

Timing Driven Floorplanning

by

Khalid Jawdat Kamel Al-Farra

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

June, 1995

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600



Timing Driven Floorplanning

BY

Khalid Jawdat Kamel Al-Farra

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

Computer Engineering

June 1995

UMI Number: 1375580

UMI Microform 1375580
Copyright 1995, by UMI Company. All rights reserved.

**This microform edition is protected against unauthorized
copying under Title 17, United States Code.**

UMI

**300 North Zeeb Road
Ann Arbor, MI 48103**

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN, SAUDI ARABIA
COLLEGE OF GRADUATE STUDIES

This thesis, written by

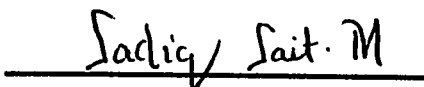
Khalid Jawdat Al-farra

under the direction of his Thesis Advisor, and approved by his Thesis committee, has been presented to and accepted by the Dean, College of Graduate Studies, in partial fulfillment of the requirements for the degree of

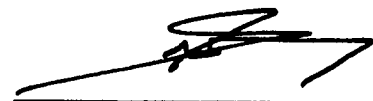
MASTER OF SCIENCE IN COMPUTER ENGINEERING

Thesis Committee :



Dr. Habib Youssef (Chairman)


Dr. Sadiq Sait (Co-Chairman)


Dr. Muhammed S. T. Benten (Member)

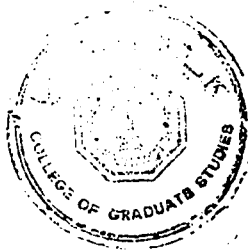


Dr. Samir H. Abdul-Jauwad
Department Chairman


Dr. Ala H. Rabeh
Dean, College of Graduate Studies

Date:

24/6/95



Timing Driven Floorplanning

MS Thesis

Khalid Jawdat Kamel Al-Farra

June, 1995

Dedicated to

My Parents,

whose prayers, guidance and inspiration

led to this accomplishment

إهداء

الى من تعهدا هذا الغرس

حتى أثمر.....

إلى والديّ العزيزين

Acknowledgment

First and foremost, all praise to the Almighty Allah Who gave me the courage and patience to carry out this work. I am happy to have had a chance to glorify His name in the sincerest way through this small accomplishment and ask Him to accept my efforts. May He guide us and the whole humanity to the right path. Peace and blessings of Allah be upon his prophet Muhammad.

Acknowledgement is due to King Fahd University of Petroleum and Minerals for providing support to this work.

My deep appreciation goes to my thesis committee chairman, Dr. Habib Youssef for his constant help, guidance and the countless hours of attention he devoted throughout the course of this work. He was always kind, understanding and sympathetic towards me. Working with him was indeed a wonderful and learning experience which I thoroughly enjoyed.

I am thankful to my thesis committee co-chairman, Dr. Sadiq M. Sait for his deep interest, and constructive criticism during the course of this work. I would also like to thank Dr. Muhammed S. T. Benten, Dean of CCSE and my thesis committee

member for his consistent support.

I am also indebted to the department chairman, Dr. Samir Abdul-Jauwad and other faculty members for their support.

I am thankful to my fellow graduate students and my friends especially Mohammed Shahid Tanvir, Khalid Nassar, and Amir Hashmi for their cooperation.

Lastly, I am very grateful to my family members for their encouragement and moral support.

Contents

Acknowledgement	i
List of Tables	vii
List of Figures	ix
Abstract (English)	xii
Abstract (Arabic)	xiii
1 Introduction	1
1.1 Overview of the System	7

1.2	Organization of the Thesis	12
1.3	Conclusion	14
2	Literature Review	15
2.1	Floorplanning Approaches	17
2.2	Work on Floorplanning	20
2.3	Work on Timing Driven Layout	24
2.4	Conclusion	27
3	Generation of Timing Constraints	28
3.1	Introduction	28
3.2	Timing Analysis Concepts	30
3.2.1	Long Path Problem	31
3.2.2	Short Path Problem	32
3.3	Delay Model	33

3.4	Graph Model	38
3.5	Critical Path(s) Prediction	40
3.5.1	The α -Critical Approach	41
3.6	Calculation of Timing Bounds on Nets	51
3.6.1	<i>Minimax</i> Approach	52
3.6.2	Minimax-PERT	55
3.7	Experimental Results	61
3.8	Conclusion	63
4	Timing Driven Floorplanning	65
4.1	Introduction	65
4.2	Preliminaries	68
4.3	Problem Definition	68
4.4	A Floorplanning Heuristic	71

4.4.1 Force Directed Topological Arrangement	71
4.4.2 Floorplan Sizing	84
4.5 Timing Verification	99
4.6 Discussion	101
4.7 Experimental Results	103
4.8 Conclusion	110
5 Conclusion	111
Bibliography	114
Vitae	121

List of Tables

3.1	Technology data for METAL-1 and METAL-2.	36
3.2	Net capacitance statistics.	43
3.3	Results for Example 3.2.	50
3.4	Delay bounds computation for Figure 3.6.	60
3.5	Test cases statistics.	61
3.6	Number of predicted paths as a function of α	62
3.7	Number of predicted paths for the <i>16-bit multiplier</i>	62
3.8	α^{max} values and delay and variance of longest path.	63
3.9	<i>Minimax PERT</i> test results.	63

4.1	Results of running <i>Cluster Growth</i> on Example 4.2.	83
4.2	Dimensions for blocks in Example 4.3.	91
4.3	Results for Example 4.3.	93
4.4	Data from the <i>Traffic Controller</i>	105
4.5	Data from the <i>Fractional Multiplier</i>	107
4.6	Test cases statistics.	107
4.7	Data from the <i>adder circuit</i>	108
4.8	Data from the <i>16-bit parity checker</i>	109
4.9	Data from the <i>8-bit parity checker</i>	109

List of Figures

1.1	Classic design process.	6
1.2	Timing driven design process.	8
1.3	Overview of the <i>TAP-TDFP</i> system.	9
1.4	An AHPL netlist file.	10
1.5	A VPNR input file.	11
3.1	A circuit example to illustrate the timing model.	36
3.2	Graph representation of a VLSI circuit.	39
3.3	Critical Path Depth-First Trace with Pruning.	47
3.4	A circuit example to illustrate DFTP.	49

3.5	Minimax-PERT Algorithm.	57
3.6	A circuit example to illustrate Minimax-PERT.	59
4.1	(a) slicing floorplan; (b) nonslicing floorplan.	69
4.2	<i>Place_Block</i> procedure.	74
4.3	Cluster_Growth algorithm description.	75
4.4	A circuit example to illustrate the target location computation. . . .	78
4.5	Target location for b_3 : (a) at (x_3^t, y_3^t) ; (b) at (x_3^c, y_3^c) ; (c) at (x_3, y_3) . . .	79
4.6	Circuit for Example 4.2.	83
4.7	Force-directed floorplan for Example 4.2.	84
4.8	(a) Illegal floorplan; (b) constraint graphs: G_H, G_V	87
4.9	<i>Sufficient_Constraint</i> algorithm.	90
4.10	(a) Overlapping floorplan; (b) edge (1,2) retained; (c) edges (1,4) and (1,3) retained; (d) sufficiently constrained set: G_H, G_V	91
4.11	A possible floorplan for Example 4.3.	93

4.12 Approximate Steiner tree. (a) $L_0 = y_2 - y_1$; (b) $L_0 = x_2 - x_1$	101
4.13 Growth in execution time.	102
4.14 A 7-block example: (a) topological assignment; (b) after sizing.	104
4.15 A floorplan of the <i>Traffic Controller</i>	106

Abstract

Name: Khalid Jawdat Kamel Al-Farra

Title: Timing Driven Floorplanning

Major Field: Computer Engineering

Date of Degree: June 1995

Increase in chip density and decrease of feature size have made the performance of modern VLSI circuits dependent on signal propagation along signal nets rather than the switching delays of the cells. Consequently, physical design should be made timing driven. A timing driven physical design tool requires necessary timing data about the design and a strategy to use this data during physical design. In this work, we address both of these issues. We describe efficient algorithms for the prediction of interconnect delay requirement prior to layout. Next, we present a timing driven floorplanning methodology. Our floorplanning approach has two major stages. The first stage constructs a timing driven topological arrangement using a force directed technique. The second stage is floorplan sizing which converts the topological arrangement to a legal floorplan. The execution time of the floorplanning algorithm grows linearly with the problem size.

Master of Science Degree
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

خلاصة الرسالة

الإسم : خالد جودت الفيرا
 عنوان الرسالة : خوارزمية تخطيط زمنية
 التخصص : هندسة الحاسب الآلي
 تاريخ الرسالة : يونيو ١٩٩٥

لقد أصبحت سرعة انتقال الإشارات الرقمية عبر الموصلات عاملاً أساسياً في تحديد سرعة المعالجة للدوائر عالية التكامل. وحيث أن طول الموصلات يتم تحديده أثناء مرحلة التصميم الفعلي.. فإن هذه المرحلة يجب أن تكون محفزة بسرعة المعالجة المطلوبة. و لكي تتمكن أداة التصميم الفعلي من تحقيق هذا الهدف يجب أن يوفر لها معلومات دقيقة عن زمنية التصميم و خوارزمية لكيفية استخدام هذه المعلومات. و تسمى ادوات التصميم المحفزة بسرعة المعالجة بالخوارزميات الزمنية. في هذه الرسالة قمنا بعمل خوارزمية للحصول على المعلومات الزمنية اللازمة. و قمنا أيضاً بعمل خوارزمية زمنية لتخطيط الدوائر عالية التكامل. تتكون خوارزمية التخطيط من مرحلتين. في المرحلة الأولى يتم ترتيب الخلايا المنطقية باستخدام مبدأ توازن القوى الفيزيائية. و يجري تحويل ترتيب الخلايا إلى مخطط صحيح هندسياً في المرحلة الثانية. و قد حصلنا على نتائج إيجابية جداً أثناء استخدام الخوارزمية الجديدة في عمل بعض التصاميم .. ووجدنا أن سرعة تنفيذ الخوارزمية تتناسب خطياً مع حجم التصميم.

درجة الماجستير في العلوم
 جامعة الملك فهد للبترول و المعادن
 الظهران - المملكة العربية السعودية
 يونيو ١٩٩٥

Chapter 1

Introduction

Very large scale integration (VLSI) has made it possible for the semiconductor industry to fabricate an integrated circuit consisting of thousands of components on a single silicon wafer. The VLSI design process spans a wide spectrum of branches in physics, chemistry, electronics and computer science. The complexity of the design process requires breaking it into a number of design steps. The hierarchical decomposition enables the designer to work at any time on a task of manageable complexity.

In most general terms, design can be viewed as a process of successive mappings or transformations of specifications from one abstraction level (or domain) into another. The VLSI design space can be divided into three areas: behavioral (func-

tional), structural, and physical . The behavioral level defines the system outputs in terms of system inputs. The structural level represents the system as a network of logic modules that implements the behavioral level. At the lowest level lies the physical representation. The physical representation gives the required details that can be used by layout synthesis tools to fabricate the desired system.

The complexity of the design process and the variety of applications have lead to different design styles. These design styles can be classified into two general classes [SY94]:

- Full-custom layout.
- Semi-custom layout.

In the *full-custom* approach, the circuit is designed manually by an expert artwork designer. The circuit elements can be placed anywhere on the layout surface. The design cost for this approach is very high because of very long design time (usually years). Thus, only high-volume production can make the manufacturing process profitable. Because of the flexibility in logic and layout design, circuits (e.g., microprocessors) produced by this style can be highly optimized for area and performance.

The *semi-custom* approach constrains the layout elements to some structure in order to reduce the complexity of the design process and consequently the design

time. This reduces the design cost, and makes low-volume production profitable. The popular examples of this design style are gate array, standard cell, and general cell layout styles.

Gate array provides a large two dimensional array of prefabricated transistors on a silicon wafer. Gates, or basic cells, are designed and realized by the manufacturer. The basic cells and their interconnection patterns are kept in a library. All basic cells have equal dimensions. The interconnection of the basic cells is customized to the desired circuit. Routing space in gate array is fixed and limited. The routing regions are called *channels*. Sea-of-gates is a special case of gate array where over the cell routing is allowed and no routing channels are provided.

The *standard cell* layout offers more flexibility. A library of standard cells gives the cell name along with information about its geometry, structure, and delay characteristics. All the cells are of the same height, but with varying widths. The cells are laid out side by side into rows. Routing space is not fixed as in gate array.

The last approach is *general cell* layout. This style provides maximum flexibility in both logic and layout design. This approach is often referred to as *macro-cell*, or *building block* layout style. Macros are permitted to vary in height and width. Consequently, macros with different degrees of sophistication can be implemented. Examples of macros include registers, RAM, ROM, ALU, etc. This approach is

similar to the full-custom approach.

The objective of the physical design steps is to map the structural representation of the design into a physical representation that can be used to produce the desired circuit. The major physical design steps are floorplanning, placement and routing. *Floorplanning* is a preparatory step to placement¹. Usually at this step, it is assumed that the areas and connectivity information of modules are known, while topology of chip and exact dimensions of modules are yet to be defined. Such flexibility represents the designer's freedom in selecting among several possible floorplan configurations. *Placement* is concerned with finding geometric positions for the circuit modules on the layout surface. For placement, the module dimensions, shapes, and pin locations are fixed. Placement is followed by *routing* which interconnects the modules according to the netlist supplied by the structural level.

The feasible solution space for the three steps is very large. If the layout surface is unbounded the solution space is infinite. The floorplanning, placement, and routing problems are NP-Hard problems. By imposing constraints and defining objective functions for the search process, we can reduce the solution space and get superior solutions. However, this transforms the problem to an optimization problem which is much harder. As a result of this complexity, heuristic techniques are used to find sub-optimal solutions that satisfy the stated design objectives and constraints.

¹Floorplanning is sometimes referred to as global or loose placement.

Common objective functions in the layout design problem are chip area, total wire length, routability, timing, or a combination of these.

Once considered to be electrically negligible, interconnects (nets) are becoming a major concern in modern VLSI design because the capacitance and resistance of wires increase rapidly as chip density grows larger and minimum feature size is reduced. In recent years, performance driven layout has become a major issue in VLSI design [DUN84, BUR85, JAC89, YOU90, NAI89, MSL89, BRA90]. For this purpose, designers use various tools, generally called *timing analyzers*, to help them control and improve the temporal properties of circuits before committing them to hardware. The timing analyzer identifies a set of circuit paths called critical paths and possibly a set of critical nets. The physical design step then tries to place the modules belonging to the critical paths and nets in a topological proximity.

