard cells inside all the clusters are placed in their optimal locations during global placement itself. Therefore no further post processing is required unlike fixed shape clustering based placers like LCPlace. [2] It is during such stages as legalization, the quality of placement degrades when the cluster boundaries are resolved and all overlaps have to be removed. The most important cause for such overlaps to remain after global placement is fixed shapes of all the clusters. So, we find that providing flexibility to the aspect ratios during cluster formation

helps to improve the quality of global placement. As far as timing values are concerned, we do see improvements in WNS values especially for the larger designs.

5.2.3 Impact of number of clusters on placement quality

Benchmark Name	Number of clusters generated	Total HPWL ($x10^8$ nm)	Total overlap (%)	Normalized score
reedsoldec	100	8.97	0.99	99.6
	70	7.09	0.43	75.2
	50	5.83	0	58.3
seq_align	100	10.80	0.91	117.1
	70	9.11	0.83	99.4
	50	7.15	0	71.5
cordic	100	5.77	0.10	58.7
	70	5.31	0	53.1
	50	4.79	0	47.9
pairing	100	107.45	0	1074.5
	70	96.23	0.26	988.3
	50	89.10	0.34	925.0
jpegenc	100	143.19	0.42	1473.9
	70	126.58	0	1265.8
	50	122.34	0	1223.4

Table 5.4 Results showing the impact of number of clusters on placement quality

We have varied the number of clusters to be formed by the partitioning tool to observe its impact on the final placement quality. Table 5.4 above shows the results of these variations on the weighted score function. These scores were normalized to make them easier to comprehend. More details about this score function can be found in chapter 4 above. We have limited parameter variations only to the number of clusters keeping the other parameters such as number of shapes formed, maximum allowed percentage for variation in height, etc. constant for

this round of experimentation. Here we have fixed the number of shapes to be formed for each cluster to be '5'. The maximum height variation percentage is fixed at '50'. All the values reported in the table above are recorded after unclustering step but without running any standard cell legalizer on it. This was done to understand the significance of the normalized scores which are essentially a combination of both HPWL and overlap.

