3D Standard-Cell Placement Based on Circuit Partitioning

Delakoura Angeliki

A Thesis presented for the degree of

Master of Science in
Electrical and Computer Engineering

Supervisor: Christos Sotiriou, Associate Professor
Committee: Nikolaos Bellas, Associate Professor
Nestoras Eumorfopoulos, Assistant Professor

University of Thessaly
Volos, Greece

June 2016

Dedicated to

My parents.

3D Standard-Cell Placement Based on Circuit Partitioning

Τρισδιάστατη Τοποθέτηση Πρότυπων Στοιχείων Βασισμένη σε Διαμερισμό Κυκλωμάτων

by

Delakoura Angeliki

Submitted to

the Department of Electrical and Computer Engineering

University of Thessaly, Volos, Greece

for the degree of Master of Science in

Electrical and Computer Engineering

June 2016

Acknowledgements

I would like to thank my advisors Dr. Nikolaos Bellas, Dr. Nestoras Eumorfopoulos and especially Dr. Christos Sotiriou for the great collaboration, the ideas, the inspiring discussions and for their guidance.

I would also like to thank specifically my good friends and colleagues Nikolaos Sketopoulos and Michalis Spyrou for all their help, insight and support during the development of my thesis.

In conclusion, I would like to thank my parents for the support they provided me through my entire life, for all the sacrifices they made on my behalf and for believing in me.

Abstract

For many years Moore's law have been accurate to its predictions and has made the work of many integrated circuit designers full of obstacles they needed to overcome. The excessive wire-length on chip, which created a great routing congestion, many delays for the designs, extreme on-chip temperatures and as a result great amounts of energy consumption, are the main reasons why the Semiconductor Industry has turned its interest towards 3D integrated circuits.

The purpose of this thesis is the study and implementation of a 3D placer that combines different methodologies and algorithms, and takes advantage of their techniques in order to achieve an overall optimal work for 3D standard cell placement that could be applied to industrial benchmarks and work as part of an actual industrial physical design tool.

Περίληψη

Για πολλά χρόνια ο νόμος του Moore ήταν ακριβής με την προβλέψη του και έχει κάνει τη δουλειά πολλών σχεδιαστών ολοκληρωμένων κυκλωμάτων γεμάτη εμπόδια, τα οποία ήταν αναγκαίο να ξεπεραστούν. Το υπερβολική wire-length στο chip, η οποία δημιούργησε μια μεγάλη δρομολόγησης συμφόρηση, πολλές καθυστερήσεις, ακραίες θερμοκρασίες ον-σηιπ και ως αποτέλεσμα μεγάλη κατανάλωση ενέργειας, αποτελούν τους κύριους λόγους για τους οποίους η Βιομηχανίας Ημιαγωγών έχει στρέψει το ενδιαφέρον της προς τα τρισδιάστατα ολοκληρωμένα κυκλώματα.

Ο σκοπός αυτής της μεταπτυχιακής εργασίας είναι η μελέτη και υλοποίηση ενός 3D placer που συνδυάζει διαφορετικές μεθοδολογίες και αλγορίθμους, και εκμεταλλεύεται τις τεχνικές τους, προκειμένου να επιτευχθεί ένα συνολικά βέλτιστο έργο για την τρισδιάστατη τοποθέτηση πρότυπων στοιχείων που θα μπορούσε να εφαρμοστεί σε βιομηχανικά designs και να λειτουργήσει ως μέρος ενός πραγματικού βιομηχανικού εργαλείου φυσικής σχεδίασης.

Declaration

The work in this thesis is based on research carried out at the University of Thessaly, Electrical and Computer Engineering Department, Greece. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose of This Thesis

Three dimensional integrated circuits (3D-ICs) have emerged as a promising solution to extend the 2D scaling trajectory predicted by Moore's Law. Current 3D-ICs are through-silicon-via (TSV) based, but the integration density is limited by the pitch of TSVs. Monolithic 3D IC is an emerging technology that enables orders of magnitude higher integration density than TSV based 3D, due to the extremely small size of the monolithic inter-tier vias (MIVs) (typically 50nm in diameter).

This thesis presents our work and our purpose, which is a 3D standard-cell placement with the minimal wire-length for the design and the minimal connections between the different chip levels. In order to achieve our purpose, we apply a 2D Quadratic Programming Placement, with respect to pin and cell connectivity, partition cells with the minimum cutsize and finally have a 3D Placement with the minimum wire-length and minimum tier-to-tier connections.

1.2 Thesis Structure

This thesis is divided in four main parts. The first part discusses background issues regarding every aspect of our work. More specifically, section 2.1 presents information about partitioning algorithms and focuses on the Fiduccia-Mettheyses

algorithm, which is the algorithm that we based our partitioning implementation on, and present an detailed example of the algorithm flow. In section 2.2 we discuss about Global Placement and different techniques and emphasise on Analytical Placement and specifically analyse Quadratic Programming Placement through an example. Finally, in section 2.3 we discuss about different 3D placement methods, present the new emerging technology that was the motivation for our work and explain how 3D placement is applied nowadays.

In the second part we present our contribution and implementation of the three different main aspects of our work, 2D Placement, FM Partitioning algorithm and 3D Placement, along with pseudocodes for the QP problem formulation and the partitioning logic, in order to understand in depth our implementation of the algorithms.

In the third part we present the experimental results that came from testing our work with industrial benchmarks. First, the experimental results regarding partitioning are shown and then results for the final 3D placement wire-length compared to 2D placement wire-length.

Finally, chapter 5 we describe the conclusions that we came to and discuss possible future work.

Chapter 2

Background

In this chapter we describe basic information regarding Global Analytical Placement, Partitioning Algorithms and 3D Placement for a better understanding of our work.

2.1 Partitioning Algorithms

Modern standard cell placement techniques must handle more and more increasing design sizes, which eventually turn out to be huge. It is practically infeasible to place flattened representations of designs of such scale, due to the difficulty and the enormous number of the computations needed. A significant move in cell placement of such large designs, is first to obtain a smaller representation of the design that depicts the global connectivity of the original design. This is what is known as partitioning and clustering. Partitioning is the method that we applied in our work and is typically used to divide a netlist into two or four sections, then recursively applied to the subsections in order for the wiring cost between them to be minimized.

Just like many other combinatorial optimization problems, circuit partitioning is NP-hard; as the problem size grows linearly, the effort needed to find an optimal solution grows faster than any polynomial function. To date, there is no known polynomial-time, globally optimal algorithm for balance-constrained partitioning. However, several efficient heuristics were developed in the 1970s and 1980s. These algorithms find high-quality circuit partitioning solutions and are virtually imple-

mented to run in low-order polynomial time – the Kernighan-Lin (KL) algorithm [1], its extensions and the Fiduccia-Mattheyses (FM) algorithm [2]. Additionally, difficult partitioning formulations can be solved by using simulated annealing for optimization.

In our work, we focus on the Fiduccia-Mattheyses partitioning algorithm, which is fully described in the following subsection along with a practical example in order to be easily understood.

2.1.1 Fiduccia-Mattheyses Algorithm

The Fiduccia-Mattheyses algorithm was our algorithm of choice for the partitioning process in our work and it is a widely-used heuristic algorithm for the Balanced Circuit Bipartitioning Problem [3].

FM has inherited many concepts from the KL algorithm, such as the way the algorithm's cost function is computed, that is each cell transfer from one partition to another occurs based on a certain move gain for each cell. One more concept inherited from KL is the hill-climbing logic of the algorithm, where each cell is moved only once during a single pass of the algorithm, but all cells are moved even though their movement may not result to a better cutsize. Thus, bad moves are accepted in order to achieve the optimal minimum solution at each pass.

Additionally, FM improves some very significant concepts of KL. First of all, FM is applied directly on hypergraphs, which is a natural way to represent circuits, instead KL is applied on an edge-weighted undirected graph. It has been shown that it is not possible to assign weights to the edges of an undirected graph G so that any cut in G correctly may represent the cutsize in the original circuit [4]. On the other hand, the cutsize computed in a hypergraph represents exactly the cutsize in the actual circuit.

Another improved concept is that FM performs *cell moves* where KL performs *cell swaps*. At each move, the cell with the maximum gain is chosen to move to the other partition. Each cell move is constrained by the area balance requirement so that the move is legal only when the area constraint is not violated after the move have been applied. This cell move improves time complexity significantly, because

we do not need an $O(n^2)$ for all-pair swap gain computation as in KL, instead we just need an $O(n)$ gain computation.

Finally, FM makes use of a special data structure called *bucket* that allows $O(1)$ search of the maximum gain and $O(1)$ update of the gain values at each move. In the beginning and before a pass of the FM is applied, the gain values of all cells are computed. Once the pass begins, the gain values of affected cells, that is the cells connected to the one that has been moved, are the only gain values updated/recalculated instead of computing once more the gain values of all cells from scratch. As a result, the complexity of each cell move is $O(1)$ instead of $O(n2)$ as in KL. Thus, the overall time complexity of FM is $O(n)$ compared to $O(n3)$ in KL.