In conventional IC design methodology, layout is generated without any knowledge about performance (timing) requirements of the design. Timing verification is invoked after each step. If timing errors are detected, the higher design step(s) is (are) repeated (see Figure 1.1). The timing problems are caused by long interconnects generated by the layout tools. Such philosophy usually results in very unpredictable and expensive iterations of the design process. The lack of timing information on the nets and paths of the design prior to layout, the quality of these information, or the lack of integration between the layout and the timing analysis

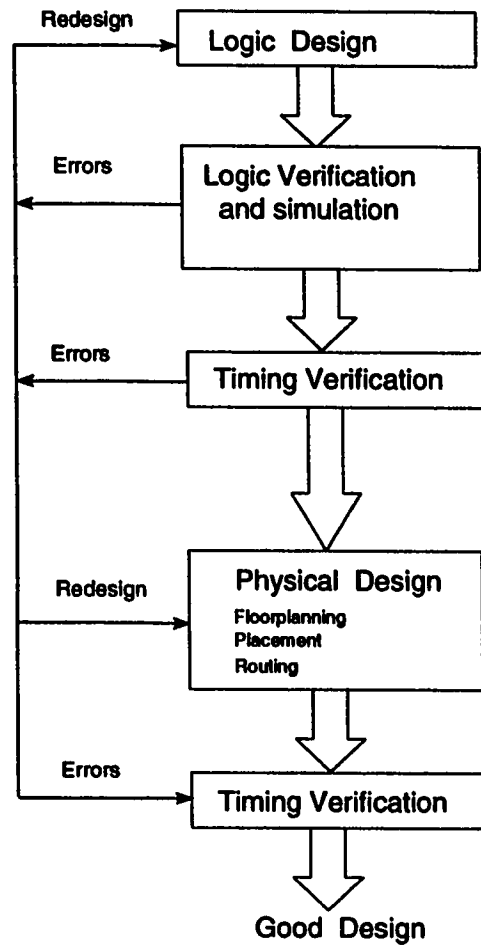


Figure 1.1: Classic design process.

steps are major reasons of the iterative nature of this methodology [YOU90].

Due to recent advances in VLSI technology, the interconnect delays have become a major factor in the overall speed performance of the circuit. The need for timing driven layouts is increasing. It is thus necessary to consider the performance issue in the layout process. In other words, the layout steps such as floorplanning, placement

etc., have to be made sensitive to the timing requirements.

In this work we adopt a design methodology that considers these issues. In this methodology, the timing analysis step is integrated with the physical design step. The timing analysis step is included after the logic design step where design has already been mapped into a specific cell library. The timing step performs two functions: (i) it computes delay constraints on all interconnects that are consistent with the required clock rate of the design, and (ii) it identifies the most critical paths in the circuit (Figure 1.2). For the timing step we implement algorithms similar to those proposed in [YOU90]. We then use the timing data to implement a timing driven floorplanning algorithm.

1.1 Overview of the System

The timing analysis algorithms are embodied in a program called *Timing Analysis/Prediction (TAP)*. The *Timing Driven Floorplanning* is implemented in a program called *TDFP*. The structure of the *TAP-TDFP* system is shown in Figure 1.3.

Two input formats are recognized by the system: AHPL [MS86] netlist or VPNR [MCN90] netlist. The AHPL netlist consists of two files: the gate list and the I/O

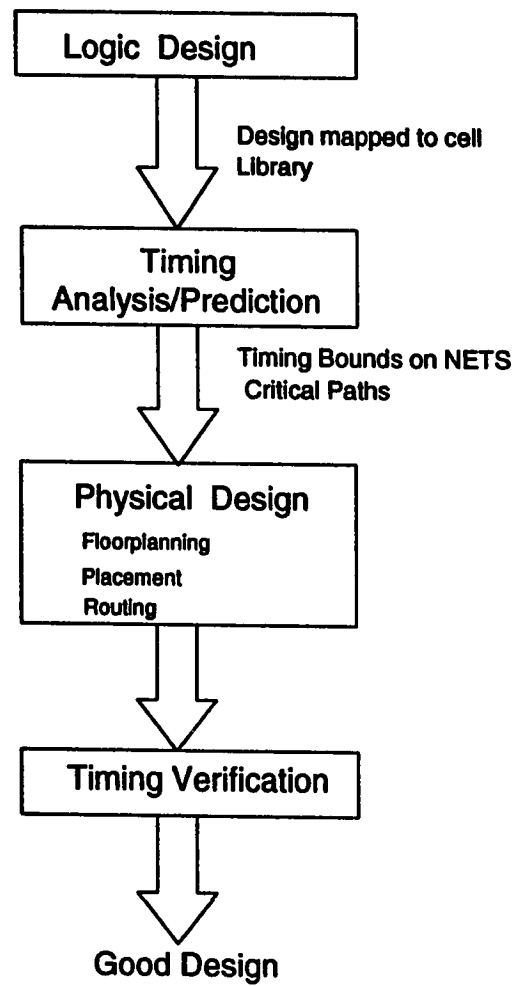


Figure 1.2: Timing driven design process.

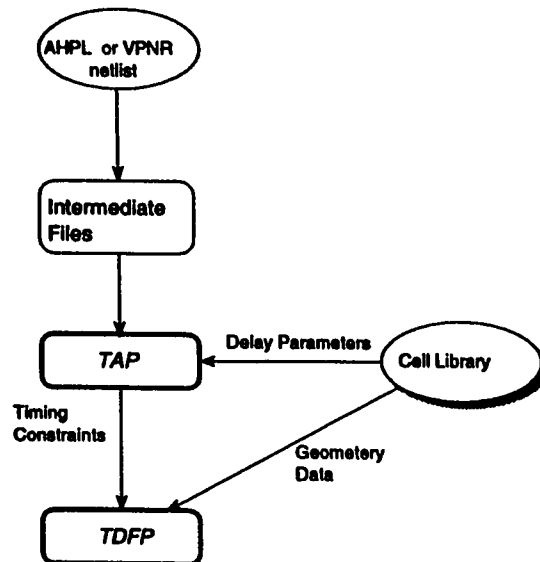


Figure 1.3: Overview of the *TAP-TDFP* system.

list. Each row in the gate list file corresponds to a single gate. The gate list gives the gate number, gate type, input link and the output link. Using either the input link or the output link along with the I/O list, the connectivity of the circuit can be derived. An example of AHPL netlist is given in Figure 1.4.

The VPNR netlist is of two types: placed and unplaced domains. The placed domain netlist specifies a complete standard-cell placement for the circuit. The netlist consists of a list of rows giving the cells in each row. The unplaced netlist consists of a single row. Figure 1.5 is a sample of an unplaced VPNR netlist. The reader is referred to [MCN90] for details of the VPNR format. Starting from a circuit description in AHPL or VPNR format, the description is compiled through a preprocessor into the *TAP* intermediate files. The cell's delay characteristics are

```

GATE LIST
GATE   TYPE   IN-LINK  OUT-LINK
101    4018    0        2
102    4018    0        4
103    4018    0        6
104    4018    0        8
107    4203    13       14
105    4203    9        10
106    4203    11       12
108    4203    15       16
109    4203    17       18
110    4018    19       0
111    4018    21       0
112    4018    0        23
113    4102    24       25
114    4018    26       0
IOLIST
LINK   GATE 1  GATE 2  NEXT LINK
2      105    0       0
4      105    0       0
6      106    0       0
8      106    0       0
10     107    108     0
12     107    109     0
14     108    109     0
16     110    0       0
18     111    0       0
23     113    0       0
25     114    0       0

```

Figure 1.4: An AHPL netlist file.


```
domain begin detect lib=scmos swap=0
profile top (0,0) (0,0);
profile bot (0,0) (0,0);
iolist
Seq_in T:(0,100) pintype=pi
Reset T:(0,100) pintype=reset
Phi1H T:(0,100) pintype=clock
Phi2H T:(0,100) pintype=clock
Phi1_test T:(0,100) pintype=clock
Out_pls B:(0,100) pintype=po
;
row 1
dsr2s INSI14 (I9,I13,Reset,Phi1H,Phi1_test,Phi2H,I11,I8)
dsr2s INSI16 (I10,I15,Reset,Phi1H,Phi1_test,Phi2H,I12,Out_pls)
ai2s INSI21 (Seq_in,I8,I20)
i1s INSI22 (Seq_in,I9)
i1s INSI23 (I20,I10)
;
domain end detect
```

Figure 1.5: A VPNR input file.

extracted from the cell library. The *TAP* program requests the user to specify a *clock period* for the circuit, and a *confidence level* for critical path analysis (Chapter 3). The output from the *TAP* system is a list of timing critical paths sorted on their criticality, and maximal delay bounds for all the nets in the circuit. Additionally, the *TAP* program generates an efficient simple netlist description for the circuit. This description has been used successfully by a *timing driven genetic placer* [NAS94].

The input to *TDFP* is a *TAP*-netlist. The necessary geometry data are extracted from the cell library. The user is requested to specify a desired chip aspect ratio. The output generated by *TDFP* is a list of the blocks, along with their heights, widths, and positions of their lower-left corners on the layout surface.

1.2 Organization of the Thesis

In this work, we are concerned with two issues: prediction of the timing requirements of a VLSI circuit, and the usage of these predictions to drive the floorplanning step.

Our goal is to tie the physical design stage to the timing analysis step. The timing analysis step performs two tasks:

- Prediction of the most timing critical paths.

- Computation of delay constraints on all signal nets.

This information is used to drive the physical design steps, so that layout will be free from timing related problems. We demonstrate the usage of the timing data in the physical design stage by describing a force-directed timing sensitive floorplanning approach for general cell design style.

In Chapter 2, we review several reported works on the *floorplanning* problem. The issues of critical paths prediction and the derivation of delay constraints on signal nets are discussed in Chapter 3. We describe a new strategy for predicting critical paths using statistical estimations. Next, we discuss a minimax algorithm proposed by Youssef [YOU90] for the derivation of maximal delay bounds on net delays, which are consistent with the path's timing constraints.

In Chapter 4, we present our timing driven floorplanning approach. Our approach consists of two major steps. The first step is concerned with constructing a timing driven topological assignment using a force directed technique. The output from this step is an overlapping floorplan satisfying all timing constraints. The second step is floorplan sizing. This step produces a legal floorplan that satisfies all geometric constraints.

Finally, in Chapter 5, we summarize our work and discuss possible future extensions.

1.3 Conclusion

In this chapter, a brief introduction of VLSI design process and motivation behind timing driven design is given. An overview of the implemented system is described. Finally, an outline of the thesis is presented.

Chapter 2

Literature Review

The intractability of the floorplanning problem has led to a large number of heuristic solution techniques. It is difficult to compare the quality of floorplans generated by these techniques. This is due to the inherent differences in the solution methods, which is brought about by the multi-objective nature of the problem. Several objectives are considered in floorplan design. The traditional objectives include minimizing chip area, minimizing total wire length, ensuring routability, or a combination of these. Recently, circuit performance has become a popular objective. The classical approaches to floorplan design solve the problem in two steps. The first step generates a topological arrangement. The next step is sizing, where actual dimensions of modules, routing area estimation and total floorplan area are

computed.

Floorplanning approaches can be classified into two general classes: *constructive* and *iterative*. *Constructive* algorithms adopt a cluster-growth strategy to build a complete solution. Some constructive algorithms require a partial placement as a starting condition. Examples of constructive approaches include dual graph method [LAI88], partitioning/slicing [LAP86], mathematical programming, and force directed methods. *Iterative* techniques on the other hand operate on complete solution and try to improve the quality of the floorplan. For iterative algorithms, the initial solution is generated randomly or by a constructive algorithm. The most widely used iterative improvement technique is Simulated Annealing (SA).

Another possible classification for floorplanning techniques is *deterministic* techniques and *probabilistic* techniques. Constructive algorithms that adopt an equation solving method or based on connectivity information are deterministic because they will always produce the same solution. Iterative techniques are usually probabilistic and generate a different solution for each run.

2.1 Floorplanning Approaches

There are many methods for solving the floorplanning problem. Among them are *force-directed, simulated annealing, dual graph formulation and analytical techniques*.

A common aspect of force-directed algorithms is the method of computing the location where a module should be positioned. This method is analogous to the computation of the *equilibrium location* (the *center of gravity*) problem in physics. The equilibrium location corresponds to a minimum energy state for the module.

Starting with Fisk et. al., 1967 [FCW67], many force-directed techniques are in existence today. There are two possible implementations of this approach: constructive and iterative. The constructive method adopts an equation solving method. The coordinates of each module are treated as variables. The equations are then solved simultaneously to get a minimum-energy state configuration. Iterative force-directed relaxation techniques try to improve the quality of an existing solution. Two strategies are used. The first one tries to move each module to its equilibrium location or to the nearest possible if the target location is occupied. The second strategy randomly selects a pair of modules and interchanges their locations. If the interchange results in a cost reduction (usually wire length) the move is accepted, otherwise it is rejected [HWA76].

The force-directed approach adopts the point model for modeling the modules. In other words, the size and shape of modules are ignored in the computation of equilibrium location of each module. This makes it unsuitable for building block layout due to the irregular dimensions of the blocks. On the other hand, this technique works well for gate array and standard-cell layouts, where points in the plane can be easily mapped to the basic cells in each row of the layout surface. The force-directed approach is best used as an initial solution generator because of its superior execution speed and global view of the overall circuit connectivity. Furthermore, the solutions generated by force-directed techniques are not restricted to slicing floorplans [WL86](see Section 4.2).

The *simulated annealing* (SA) algorithm gets its name from the annealing process in metals. In this process, a clean crystal structure of a certain metal can be restored by heating it to a very high temperature, then cooling it very slowly. The first attempts of applying SA to VLSI-CAD problems were reported in [KGV83, JG83]. Since then, several works have used SA to solve the floorplanning problem [WL86, OG84, WT89].

The basic idea of SA is to perturb the current solution and to accept all perturbations that result in a reduction in cost. Moves that cause the cost to increase are accepted with a probability that decreases with increasing cost and temperature values. This prevents the algorithm from getting trapped at a local minimum.

This process is repeated until a given stopping criterion is met. The quality of the solutions produced by SA is excellent. A major disadvantage of SA is that it is computing intensive.

In the *dual graph* technique a layout structure graph is first transformed into a planar graph by deleting a minimum number of edges and/or adding crossover vertices. Next, an optimal rectangular dual is found for the planar graph. The faces of the dual correspond to modules and the edges of the dual represent module adjacencies. It is not clear how this approach can be modified to take into account the various constraints imposed by practical applications. More details about this technique can be found in [SY94].

Floorplanning is an optimization problem. Thus, analytical techniques can be used to solve the problem. A feasible floorplan is formulated as a set of mathematical equations. Then, mathematical programming techniques are used to solve these equations. This approach is confronted with two main problems. The first problem is the number of equations describing a feasible floorplan is very large resulting in a very large mathematical program. To tackle this problem, a divide and conquer strategy is adopted. The second problem is the nonlinearity of the floorplanning problem leading to a nonlinear program. To overcome this problem, the nonlinear program is linearized using approximation techniques. However, approximation will impact the quality of the solution. A survey of the application of analytical tech-

niques to placement is given in [SM91]. The reader is referred to Chapter 3 of [SY94] for a detailed description of floorplanning.

2.2 Work on Floorplanning

In this section, we review several other reported works on the floorplanning problem.

In [BRA90], a floorplanning algorithm and a global router that uses a sequence of gradient descent operations based on force-directed functions were presented. The best floorplan is selected and simulated annealing is then applied to remove cell overlaps. Circuit timing is considered among other objectives of the floorplan. The timing constraints are specified as critical net/path weights, maximum path length, and maximum net wire length.