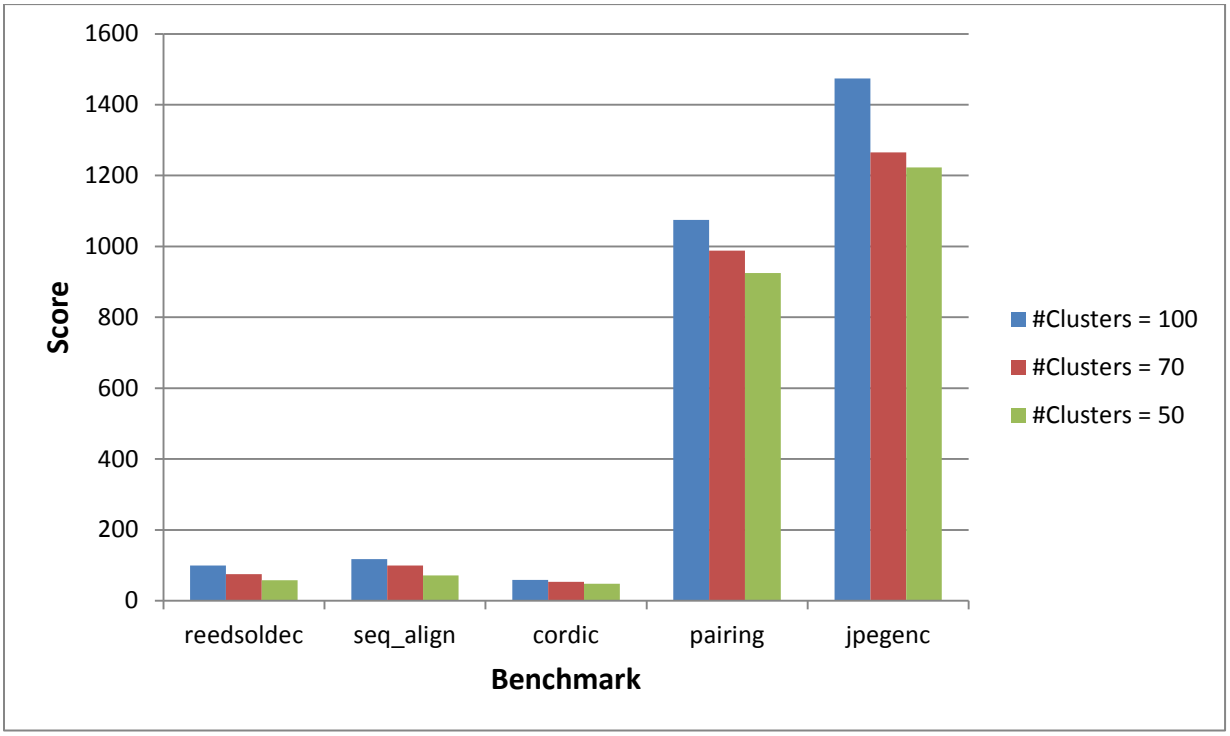


Figure 5.5 Effect of number of clusters on quality of placement

5.2.4 Analysis of the results with variation in number of clusters

The experimental results present in table 5.4 above show how the HPWL values vary when the numbers of clusters are varied from 100 to 50. Our findings from this experiment are that when the number of clusters is reduced, the HPWL decreases i.e. the placement quality improves. This happens because the smaller number of clusters allows the macro placer to perform better

in terms of optimizing a placement when compared to a situation where the number of clusters to be placed is more. We also notice in the results above that the percentage of overlap decreases when the number of clusters formed are relatively less. This stems from the fact that the probability of finding a placement solution with absolutely no overlaps or a negligible overlap is much more for a problem with less number of clusters.

5.2.5 Effect of number of shape variants on the placement solution

We have seen in the previous chapter that the number of shapes generated for a cluster can be varied by providing specific values to the number of variation steps which is taken as an input. We conducted a few experiments to find out the optimal number of shapes that are required to be generated when using this cluster shaping technique. The number of shapes created was varied from 3 to 7 which seemed to be a reasonable range. Throughout all the experiments the number of clusters formed was kept fixed to a value of 50. The maximum percentage of height variation for all the shapes is fixed to be 50. We have recorded the results of the experimental flow only till the unclustering stage. No legalization is performed on the resultant netlist. Also the following steps such as extraction, routing, etc. are not executed for these experiments primarily because here we are only trying to observe the variation in any one of the performance metrics when the other input parameters are kept constant. Execution of those steps incurs a lot of unnecessary runtime overhead during experimentation.

Benchmark Name	Number of shapes generated	Total HPWL (x10^8 nm)	Total overlap (%)	Normalized score
reedsoldec	3	7.25	0	72.5
	5	5.83	0	58.3
	7	5.80	0	58.0
seq_align	3	9.23	0.19	92.49
	5	7.15	0	71.5
	7	7.11	0	71.1
cordic	3	5.34	0	53.4
	5	4.79	0	47.9
	7	4.98	0	49.8
pairing	3	107.9	0.46	1125.1
	5	89.10	0.34	925.0
	7	87.23	0	872.3
jpegenc	3	164.37	0.17	1645.4
	5	122.34	0	1223..4
	7	121.46	0	1214.6

Table 5.5 Results showing the impact of number shape variants on placement quality