Algorithm Overview

The algorithm begins with an initial balanced bipartitioning solution (P_1, P_2) of the given hypergraph and is usually obtained randomly. For a cell $c \in P_1$, we define $FS(c)$ as the number of nets that have c as the only cell in P_1 and $TE(c)$ as the number of nets that contain c and are entirely located in P_1. Finally, the gain of moving c from P_1 to P_2 is defined as:

$$gain(c) = FS(c) - TE(c)$$

Before the first pass of the algorithm is applied, three actions need to be performed:

- unlock all cells
- compute the gain of all cells based on the initial random partitioning
- add cells in the bucket structure

Once the pass of the algorithm begins, the following steps are repeated at each move until all cell are defined as locked:

- the cell with the maximum gain is selected under the condition that moving the cell to the other partition does not violate the balance constraint

- the chosen cell is moved and locked in the destination partition
- the gain values of the cells connected to the moved cell are updated and so are their positions in the bucket structure
- the gain and cutsize are recorded

At the end of the pass, the first K moves that lead to the minimum cutsize discovered during the entire pass are identified and accepted. If the initial cutsize has reduced during the current pass, another pass is attempted using the best solution discovered from the current pass as the initial solution, otherwise the algorithm terminates.

Algorithm Example

Figure 2.1 shows a gate-level circuit and the same circuit modeled as an non-weighted hypergraph.

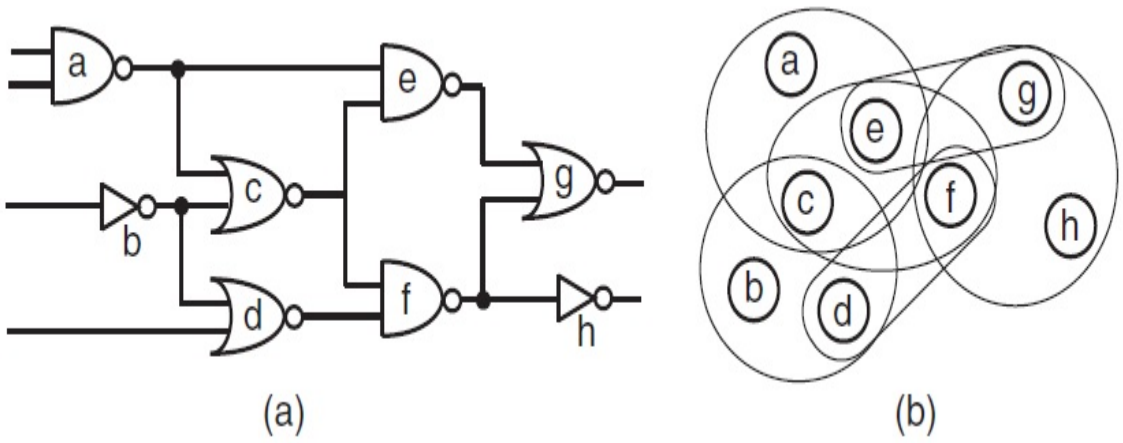


Figure 2.1: (a) Gate-level circuit, (b) Hypergraph representation.

Suppose an initial partition $\{acdg, befh\}$, which can be seen in Figure 2.2. The gain of the cells in the left partition is computed as follows:

1. Cell a: a is contained in net $n1 = \{a, c, e\}$. But a is not the only cell in $n1$ that is located in the left partition, so $FS(a) = 0$. In addition, $n1$ is not entirely located in the left partition. So, $TE(a) = 0$. Thus, $gain(a) = FS(a) - TE(a) = 0$.

2. Cell c: c is contained in net $n1 = \{a, c, e\}$, $n2 = \{b, c, d\}$, and $n3 = \{c, f, e\}$. $n3$ contains c as its only cell located in the left partition, so $FS(c) = 1$. In addition, none of these three nets are located entirely in the left partition. So, $TE(c) = 0$. Thus, $gain(c) = 1$.
3. Cell d: d is contained in net $n2 = \{b, c, d\}$ and $n5 = \{d, f\}$. $n5$ contains d as its only cell located in the left partition, so $FS(d) = 1$. In addition, none of these two nets are located entirely in the left partition. So, $TE(d) = 0$. Thus, $gain(d) = 1$.
4. Cell g: g is contained in net $n6 = g, e$ and $n4 = g, f, h$. Both $n6$ and $n4$ contain g as their only cell located in the left partition, so $FS(g) = 2$. In addition, none of these two nets are located entirely in the left partition. So, $TE(g) = 0$. Thus, $gain(g) = 2$.

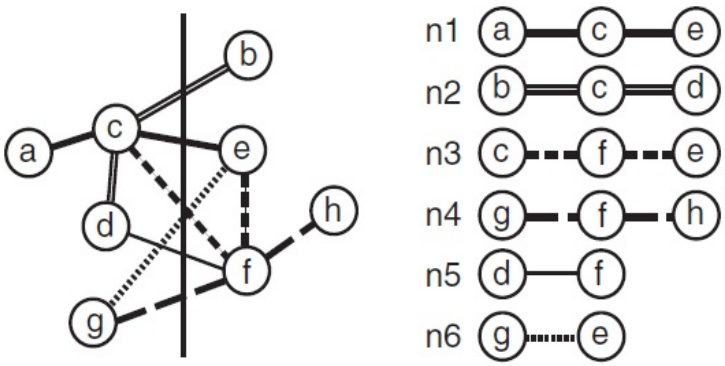


Figure 2.2: The initial partition of the circuit in Figure 2.1, cutsize = 6, and the nets of the circuit.

The gain of the cells in the right partition is computed as follows:

1. Cell b: b is contained in net $n2 = \{b, c, d\}$. $n2$ contains b as its only cell located in the right partition, so $FS(b) = 1$. In addition, $n2$ is not entirely located in the right partition, so, $TE(b) = 0$. Thus, $gain(b) = 1$.
2. Cell e: e is contained in net $n3 = \{c, f, e\}$, $n6 = \{g, e\}$, and $n1 = \{a, c, e\}$. Both $n6$ and $n1$ contain e as their only cell located in the right partition, so $FS(e) = 2$. In addition, none of these three nets are entirely located in the right partition, so, $TE(e) = 0$. Thus, $gain(e) = 2$.

3. Cell f: f is contained in net $n3 = \{c, f, e\}$, $n5 = \{d, f\}$, and $n4 = \{g, f, h\}$. $n5$ contains f as its only cell located in the right partition, so $FS(f) = 1$. In addition, none of these three nets are entirely located in the right partition, so, $TE(f) = 0$. Thus, $gain(f) = 1$.
4. Cell h: h is contained in net $n4 = \{g, f, h\}$. But, h is not the only cell in $n4$ that is located in the right partition, so $FS(h) = 0$. In addition, $n4$ is not entirely located in the right partition. So, $TE(h) = 0$. Thus, $gain(h) = 0$.

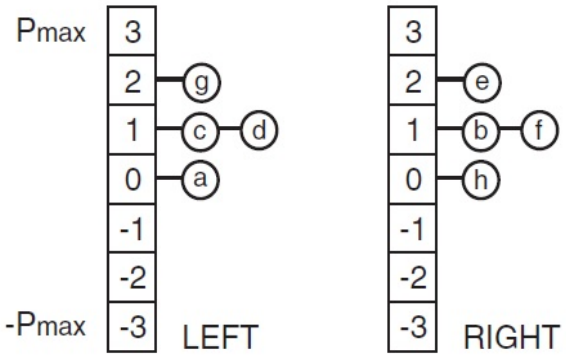


Figure 2.3: Bucket structure based on Figure 2.2

In Figure 2.3 we see the bucket structure for this design and how the cells are sorted based on their gain. In the example that follows, the moves of a single pass of the FM algorithm are described based on the area constraint [3,5], which means a cell's move should not result in source partition's total being smaller than 3 and destination partition's total larger than 5. Gain ties should be broken in alphabetical order.

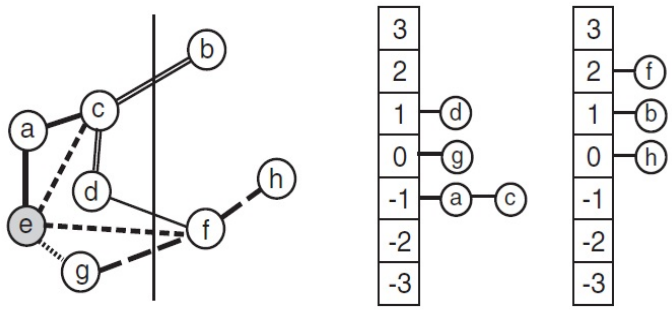


Figure 2.4: After moving e. Cutsize = 4

a) Move 1: From Figure 2.2, can be seen that cells g and e have the maximum gain and can be moved without violating the area constraint. e is moved based on alphabetical order. Figure 2.4 shows the resulting hypergraph. Then, the gains of the unlocked neighbors of e are updated as follows: $gain(a) = FS(a) - TE(a) = 0 - 1 = -1$, $gain(c) = 0 - 1 = -1$, $gain(g) = 1 - 1 = 0$, $gain(f) = 2 - 0 = 2$. Figure 2.4 shows the updated bucket structure.

b) Move 2: f is the cell with the maximum gain, but moving f will violate the area constraint. So d is moved based on its gain and alphabetical order. Figure 2.5 shows the resulting hypergraph. Then, the gains of the unlocked neighbors of d are updated as follows: $gain(b) = 0-0 = 0$, $gain(c) = 1-1 = 0$, $gain(f) = 1 - 1 = 0$. Figure 2.5 shows the updated bucket structure.