In [WIM89], a branch and bound algorithm for selecting optimal aspect ratios of building blocks in non-slicing floorplans was proposed. Each block is assigned to a level of the enumeration tree. Each node in the tree corresponds to a partial floorplan and each "root to leaf" path is a complete floorplan.

In [DON89], an iterative improvement floorplan procedure using constrained graphs was described. The procedure has two steps. In the first step, the dimensions of all the blocks are iteratively computed based on the length of the longest path

passing through the block in the constraint graph. In the second step, the blocks are placed according to the imposed constraints. If the resulting floorplan has overlaps additional constraints are introduced and the process is repeated. Such an iteration is repeated until an acceptable solution is achieved.

In [SUT90], a general floorplan design algorithm based on a linear mixed integer programming model was proposed. The objective is to minimize the overall area of the rectangle enclosing all the basic rectangles. Successive augmentation is used in order to avoid very large mathematical programs. Successive augmentation consists of constructing a complete floorplan by optimally adding a new set of modules to a partial floorplan until all modules are positioned. Each basic rectangle is inflated by an estimate of the routing space of the corresponding module.

In [VIJ91], an approach based on constraint reduction and block reshaping to find floorplans with optimal areas was employed. Two directed acyclic graphs (G_H, G_V) are used to represent the left-right and bottom-top relations between blocks. A complete topological constraint set is derived from a topological arrangement. A constraint set is complete if it contains at least one constraint for each pair of blocks. Then, redundant constraints (i.e., edges) are removed from the critical paths in either G_H or G_V . A constraint is redundant if it is present in both G_H and G_V . Flexible blocks are then reshaped in order to reduce floorplan area. For large designs, the derivation of a complete constraint set and its optimization by removing redundant

constraints may require enormous amount of computation time.

In [YIN89], an analytical technique for floorplanning rectangular blocks is described where the objective is the minimization of the total interconnect length and total floorplan area. The approach consists of two phases. The first phase is a relative placement based on a potential energy model to layout the blocks such that interconnects and size of blocks requirements are satisfied. The second phase is a spacing phase to remove overlaps between blocks.

In [LAI88], a graph theoretic approach to construct rectangular floorplans was presented. The key point in this work was the reduction of the rectangular dualization problem to a matching problem in bipartite graphs. The resulting floorplan was then optimized using a sequence of rectangular dual transformations while preserving the adjacency requirements implied by the original structure graph.

In [DAI87], a constructive hierarchical floorplanning approach combined with global routing was proposed. The approach consists of a clustering step based on the connectivity information. This is followed by a mapping of the clusters onto floorplan templates. Global routing is considered as a part of the floorplanning procedure. Each resulting floorplan is evaluated against an area goal and an I/O pad goal.

LaPotin et. al., [LAP86] proposed a floorplanning approach which combines min-

cut technique and slicing in order to reduce the complexity of the routing problem. Multiple floorplans can be obtained by traversing the slicing tree. Floorplans are evaluated in terms of area and overall wire length using a global router. Module dimensions are derived based on the global routing area.

In [UED85], a semi-automatic VLSI chip floorplanning algorithm was proposed. The algorithm has two stages: initial block placement, and block packing process. The target for the initial placement is to place highly connected blocks in close proximity, whereas weakly connected blocks are placed far from each other. The initial placement is generated by an attractive and repulsive force method (AR method). In the block packing process, overlaps between blocks or blocks and chip boundary are removed by gradually shifting and reshaping blocks with chip boundary shrinking. The block packing process is repeated until an acceptable layout is obtained. A disadvantage of the AR method is that it places weakly connected blocks far from each other. This may not be suitable for timing driven floorplanning because those weakly connected blocks could be part of a timing critical path; if the blocks are far apart, this path will most likely have a timing problem in the final layout.

2.3 Work on Timing Driven Layout

Numerous attempts have been reported on timing driven physical design. These attempts can be classified into three general approaches.

One approach to correct timing errors is to modify some of the logic on the critical paths [MIC86, KIC87].

Another approach relies on transistor re-sizing to reduce delays along the slow paths [JOU87, MIC86].

The third approach avoids any logic modification by imposing delay constraints on the interconnects and paths of the design. Then, the objective for the physical design step is to satisfy these constraints. The possibility of increasing the clock speed of a layout system by 5%-30% without making changes to the logic design makes this approach superior to the other two approaches. Representative works of this approach are reported in [DUN84, HAU87, NAI89, YOU92, SRI92, SUT93]. Our work adopts this approach.

Timing analysis is path oriented as opposed to layout tools which are net oriented. This suggests two approaches for timing driven design: (1) path oriented approach, and (2) net oriented approach. The path oriented approach predicts a set of paths called *critical paths*. This set is then used by the physical design step.

On the other hand, the net oriented approach computes delay constraints for each net based on paths slacks [YOU90, HAU87]. Another net oriented approach assigns a level of criticality to each net [BUR85, DUN84]. A disadvantage of the latter approach is that minimizing the length of the critical nets often causes other nets to become excessively long.

In [DUN84], nets were assigned weights based on a delay analysis performed first with the assumption that net delays are function of their fanouts. The paths with maximum delays are considered as critical and nets covered by these paths are the critical nets. Critical nets are given higher weights which are used to bias the placement and routing steps. After a first layout, actual routing data are passed back to the timing analysis step. This process is repeated until the layout has no failing paths for the desired working clock.

In [BUR85], a preliminary timing analysis prior to layout is performed based on the assumption that all interconnects are equal to an average length. A net-slack for each net is defined to be the slack of the worst path traversing this net divided by the number of nets along that path. The computed net slacks are used to categorize nets into three classes: critical, admissible and regular. The value of the net slack determines its degree of criticality.

In [HAU87], delay bounds on all the nets are computed using a procedure called

Zero Slack Algorithm (ZSA). The computed bounds are transformed into length constraints on the nets and supplied to the physical design steps. Penalties (0, 1, 2) are raised when violations of these constraints occur. Higher penalties imply more attention is given to circuit timing.

In [JAC89], linear programming has been invoked at each stage of recursive partitioning to track path constraints dynamically during placement. Constraints on the physical and timing characteristics of the design are considered in formulating the linear program. Timing constraints on the paths are transformed into constraints on cell locations. Net length estimation is performed using the half-perimeter method. This method for net length estimation is very optimistic, which will hide all timing problems that the layout might have. Furthermore, solving a linear program at each stage slows down the system significantly.

In [FRA92], an algorithm, called the *limit-bumping* algorithm, for computing upper limits on interconnect delays is proposed. This algorithm is similar to the *Iterative-Minimax* approach in [YOU90]. These upper limits are used to improve the layout performance. This is achieved by iteratively decrementing the upper limits on connection delays without violating the lower limits. Frankle shows the necessity for having lower limits on connection delays in order to get realistic upper limits; however, the derivation of such lower limits is not clear. The timing data has been used by a timing driven *FPGA* router. An average of 14% improvement

in system clock periods has been reported.

2.4 Conclusion

In this chapter, we reviewed several approaches to floorplanning as reported in the literature. A classification of solution methods was given. Several timing driven design approaches were reviewed.

Chapter 3

Generation of Timing Constraints

3.1 Introduction

With advances in integrated electronics technology, the interconnect delays have become a significant factor in determining the timing characteristics. This situation makes it almost impossible to verify the required clock speed at the logic design step. It is no longer the case that timing verification can be done prior to physical design, and physical design itself should be governed by timing requirements. Timing driven layout has become a popular topic among researchers in the CAD area.

In order to produce layouts optimized for timing, the timing analysis/ prediction

step should guide the layout tool by supplying necessary and accurate information about the timing aspects of the design. This information may be supplied as maximal constraints on interconnect delays, or a list of the most critical paths in the design, or both.

In this chapter, we present a new approach to predict timing critical paths in a given VLSI design, which is a variation of one of the approaches reported in [YOU90]. The predicted critical paths' information has been successfully used by a *timing driven genetic* placer for standard cell design style. The critical paths enumerated after placement were a proper subset of the predicted paths. For details of this layout system, the reader is referred to [NAS94]. Next, the implementation of an algorithm due to [YOU90] for computing delay upper bounds on all the nets will be presented. The net bounds are functions of the nets' electrical characteristics.

The rest of the chapter is organized as follows. In Section 3.2 we present some basic timing analysis concepts. Section 3.3 discusses the delay model adopted in this work. In Sections 3.4 to 3.6, we focus on the development of timing information about the design at hand. Experimental results are presented in Section 3.7. We conclude in Section 3.8.

3.2 Timing Analysis Concepts

Timing simulation¹ and timing analysis are two popular approaches to verify the timing behavior of a digital circuit. By exercising the circuit for a large set of input signals, the simulator checks the functional and timing behavior of the design. Timing analysis, on the other hand, ignores the logic properties of the circuit elements and checks only the timing behavior of the circuit. This fact makes timing analysis very time efficient compared to simulation. Ignoring the functionality of the circuit elements is however, responsible for the main difficulty in timing analysis: the *false path problem*. This problem can be eliminated by using a path sensitization criterion to determine whether a path is sensitizable or not [CD93].

Nowadays, timing analysis is replacing simulation for VLSI circuits. Simulation is used only for testing the functionality of the circuit.

Timing analysis tools can be classified into two classes: transistor/switch level analyzers, and gate/macro level analyzers. Switch level analysis is usually more accurate than gate level analysis; however, it is suitable only after physical design (placement and routing) has been completed. Switch level analyzers are very demanding in terms of computing resources. Gate level analysis can be done before or after physical design. Post-layout gate level analysis includes interconnect delay

¹Simulation is sometimes referred to as dynamic timing analysis.

in the verification process. The accuracy of gate level analysis is contingent on the quality of the gate (also block, or cell) delay model. In this work, we perform timing analysis at the cell level.

In general, timing analysis of VLSI designs is concerned with checking for long and short path problems. A *path* is an alternating sequence of circuit elements and signal nets. The first element of the path is called the *source*, and the last element is the *sink*. Input pads and outputs of storage elements are sources. Output pads and input pins of storage elements are sinks. In this work, storage elements are assumed to be *flipflops*. Both the source and sink of a path are controlled by the same clock.

3.2.1 Long Path Problem

The long-path slack of a path π is defined as

$$SLACK_{\pi} = LRAT_{\pi} - T_{\pi} \quad (3.1)$$

where $LRAT_{\pi}$ and T_{π} are the “latest required arrival time” and the “actual arrival time” of the signal at the path sink, respectively. A design has a long path problem if for some path, the long-path slack is negative. A negative slack indicates that the signal will fail to propagate through the path within the required time. For flipflops, $LRAT_{\pi}$ is given by

$$LRAT_{\pi} = CP - T_{cs} - T_{stp} \quad (3.2)$$

where,

CP is the clock period in nanoseconds,

T_{cs} is the maximum clock skew, and

T_{stp} is the setup time at the path sink.

3.2.2 Short Path Problem

The short-path slack of a path π is defined as

$$SLACK_{\pi} = ERAT_{\pi} - T_{\pi} \quad (3.3)$$

where $ERAT_{\pi}$ is the earliest required arrival time.

A design is said to have a short path problem if $SLACK_{\pi}$ is positive. A positive short slack indicates that the signal is arriving earlier than what is required and hence will cause a premature gating. For flipflops, $ERAT_{\pi}$ is given by,

$$ERAT_{\pi} = T_{cs} + T_{hold} \quad (3.4)$$

where,

T_{hold} is the hold time.

Now the challenge for performance driven layout tools is to produce designs such that the total path delay for any path is greater than its earliest required arrival time and less than its latest required arrival time.

While short path problems can be detected and corrected during logic design, long path problems are most difficult to avoid. The maximum clock period for a given circuit is determined by the total delay of its longest path. In this thesis, we concentrate on the long path problem and ignore the short path problem. Short paths are easier to eliminate and are less likely to occur in VLSI designs.

For the timing analysis/predictions to be of any value, a suitable delay model should be adopted. The delay model must accurately abstract the essential electrical properties of the circuit elements.

3.3 Delay Model

Timing analysis at any level strongly depends on the *delay model* used to abstract the timing behavior of circuit elements. The model should accurately illustrate the temporal properties of the components of the design. Approximations should be avoided as much as possible as this will dramatically affect the quality of the analysis and hence the quality of the final layout.

A suitable delay model should consider the following essential characteristics:

- The switching delay of a cell consists of two components: the base (intrinsic)

delay of the cell and the loading delay.

- Unateness (polarity) information. Some cells invert the incoming data signal (e.g., NAND); others do not (e.g., AND).
- Differences between rising and falling delays can be order of magnitude apart.

In this work, a linear cell delay model which considers these characteristics has been adopted. With this model, the time needed to charge the capacitances at the loading pins of a given net is the same. Using this model, the total switching delay of a given cell v is given by

$$CD_v = BD_v + LF_v \times AcL_v \quad (3.5)$$

where,

BD_v is the base (intrinsic) delay of cell v in *nanoseconds*.

LF_v is the load factor in *Kilo Ω s*. It is the resistance as seen at the cell output pin.

AcL_v is the input capacitance on the loading pins.

For ease of explanation, the unateness of cells as well as differences between rising and falling delays are ignored (but not in the actual calculations of the *Timing Analysis Program (TAP)* system).

Using the *lumped RC model*, the total delay seen by the output pin of cell v is

expressed as follows:

$$TD_v = CD_v + ID_v \quad (3.6)$$

where,

ID_v is the interconnect delay of the net driven by cell v output pin, and is expressed as follows,

$$ID_v = LF_v \times C_v + R_v \times (AcL_v + C_v) \quad (3.7)$$

where,

C_v is the total interconnect capacitance (area + fringe) of the net driven by cell v output pin.

R_v is the total interconnect resistance of the net driven by cell v output pin.

Based on this delay model, the interconnect delay is a function of three parameters: the load factor of the driving cell which is layout independent, the interconnect capacitance, C_v , which is layout dependent, and the interconnect resistance which is also layout dependent. As will be demonstrated with an example, for metallic wires the contribution of the resistance parameter to the interconnect delay is negligible compared to the capacitance parameter. In most timing analysis tools, when nets are included in the verification process, only their capacitive effect is considered while their resistive effect is ignored, or at best lumped with the estimated resistance of the driver.

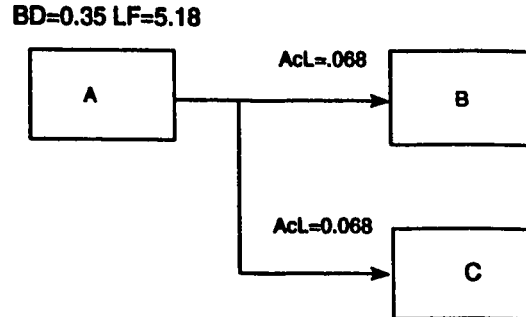


Figure 3.1: A circuit example to illustrate the timing model.

METAL	Sheet_Resistance Ω/\square	Area_Capacitance $10^{-4}pF/\mu^2$	Fringe_Capacitance $10^{-4}pF/\mu$
1	0.06	0.26	0.82
2	0.033	0.15	0.85

Table 3.1: Technology data for METAL-1 and METAL-2.