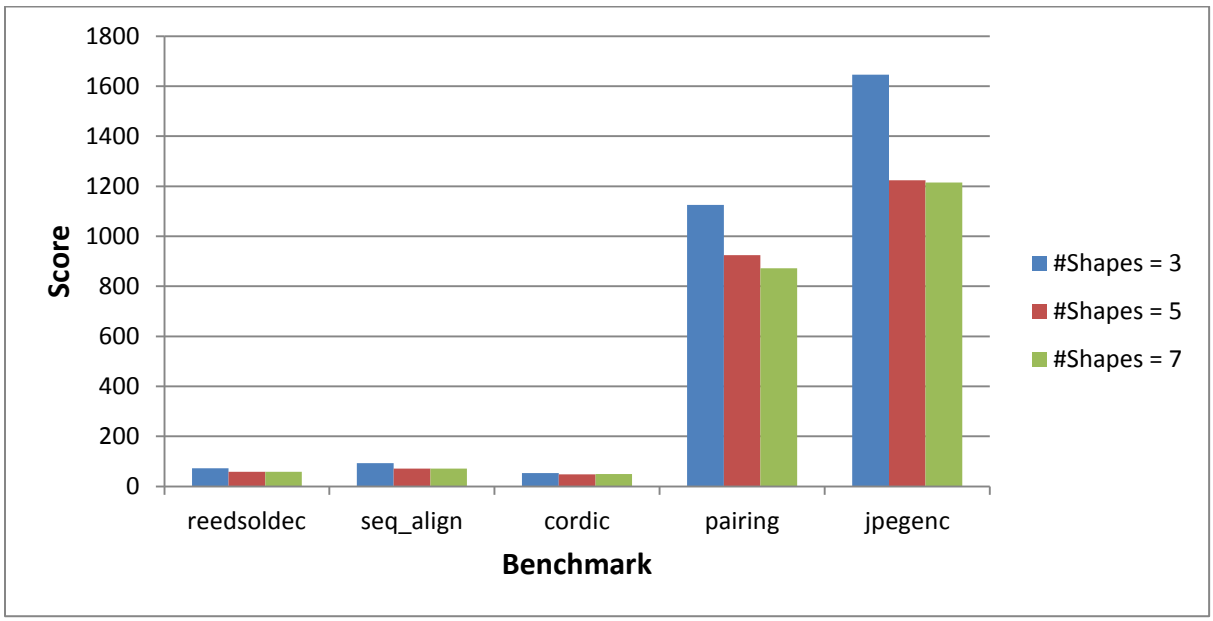


Figure 5.6 Effect of number of shapes on quality of placement

5.2.6 Analysis of results produced by varying the number of shapes for a cluster

Table 5.5 above shows the variation of HPWL when the number of shapes to be generated for a cluster are varied. We have used a range of 3 to 7 as the input to the number of steps for variation because the minimum number of shape variants that should be generated for a cluster is 3 (square, wide variant, narrow variant) and if we try to generate more than 7 shapes for each cluster then the runtime overhead is very large. It will affect the runtime significantly, also because the academic placer used to place the cells inside a cluster has to be run for more than 7 times for a single cluster. As a result these values were chosen as a reasonable number of shape variants for a cluster. From the results listed in table 5.5 above we find that in most of the cases the score improves when the number of shapes is increased from 3 to 5 and then to 7. This can be explained from the fact that as we increases the number of shapes for each cluster the shape selection procedure now has a much larger solution space to choose from during each round of annealing. A larger solution space increases the probability of finding a better solution which is the reason behind getting better scores with more shapes for a cluster. But this improvement in placement quality comes at the cost of runtime which increases with the increase in solution space. So, in other words when the solution space for a problem increases we do find much better solutions but the time taken to find those solutions is much more than that compared to a smaller solution space.

5.2.7 Overlap reduction due to cluster shaping

We wanted to understand the impact of cluster shaping on overlap reduction during global placement. So we performed a few experiments by fixing the cluster shapes and measuring the overlaps and HPWL values right after global placement. These were compared against our regular placement flow with shaping and the results of these experiments are shown in table 5.6 below.

Benchmark Name	HPWL with square shapes (x 10^8 nm)	HPWL with varied shapes (x 10^8 nm)	Improvement in HPWL (%)	Percentage of overlap with square shapes (%)	Percentage of overlap with varied shapes (%)	Improvement in overlap (%)
reedsoldec	5.72	5.83	-3.34	4.81	0	4.81
seq_align	6.45	7.05	-9.30	4.33	0.35	3.98
cordic	4.38	4.79	-9.36	5.82	0	5.82
pairing	84.15	86.19	-2.40	6.97	0.12	6.85
jpegenc	110.38	121.36	-9.90	9.67	0.16	9.51

Table 5.6 Results showing the impact of shaping on overlap reduction

5.2.8 Analysis of results representing the impact of shaping on overlap

In table 5.6 shown above we recorded the results of our experiments only till the unclustering stage of the execution flow. Legalization is not performed on the output netlist. So, the HPWL values for both square as well as varied shapes are the values right after global placement. We can clearly identify a tradeoff between percentage of overlap amongst clusters and total HPWL after global placement. The percentage overlap values are much less when multiple shapes for clusters are used. These came at a cost of degradation in HPWL and these results met our expectations. The HPWL values for executions with square shaped clusters only, further degrades when the residual overlap is removed using a legalizer whereas the HPWL values for execution with variable shapes for clusters degrade by a negligible quantity due the presence of almost no overlaps in the design after global placement.