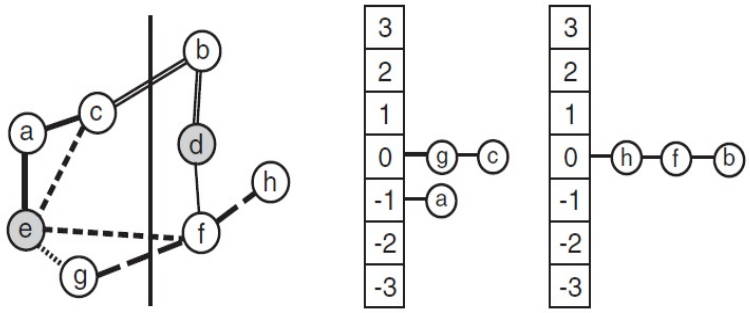


Figure 2.5: After moving d. Cutsize = 3

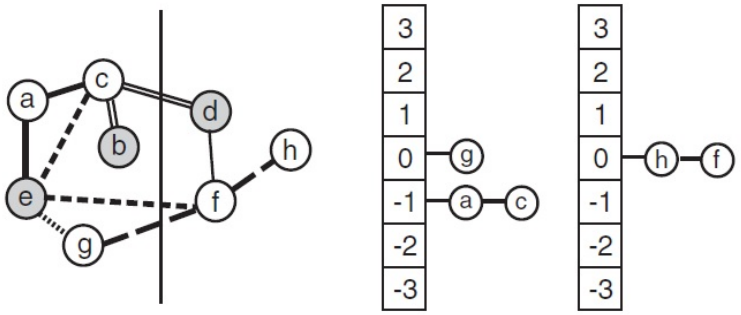


Figure 2.6: After moving b. Cutsize = 3

c) Move 3: Since cells $\{g, c, h, f, b\}$ all have the maximum gain, cell b is selected based on alphabetical order. Figure 2.6 shows the resulting hypergraph. Then,

the gains of the unlocked neighbors of b are updated as follows: $gain(c) = 0 - 1 = -1$. Figure 2.6 shows the updated bucket structure.

d) Move 4: Among ells $\{g, h, f\}$ with maximum gain, g based on the area constraint. Figure 2.7 shows the resulting hypergraph. Then, the gains of the unlocked neighbors of g are updated as follows: $gain(f) = 1 - 2 = -1$, $gain(h) = 0 - 1 = -1$. Figure 2.7 shows the updated bucket structure.

e) Move 5: a is selected based on alphabetical order. Figure 2.8 shows the resulting hypergraph. Then, the gain of the unlocked neighbor of a is updated as follows: $gain(c) = 0 - 0 = 0$. Figure 2.8 shows the updated bucket structure.

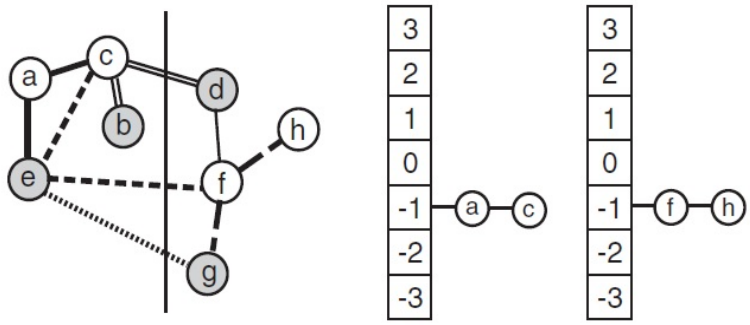


Figure 2.7: After moving g. Cutsize = 3

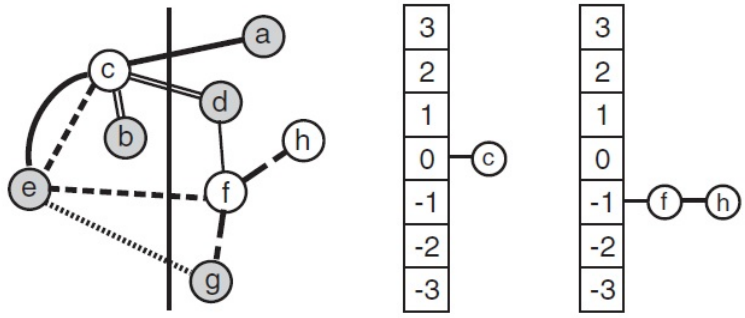


Figure 2.8: After moving a. Cutsize = 4

f) Move 6: f is selected based on the area constraint and alphabetical order. Figure 2.9 shows the resulting hypergraph. Then, the gains of the unlocked neighbors of f are updated as follows: $gain(h) = 0 - 0 = 0$, $gain(c) = 0 - 1 = -1$. Figure 2.9 shows the updated bucket structure.

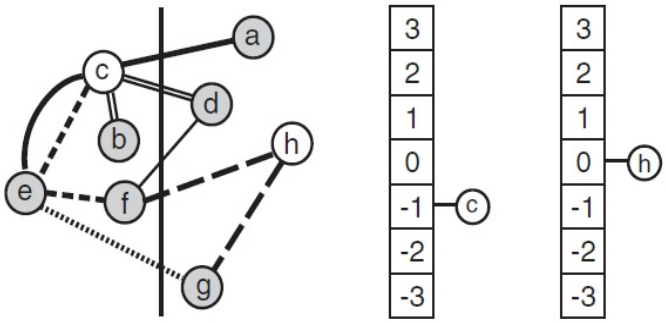


Figure 2.9: After moving f. Cutsize = 5

g) Move 7: h is moved. Figure 2.10 shows the resulting hypergraph and bucket structure. h has no unlocked neighbors.

h) Move 8: c is moved. Figure 2.11 shows the resulting hypergraph.

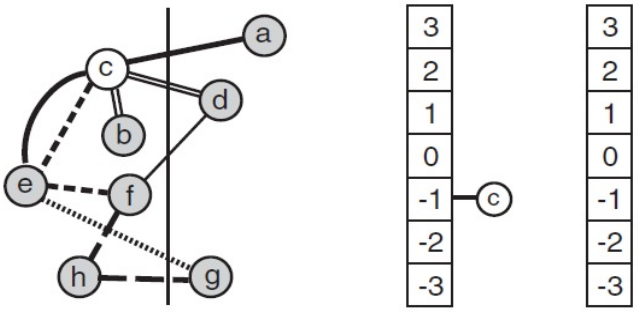


Figure 2.10: After moving h. Cutsize = 5

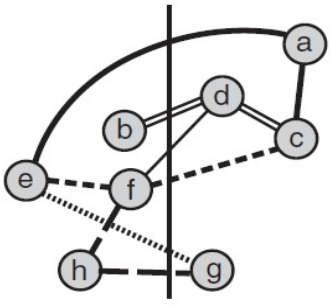


Figure 2.11: After moving c. Cutsize = 6

The cutsize after each move is recorded in a history log from which the k first moves that result in the minimum cutsize can be found. Table 2.1 shows the initial

cutsize, the cutsize after each move and the move at which the minimum cutsize was reached. Also, the maximum and minimum cutsize can be seen.

i	Cell	$Gain(i)$	Cutsize
0	-	-	6
1	e	2	4
2	d	1	**3**
3	b	0	3
4	g	0	3
5	a	-1	4
6	f	-1	5
7	h	0	5
8	c	-1	6

Table 2.1: History log of all the moves for one pass of FM partitioning algorithm.

2.2 Global Placement

After floorplanning determines block outlines and pin locations, placement seeks to assign to standard cells or logic elements within each block their locations while satisfying certain optimization objectives. The techniques of global cell placement are divided into three different categories.

In *partitioning-based algorithms*, the circuit and the layout are divided into smaller sub-circuits and sub-regions, respectively, according to cost functions based on cutsize. This process is repeated until each sub-circuit and sub-region is small enough to be tackled in an optimal way. An example of this approach is *min-cut placement*.

In *analytical algorithms* the placement problem is modeled using an objective-cost function, which can be maximized or minimized through mathematical analysis. The cost function can be quadratic or in other cases non-convex. Examples of such techniques are *quadratic placement* and *force-directed placement*.

In *stochastic algorithms*, randomized moves are used to optimize the cost function. This means that in search of a globally optimal solution and in order to reach it, locally worse moves are accepted. An example of this approach is simulated annealing.