Example 3.1 Consider the circuit depicted in Figure 3.1 where cell A is driving cells B and C. The switching delay of A is given by $0.35 + 5.18 \times (0.068 + 0.068) = 1.054ns$.

To compute the interconnect delay for the net in Figure 3.1, assume that the vertical connection is in METAL-1 and its height and width are $1000\mu, 3\mu$ respectively. The horizontal connection is implemented in METAL-2 with 1000μ height and 3μ width. Table 3.1 gives the worst case values of sheet resistance and capacitance (area and fringe) for METAL-1 and METAL-2 as provided by Orbit Semiconductor [Orb92]. The interconnect capacitance consists of two components: the area (surface) and the fringe capacitances. Let C_{M1} be the interconnect capacitance due to METAL-1 and C_{M2} be the interconnect capacitance for METAL-2. Then, using the

technology data in Table 3.1 and the dimensions for METAL-1 and METAL-2 given above, C_{M1} is computed as follows:

$$\begin{aligned}
 C_{M1} &= \text{area_capacitance} + \text{fringe_capacitance} \\
 &= 1000 \times 3 \times 0.26 \times 10^{-4} + 2 \times (1000 + 3) \times 0.82 \times 10^{-4} \\
 &= 0.24249 \text{ pF}
 \end{aligned}$$

Similarly, we get $C_{M2} = 0.21551 \text{ pF}$. The total interconnect capacitance is $C_{int} = C_{M1} + C_{M2} = 0.458 \text{ pF}$.

The total resistance is given by

$$\begin{aligned}
 R_{int} &= R_{M1} + R_{M2} \\
 &= 0.06 \times \frac{1000}{3} + .033 \times \frac{1000}{3} \\
 &= 0.033 \text{ K}\Omega
 \end{aligned}$$

ID_A , the total interconnect delay seen by cell A output pin is therefore,

$$\begin{aligned}
 ID_A &= LF_A \times C_{int} + R_{int} \times (AcL_A + C_{int}) \\
 &= 5.18 \times 0.458 + 0.033 \times (0.136 + 0.458) \\
 &= 2.392 \text{ ns}
 \end{aligned}$$

Observe that the contribution of the delay due to resistance R_{int} is only 0.0196 ns. Thus, interconnect delay is dominated by its capacitance. This value of ID_A is more than double the delay seen by the signal when the interconnect delay is ignored

(1.054 ns). These values are for 2μ technology. This ratio of interconnect delay to cell delay is expected to increase as feature size continues to decrease.

In the subsequent discussions, we shall ignore the resistance effect in interconnect delay computations. Thus, Equation 3.7 can be simplified as,

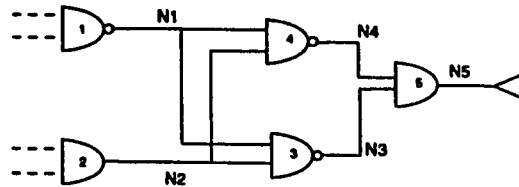
$$ID_v = LF_v \times C_v \quad (3.8)$$

A timing driven physical design tool expects necessary timing information. This information may consist of either or both of the following: (i) a list of the most critical paths, (ii) timing constraints on all the nets. Before we describe the algorithms used to derive such information, we shall first describe a graph model for representing a VLSI circuit.

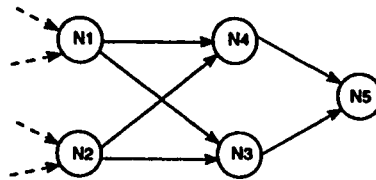
3.4 Graph Model

The netlist information is transformed into an acyclic directed graph $G = (V, E)$, where V is the set of vertices representing the signal nets, and E is the set of edges representing adjacency relations between signal nets. Vertices with zero in-degree are called *source vertices* (e.g., input pads, output pins of flipflops), and vertices with zero out-degree are *sink vertices* (e.g., output pads, input pins of flipflops). All

other vertices are called the *internal vertices*. We shall denote the source vertex set, the sink vertex set, and the internal vertex set by V_s , V_t , and V_n , respectively. We shall use the symbol π to refer to a path in G . A path is a sequence of vertices. A path is a complete path if it starts at a source vertex and terminates at a sink vertex, otherwise it is a partial path. Figure 3.2 illustrates this graph representation. The



(a)



(b)

Figure 3.2: Graph representation of a VLSI circuit.

timing information related to the logic cells (i.e., BD , LF , AcL) are stored in the *net-vertices*. Therefore, we do not need to include the logic cell in the graph model. This will significantly reduce the memory space required by the model.

3.5 Critical Path(s) Prediction

Since the total number of paths in a VLSI design may grow exponentially with the size of the design, it is impractical to enumerate and monitor every path in the circuit. However, usually a small subset of these paths are timing critical. A path π is critical if its total delay, T_π , is very close to its latest required arrival time $LRAT_\pi$. If T_π exceeds $LRAT_\pi$, path π becomes a long path. The task of the timing tool is to predict those critical paths. The accuracy of this prediction has a direct impact on the temporal performance of the final layout.

One approach to path prediction is to report the most critical path in the design. This approach is also referred to as the *Block Oriented Approach*. Representative examples of this approach are reported in [HIT82, KIRK66]. Reporting the most critical path is not sufficient to correct/prevent timing errors for the whole layout. The second approach reports all the paths which are greater than or less than a given *threshold*. A variation of this technique is to report the K most critical paths [YEN89]. In these approaches the criticality of a given path is measured based on the *path slack*. The smaller the slack, the more critical the path is. For very dense designs, the slack metric may not lead to a good prediction.

The third approach is based on the notion of *categorization* [YOU90]. In this approach, a score is computed for each enumerated path. The score is a function

of parameters that are correlated with the total path delay. Examples of these parameters include: load factors, number of nets on the path, slack, etc. The K paths with worst scores are the K most critical. As in the second approach K could be fixed or a function of a given *threshold*.

For path enumeration, all the above mentioned strategies are based on a depth first search (DFS) with/without pruning, breadth first search (BFS), or PERT-like trace [KIRK66]. The time complexity of these algorithms is proportional to the reported number of paths. The path enumeration phase is always preceded by a graph construction phase. Next, we present a new strategy for critical path prediction.

3.5.1 The α -Critical Approach

As stated earlier, for VLSI designs, the interconnect delay is a major part of the overall path delay. It is then of prime importance, for pre-layout timing analysis, to predict the interconnect delay requirements.

Youssef [YOU90] experimented with the *linear regression* approach to predict interconnect delays at the net level as well as at the path level. In *net-level* point prediction, data about delays of individual nets were collected and used to build a regression model. The model is then used to predict the response for new cases.

On the other hand, in *path-level* prediction, data about total path's interconnect delay along with other path related parameters (e.g., number of nets on the path, sum of load factors, etc.,) are used to build the predictor (model). The following lines are extracted from [YOU90]:

regressional approaches to point prediction of the interconnect delay of paths, and especially of nets, for the purpose of predicting before layout all paths with tight timing requirements, do not produce desirable results.

The deficiency of this approach is due to the low prediction power (i.e., large prediction errors) of the model. The situation is not as bad as it seems. Net-level prediction can still provide the desired accuracy. That depends on the design size, design density, and the variability in net length compared to the overall path interconnection length. We wish to have the *variances* on net lengths much smaller than the total path interconnect length. The α -critical approach is a variation of the above mentioned approach. A description of this approach follows.

The total path delay consists of two components: the total switching delays of logic cells which is known prior to layout and the total interconnect delay which is unknown. As demonstrated in Example 3.1, the interconnect capacitance is a key element in the total interconnect delay. The average and standard deviation of net length for different types of nets (2-pin, 3-pin, . . . , k -pin) are collected from past

designs of similar complexity. These are transformed into interconnect capacitances. These statistics are shown in Table 3.2. Nets are classified according to the number of pins they are connecting. For example, a 2-pin net has an average capacitance of $0.0496613614 pF$, and a standard deviation of $0.0381922414 pF$.

Pins	Mean (pF)	Standard Deviation (pF)
2	0.0496613614	0.0381922414
3	0.0642885151	0.0269860004
4	0.1045077916	0.0474687684
5	0.1239954827	0.0564270349
6	0.1325196250	0.0574357545
7	0.1486822222	0.0440815179

Table 3.2: Net capacitance statistics.

Let $\pi = \{v_1, v_2, \dots, v_k\}$ be a path in the circuit graph, where v_1 is the source and v_k is the sink. The total delay on π is given by,

$$T_\pi = \sum_{i=1}^{k-1} (CD_{v_i} + ID_{v_i}) \quad (3.9)$$

where,

CD_{v_i} is the switching delay of net vertex v_i as defined by Equation 3.5, and

ID_{v_i} is the interconnect delay of net v_i as defined by Equation 3.8.

But, C_{v_i} , the total interconnect capacitance of net v_i , in Equation 3.8 is unknown prior to layout because it is a function of the net length. Therefore, we characterize each net vertex v by two elements:

1. an average capacitance \bar{c}_v , and
2. a standard deviation s_v .

The expected delay on net v , \overline{ID}_v , is computed as,

$$\overline{ID}_v = LF_v \times \bar{c}_v \quad (3.10)$$

The delay variance on net v is computed as

$$S_v^2 = LF_v^2 \times s_v^2 \quad (3.11)$$

Assume that the nets are statistically independent. Thus the expected delay on any path π can be expressed as,

$$T_\pi = \sum_{i=1}^{k-1} (BD_{v_i} + LF_{v_i} \times (AcL_{v_i} + \bar{c}_{v_i})) \quad (3.12)$$

The delay variance on path π is defined as the sum of the delay variances of its nets. Using the already introduced notation, the delay variance of path π is expressed as follows,

$$S_\pi^2 = \sum_{i=1}^{k-1} S_{v_i}^2 \quad (3.13)$$

Let T_{max} be the expected delay of the longest path in the circuit. T_{max} can be expressed as follows,

$$T_{max} = \max_{\pi \in \Pi} (T_\pi) \quad (3.14)$$

where Π is the set of all paths in the circuit graph G .

Definition 1 A path π is α -critical iff:

$$T_\pi + \alpha\sqrt{S_\pi^2} \geq T_{max} \quad (3.15)$$

The parameter α (interpreted as a *confidence level*) is a user supplied input. $T_\pi + \alpha\sqrt{S_\pi^2}$ means that we are $\alpha\sqrt{S_\pi^2}$ ns confident that path π is critical. The higher α is, the larger the number of reported paths will be, and the higher is the probability of capturing all the critical paths. Reasonable values of $\alpha\sqrt{S_\pi^2}$ are ≤ 4 ns.

Lemma 1 The confidence level that will cause full path enumeration is given by,

$$\alpha_{max} = \frac{T_{max} - T_{min}}{\sqrt{S_{min}^2}} \quad (3.16)$$

where T_{min} is the delay of the path with minimum T_π and S_{min}^2 is its delay variance.

The algorithm used to extract paths according to Definition 1 is a variation of the algorithm in [AS85]. Figure 3.3 gives a formal description of this algorithm. We call this algorithm *Depth_First_Trace_with_Pruning* (DFTP). An informal description of this algorithm follows.

Let $\pi = \{s, v_1, v_2, \dots, v_k, v\}$ be a partial path starting at source s and terminating at v ; T_π and S_π^2 are the expected delay and variance along that partial path. When

ALGORITHM DEPTH_FIRST_TRACE_WITH_PRUNING**NOTATION**

$\Gamma(v)$: all successors of vertex v
 T_{max} : delay of longest path
 T_π : delay of path π (i.e. $\sum t_v$)
 MDS_v : Maximum delay to sink of vertex v
 MVS_v : Maximum variance to sink of vertex v
 t_v : delay of v including estimated interconnect capacitance
 S_π^2 : Variance of path π
 S_v^2 : Variance of the net driven by cell v

INITIALIZE

FOREACH $v \in (V_s \cup V_n)$
 COMPUTE $S_v^2 = LF_v^2 \times s_v^2$;
 ENDFOREACH

GENERAL STEP DFTP**BEGIN**

1 **step 1: Backward Trace**
 2 FOREACH $v \in (V_n \cup V_s)$ DO
 3 $MDS_v \leftarrow t_v + \max_{u \in \Gamma(v)}(MDS_u)$;
 4 $MVS_v \leftarrow S_v^2 + \max_{u \in \Gamma(v)}(MVS_u)$;
 5 ENDFOREACH
 6 **step 2: Forward Trace**
 7 FOREACH $v \in V_s$
 8 PUSH(v);
 9 ENDFOREACH
 10 **step3: Path Trace**
 11 $\pi \leftarrow \{\}$; $K \leftarrow 0$;
 12 WHILE (stack is not EMPTY) DO
 13 BEGIN
 14 $v \leftarrow \text{TOP}(\text{stack})$;
 15 IF ($v \in V_t$) THEN
 16 BEGIN
 17 SavePath($(\pi!, v)$); $K \leftarrow K + 1$;
 18 POP(stack);
 19 END

```

20     ELSEIF (v is marked) THEN
21         BEGIN
22             POP(stack);
23             REMOVE v from  $\pi$ ;
24              $T_\pi \leftarrow T_\pi - t_v$ ;
24              $S_\pi^2 \leftarrow S_\pi^2 - S_v^2$ ;
25         END
26     ELSEIF ( $T_\pi + MDS_v + \alpha \times \sqrt{(S_\pi^2 + MVS_v)} \geq T_{max}$ ) THEN
27         BEGIN
28             MARK(v);
29             FOREACH  $u \in \Gamma(v)$  DO
30                 PUSH(u);
31             ENDFOREACH
32              $T_\pi \leftarrow T_\pi + t_v$ ;
33              $S_\pi^2 \leftarrow S_\pi^2 + S_v^2$ ;
34              $\pi \leftarrow [!\pi, v]$ ;
35         END
36     ELSE
37         BEGIN
38             POP(stack);
39             UNMARK(v);
40         END
41     ENDIF
42 ENDIF
43 ENDIF
44 ENDIF
45 ENDWHILE
46 step 4: Report Critical Paths
   Let  $\Pi = \{\pi_1, \pi_2, \dots, \pi_K\}$  such that
    $SLACK(\pi_i) \leq SLACK(\pi_{i+1})$ ;
   FOR  $i = 1$  TO  $K$ 
       PRINT path  $\pi_i$ ;
END_DFTP.
```

Figure 3.3: Critical Path Depth-First Trace with Pruning.

v is reached, the criticality test (line “26” in Figure 3.3) is performed using T_π , the maximum v -to-sink delay $MDS(v)$, and S_π^2 , the maximum v -to-sink variance $MVS(v)$. If the test is positive the search continues, otherwise the search terminates and backtracks to the immediate predecessor. Hence, the knowledge of $MDS(v)$ and $MVS(v)$ is necessary to prune away all paths prefixed by the partial path π .

Informally, the algorithm proceeds as follows. The first step (step “1” in Figure 3.3) consists in performing a backward trace from the graph sinks to the sources. This step saves within each vertex: (i) its maximum delay to the sinks, and (ii) its maximum variance to the sinks. The second step (step “2” in Figure 3.3) consists in performing a forward trace from the sources to the graph sinks. The aim of this step is to enumerate the critical paths.