5.2.9 Reference placer with detailed placement vs placer with shaping

This experiment was performed to compare the final results of LCPlace [2] along with its optimization techniques with our cluster shaping technique. Optimization techniques such as

cluster swapping, flipping as mentioned in [2] was enabled in our reference flow and our cluster shaping technique was used in the other flow by fixing the number of clusters formed to be 50 in both the cases. The results of such experiments is shown below in table 5.7

Benchmark Name	**Number of shapes**	**LCPlace with flipping and swapping**	**Our placer with shaping**	**Improvement (%)**
		HPWL (x 10^8 nm)	HPWL (x 10^8 nm)	
reedsoldec	5	5.26	5.83	-10.83
seq_align	5	7.29	7.15	1.9
cordic	5	4.50	4.79	6.4
pairing	5	89.54	87.23	2.5
jpegenc	5	114.56	122.34	-6.7

Table 5.7 Comparing LCPlace with flipping, swapping and our optimization technique

5.2.10 Comparison of execution times of LCPlace with our cluster-shaping based placement

Benchmark Name	**LCPlace**	**Our placer with shaping**	**Ratio (Ourplacer/LCPlace)**
	Runtime (s)	Runtime (s)	
reedsoldec	41.35	98.54	2.38
seq_align	55.12	135.76	2.46
cordic	26.43	75.39	2.85
pairing	738.48	2450.51	3.31
jpegenc	495.12	1137.72	2.29

Table 5.8 Comparing execution times of LCPlace with our placer

In table 5.8 above we compare the execution times of LCPlace with our cluster shaping based placer. It can be seen that the execution times have increased by 2X to 3X times when shaping technique is incorporated in placement. The reason for this is when multiple shapes are generated for all the clusters; the standard cell placer has to be executed multiple times (5 per cluster in table 5.8 above) in order to fill all the shape variants for a cluster unlike that of LCPlace [2] where the standard cell placer is only executed once per cluster. This increases the overhead incurred significantly thus contributing to the increase in overall execution times. Also file i/o has to be done multiple times when the required files have to be written out in the bookshelf format for all the shape variations and after filling of each shape of a cluster is complete it has to be read back into memory. It is possible to improve the execution times by some of the techniques mentioned in the future work below.

5.2.12 Plots of variably shaped clusters vs square clusters

In figures 5.6 – 5.11 given below we can clearly see how variable shapes are preferred during placement when compared to only square shapes. Each of the figures shown below represents the final placement of all the clusters with their best shape that was selected as a result of annealing and after placing them at their optimal positions. Figure 5.6 is given as a reference to compare placement of clusters with variable shapes against that of square cluster placement. It can also be seen from the plots shown below that the overlapping area has reduced significantly when compared to square shaped clusters. All the plots shown below represent the placement of clusters before unclustering and final legalization is done.