In this work we focus on *analytical placement* and specifically on *quadratic placement*. In the following subsection, we explain analytical and quadratic placement in more depth.

2.2.1 Analytical Placement

Analytical placement minimises or maximises a given objective depending on the goal, for example in some cases tries to minimise wire-length or circuit delay, using mathematical techniques such as numerical analysis, quadratic programming or linear programming. The use of such methods often requires certain assumptions, such as treating placeable objects as dimensionless points. After such algorithms have been applied and cells have been placed too close, that is creating overlaps, the cell locations need to be spread further apart by post-processing methods, so as to remove overlaps.

Quadratic Placement [5]

In quadratic placement the aim is to minimise the total length of connections between elements, that is minimise the total wire-length of the given design. The elements need to be treated as dimensionless points and later after the elements have been placed, the dimensions are being considered too.

The function that best models the total length of connections is the *squared Euclidean distance*,

$$L(P) = \sum_{i=1,j=1}^{n} c(i,j)\Big((x_i - x_j)^2 + (y_i - y_j)^2\Big)$$

which gives the total distances between the cells. n is the total number of cells and $c(i,j)$ is the connection cost between cells i and j, which is equal to zero, $c(i,j) = 0$, when cells i and j are not connected. Terms $(x_i - x_j)^2$ and $(y_i - y_j)^2$ respectively give

the squared horizontal and vertical distances between the centers of i and j. The quadratic formulation emphasises on the minimisation of long connections, which tend to have negative impacts on timing.

In quadratic placement, each dimension can be considered independently. Therefore, the cost function $L(P)$ can be separated into x- and y-components

$$L_x(P) = \sum_{i=1,j=1}^{n} c(i,j)(x_i - x_j)^2 \quad \text{and} \quad L_y(P) = \sum_{i=1,j=1}^{n} c(i,j)(y_i - y_j)^2$$

With these cost functions, the placement problem turns into a convex quadratic optimization problem, which implies that any local minimum solution is also a global minimum and therefore, the optimal x- and y-coordinates can be found by setting the partial derivatives of $L_x(P)$ and ${}_Ly(P)$ to zero

$$\frac{\partial L_x(P)}{\partial X} = AX - b_x = 0 \quad \text{and} \quad \frac{\partial L_y(P)}{\partial Y} = AY - b_y = 0$$

Two systems of linear equations are formulated, where A is a matrix with $A[i][j] = -c(i,j)$ when $i \neq j$, and $A[i][i] =$ the sum of all the connection weights of cell i. X is the solution vector of all the x-coordinates of the non-fixed cells, and b_x is a vector where $b_x[i] =$ the sum of x-coordinates of all fixed elements connected to i. The same applies for Y and b_y, but for the y-coordinates. The solution of these two linear systems give the final positions of all the cells in the design, without using an initial position.

Example

Figure 2.12 shows a placement P with two fixed points $p_1(100, 175)$ and $p_2(200, 225)$ and three blocks a, b, c that need to be placed with a minimum total wire-length.

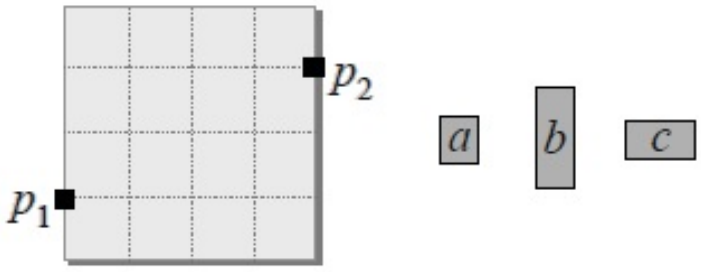


Figure 2.12: Placement P and blocks a, b, c.

Suppose the connections $N_1(p_1, a), N_2(a, b), N_3(b, c)N_4(c, p_2)$ the linear systems that will result in the coordinates of blocks $(x_a, y_a), (x_b, y_b)$ and (x_c, y_c) are formulated with the following procedure.

First, the x-coordinates system is formulated and solved. Based on the given nets and the coordinates of the fixed points, the cost function for x-coordinates is

$$L_x(P) = (100 - x_a)^2 + (x_a - x_b)^2 + (x_b - x_c)^2 + (x_c - 200)^2$$

After applying the partial derivative for each unknown x-coordinate, the functions that are generated are

$$\frac{\partial L_x(P)}{\partial x_a} = -2(100 - x_a) + 2(x_a - x_b) = 4x_a - 2x_b - 200 = 0$$

$$\frac{\partial L_x(P)}{\partial x_b} = -2(x_a - x_b) + 2(x_b - x_c) = -2x_a + 4x_b - 2x_c = 0$$

$$\frac{\partial L_x(P)}{\partial x_c} = -2(x_b - x_c) + 2(x_c - 200) = -2x_b + 4x_c - 400 = 0$$

Finally, the linear system $AX = b_x$ that arises from the above equations is formulated as

$$\begin{bmatrix} 4 & -2 & 0 \\ -2 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} = \begin{bmatrix} 200 \\ 0 \\ 400 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} = \begin{bmatrix} 100 \\ 0 \\ 200 \end{bmatrix}$$

After solving the resulting system for X, the solution for x-coordinate is $x_a = 125$, $x_b = 150$ and $x_c = 175$.

Next, the y-coordinates system is formulated and solved. Based on the given nets and the coordinates of the fixed points, the cost function for y-coordinates is

$$L_y(P) = (175 - y_a)^2 + (y_a - y_b)^2 + (y_b - y_c)^2 + (y_c - 225)^2$$

After applying the partial derivative for each unknown y-coordinate, the functions that are generated are

$$\frac{\partial L_y(P)}{\partial y_a} = -2(175 - y_a) + 2(y_a - y_b) = 4y_a - 2y_b - 350 = 0$$

$$\frac{\partial L_y(P)}{\partial y_b} = -2(y_a - y_b) + 2(y_b - y_c) = -2y_a + 4y_b - 2y_c = 0$$

$$\frac{\partial L_y(P)}{\partial y_c} = -2(y_b - y_c) + 2(y_c - 225) = -2y_b + 4y_c - 450 = 0$$

Finally, the linear system $AY = b_y$ that arises from the above equations is formulated as

$$\begin{bmatrix} 4 & -2 & 0 \\ -2 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{bmatrix} y_a \\ y_b \\ y_c \end{bmatrix} = \begin{bmatrix} 350 \\ 0 \\ 450 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} y_a \\ y_b \\ y_c \end{bmatrix} = \begin{bmatrix} 175 \\ 0 \\ 225 \end{bmatrix}$$

After solving the resulting system for Y, the solution for y-coordinate is $y_a = 187.5$, $x_b = 200$ and $x_c = 212.5$.

The final solution that gives the final positions of cells a, b and c is $a(125, 187.5)$, $b(150, 200)$ and $c(175, 212.5)$ and the result can be seen in Figure 2.13.

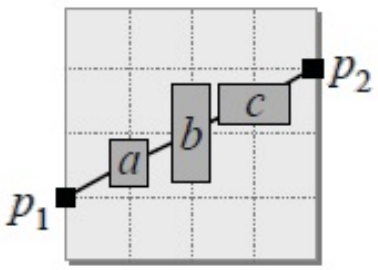


Figure 2.13: Placement P of blocks a, b, c.

2.3 Monolithic 3D Placement

Three dimensional integrated circuits (3D-ICs) have been introduced as a promising solution that makes it possible to extend the limiting 2D scaling prediction that was made by Moore's Law. Current 3D-ICs are through-silicon-via (TSV) based, but the pitch of TSVs is limiting to the integration density. Monolithic 3D IC is a new emerging technology that allows higher orders of magnitude in integration density than TSV based 3DICs, since the monolithic inter-tier vias (MIVs) have an extremely small size. Instead of bonding two pre-fabricated dies, in monolithic 3D

integration technology two or more tiers of devices are fabricated sequentially. This way the need for die alignment is eliminated, enabling smaller via sizes, where each MIV has practically the same size as an intra-tier via (<100nm diameter) [6].

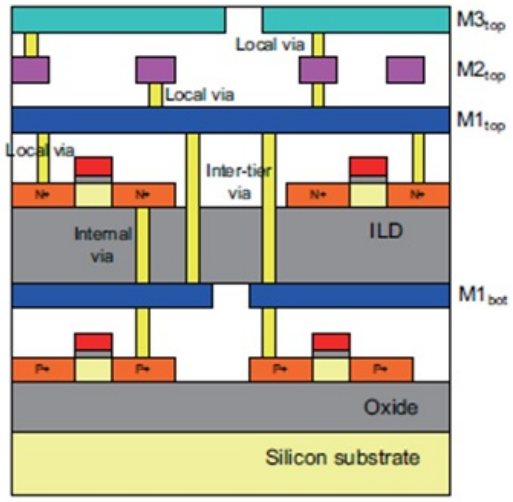


Figure 2.14: Monolithic 3D structure.

2.3.1 Advantages

Monolithic 3D integration allows ultra fine-grained vertical integration of devices and interconnects, because of the extremely small size of inter-tier vias, which are typically 50nm in diameter, and provides more design freedom.