During the enumeration process, three possibilities may occur:

1. *A sink is reached: a full path is found.* When a full path π is found, its *SLACK* is computed by $LRAT_\pi - \sum_{v \in V_\pi} CD_v$ where V_π is the set of vertices traversed by path π excluding the sink vertex, and CD_v is the switching delay of net vertex v .
2. *A pruning case :* let $\pi = \{s, v_1, v_2, \dots, v_k, v\}$ be a partial path starting at source s and terminating at v ; T_π and S_π^2 are the expected delay and variance

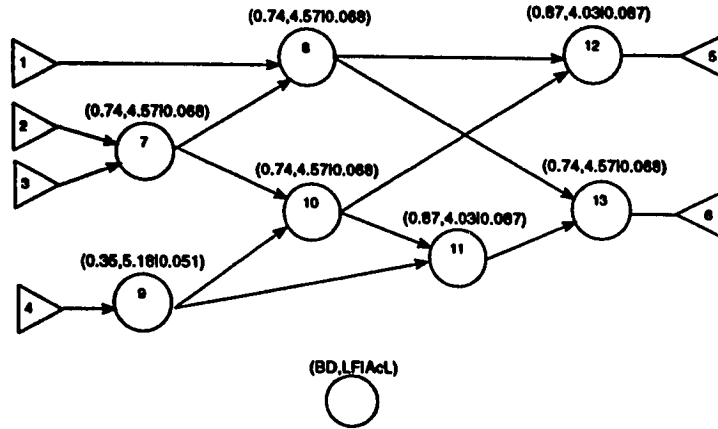


Figure 3.4: A circuit example to illustrate DFTP.

along that partial path excluding v . Because

$$T_{\pi} + MDS_v + \alpha \times \sqrt{(S_{\pi}^2 + MVS_v)} < T_{max}$$

the trace stops at v , and backtracks to v_k .

3. *The α -critical test is successful:* the trace continues.

In the algorithmic description of Figure 3.3, the elements of a sequence are enclosed within brackets (“[” and “]”). The notation “!S” should read, the elements of sequence “S”. The operations PUSH, POP, TOP are the standard stack operations.

Example 3.2 The graph in Figure 3.4 is used to illustrate the α -critical approach. For this graph, $V_s = \{1, 2, 3, 4\}$, $V_t = \{5, 6\}$, and $V_n = \{7, 8, 9, 10, 11, 12, 13\}$. Assume that for 2-pin and 3-pin nets, the average interconnect capacitance, \bar{c}_v , is 0.04966 pF and 0.06428 pF respectively, and the standard deviation, s_v , is

VERTEX	$LF^2 \times s^2$	MDS_v	MVS_v	$BD + LF \times (AcL + \bar{c})$
1	0.001459	2.933975	0.047132	0.117661
2	0.001459	5.917657	0.086031	0.117661
3	0.001459	5.917657	0.086031	0.117661
4	0.001459	5.731253	0.090362	0.100661
7	0.015209	5.799995	0.084572	1.655319
8	0.015209	2.816314	0.045673	1.742149
9	0.019541	5.630591	0.088903	1.485914
10	0.015209	4.144677	0.069363	1.828979
11	0.023690	2.315698	0.054153	1.344175
12	0.023690	1.074165	0.023690	1.074165
13	0.030464	0.971522	0.030464	0.971522

Table 3.3: Results for Example 3.2.

0.03819 pF and 0.02698 pF respectively. The results of executing step 1 in Figure 3.3 are shown in the third and fourth column of Table 3.3. The last column in Table 3.3 shows the switching delay for each vertex including interconnect delay. If the *confidence level* is set to "1" then only the following paths will be enumerated:

- $\pi_1 : \{2, 7, 10, 11, 13, 6\}$ $T_{\pi_1} = 5.918$ nanoseconds $S_{\pi_1}^2 = 0.086$
- $\pi_2 : \{3, 7, 10, 11, 13, 6\}$ $T_{\pi_2} = 5.918$ nanoseconds $S_{\pi_2}^2 = 0.086$
- $\pi_3 : \{4, 9, 10, 11, 13, 6\}$ $T_{\pi_3} = 5.731$ nanoseconds $S_{\pi_3}^2 = 0.09$

For this example $T_{max} = 5.918$ nanoseconds.

The question now is about the domain of applicability of the α -critical prediction strategy. The accuracy of the prediction results is dependent upon the quality of the

database supplying the estimated interconnect capacitance data. For example, for the test cases that we experimented with the database was constructed from previous designs with similar complexity. It is not practical to use the same database for analyzing other circuits of different complexity as this will significantly impact the prediction quality. One way to overcome this limitation is to construct a database for each class of circuits (1K, 2K, 100K, . . . , etc). Then depending on the circuit size the timing analyzer will select the appropriate database. Another point is that if the net capacitance distributions significantly deviate from Gaussian distributions, then the possibility of using the *confidence level* to draw conclusions about path criticality is weakened. The *Central Limit Theorem* from the *theory of statistics* indicates that this effect is usually minimal especially for large sample sizes. The *Central Limit Theorem* applies in our case.

3.6 Calculation of Timing Bounds on Nets

Layout tools work on individual nets as opposed to timing analysis tools which work on paths. Thus, a logical approach would be to transform the path constraints into constraints on nets. While path constraints ought to be satisfied, net constraints need not be satisfied in their entirety. In Chapter 4, we shall demonstrate the usage of net constraints in driving the floorplanning step.

There are three approaches to develop timing information on nets. The first approach classifies nets according to their covering paths [DUN84]. The nets belonging to the critical paths are the critical nets and have highest priority during physical design. The second approach transforms the path constraints into constraints on cell positions [JAC89]. The third approach assigns weights to nets according to the timing requirements of their covering paths [HAU87, YOU90]. Examples of these weight functions for of a net v are

1. $W_1(v) = 1$ [HAU87]

2. $W_2(v) = LF_v \times AcL_v$ [YOU90]

3. $W_3(v) = source_fanout(v)$ [HAU87]

A good weight function (e.g., $W_2(v)$) should consider the electrical and physical characteristics of the nets in distributing the path slack on its constituent nets.

3.6.1 *Minimax* Approach

In this work, we implemented an algorithm due to Youssef [YOU90]. The algorithm is called *minimax* and consists of deriving upper bounds on interconnect delays which are consistent with their covering paths timing constraints. The key idea of *minimax* comes from the fact that a net usually belongs to more than one path, and

hence the propagation delay on this net should be consistent with the longest path delay traversing this net. Next, we briefly explain the *Minimax* approach.

Let π be a circuit path in the graph $G=(V, E)$, and V_π is the set of vertices it traverses. The delay along path π is given by,

$$T_\pi = \sum_{v \in V_\pi} CD_v + \sum_{v \in V_\pi} ID_v \quad (3.17)$$

where, CD_v (Equation 3.5) and ID_v (Equation 3.8) are the switching delay and the interconnect delay of net vertex v , respectively.

Path π will not have a *long path problem* iff:

$$T_\pi \leq LRAT_\pi \quad (3.18)$$

But T_π has two components: a layout independent component (i.e., sum of the switching delays) and a layout dependent component which is the sum of interconnect delays. Therefore, for the final layout to be free from *long path problems*, the interconnect delays must satisfy the following constraint:

$$\sum_{v \in V_\pi} ID_v \leq LRAT_\pi - \sum_{v \in V_\pi} CD_v \quad \forall \pi \in \Pi \quad (3.19)$$

where Π is the set of all paths in the circuit graph G .

Now the problem for deriving timing constraints on all nets can be stated as follows: *Given a VLSI design represented by a graph $G = (V, E)$, find net delays ID_v for each net $v \in V$ such that inequality (3.19) holds.*

Minimax consists of computing upper bounds on interconnect delays (i.e., ID_v) which satisfy inequality (3.19). This is achieved as follows. Each net v on path π is assigned a timing bound given by the following expression,

$$u_v^\pi = Slack_\pi \times \frac{w_v}{\sum_{u \in V_\pi} w_u} \quad (3.20)$$

where,

w_v : the weight of net v defined as the product of the capacitance on the loading input pins driven by the net, and the load factor of its driving output pin, i.e.,

$$w_v = LF_v \times AcL_v;$$

$Slack_\pi : LRAT_\pi - \sum_{v \in V_\pi} CD_v$ and CD_v is as defined in Equation 3.5.

Let Π_v be the set of paths going through net v . A consistent timing bound for net v should be the smallest bound among those bounds for paths traversing this net. Such a bound is derived as follows:

$$u_v^* = \min_{\pi \in \Pi_v} u_v^\pi \quad (3.21)$$

The reader can observe that the problem of finding u_v^* requires enumerating all the paths in the design. But the number of paths in a VLSI circuit can be very high. It was demonstrated in [YOU90] that the problem of computing u_v^* for all nets is NP-Hard. Three efficient approximation algorithms for the *Minimax* problem were proposed in [YOU90]. These algorithms try to enumerate a small subset of paths such that all nets are covered, and every selected path has minimum slack. The

selected paths are then used in the derivation of the timing bounds for all nets as dictated by the *Minimax*.

In this work, we implemented one of those algorithms namely the *Minimax-PERT* algorithm. Our selection was based on the execution time.

3.6.2 Minimax-PERT

The path slack $LRAT_{\pi} - \sum_{v \in V_{\pi}} CD_v$ and the path weight functions are additive functions and hence satisfy the optimality principle [HOR78]. The *Minimax-PERT* algorithm obtains an approximate solution for Equation 3.21 by exploiting this idea.

For the *Minimax-PERT* the problem of finding u_v^* can be formulated as,

$$u_v^* = w_v \times \frac{\min_{\pi \in \Pi_v} Slack_{\pi}}{\max_{\pi \in \Pi_v} \sum_{u \in V_{\pi}} w_u} \quad (3.22)$$

The algorithm enumerates a polynomial number of paths using a PERT-like [KIRK66] trace of the graph. The algorithm consists in finding for each net v the slack of the longest path traversing v , and the weight of the path that has maximum weight among all paths traversing v . Figure 3.5 is a complete description of the algorithm. Our implementation of the *Minimax-PERT* depends on the following definition for the *slack* of net v .

Definition 2 Let AAT_v and $LRAT_v$ be the actual and latest required arrival times

ALGORITHM *Minimax PERT***NOTATION**

- $\Gamma^+(v)$: all successors of vertex v
 $\Gamma^-(v)$: all predecessors of vertex v
 $Slack_v^{min}$: slack of worst path traversing net v
 W_v^{max} : $\max_{\pi \in \pi(v)} (W_\pi + W_v^+)$
 W_v^+ : maximum weight to sink for net v
 D_v : delay of longest path segment terminating at v inclusive
 W_v : weight of the longest path terminating at v inclusive
 L_v : latest required arrival time at net v
 CD_v : switching delay of v
 w_v : weight of net v
 u_v : Delay bound

step 1: Initialize

```

FOREACH  $v \in (V_s \cup V_n)$  DO
  (# $BD$ ,  $LF$ ,  $AcL$  ARE FROM CELL LIBRARY#)
   $CD_v = BD_v + LF_v \times AcL_v$ ;
   $w_v = LF_v \times AcL_v$ ;
ENDFOREACH
 $wave \leftarrow V_s$ ;

```

step 2: Forwardtrace

```

WHILE ( $wave \neq \phi$ ) DO
  BEGIN
     $newwave \leftarrow []$ ;
    FOREACH  $v \in wave$  DO
       $D_v \leftarrow CD_v + \max_{u \in \Gamma^-(v)} D_u$ ;
       $W_v \leftarrow w_v + \max_{u \in \Gamma^-(v)} W_u$ ;
       $newwave \leftarrow newwave \cup \Gamma^+(v)$ ;
    ENDFOREACH
     $wave \leftarrow newwave$ ;
  ENDWHILE

```

step 3: Backwardtrace

```

wave ←  $V_i$ ;
WHILE (wave ≠  $\phi$ ) DO
  newwave ←  $\emptyset$ ;
  FOREACH ( $v \in \textit{wave}$ ) DO
    (#Compute the latest required arrival time at net  $v$  #)
     $L_v \leftarrow \min_{u \in \Gamma^+(v)} (L_u - d_u)$ ;
    (#Compute the maximum weight to sink for each net  $v$  exclusive #)
     $W_v^+ \leftarrow \max_{u \in \Gamma^+(v)} W_u^+$ ;
    newwave ← newwave  $\cup \Gamma^-(v)$ ;
  ENDFOREACH
  wave ← newwave;
ENDWHILE
step 4: Find_Bounds
FOREACH ( $v \in V_n$ )
   $Slack_v^{min} = L_v - D_v$ ; (#Minimum SLACK #)
   $W_v^{max} = W_v + W_v^+$ ; (#Maximum weighted path #)
  (#Delay bound assigned to net  $v$  #)
   $u_v = w_v \times \frac{Slack_v^{min}}{W_v^{max}}$ ;
ENDFOREACH
END Minimax_PERT.

```

Figure 3.5: Minimax-PERT Algorithm.

at the loading pins of net v . The slack of net v is given by,

$$Slack_v = LRAT_v - AAT_v \quad (3.23)$$

As pointed out in [YOU90], a single run of the *Minimax-PERT* algorithm is not sufficient to distribute all available slacks on the nets. Consequently, the resulting bounds will be loose. To make the computed delay bounds tight (maximal), the remaining path's slacks should be distributed over nets in proportion to their weights. After the first pass of the *Minimax-PERT* algorithm, we compute the remaining slack for all nets. Then, the delay bound of each net v is incremented by,

$$\frac{w_v}{W_v^{max}} \times Slack_v \quad (3.24)$$

where,

w_v is the weight of net v ,

W_v^{max} is the maximum-weight path traversing vertex v , and

$Slack_v$ is the remaining slack on vertex v .

This process is repeated until slacks are near zero. Observe that we need to compute $\frac{w_v}{W_v^{max}}$ only once for each net, because it is a function of static parameters (i.e., LF_v , AcL_v). It was demonstrated in [YOU90] that following this iterative approach, the slacks go monotonically to zero.

Example 3.3 The partial graph of Figure 3.6 is used to illustrate the execution

of the *Minimax-PERT* algorithm. The results are summarized in Table 3.4. The table has four columns, one for each nonterminal vertex, and five rows. The first row labeled " $Slack_v^{min}$ " gives the slack of the longest path traversing net v . The second row " W_v^{max} " gives the weight of the path with maximum weight among all the paths traversing net v . The third row " D_v^{max} " is the delay of the longest path traversing net v . The fourth row " w_v " gives the weight of each net v , and the last row " u_v " indicates the timing bound of net v as computed by *Minimax-PERT*. The latest required arrival time at v_5 is assumed to be 18 *nano*seconds.

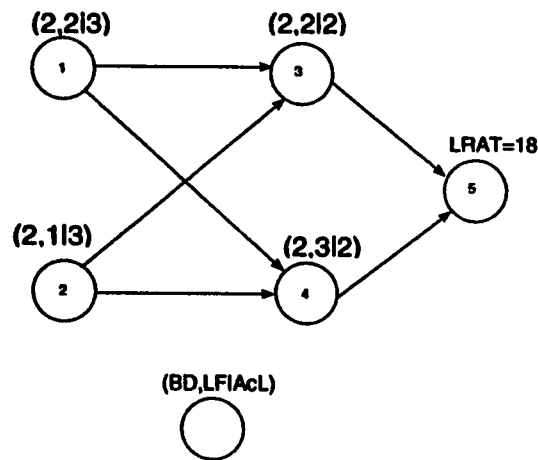


Figure 3.6: A circuit example to illustrate Minimax-PERT.