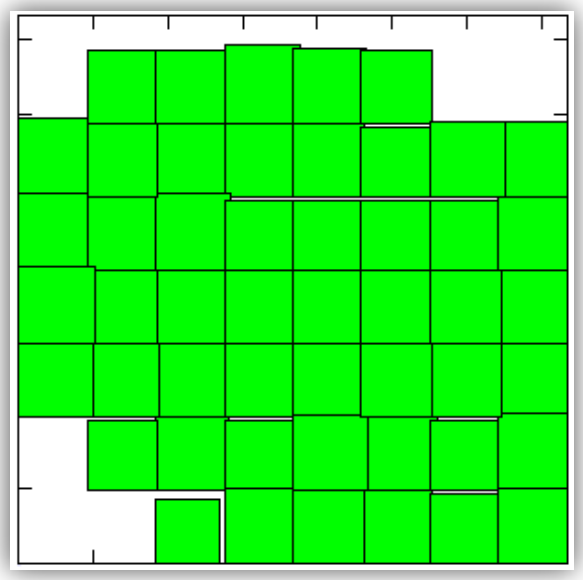


Figure 5.7 Final placement for reedsoldec using only square clusters

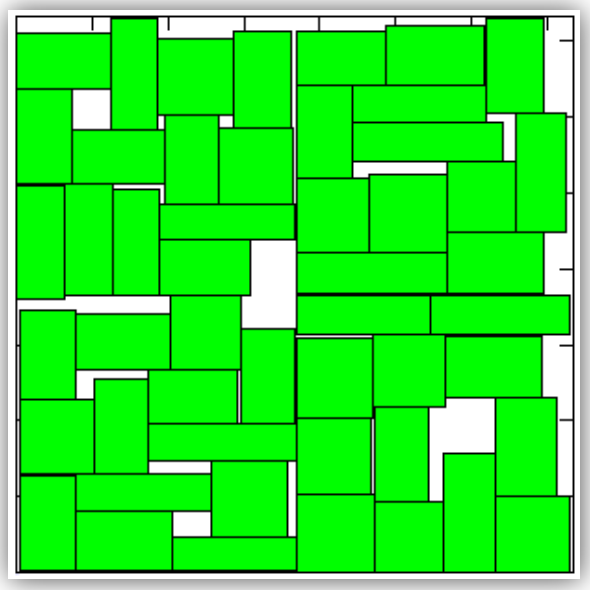


Figure 5.8 Final placement for reedsoldec using variably shaped clusters

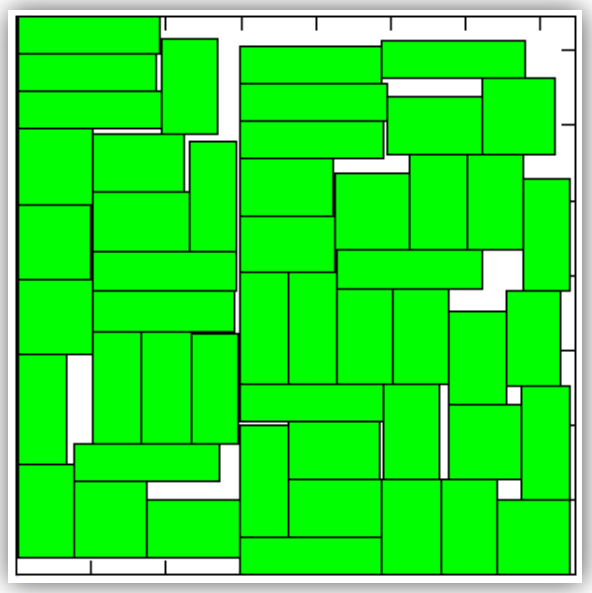


Figure 5.9 Final placement for seq_align using variably shaped clusters

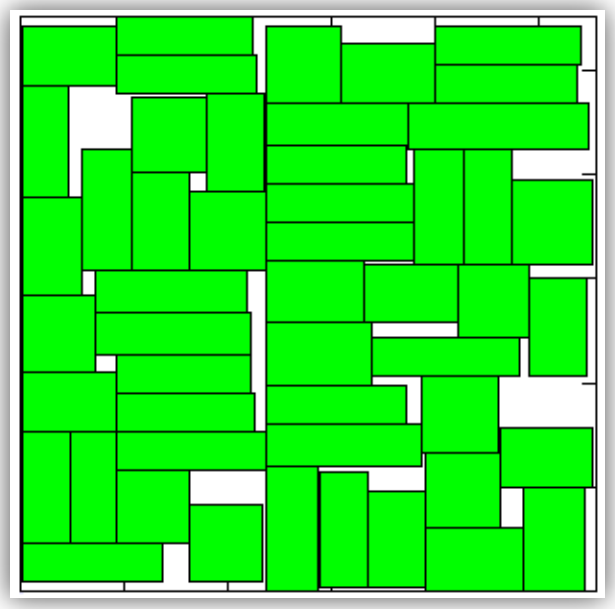


Figure 5.10 Final placement for cordic using variably shaped clusters

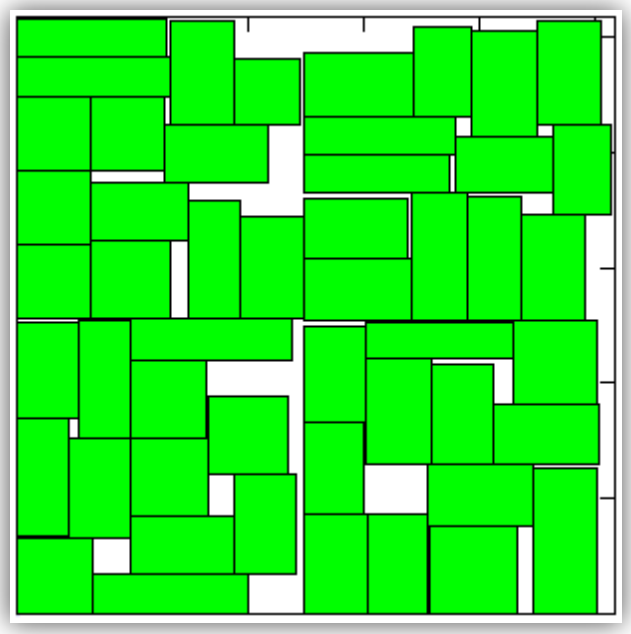


Figure 5.11 Final placement for pairing using variably shaped clusters

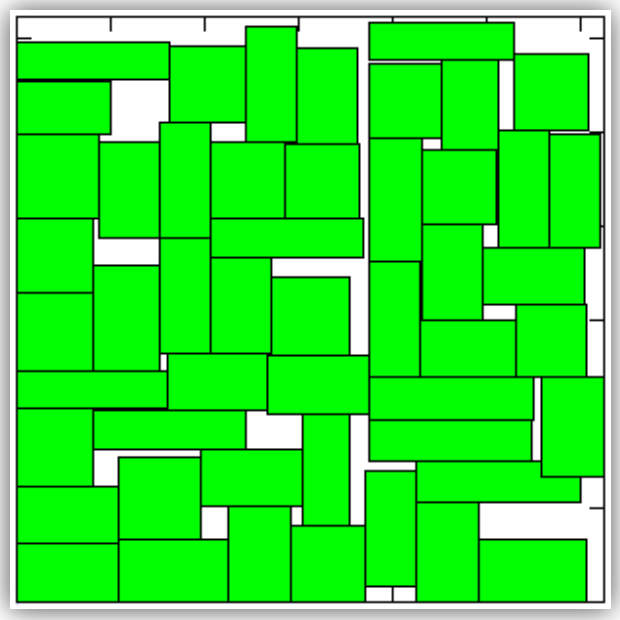


Figure 5.12 Final placement for jpegenc using variably shaped clusters

Chapter 6 CONCLUSIONS AND FUTURE WORK

In this chapter we summarize our findings so far and discuss the conclusions. We also provide an insight on how to pursue further with research along these lines

6.1 Conclusions

In our research on placement optimization, we have proposed a new optimization technique to improve the quality of solutions when clustering is used to form large clusters. We have used the clustering methodology of LCPlace [2] to form large clusters. Once square clusters are created we have generated a 'shape-bank' containing several variations of the square shape in the form of rectangles for each of the clusters. After these shape variants are generated we have placed the cells belonging to the cluster inside all these shapes using a standard academic placer. After all the possible shapes were generated we have implemented several techniques to use these variably sized clusters and found that using a simulated annealing based procedure to select the best shape for a cluster yielded superior results. We have also found that only changing the shape of clusters is not sufficient to give good results. Moving these mixed size clusters to their optimal locations is also a very significant step for improving the final placement solution. We also modified our cost function to include the sum of global as well as internal HPWL and the total overlapping area to evaluate a placement solution. We tried to use a force directed placement technique to move the clusters to their optimal location after variable shape assignment but we were not able to include the overlapping area as a part of the cost function and the placement of clusters using this technique was also not the most efficient one. So, we decided to use a non-deterministic approach to placement of cluster and finally used a mixed size placer used for macro placement, MetaPlacerCapo [39].

Comparing our results to that of LCPlace we found that our cluster shaping technique was able to improve the HPWL values by 8% on an average at the cost of a slight increase in runtime.

6.2 Future Work

In this section we provide insights on possible directions of further research along these lines

6.2.1 Non-linear placement of clusters

In our work we have used a mixed size placer based on a simulated annealing based technique. This technique was able to incorporate multiple parameters (HPWL, overlap) in its objective function but the runtime of this technique is comparatively high. So using a non-linear based placement technique which is faster compared to an annealing based approach is an area of research that promises to yield superior results.