The 3D vertical integration can be categorized into several levels in terms of partitioning granularity. The first one is core level integration, where a typical example is core with memory stack, which provides very high memory access bandwidth. The second one is block-level integration, where functional blocks are partitioned into different tiers based on their logical connections. The third one is gate-level integration, where tiers are partitioned based on each single gate. Since the number of gate is huge in a digital system, the demand for vertical interconnection is very aggressive. The last one is transistor-level integration, which partitions the transistors into different tiers.

Inter-tier vias have a much better electrical performance than TSVs, regarding parasitic capacitance, mechanical stress and electrical coupling, due to its small size.

Since TSVs accommodate much larger parasitic capacitance, which means that for the same timing performance TSV-based designs need more effort on buffering and gate sizing, which will eventually increase the power consumption [7].

2.3.2 Placement Approach

The "Flattened Half-Perimeter wire-length" (HPWL) is defined as the HPWL of a monolithic 3D-IC as if all the gates have been projected onto a single placement layer. Also, the total routing overflow is defined as the sum of routing demand minus routing supply on all global routing bins that are congested [6].

Given an initial monolithic 3D placement, the gates are repartitioned in such way that the total routing overflow is minimized, without changing the flattened HPWL.

It is possible to have a HPWL constraint, since changing only the z location of a cell does not change its flattened HPWL. The extremely small z height in monolithic 3D-ICs guarantees that the impact on the total 3D half-perimeter bounding-box will be minimal.

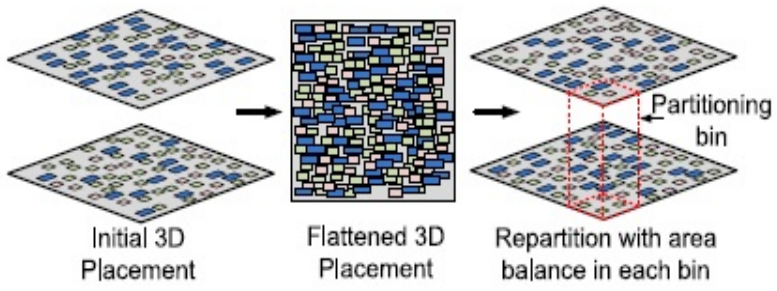


Figure 2.15: The initial monolithic 3D placement is flattened into 2D, and then repartitioned with area balance in each partitioning bin [6].

An overview of this methodology is shown in Figure 2.15. The methodology first starts with a monolithic 3D global placement, which is then projected to 2D, so that a Flattened 3D Placement can be obtained. The partitioning bins are defined, and this flattened placement is repartitioned to 3D, while respect to local area balance within each partitioning bin. The flattening step is used in order to take advantage

of existing 2D placers to generate a flattened monolithic 3D placement, without actually running any 3D placement.

2D placement engines have the concept of chip capacity or utilisation, which is the maximum number of standard cells that can be placed in a certain area. A two-tier monolithic 3D chip has half the footprint area of a 2D chip, and since all the gates need to be fitted into half the area, the area of the chip is halved and the capacity of the chip is doubled.

Once the 3D flattened placement is generated, it is splitted into two tiers, minimising the change to the placement solution. First a min-cut partitioning heuristic is used as an initial solution (FM), a routing demand model for monolithic 3D-ICs, and finally a min-overflow partitioning heuristic that uses the routing demand model to minimize the routing overflow.

Chapter 3

Contribution and Implementation

In this chapter, first we describe the existing infrastructure on which our work was based and then we introduce the contribution and implementation of our work, that is the implementation of the QP Placement, the implementation of the FM partitioning algorithm and finally the implementation of the 3D Placer. We describe how our work is applied on and cooperates with all the existing structures and functionalities.

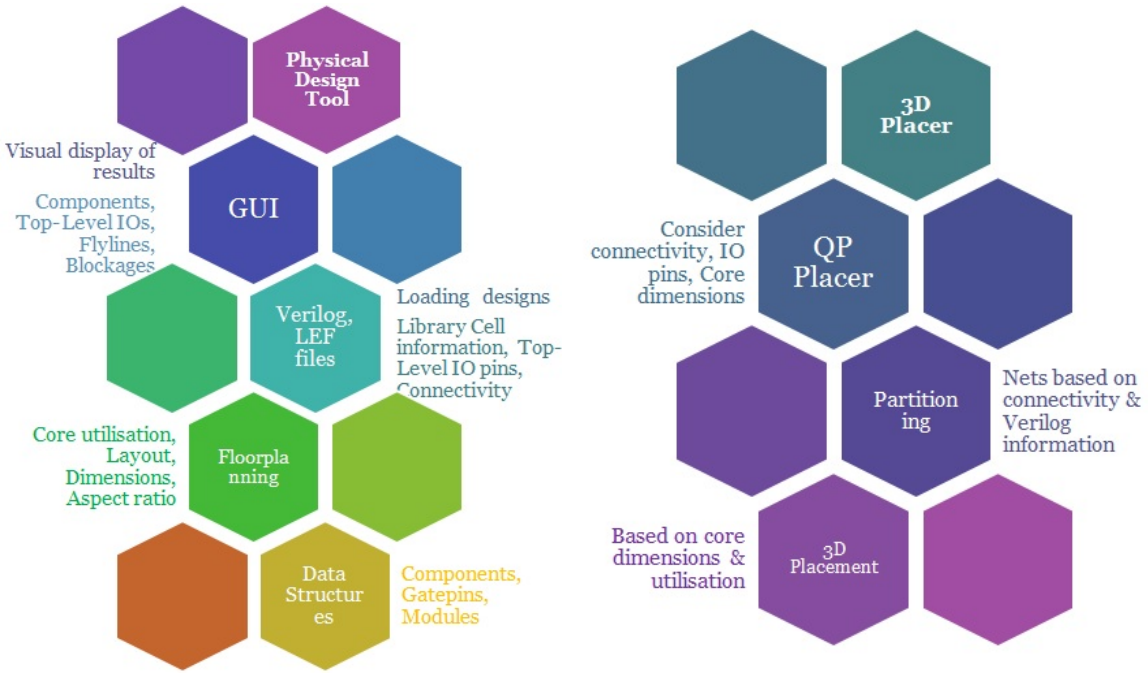


Figure 3.1: *a) Existing Infrastructure and b) Our Work.*

Infrastructure

Figure 3.1 shows the existing infrastructure which is an actual industrial physical design tool and consists of a GUI, a Floorplanning functionality and a Verilog and LEF parser.

The GUI is used to visualise the results from all the functions embedded in the tool and see the actual placement of the cells. The whole core area is displayed along with the floorplanning layout and possible blockages, the cells in their positions and the connections with the Top-level IO pins and between the cells.

The Verilog and LEF parser loads industrial benchmarks from Verilog and LEF files that have all the information we need for each design. We load information regarding cell connectivity and connectivity with Top-level IO pins, and library cell, for example cell dimensions and specifications needed.

The Floorplanning algorithm is responsible for the core layout, that is how the core is divided into bounding boxes based on the modules from each design. This function also provides us with the core dimensions based on the core utilisation that we want for the design and vice versa and core's aspect ratio.

All the information loaded from the design files and the information that comes from the other functionalities is stored in data structures created for the purposes of the physical design tool. There are structures regarding the modules of each design and their hierarchy, the cells, their dimensions, connections, possible positions on the core, and finally structures regarding gatepins, their type - input, output or both - and possible connections.

Combination

Figure 3.1 also shows the different parts and algorithms that were implemented in our work and needed to be applied on the already existing infrastructure and that each function takes advantage of different aspects of the existing information.

The Quadratic Programming (QP) Placer in order to calculate cells' positions in the core area, takes into consideration cell connectivity, possible connections with Top-level IO pins and the core's dimensions that result from the Floorplanning algorithm.

The partitioning algorithm divides cells according to the nets to which each cell is connected. The data structures for the nets are based on cell connections and information from each design's Verilog files.

Finally, 3D placer uses the core dimensions and utilisation that come as a result from the floorplanning algorithm in order to create to different placement levels, each one with half the initial core area.

3.1 QP Placement Algorithm

Contribution

Our contribution regarding QP Placement was to the extent of testing the many different tools and libraries that could solve the QP problem. In order to avoid the cost of creating our own solver just to solve the QP problem, the wise move was to find the most suitable existing solver for the purposes of our work.

The tools that we tested were MATLAB, a well known mathematical tool and CVX, which is an expansion tool to the functions of MATLAB. The libraries tested were Intel Math Kernel Library (IMKL), GNU Scientific Library (GSL) and SuiteSparse.

All the forementioned tools and libraries were able to solve QP successfully. Having in mind additional work that could be done in the future, some of the choices proved not to be suitable. Mathematical functions that could be used for the cell overlap minimisation could not be solved and were not supported by MATLAB and CVX.

From the three libraries left, we chose to work with SuiteSparse due to the many different solver packages available, the ease of use and the very quick execution times when solving the QP problem.