Time Complexity

Let N be the number of vertices in the circuit graph G ; N represents the number of *nets* in the circuit. Let k be the *in-degree* of any vertex in G . Further assume that

	v_1	v_2	v_3	v_4
$Slack_v^{min}$ ns	2	5	4	2
W_v^{max} ns	12	9	10	12
D_v^{max} ns	16	13	14	16
w_v ns	6	3	4	6
u_v ns	1	1.667	1.6	1

Table 3.4: Delay bounds computation for Figure 3.6.

the graph G is levelized. An acyclic directed graph G with n vertices is said to be levelized if the level of any vertex v is defined as follows,

1. All source vertices are assigned level zero.
2. The level of a non-source vertex v is,

$$l(v) = 1 + \max_{u \in \Gamma^-(v)} l(u)$$

where $\Gamma^-(v)$ is the set of predecessor vertices of vertex v .

For each *net*, computing $Slack_v^{min}$ and W_v^{max} (Figure 3.5) requires $O(k)$. Since the graph is levelized, then each vertex will be processed exactly once. Thus, the time complexity of this algorithm is $O(kN)$. If k is small then the algorithm will exhibit a linear time complexity of $O(N)$.

chip	Description	Format	IOPads	Cells	CLOCK
fract	fractional multiplier	VPNR	24	125	38 ns
strcut	16-bit multiplier	VPNR	64	1888	140 ns
highway	traffic light controller	VPNR	11	45	20 ns

Table 3.5: Test cases statistics.

3.7 Experimental Results

The algorithms described in this chapter were implemented in C language on *NEXT* workstations. The critical paths predicted by the *DFTP* algorithm were used by a timing-driven genetic placer [NAS94] developed at KFUPM. The critical paths enumerated after layout were a proper subset of the predicted paths. The implementation was tested on three benchmark designs (Table 3.5). All the test cases are implemented in the standard-cell design style for 2μ *p*-well SCMOS technology.

Input to the system is either a VPNR or AHPL net list. There are three stages in the program execution. The first stage is responsible for reading the cell delay parameters as described in Section 3.2. The necessary information is extracted from the OASIS standard-cell library [MCN90].

The second stage transforms the netlist description of the design into an acyclic directed graph. The vertices of the graph correspond to the circuit elements and the edges represent the connections.

α	fract	highway
0	1	1
1	3	2
2	7	3
3	8	4
4	9	4
5	11	5
α^{max}	369	65

Table 3.6: Number of predicted paths as a function of α .

α	0	0.15	0.5	1
Critical paths	28	420	13160	> 90,000
CPU (min)	2.17	3.33	4.17	> 11

Table 3.7: Number of predicted paths for the *16-bit multiplier*.

The third step consists of using user specified *confidence level* α and *clock period* for validating the specified *clock period*², prediction of critical paths and the derivation of delay bounds on all the interconnects.

Table 3.6 shows the number of predicted critical paths for various values of α for three test cases. The last row in Table 3.6 gives the number of critical paths for α^{max} . All the test cases in Table 3.5 took less than 105 *seconds* of CPU time with α^{max} as *confidence level*. Table 3.7 gives the results of executing *DFTP* on the *16-bit multiplier*. Table 3.8 gives the delay and variance of the longest path, and the value of α^{max} for each design.

²If the clock period is not large enough, the system will report paths with negative slacks.

	fract	highway	struct
Delay of longest path (ns)	34.141	15.524	122.183
Variance of longest path	0.542	0.328	1.659
α^{max}	163	62	132

Table 3.8: α^{max} values and delay and variance of longest path.

Design	$T_{achieved}$ (ns)	iterations (ns)	smallest bound (ns)	largest bound (ns)
fract	38	5	0.11	7.3
highway	20	3	0.038	6.464

Table 3.9: *Minimax-PERT* test results.

The results of executing *Minimax-PERT* algorithm on two test cases are summarized in Table 3.9. The first column gives the minimum clock period which the *genetic placer* in [NAS94] was able to achieve. The second column gives the number of iterations required to distribute the remaining slacks. The third and fourth columns give the minimum and maximum delay bounds.

3.8 Conclusion

We have introduced the α -critical approach, a new prediction strategy for timing critical paths prior to layout. This strategy is based on the usage of estimated interconnect capacitance information from past designs in measuring the criticality of paths. A confidence interval is built around the longest path in the design. The

paths that fall inside the interval are the critical paths. The span of the interval depends on the confidence level. Experimentation with real circuits has shown that the critical paths enumerated after layout were a proper subset of the predicted paths.

We implemented *Minimax-PERT*, a procedure proposed in [YOU90]. This algorithm which is based on minimax ideas to derive maximal delay bounds for nets has been described. The timing bounds are maximal in the sense that all slacks are distributed among the nets, thus maximizing degrees of freedom for physical design tools.

In the next chapter, we demonstrate the usage of timing constraints on nets to implement a timing driven floorplanner.

Chapter 4

Timing Driven Floorplanning

4.1 Introduction

As any other design step, floorplanning represents a certain level of abstraction. The floorplanning process views the circuit as a set of rectangular blocks interconnected by signal nets. The goal of floorplanning is to come up with a placement plan that will decide topological proximity as well as appropriate shapes and orientations of each block. A floorplan solution is supposed to satisfy geometric constraints while optimizing certain objective function, e.g., chip area, performance, wirelength, or a combination of two or more of these.

Floorplanning is a preparatory step to placement. It can be seen as a feasibility study of the placement. If floorplanning is not successful, then placement will also be not successful. Floorplanning is an essential step in a hierarchical design methodology. Floorplanning helps in solving problems such as overall required area, sizes and shapes of blocks, pin and pad locations, etc.

A top-down hierarchical IC design methodology reduces the computational complexity of the design process by decomposing the design process into several steps of reasonable complexity. During the early synthesis steps, designers usually make several decisions on block parameters, e.g., timing, area, aspect ratio, and I/O pad locations. It is hoped that, at later design stages, these decisions can be realized in an efficient manner in terms of layout. However, it is difficult to predict the consequences of these decisions on the final layout. In fact, the consequences of the initial decisions may only appear at a much later stage of the IC design process. As a result, a significant amount of iteration is required between logic design and physical layout steps to converge to a good layout [YOU90].

The amount of iteration can be significantly reduced if all the design steps are consistently made sensitive to the stated objectives and constraints. For example, suppose we want to design an IC chip which should run at the best possible clock period, then it is more logical to make all the design steps sensitive to this objective.

As stated in Chapter 1, this work adopts a design methodology for which *timing*, *interconnection length* and *area* are the main objectives. The floorplanning step has a dominant effect on circuit performance. It is of extreme importance to make the floorplanning step timing-sensitive since this step helps decide several major questions with respect to the structure and timing performance of the circuit. In this chapter, we show how to make the floorplanning step *sensitive* to the *timing predictions* generated by the timing analysis step.

Our floorplanning heuristic can be summarized in the following steps:

1. Construction of a timing driven topological arrangement using a force directed approach.
2. Conversion of the topological arrangement into a legal floorplan (*floorplan sizing*). This step consists of two tasks:
 - derivation of efficient graph models which capture the topological arrangement;
 - block resizing in order to optimize area and satisfy geometric constraints, this is followed by two steps:
 - computing block locations on the XY-plane;
 - calculating an enveloping rectangle.

The rest of the chapter is organized as follows. Section 2 presents some concepts related to floorplanning. A formal definition of the floorplanning problem is given in Section 3. In Sections 4-5, we present our solution approach to timing driven floorplanning. Advantages and disadvantages related to our floorplanning methodology are discussed in Section 6. Experimental results will be provided in Section 7. We conclude in Section 8.

4.2 Preliminaries

A *floorplan* of k blocks is a partitioning of a rectangular layout surface into n basic rectangles (blocks) such that each basic rectangle is wide enough to accommodate one block. A floorplan is a *slicing* floorplan if it consists of a single block, or there is a horizontal or vertical line that cuts the floorplan into two blocks such that each block is a slicing floorplan. Otherwise, the floorplan is *nonslicing*. Figure 4.1 shows examples of slicing and nonslicing floorplans. Our floorplanning approach generates general floorplans. It is not restricted to slicing structures.

4.3 Problem Definition

Our floorplanning problem can be formulated as follows:

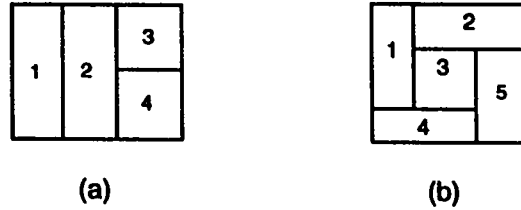


Figure 4.1: (a) slicing floorplan; (b) nonslicing floorplan.

Given:

1. A set of rectangular blocks $B = \{b_1, b_2, \dots, b_i, \dots, b_k\}$. For each $b_i \in B$ we have
 - w_i, h_i : width and height of b_i . These are constants for rigid blocks and variables for flexible blocks.
 - w_i^{min}, w_i^{max} : lower and upper bounds on the width of b_i if b_i is a *variable-shape* block.
 - a_i : area of b_i (i.e., $a_i = w_i \times h_i$). a_i is constant.
2. A set of *nets* $N = \{n_1, n_2, \dots, n_i, \dots, n_k\}$ describing the connectivity information. Each n_i is assigned a weight u_i equal to the delay upper bound computed by the *Minimax-PERT* algorithm (Chapter 3).
3. A set of timing critical paths. Each path is associated with a *slack* value.

4. Desirable floorplan aspect ratio : $\rho = H/W$ where H and W are the height and width of the floorplan.

Output:

A legal floorplan, that is, a floorplan satisfying the following constraints and objectives:

1. each block b_i is assigned to a location (x_i, y_i) ,
2. minimize chip area,
3. no overlaps between blocks,
4. $w_i^{min} \leq w_i \leq w_i^{max}$ and $a_i = w_i \times h_i$ for each flexible block b_i ,
5. meet chip aspect ratio constraint, and
6. meet timing constraints on the critical paths.

The reader can see that the above problem is an NP-Hard problem [HOR78]. If all the blocks are fixed-shape blocks, the problem reduces to a placement problem. The placement problem is a generalization of an NP-Hard problem: the quadratic assignment problem.

In the next section, we present our approach to timing driven floorplanning. Our approach consists of two steps. The first step builds a floorplan that satisfies

all timing constraints on the critical paths using a constructive force directed approach. The second step is a floorplan sizing step aiming at satisfying the remaining constraints and objectives while preserving the relative positions of the blocks as derived from the first step.

4.4 A Floorplanning Heuristic

The number of feasible solutions to any physical design problem is at least exponential with respect to the number of blocks being placed. Thus, heuristic methods should be used to get suboptimal solutions to such problems and which satisfy all the stated constraints. Among several possible solution methods to the floorplanning problem, we selected a solution method in which the problem is divided into two major steps. In the first step, we adopt a *cluster growth* technique that enables us to constructively build a *timing* sensitive solution by adding blocks to an already constructed solution. This is followed by an iterative reshaping step to optimize area and satisfy the remaining constraints on geometric fit and aspect ratio.

4.4.1 Force Directed Topological Arrangement

The constructive floorplan consists of two main procedures:

- selection of a block for assignment, and
- finding a suitable location for the block.

The *selection* procedure combines two criteria: *timing* and *connectivity*. The timing criterion is an evaluation of the timing constraints on the nets connecting the block to the partial floorplan. The connectivity criterion is the number of interconnects between the block and the partial floorplan.

The delay bounds computed by the *Minimax-PERT* algorithm are transformed into timing costs. The timing cost is increasing with decreasing values of delay bound. The delay to cost transformation is done using the following function,

$$cost_i = \frac{CLOCK - u_i}{CLOCK} \quad (4.1)$$

where, *CLOCK* is the clock period and u_i is the delay bound on net $n_i \in N$. *CLOCK* is an upper bound on u_i , and therefore, all the $cost_i$'s will be in the real interval $[0,1]$. Binding the $cost_i$'s to a certain range is very crucial to avoid possible numerical stability problems during program implementation.

A *gain* function that combines both timing and connectivity is evaluated for each unassigned block having connections to the partial floorplan. The objective function for block b_i has the following form:

$$G_i = \sum_{j \in F_k} c_{ij} + \beta \sum_{i \in B_j, j \in N_k} (1 - p_j) \times cost_j \quad (4.2)$$

where,

F_k : set of blocks in the partial floorplan at step k of floorplanning,

B_j : set of blocks interconnected by net n_j ,

N_k : set of partial nets¹ at step k of the floorplanning process,

p_j : percentage of placed blocks of B_j , where $0 \leq p_j \leq 1$, (p_j is initially zero and increases as more blocks of B_j get added to the partial floorplan),

c_{ij} : connectivity of block b_i to block $b_j \in F_k$, and

β : real positive weight coefficient.

The *gain* function G_i consists of two terms: connectivity of block b_i to the partial floorplan F_k , and a weighted sum of timing costs on nets connecting b_i to blocks in F_k . Selecting the block with the maximum number of connections will minimize the connection length on the floorplan, and it will in some sense group highly connected blocks together. The weighted sum of timing costs (the second term in G_i) will favor the selection of those blocks that are involved in tight timing constraints (i.e., high timing cost).

Unassigned blocks are selected one at a time. The block with maximum gain, G_i , is selected next and passed to the *Place_Block* procedure for positioning (see Figure 4.2).

¹A net connects a set of blocks. If some of these blocks are already in the partial floorplan, we call the net a partial net.

ALGORITHM *Place_Block***NOTATION*****b*** : selected block for placement***D(l, l')***: Manhattan distance between locations *l, l'***BEGIN** Compute *target* location, *l*, for block *b*; **IF** (*l* is VACANT) **THEN** Assign *b* to *l*, and mark *l* as OCCUPIED; **ELSE** Find a VACANT location *l'* such that *D(l, l')* is minimum; Assign *b* to *l'*, and mark *l'* as OCCUPIED; **ENDIF****END.**Figure 4.2: *Place_Block* procedure.

The *Cluster_Growth* procedure grows greedily a topological assignment using a variation of the *force-directed* approach [SM91] (see Figure 4.3).