6.2.2 Single placer to place both in-cluster and top level clusters

In our approach we have used two different techniques for placement of cells inside a cluster and placement of the clusters themselves. Overhead is incurred to prepare the inputs in the required format by the external placers and reading them back once again into memory. Instead if we could combine both these objectives into one single placer the execution time would be improved further

6.2.3 Integration with static timing analysis

Integrating a static timing analyzer as a part of our placement is bound to improve the timing values of the placement solution and result in timing closure of the design

6.2.4 Predicting optimal shapes before placement

Another promising area for research would be to come up with a technique to predict the possible best shape for a cluster without actually placing it. This would improve the runtime of this shaping technique a lot by eliminating the need to execute a macro placer for multiple iterations

6.2.5 Parallelizing standard cell placement in all the shapes

If the placement of standard cells inside all the possible shapes for a cluster was done in parallel, then it would also save a lot on runtime overhead. This would decrease the overall runtime requirements to a large extent.

BIBLIOGRAPHY

[1] Multilevel k-way hypergraph partitioning, *Proceedings of the IEEE/ACM Design Automation Conference*, p. 343–348, New Orleans, LA, 1999

[2] Tirumalai Nakul, "*LCPlace: A novel VLSI placement methodology based on large cluster formation*", MS thesis, University of Cincinnati, 2014

[3] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, Y.-W. Chang, "NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints", *IEEE Transactions on Computer-Aided Design and Systems*, p. 1228-1240, Vol. 27, No. 7, 2008

[4] http://www.willnaylor.com/mantext/wnanl.txt

[5] J. A. Roy, S. N. Adya, D. A. Papa and I. L. Markov, ``Min-cut Floorplacement'', *IEEE Trans. on Computer-Aided Design*, vol. 25, no. 7, pp. 1313-1326, 2006

[6] Viswanathan, Natarajan, Chu, Chris C.-N., "FastPlace: Efficient Analytical Placement Using Cell Shifting, Iterative Local Refinement, and a Hybrid Net Model", *IEEE Transactions on Computer-Aided Design of Integrated Circuits And Systems,* p. 722-733, Vol. 24, No. 5, 2005

[7] OpenCores, "OpenCores", http://www.opencores.org

[8] *ISPD 2005 Placement Contest*, http://www.sigda.org/ispd2005/contest.htm

[9] *ISPD 2006 Program*, http://www.ispd.cc/program.html

[10] *ISPD02 IBM-MS Mixed-Size Placement Benchmarks*, S.N. Adya and I. L. Markov, http://vlsicad.eecs.umich.edu/BK/ISPD02bench

[11] Wang L, Chang Y, Cheng K, “Electronic Design Automation: synthesis, verification and test”, p 635-638, Morgan Kaufmann publications, 2009

[12] Viswanathan, Natarajan, "Placement techniques for the physical synthesis of nanometer-scale integrated circuits" (2009). *GraduateTheses and Dissertations.* Paper 10758.

[13] Ulrich Brenner, Anna Pauli, and Jens Vygen, “Almost optimum placement legalization by minimum cost flow and dynamic programming”, ISPD, page 2-9. ACM, (2004**)**

[14] Ren, Haoxing, Pan, David Z., Alpert, Charles J., Villarrubia, Paul G. and Nam, Gi-Joon. "Diffusion-Based Placement Migration With Application on Legalization”. *IEEE Trans. on CAD of Integrated Circuits and Systems* 26, no. 12 (2007): 2158-2172.

[15] Brenner, U, “VLSI legalization with minimum perturbation by iterative augmentation”, *in Wolfgang Rosenstiel & Lothar Thiele, ed., ‘DATE’, IEEE,* (2012) pp. 1385-1390.