Implementation

In order to solve the QP problem, first we need to create and formulate the system that will give us the final positions of the cells. The system is of the form of $Ax = b$.

In algorithm 1 we present the pseudocode for the formulation of the system, which actually is a double system, one for the x-coordinates and one for the y-coordinates.

In order to formulate the system, we use as input the number of cells, their connectivity to other cells and IO pins and the positions of IO pins. The result is a vector with the positions of all cells.

The diagonal of matrix A, that is element $A[i][i]$ has the number of cells and the number of IO pins connected to cell i. Elements $A[i][j]$ where cell i is connected to cell j have the value -1 in order to show the adjacency between the cells. Elements $b_x[i]$ and $b_y[i]$ have the x- and y-coordinates of IO pins respectively.

Algorithm 1 Matrix and Vector Formulation for QP

Input: The components of our design, the number and position of pins and the number of nets to which each of them is connected

Output: Matrix A and vector b for the system regarding our QP

1: $i = 0$
2: **for** *each component c in design D* **do**
3: $A[i][i]$ = Number of connected components
4: $matrixAposition[c] = i$
5: **for** *each IO pin p connected to c* **do**
6: $A[i][i]++$
7: $bx[i] = pinXposition[p]$
8: $by[i] = pinYposition[p]$
9: $i++$
10: **for** *each component c in design D* **do**
11: **for** *each net n connected to c* **do** (Could be avoided)
12: **for** *each component cc connected to component c (through n)* **do**
13: $ni = matrixAposition[c]$
14: $nj = matrixAposition[cc]$
15: $A[ni][nj] = -1$

Since cells are treated as dimensionless points while solving the QP problem, the results may lead some cells to be placed outside the core area. To fix the positions so that the cells are placed inside the core area, we take into consideration the dimensions of each cell, check whether they are located outside the core bounds and quantise their positions so that they are located in the core area without altering much the initial position calculated by the algorithm.

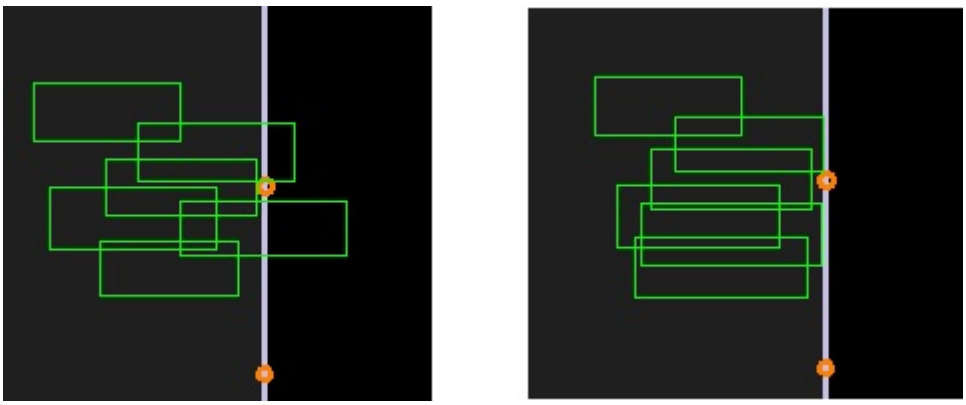


Figure 3.2: a) Cells out of core bounds b) Cells in core bounds after quantisation.

3.2 FM Partitioning Algorithm

Contribution

Our contribution for the FM Partitioning algorithm lies to three different aspects. First, the organisation of cells in *nets*. The logic behind the nets idea was introduced in the FM algorithm. We take the general idea and apply it in the scope of gatepins. This means that from each components output gatepin, we find all the connected input gatepins of other cells and organise the connected cells into a net.

Second, the organisation of the partitions as a *binary tree*. In the beginning, the design is initially cut randomly into two different partitions and with the help of the binary tree structure, each of these partitions can be furthermore cut into two other partitions.

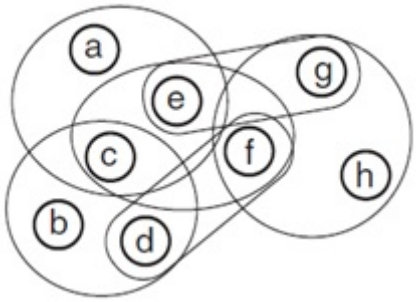


Figure 3.3: Nets from a gate-level circuit.

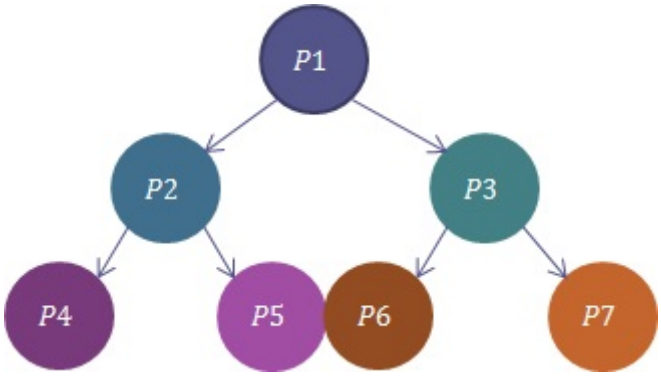


Figure 3.4: Partitions organised as a binary tree.

Finally, the implementation of gain sorting with *binary heap structures*. As the FM algorithm makes use of the bucket structures mentioned in section 2.1, we use binary heap structures to sort each partition's cell gains and easily find the cell to be moved. Compared to linear sorting, which has time complexity $O(n^2)$, the binary heap sorting has average time complexity $O(\log n)$ and pick or extract maximum time complexity $O(1)$.

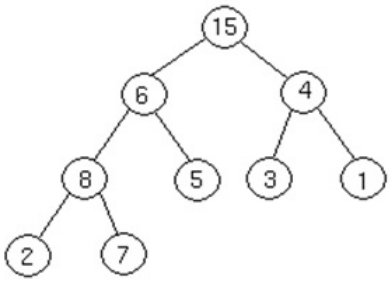


Figure 3.5: Gain sorting binary heap.

Implementation

Given a gate-level design, the cells are divided into two partitions by approximately half. Thus, as the cells are parsed in a certain order, based on this order the first half go to the left partition and the rest to the right partition. Then we calculate the cell gain for these two initial partitions and start moving the cells.

At first, we apply a tentative movement to all cells in order to check whether or not the cutsize will be smaller compared to the starting cutsize. The process of moving each cell is **a)** find the cell with the maximum gain and whose movement does not violate the balance constraint, **b)** move cell to the other partition and declare it locked, **c)** update the gain of connected cells to the one moved, **d)** keep the new cutsize and the move in the history log, **e)** check if there are any cells left unlocked.

After all cells have been moved, we find the minimum cutsize from this pass of the algoithm. If this cutsize is smaller than the initial cutsize, we apply the first k moves that result in this cutsize, else the algorithm terminates. When the final moves have been applied, we attempt another tentative pass of the algorithm.

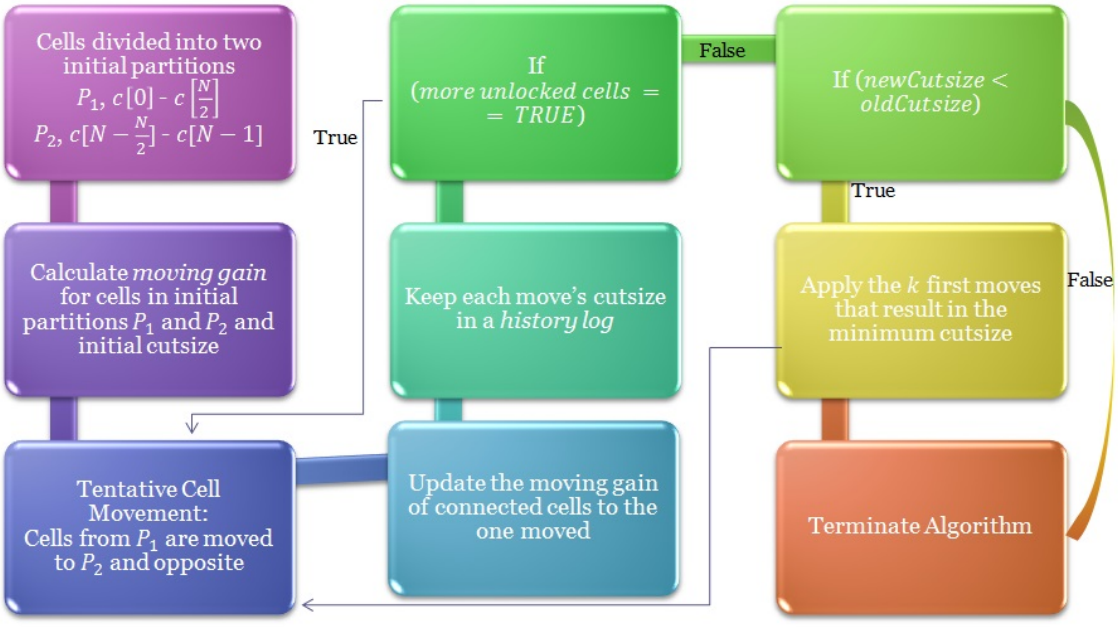


Figure 3.6: Partitioning algorithm flow.