Two types of attraction forces act on a pair of connected blocks: a *timing-based force* and a *connectivity-based force*. The timing-based force is a function of the delay bounds on the nets interconnecting the blocks. This force is inversely proportional to the value of the delay bound. The smaller the timing bound, the higher is the attraction force. As a result, the blocks which are connected by timing critical nets will be assigned locations in topological proximity. On the other hand, the connectivity-based force is directly proportional to the number of connections between the blocks. Thus, highly connected blocks will be placed close to each other. This will minimize connection length. The *timing-based force* f_i^t is set equal to the

ALGORITHM *Cluster_Growth***NOTATION** **B** : set of blocks to be placed; **B_p** : set of already placed blocks (*placed set*); **B_a** : set of blocks having common connections with blocks in B_p (*adjacent set*); **N_i** : incidence set of nets for block b_i ; **(b_i, b_j)** : connection between b_i, b_j ; **G_i** : gain function for selecting block b_i (Equation 4.2);**BEGIN****Step 1: Initialize** Compute initial chip dimensions H, W ; Select *seed* block b_0 ; Assign b_0 to a *seed* location (center of chip $(W/2, H/2)$); $B_p \leftarrow B_p \cup \{b_0\}$; $B_a \leftarrow B_a \cup \{b_i \mid b_i \in B \text{ and } (b_0, b_i) \in N_{b_0}\}$; $B \leftarrow B - \{b_0\}$;**Step 2: Cluster Growth** **WHILE** ($B \neq \phi$) **Do** **BEGIN** **IF** ($B_a \neq \phi$) **THEN** Select block b_s such that $G_s = \max_{b_i \in B_a} g_i$; **CALL** *Place_Block*; $B_p \leftarrow B_p \cup \{b_s\}$; $B_a \leftarrow B_a \cup \{b_j \mid b_j \in B \text{ and } (b_j, b_s) \in N_{b_s}\}$; $B_a \leftarrow B_a - \{b_s\}$; $B \leftarrow B - \{b_s\}$; **ELSE** (# The circuit has disconnected components #) Select a new *seed* b_0 ; $B_p \leftarrow B_p \cup \{b_0\}$; $B_a \leftarrow B_a \cup \{b_i \mid b_i \in B \text{ and } (b_0, b_i) \in N_{b_0}\}$; $B \leftarrow B - \{b_0\}$; **ENDIF** **ENDWHILE****END.**

Figure 4.3: Cluster_Growth algorithm description.

timing cost, $cost_i$ (Equation 4.1),

$$f_i^t = cost_i \quad (4.3)$$

Let c_{ij} represent the number of connections between blocks b_i and b_j , The exerted *connectivity-based force* on b_i due to b_j is given by,

$$f_j^c = c_{ij} \quad (4.4)$$

The *Place-Block* procedure utilizes the above mentioned forces in the derivation of a zero-force location for the selected block. The zero-force location is called the *target location*. Let b_s be the block selected at step k of the floorplanning process, and $B_s = \{b_{s_1}, b_{s_2}, \dots, b_{s_i}, \dots, b_{s_n}\}$ be the set of blocks which are connected to b_s and have already been positioned. Let f_i^t and f_i^c be the forces between b_s and $b_{s_i} \in B_s$. The zero-force timing sensitive location (x_s^t, y_s^t) and the zero-force connectivity sensitive location (x_s^c, y_s^c) are computed as follows:

$$x_s^t = \frac{\sum_{i=1}^n p_{s_i} f_{s_i}^t x_{s_i}}{\sum_{i=1}^n p_{s_i} f_{s_i}^t} \quad (4.5)$$

$$y_s^t = \frac{\sum_{i=1}^n p_{s_i} f_{s_i}^t y_{s_i}}{\sum_{i=1}^n p_{s_i} f_{s_i}^t} \quad (4.6)$$

$$x_s^c = \frac{\sum_{i=1}^n p_{s_i} f_{s_i}^c x_{s_i}}{\sum_{i=1}^n p_{s_i} f_{s_i}^c} \quad (4.7)$$

$$y_s^c = \frac{\sum_{i=1}^n p_{s_i} f_{s_i}^c y_{s_i}}{\sum_{i=1}^n p_{s_i} f_{s_i}^c} \quad (4.8)$$

where, p_{s_i} is percentage of placed blocks belonging to the net connecting b_s and b_{s_i} .

The *target* location for b_s is derived as follows:

$$x_s = \alpha_1 x_s^t + \alpha_2 x_s^c \quad (4.9)$$

$$y_s = \alpha_1 y_s^t + \alpha_2 y_s^c \quad (4.10)$$

where, $0 \leq \alpha_1, \alpha_2 \leq 1$, and $\alpha_1 + \alpha_2 = 1$. α_1 and α_2 are weight coefficients to define the relative importance of (x_s^t, y_s^t) and (x_s^c, y_s^c) . (x_s^t, y_s^t) and (x_s^c, y_s^c) as defined in Equations (4.5-4.8), minimize the weighted sum of the squared Euclidean distances from b_s to its adjacent blocks in the partial floorplan. We illustrate the above computations with an example.

Example 4.1 Referring to Figure 4.4, suppose the selected block for positioning is b_3 . Assume also that b_1 and b_2 are located at (5,5) and (1,7) respectively. The forces exerted on b_3 due to b_1 for $CLOCK = 5ns$ are,

$$f_1^t = \frac{5-2}{5} + \frac{5-0.5}{5} = 1.5, \quad f_1^c = 2$$

The forces exerted on b_3 due to b_2 are,

$$f_2^t = \frac{5-0.2}{5} = 0.96, \quad f_2^c = 1$$

Thus, the timing sensitive location of b_3 is,

$$x_3^t = \frac{5 * 1.5 + 1 * 0.96}{1.5 + .96} = 3.43$$

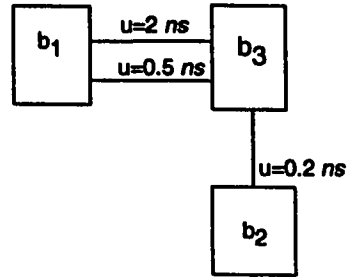


Figure 4.4: A circuit example to illustrate the target location computation.

$$y_3^t = \frac{5 * 1.5 + 7 * 0.96}{1.5 + .96} = 5.78$$

Similarly, the connectivity sensitive location is,

$$x_3^c = 3.67, \quad y_3^c = 5.67$$

Let α_1, α_2 in Equations (4.9-4.10) be equal to 0.5, then the *target* location of b_3 is given by,

$$x_3 = [0.5 \times (3.43 + 3.67)] = 4, \quad y_3 = [0.5 \times (5.78 + 5.67)] = 6.$$

Figure 4.5 shows three possible assignments for the circuit in Figure 4.4: (a) when b_3 is assigned to its *timing* sensitive location, (b) b_3 at its *connectivity* sensitive location, and (c) when both criteria are considered.

In computing a block's *target* location, the Place_Block procedure considers the connections of the block to the I/O pads. At the floorplanning step, we assign I/O pads to sides of the floorplan, but without identifying their exact locations on the floorplan boundary. The I/O pads assigned to a particular side are assumed

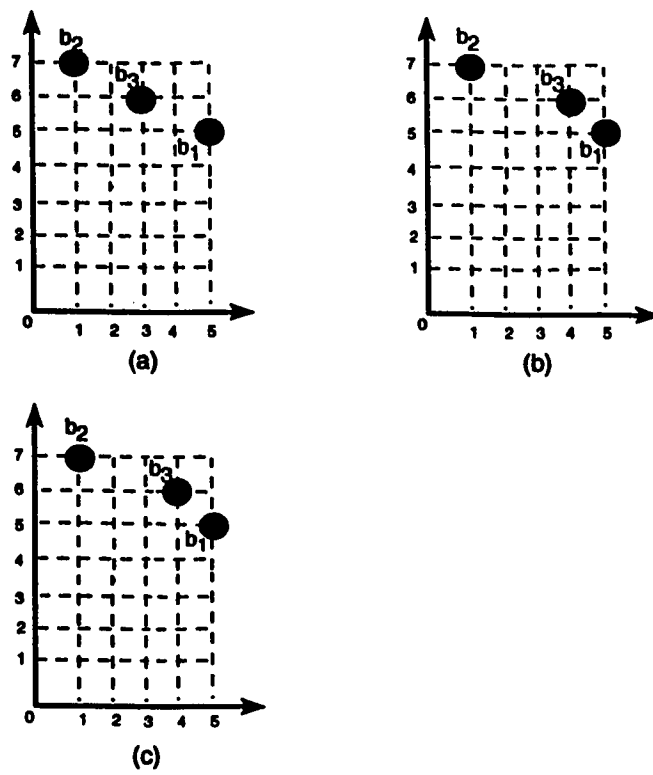


Figure 4.5: Target location for b_3 : (a) at (x_3^t, y_3^t) ; (b) at (x_3^c, y_3^c) ; (c) at (x_3, y_3) .

to reside in the middle of that side. The I/O pads of the design are distributed equally on the four sides. Our strategy for considering the I/O pad locations in the *target* location computation can be summarized as follows. If the block is connected to some unassigned pad, we compute the target location without considering its connection to the unassigned I/O pad. Next, we assign this pad to the closest side to the computed target location. Then, recomputation of the target location for the block is carried out (considering the location of the I/O pad). Considering the I/O pads in computing target locations will pull those blocks that have connections to I/O pads toward the periphery of the chip. As our experiments demonstrated, including the I/O pad locations in the target location computations, reduces the probability of finding the target location of some block being occupied by another block, and to some extent improves the circuit timing.

The computation of the *target* location according to Equations (4.5-4.10) is similar to finding the *center of gravity* in physics. It is an intrinsic nature of this method that it will assign most of the blocks to the center of the chip, as this will definitely lead to a minimum-energy state. As a result, it is very likely that the *target* location for the new selected block will be occupied by an already placed block. In case the target location of the new block is occupied, the following strategy is adopted to avoid displacing a block very far from its target location. The new block is moved to the nearest free location (using minimum Manhattan distance measure). Figure 4.2

is a general description of the *Place_Block* procedure. The complete algorithm for generating the topological arrangement of the blocks is presented in Figure 4.3. We call this algorithm *Cluster_Growth*. An informal description of this algorithm follows.

The algorithm maintains three disjoint sets: a placed set, an adjacent set, and an unplaced set [KAN83]. The *placed set* contains the already positioned blocks. The set of blocks having common connections with the elements of the *placed set* constitute the *adjacent set*. The *unplaced set* contains the remaining blocks of the design.

The algorithm starts by computing initial dimensions for the chip using the supplied aspect ratio, ρ , and the total area of the blocks. The initial height and width of the chip are computed as follows:

$$H = \sqrt{\rho \sum_i a_i} \quad (4.11)$$

$$W = \frac{H}{\rho} \quad (4.12)$$

The calculation of initial H and W is required for making the growth of the chip controlled by the supplied aspect ratio. Next, the algorithm selects a *seed* block. We experimented with three seed selections. The first seed selection was the block with maximum connections. The second seed selection was a block belonging to the

most *critical* path. The third seed was a batch seed (a group of blocks belonging to the most critical path). The second and third seeds exhibited similar results, and produced superior floorplan solutions than those obtained with the first seed selection. As a general guideline, one should avoid selecting a seed which is connected to I/O pads (especially if the seed will end up at the center of the chip). After initializing the above mentioned sets based on the selected seed (Step 1 in Figure 4.3), the algorithm proceeds to Step 2. Suppose that there are n blocks in the *adjacent set* at the k th iteration of the floorplanning process. The algorithm computes gain function G_i for each of the n blocks (Equation 4.2). The block with maximum connections and worst timing constraints is selected: this corresponds to the block with maximum G_i . The selected block is then passed to the *Place_Block* procedure for positioning. This process is repeated until all the blocks of the design are placed. The output from this algorithm is a list of the blocks with their xy -coordinates, and a list of the I/O pads and their assigned sides. Since the force-directed technique models the blocks as *points*, the xy -coordinates are interpreted as the block's center locations.

Example 4.2 Referring to Figure 4.6, let b_5 be the *seed* block. Assume that the clock period is $10ns$, $\beta = 0.5$ in Equation 4.2, and $\alpha_1 = \alpha_2 = 0.5$ in Equations (4.9-4.10). Table 4.1 summarizes the results of running the *Cluster Growth* algorithm. The floorplan for this circuit is shown in Figure 4.7. In Figure 4.7, the blocks are

modeled as dots because, as mentioned earlier, the force-directed technique ignores the shape and size of the blocks and considers them as points.

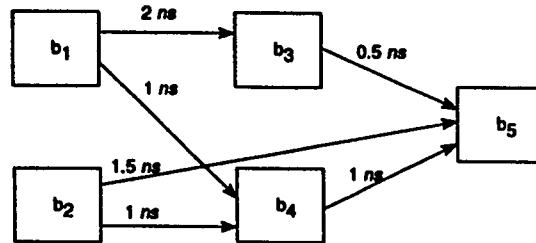


Figure 4.6: Circuit for Example 4.2.

iteration	placed set	adjacent set	G values
0	b_5	b_3^*, b_4, b_2	$G_2 = 1.42, G_3 = 1.48, G_4 = 1.45$
1	b_5, b_3	b_4^*, b_2, b_1	$G_2 = 1.42, G_1 = 1.4, G_4 = 1.45$
2	b_5, b_3, b_4	b_2^*, b_1	$G_2 = 2.88, G_1 = 2.85$
3	b_5, b_3, b_4, b_2	b_1^*	$G_1 = 2.85$
4	b_5, b_3, b_4, b_2, b_1	–	–

Table 4.1: Results of running *Cluster Growth* on Example 4.2.

Time Complexity of *Cluster Growth*

Let K be the number of blocks in the *adjacent set* at step k of the floorplanning process, and m be the average number of connections per block. Selecting one block for positioning requires $O(mK)$ time. The computation of the *target* location is almost constant, $O(m)$. Therefore, the *Cluster Growth* algorithm has an overall time complexity of $O(mK)$.

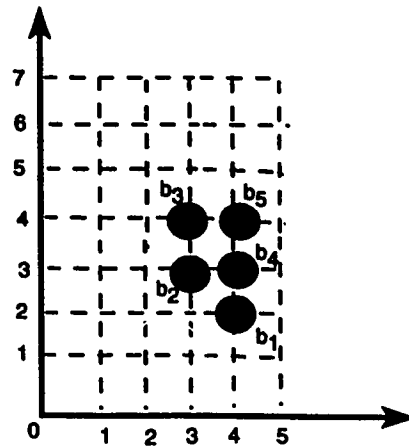


Figure 4.7: Force-directed floorplan for Example 4.2.

Although, the floorplan generated by the *Cluster Growth* algorithm is optimized for timing, it does not satisfy geometric fit. Thus, the next logical step is to legalize the resulting floorplan and optimize its area. In the next section, we describe an algorithm that converts the topological arrangement to a *legal* floorplan. This step is known as floorplan sizing.

4.4.2 Floorplan Sizing

In this section, we are concerned with the actual floorplanning where block attributes (e.g., absolute locations, dimensions for the variable-shape blocks) are defined and constraints on geometric fit and aspect ratios are satisfied. The output from the *Cluster-Growth* algorithm is a topological arrangement optimized for timing and

connectivity. In order to get the final legal floorplan, we must remove all overlaps. This step is called *floorplan sizing* and is performed without undoing any of the decisions of the timing-sensitive force-directed step, i.e., blocks are not allowed to jump over each other.

We adopted a constraint-based approach to convert the topological arrangement into a *legal* floorplan. This floorplan sizing phase consists of two major steps:

1. construction of constraint set, and
2. shape optimization.

Before we describe the details of each step, we shall discuss the graph model used to represent the constraint set.