[16] A. R. Agnihotri, S. Ono, and P.H. Madden, “Recursive Bisection Placement: Fengshui 5.0, implementation details”, p. 230-232, *Proc. of ISPD,* 2005

[17] T.-C. Chen, T.-C. Hsu, Z.-W. Jiang and Y.-W. Chang, “NTUPlace: a ration partitioning based placement algorithm for large-scale-mixed size designs”, p. 236-238, *Proc. of ISPD,* 2005

[18] Sait, M. Sadiq, Youssef, Habib, “VLSI Design Automation, Theory and Practice”, p. 179-189, World Scientific Publishing Co. Pte. Ltd., 1999

[19] C. Sechen and A. L. Sangiovanni-Vincentelli, “TimberWolf 3.2: A new standard cell placement and global routing package,” in *Proc.ACM/IEEE Design Automation Conf.*, pp. 432-439, 1986

[20] M. Wang, X. Yang, and M. Sarrafzadeh, “Dragon2000: Standard-cell placement tool for large industry circuits,” in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pp. 260-263, 2000

[21] Kleinhans M., Jürgen, Sigl, George, Johannes M., Frank, Antreich J., Kurt, “GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization”, *IEEE Transactions on Computer-Aided Design*, p. 356-365, Vol. 10, No. 3, 1991

[22] Z.-W. Jiang, H.-C Chen, T.-C. Chen and Y.-W. Chang, “Challenges and Solutions in Modern VLSI Placement”, *VLSI-DAT*, p. 1-5, 2007

[23] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, “Optimization by simulated annealing”, *Science*, New Series, Vol. 220, No. 4598, p. 671-680 1983

[24] A. Agnihotri et al., “Fractional cut: Improved recursive bisection placement”, *ICCAD*, San Jose, CA, p. 307–310, 2003

[25] P. R. Suaris and G. Kedem, “An algorithm for quadrisection and its application to standard cell placement”, *IEEE Transactions on Circuits and Systems*, p. 294-303, Vol. 35, No. 3, 294–303, 1988

[26] Wang L, Chang Y, Cheng K, “Electronic Design Automation: synthesis, verification and test”, p 653-655, Morgan Kaufmann publications, 2009

[27] Alpert C. J., Mehta D. P, Sapatnekar S. S., “Handbook of algorithms for physical design automation”, p. 328-331, New ed., New York: CRC, 2009. Print.

[28] Viswanathan N., Gi-Joon Nam, Alpert, C.J, Villarubia, P., Haoxing Ren, Chu C., “RQL: Global Placement via Relaxed Quadratic Spreading and Linearization”, *Design Automation Conference*, p. 453-458, 2007

[29] Tirumalai Nakul, "*LCPlace: A novel VLSI placement methodology based on large cluster formation*",p 14-16 MS thesis, University of Cincinnati, 2014

[30] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain", *Proceedings of the IEEE/ACM Design Automation Conference*, p. 526–529, Anaheim, CA, 1997

[31] Tirumalai Nakul, "*LCPlace: A novel VLSI placement methodology based on large cluster formation*", MS thesis, University of Cincinnati, 2014

[32] J. Cong, S. K. Lim, "Edge Separability-Based Circuit Clustering With Application to Multilevel Circuit Partitioning", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, p. 346-357, Vol. 23, No. 3, 2004

[33] Hu, Bo, M.-Sadowska, Malgorzata, "Fine Granularity Clustering-Based Placement", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems"*, p. 527, Vol. 23, No. 4, 2004

[34] Alpert, C., Kahng A., Gi-Joon Nam, Reda, S., Villarrubia, Paul, "A Semi-Persistent Clustering Technique for VLSI Circuit Placement", *Proc of ISPD*, p. 200-207, 2005

[35] Li, Jianhua, Behjat, Laleh, Kennings, Andrew, "Net Cluster: A Net-Reduction-Based Clustering Preprocessing Algorithm for Partitioning and Placement", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, p. 669-679, Vol. 26, No. 4, 2007

[36] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions", *Proceedings of the ACM Design Automation Conference*, p. 175–181, Washington D.C., 1982

[37] Yan Z. Jackey, Chu, Chris, Mak, W.-K., “SafeChoice: A Novel Approach to Hypergraph Clustering for Wirelength-Driven Placement”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, p. 1020-1033, Vol. 30, No. 7, 2011

[38] Rodrigues N. J., Kamuf M., Hedburg H., Owall V., “A Manual on ASIC Front to Back End Design Flow”, *IEEE International Conference on Microelectronic Systems Education,* 2005

[39] http://vlsicad.eecs.umich.edu/BK/PDtools/Capo