In algorithm 2, we present the pseudocode for the FM Partitioning algorithm

that we implemented as it was described above.

Algorithm 2 Our Implementation of Fiduccia-Mattheyses

Input: The components of our design divided typically in half as two initial random partitions (A,B)

Output: The initial partitions (A,B) with the minimum nets' cut-size between them

```
1: i = 0
2: for each component c in design D do
3:     Gain(c) = FS(c) − TE(c)
4:     cutSize(i) = CalculateCutSize(FS(c))
5: while sizeOf(Heap(A)) > 0 && sizeOf(Heap(B)) > 0 do
6:     cell = MaxGain(Heap(A), Heap(B))
7:     if cell in Heap(A) then
8:         TentativeMove(cell,A,B)
9:         Remove cell from Heap(A)
10:    else
11:        TentativeMove(cell,B,A)
12:        Remove cell from Heap(B)
13:    i = i + 1
14:    for each component c connected to cell do
15:        Re-calculate Gain(c) = FS(c) − TE(c)
16:        cutSize(i) = CalculateCutSize(FS(c))
17: minCutSize = cutSize(0)
18: for i = 1; i <= ComponentsNumber; i = i + 1 do
19:    if cutSize(i) < minCutSize then
20:        minCutSize = cutSize(i)
21:        moveNum = i
22: ApplyMoves(moveNum)
23: return minCutSize
```

3.3 3D Placement Algorithm

Contribution

Our contribution regarding 3D placement is the partitioning of cells to different chip levels, but having achieved the minimum partitioning cutsize and the minimum wire-length. The FM partitioning algorithm gives the minimum possible cutsize for the initial partitioning of the design and the QP placement results in the best positions for the cells based on their connectivity, which is also taken into account during the 3D placement and therefore achieving a minimal wire-length.

Also, with the organisation of partitions in a binary tree there is no limit to the number of possible partitions. To this date, 3D chips had only up to 3 different levels. We take 3D placement to another level by making it possible to have more than just two or three different levels and minimising chip area even more this way.

Implementation

For the 3D placement we want to fit a 2D placement in its half area. This way many cell overlaps occur, which is not an issue since we aim to a 3D chip. Also, the fact that connected cells may be found on different levels may raise concerns about the wire-length. With respect to cells connectivity and placement and with a tiny vertical distance, wire-length will actually be reduced.

We assume a 70% utilisation 2D placement, which we want to fit in the half initial area. This means that with half the initial area, the initial utilisation is doubled and we get a 140% utilisation. Now almost all cell overlap with each other, but this is not an issue since we will split them into different levels based on the partitioning.

As the vertical distance is tiny, approximately $50nm$, the wire-length is minimised compared to the 2D wire-length, where the on-tier distances are much larger. Since we take into consideration cell connectivity, even in different levels they are placed closely and we only consider an average 2D wire-length based on the Half Perimeter Bounding Box (HPBB) that encloses each net and the negligible vertical distance.

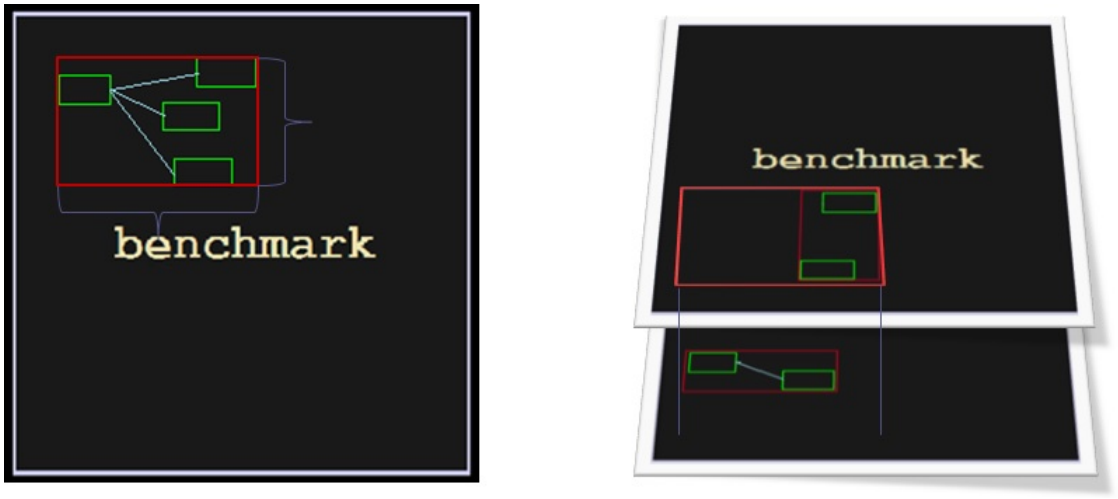


Figure 3.7: a) 2D wire-length and b) 3D wire-length calculation.

Chapter 4

Experimental Results

Partitioning Experimental Results

Table 4.1 shows the experimental results regarding 13 different industrial benchmarks, each one of which has a different number of cells. For each benchmark we present the initial cutsize for the initial partitioning, the resulting minimum cutsize after applying the FM partitioning algorithm, the number of algorithm passes that lead to the minimum solution and the execution time of the algorithm.

Benchmark	# Components	# Nets	Initial Cut	Minimum Cut	# Partitioning Pass	Execution Time (sec)
benchmark1	382	383	86	71	5	0,026517
benchmark2	546	547	21	16	2	0,048908
benchmark3	718	719	90	45	4	0,169194
benchmark4	2346	2347	199	39	4	0,378867
benchmark5	18796	18797	2674	1354	20	41,193311
benchmark6	30675	30676	3735	2179	37	77,739038
benchmark7	32281	32282	3900	984	36	60,153516
benchmark8	41601	41602	3840	1660	27	79,029320
benchmark9	130661	130662	17155	6392	59	440,402364
benchmark10	155325	155326	17803	7868	37	373,614336
benchmark11	164891	164892	17560	8536	55	883,252607
benchmark12	219268	219269	29525	15885	83	26089,862415
benchmark13	794286	794287	94853	54081	94	40251,403348

Table 4.1: Experimental results for the partitioning algorithm.

As it can be seen in Table 4.1, applying the algorithm makes a great difference on each benchmark's cutsize and in many cases in very little execution time.

In Figure 4.1 we see the graphical representation of the results in Table 4.1, which gives us a better idea of how much the cutsize has been optimised for each benchmark after applying the FM partitioning algorithm.

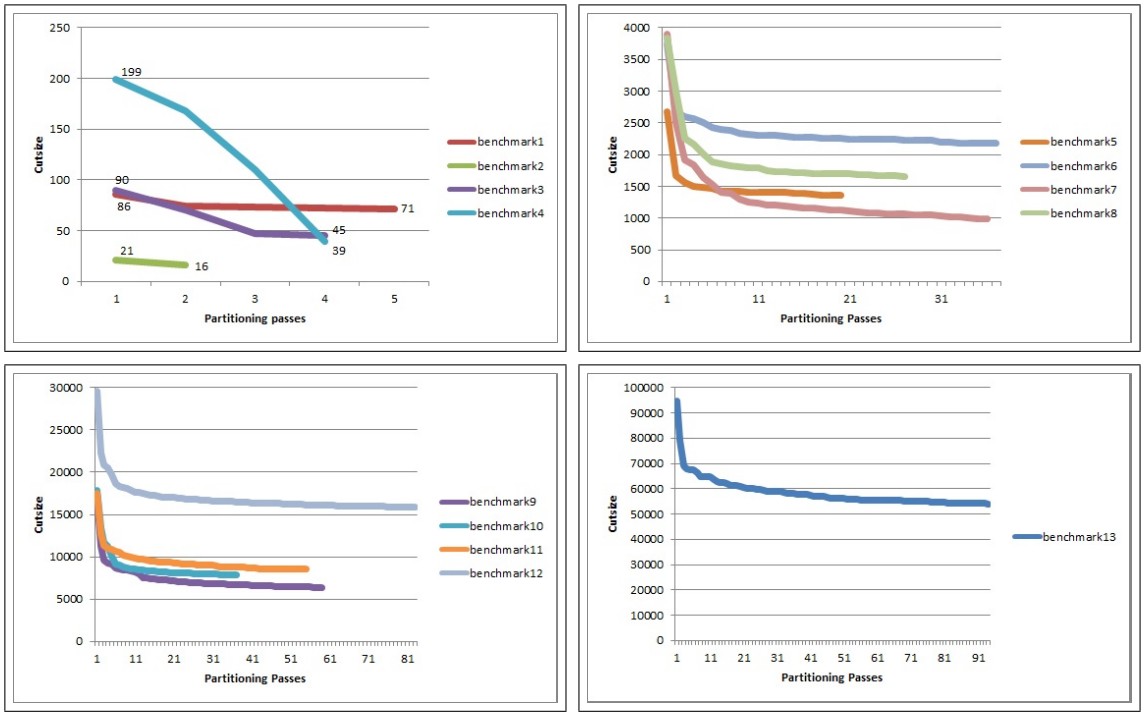


Figure 4.1: *This figure shows how the cutsize is reduced as the partitioning algorithm is applied for more passes.*

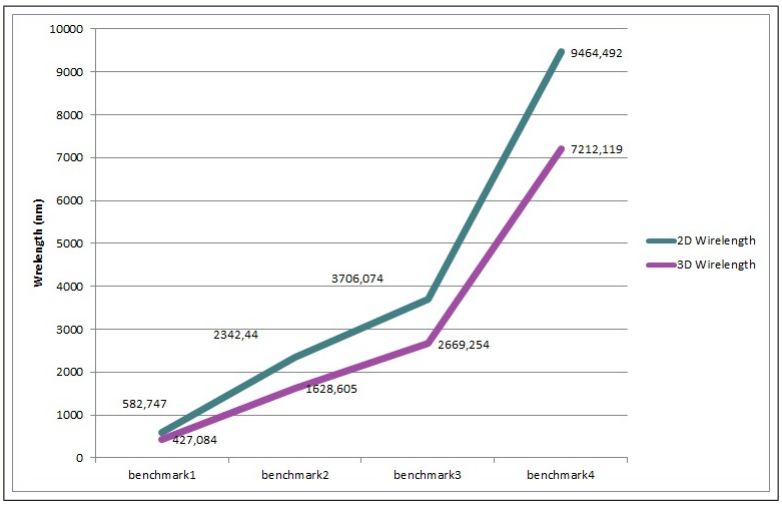
3D Placement Experimental Results

Table 4.2 shows the experimental results regarding 12 different industrial benchmarks, each one of which has a different number of cells, covering a wide range from a few hundred cells to hundreds of thousand cells. In Table 4.2, we present the wire-length and area for 2D placement for each benchmark, the wire-length and area for 3D placement and percentage decrease of wire-length after applying 3D placement. The average wire-length decrease is of the order of 25%, while for some benchmarks a decrease of approximately 30% can be observed.

Benchmark	# Components	2D Wirelength	2D Area	3D Wirelength	3D Area	% WL Decrease
benchmark1	382	582,747	131,674	427,084	65,318	26,712
benchmark2	546	2342,44	387,763	1628,605	192,845	30,474
benchmark3	718	3706,074	640,950	2669,254	319,801	27,976
benchmark4	2346	9464,492	1659,969	7212,119	829.44	23,798
benchmark5	18796	66077,126	8468,582	48026,681	4235,276	27,317
benchmark6	30675	125715,78	72792,000	91695,314	36252,000	27,061
benchmark7	32281	1142215,655	83232,000	806855,267	41616,000	29,361
benchmark8	41601	45408,484	92659,200	33728,263	46483,200	25,723
benchmark9	112644	247690,214	255833,600	178742,016	128020,800	27,836
benchmark10	130661	1558700,527	298116,000	1094661,287	148996,000	29,771
benchmark11	155325	4352560,732	346214,400	3037575,280	173056,000	30,212
benchmark12	219268	411724,839	1086493,400	317855,871	543979,800	22,799

Table 4.2: Experimental results for the partitioning algorithm.

In Figure 4.2 we see the graphical representation of the results in Table 4.2, which gives us a better idea of how much the wire-length has been optimised for each benchmark after applying 3D placement and dividing the cells into to levels.



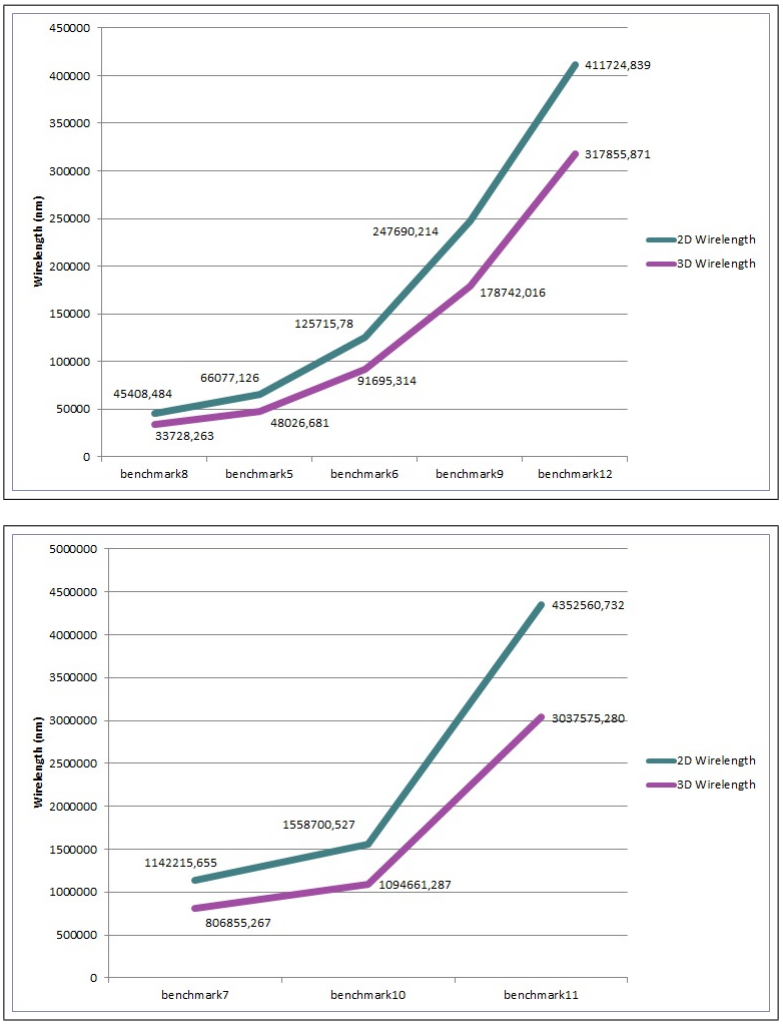


Figure 4.2: *This figure shows the decrease in wire-length after we have applied 3D placement as opposed to the 2D placement.*

Chapter 5

Conclusions and Future Work

3D standard-cell placement is a new emerging and promising technology in Semiconductor Industry and is presented as a solution and an extension to Moore's prediction. The huge number of transistors that are able to fit in a chip area nowadays, makes it more and more difficult for IC designers to create effective and low power consuming chips. Monolithic 3D IC is an emerging technology that enables high integration density on multiple tiers.

Three dimensional placement delivers more design prospects to the Semiconductor Industry as chips may be expanded to more than three levels and 3D placement may be applied in many different ways and methods.

Finally, it has been proven that it is possible to achieve a 25% decrease in wirelength through 3D placement. This leads to less power consumption for chips, less circuit delay and finally lower temperatures, as there is no excessive congestion on the chip with the cells distributed in different levels.

However, there are some additions to our work that we would like to address in the future.

First, we would like to apply 3D placement for more levels. The results presented in our work refer to two-level 3D placement and we would like to take experimental

results for four or more levels. This expansion to our work is quite easy thanks to our binary tree structure for the different partitions.

We would also like to apply the partitioning algorithm with a different cost function. Most partitioning algorithms aim to minimise the cutsize between the partitions. We would like to run the same benchmarks and compare the results for a different cost function, i.e. try to minimise routing congestion.

Finally, another addition to our work would be the elimination of cell overlaps. Even though we have different chip levels, it is still possible that overlaps may occur for large designs. The goal would be to minimise cell overlap without affecting wire-length and to spread cells based on chip's regional density.

Bibliography

[1] B. W. Kernighan and S. Lin, ***"An Efficient Heuristic Procedure for Partitioning Graphs",*** *Bell Sys. Tech. J.* 49(2) (1970), pp. 291-307.

[2] C. M. Fiduccia and R. M. Mattheyses, ***"A Linear-Time Heuristic for Improving Network Partitions",*** *Proc. Design Autom. Conf.*, 1982, pp. 175-181

[3] Sung Kyu Lim, ***"Practical Problems in VLSI Physical Design Automation",*** *Springer Science + Business Media B.V.*, 2008

[4] E. Ihler, D. Wagner and F. Wager, ***"Modeling hypergraph by graphs with the same min-cut properties",*** *Info. Proc. Letter*, 1993, 45:171–175

[5] A. B. Kahng, J. Lienig, I. L. Markov and J. Hu, ***"VLSI Physical Design: From Graph Partitioning to Timing Closure",*** *Springer Science + Business Media B.V.* , 2011

[6] S. Panth, K. Samadi, Y. Du and S. K. Lim, ***"Placement-Driven Partitioning for Congestion Mitigation in Monolithic 3D IC Designs",*** *ISPD'14*, 2014, pp. 47-54

[7] C. Liu and S. K. Lim, ***"A Design Tradeoff Study with Monolithic 3D Integration",*** *ISQED* , 2012