Graph Model

The topological arrangement produced by the *Cluster-Growth* algorithm is interpreted as a set of topological constraints. An example of a constraint is that one block should be placed to the left or above another block. Two directed acyclic graphs are used to capture the *constraint set*: a horizontal constraint graph G_H and a vertical constraint graph G_V . The vertex set of G_H is the set of blocks plus two dummy vertices: L and R . Similarly, the vertex set of G_V is the set of blocks

plus two dummy vertices: T and B . The dummy vertices L , R , T , B correspond, respectively, to left, right, top, and bottom boundaries of the chip. The edge set of G_H models the *to-the-left/to-the-right* relationships. Hence, an edge $(b_i, b_j) \in G_H$ indicates that b_i is to the left of b_j . All vertices in G_H are to the left of the dummy vertex R , and to the right of vertex L . Thus, G_H contains the edges (L, b_i) and (b_i, R) for each block b_i . The edge set of G_V models the *on-the-top/on-the-bottom* relationships. Hence, an edge $(b_i, b_j) \in G_V$ indicates that b_i is below b_j . All vertices in G_V are above the dummy vertex B , and below vertex T . Thus, G_V contains the edges (B, b_i) and (b_i, T) for each block b_i . Figure 4.8 shows an example of an overlapping floorplan and its corresponding topological constraint graphs.

Construction of Constraint Graphs

Two blocks are constrained if the center of one block must be to the left/below the center of the other. Vijayan and Tsay [VIJ91] introduced the notions of *completeness* and *strong completeness* for a topological constraint set. A constraint set is *complete* if there exists a directed path between every pair of blocks (b_i, b_j) either in G_H , in G_V , or in both. In a *strong complete* set each pair of blocks is *adjacent* either in G_H , in G_V , or in both. Two blocks are adjacent if they are the end vertices of some edge (i.e., constraint). It is clear that a floorplan that satisfies a (strongly) complete set will have no overlaps. Two blocks are *overconstrained* if they are constrained in

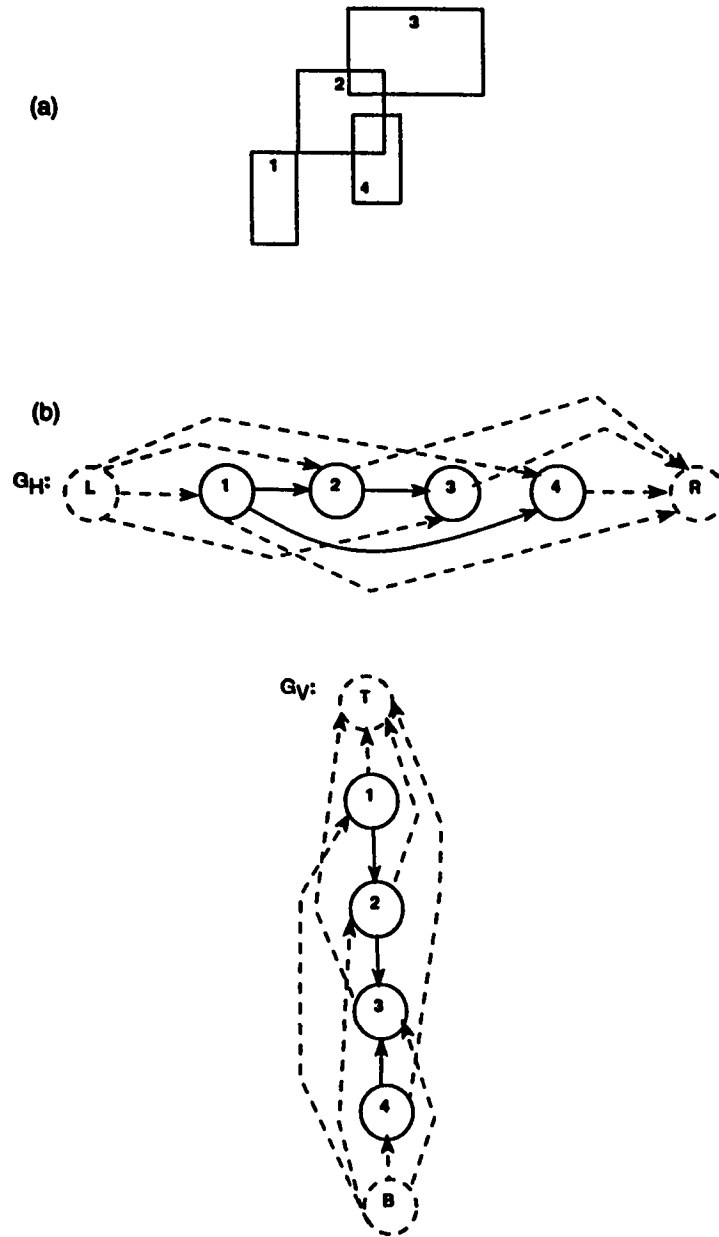


Figure 4.8: (a) Illegal floorplan; (b) constraint graphs: G_H, G_V .

both the horizontal and vertical directions; trying to produce a floorplan satisfying both the constraints will negatively affect the area optimality of the floorplan. A timing-optimized topological arrangement usually requires the satisfaction of one constraint; the other constraint is redundant.

A key observation to our approach is that for two blocks not to overlap, only one constraint (either in the horizontal or the vertical direction) is necessary and sufficient. A key question then is which of the constraints to keep, the horizontal or the vertical constraint? Before we describe our approach for constructing a constraint set (G_H, G_V) , we introduce the following definition.

Definition 1 *A constraint set (G_H, G_V) is sufficiently constrained if there exists an edge between every pair of blocks (b_i, b_j) either in G_H or G_V .*

Clearly, a *sufficiently constrained* set (G_H, G_V) is a strongly complete set. The approach in [VIJ91] starts with an overconstrained set, i.e., a set with many overconstrained blocks. This set is then reduced to a *sufficiently constrained* set by removing redundant constraints from only the longest paths in G_H and G_V . A problem with this approach is that the size of the overconstrained set could be very large, and hence may require large computing resources (i.e., memory and time).

We believe that a better approach would be to build directly a *sufficiently con-*

strained set using a constructive (greedy) procedure. If two blocks are overconstrained (i.e., horizontally and vertically), the constructive procedure will select either to constrain the two blocks in the horizontal direction or in the vertical direction. The selection is based on which of the constraints will lead to a smaller-area floorplan. If two blocks are constrained in only one direction (i.e., they have the same x or y coordinate), the algorithm in this case has only one choice. This process greedily generates a constraint set (i.e., G_H, G_V) according to Definition 1 and, at the same time, eliminates all redundant constraints right from the beginning. This is different from the algorithm given in [VIJ91], where only redundant edges belonging to the critical paths in G_H and G_V are examined. Removing all redundant constraints produces a more compact floorplan and as our experiments demonstrated does not create timing violations. The algorithm for generating a *sufficiently constrained* set is presented in Figure 4.9. The algorithm examines each pair of blocks and inserts topological constraints in G_H and G_V according to their centers. If two blocks i, j are constrained in the horizontal and vertical directions, i.e., $x_i \neq x_j$ and $y_i \neq y_j$ (CASE 1 in Figure 4.9), the algorithm inserts an edge (i, j) in the horizontal graph, G_H , and similarly an edge (i, j) in the vertical graph, G_V . Next, the algorithm enumerates the longest path $\ell_H(i, j)$ ($\ell_V(i, j)$) that goes through the inserted edge (i, j) in G_H (G_V). The edge that yields the shorter path will be retained and the other will be removed (i.e., IF statement in Figure 4.9). The other case (CASE 2 in Figure 4.9) is that the blocks i and j are constrained in only one direction

ALGORITHM *Sufficient_Constraint***NOTATION*****B***: Topological arrangement; $\ell_H(i, j)$: length of longest path traversing edge (i, j) in G_H ; $\ell_V(i, j)$: length of longest path traversing edge (i, j) in G_V ;**BEGIN****FOREACH** pair of blocks (i, j) in B **DO****BEGIN****CASE 1:** i, j are overconstrained ($x_i \neq x_j$ and $y_i \neq y_j$) Insert edge (i, j) in G_H and G_V ; Compute $\ell_H(i, j), \ell_V(i, j)$; **IF** $(\ell_H(i, j) < \ell_V(i, j))$ **THEN** Remove edge (i, j) in G_V ; **ELSE** Remove edge (i, j) in G_H ; **ENDIF****CASE 2:** i, j are constrained in only one direction i is to the left of j ($x_i < x_j$): insert (i, j) in G_H ; i is below j ($y_i < y_j$): insert (i, j) in G_V ;**ENDFOREACH****END.**Figure 4.9: *Sufficient_Constraint* algorithm.

(either to the left or below, i.e., $x_i = x_j$ or $y_i = y_j$). In this case the algorithm has only one choice and therefore the enumeration of longest paths is not required; the algorithm inserts the appropriate edge in the corresponding graph. We next give a simple example to illustrate the above algorithm.

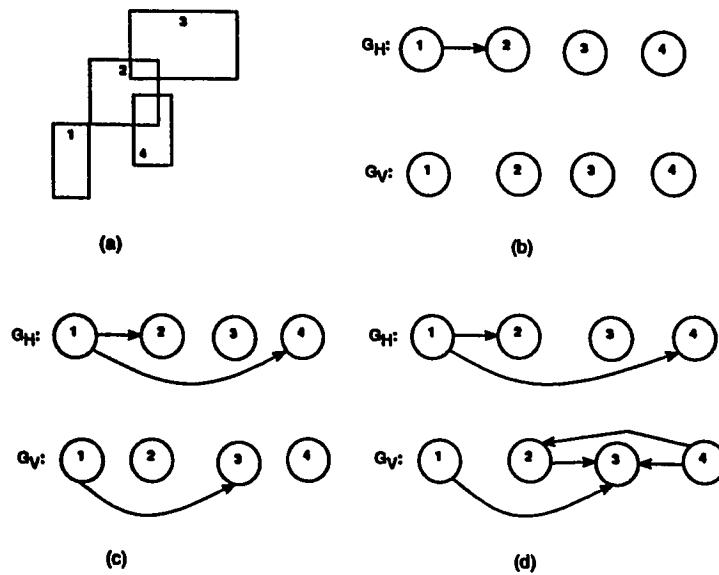


Figure 4.10: (a) Overlapping floorplan; (b) edge (1,2) retained; (c) edges (1,4) and (1,3) retained; (d) sufficiently constrained set: G_H , G_V .

Block	Height	Width
1	2	1
2	2	2
3	1	3
4	2	1

Table 4.2: Dimensions for blocks in Example 4.3.

Example 4.3 Referring to the illegal floorplan of Figure 4.10(a), the horizontal constraints are (1,2), (1,4), (1,3), (2,3), and (2,4). The vertical constraints are (1,2), (1,3), (2,3), (4,2), and (4,3). Assume that the dimensions of these blocks are as given in Table 4.2. Let $\ell_H(i,j)$, $\ell_V(i,j)$ be the length of the longest paths traversing the newly created edge (i,j) in G_H , G_V respectively. The *length* of a path is the sum of the widths (heights) of the blocks along that path in G_H (G_V). For the edge (1,2), the algorithm computes $\ell_H(1,2)$ and $\ell_V(1,2)$ in the partially constructed graphs G_H and G_V . $\ell_H(1,2)$ is 3, while $\ell_V(1,2)$ is 4; therefore edge (1,2) is kept in G_H and removed from G_V . The result of this step is shown in Figure 4.10(b). The contents of G_H and G_V after processing edges (1,4) and (1,3) are shown in Figure 4.10(c). Table 4.3 summarizes the steps in constructing a sufficiently constrained set (Figure 4.10(d)) for the floorplan of Figure 4.10(a). A “-” in the $\ell_H(i,j)$, $\ell_V(i,j)$ columns in Table 4.3 indicates that the two blocks are constrained in only one direction and hence the algorithm inserts the edge in the corresponding graph without computing the longest paths traversing the new edge because it is the only choice. A possible floorplan satisfying the constraint set is shown in Figure 4.11.

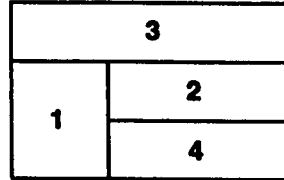


Figure 4.11: A possible floorplan for Example 4.3.

Constraint	$\ell_H(i, j)$	$\ell_V(i, j)$	Retained by
1,2	3	4	G_H
1,4	-	-	G_H
1,3	4	3	G_V
2,3	5	3	G_V
4,2	-	-	G_V
4,3	-	-	G_V

Table 4.3: Results for Example 4.3.

Time Complexity of Constraint Graph Construction

Checking all possible block pairs requires $O(K^2)$ for K blocks. This is the “FORE-ACH” loop in Figure 4.9. For the computation of longest paths, we implemented a greedy algorithm proposed by Youssef [YOU90]. If the constraint graphs are leveled, this algorithm exhibits an $O(K)$ time complexity. The number of calls to the *longest path* procedure depends on the number of over-constrained blocks.

Block Reshaping

The next step in our floorplanner consists of resizing the variable-shape blocks in order to optimize the floorplan area and satisfy the remaining constraints on block/chip aspect ratios. Resizing is done such that the topological constraints as stated by the constraint set are preserved. A two-dimensional block resizer algorithm has been implemented. The resizer determines dimensions and positions of the blocks so that floorplan area is minimized and constraints on block shapes are satisfied. This is achieved by iteratively reshaping flexible blocks on the critical paths.

In this work, we adopt the *general cell* design style. In this design style, block dimensions are considered as continuous variables. Thus, if block b_i has width w_i and area a_i , then its height h_i , is given by a_i/w_i .

The *resizing* algorithm utilizes the constraint graphs G_H and G_V developed in the previous section to compute the dimensions of the floorplan and decide on a candidate block for resizing.

Suppose we want to reduce the floorplan dimension in the Y -direction without enlarging the floorplan in the X -direction. This can be achieved as follows: let $\ell(\pi_H)$ and $\ell(\pi_V)$ be the length of the *critical* paths in G_H and G_V respectively. Let block b_i be such that $b_i \in \pi_V$ and $b_i \notin \pi_H$. If π_H^i is the longest path traversing b_i in G_H , then

the width, w_i of b_i can be increased by an amount $\delta_i^x = \ell(\pi_H) - \ell(\pi_H^i) - \epsilon$ without increasing the overall area of the floorplan. Hence, δ_i^x is the maximum block's width increment that is guaranteed not to cause an increase in the length of the critical path, π_H , in G_H . The constant ϵ is used to avoid wasting CPU time in resizing iterations which will result in negligible area reductions, and to avoid oscillation, i.e., the same block keeps getting selected in G_H then in G_V and so on. But, usually the block width has an upper bound w_i^{max} , therefore the legal δ_i^x is given by,

$$\delta_i^{x_{legal}} = \min(\delta_i^x, w_i^{max} - w_i) \quad (4.13)$$

Thus, the new dimensions w'_i, h'_i for block b_i are derived as follows,

$$w'_i = w_i + c \times \delta_i^{x_{legal}} \quad (4.14)$$

$$h'_i = a_i / w'_i \quad (4.15)$$

where c is a positive real number ($c \leq 1$) specified by the user to control how large the x -*increment* should be. We have observed that resizing the blocks in small increments ($c \leq 0.5$) helps achieve a smaller floorplan with the correct aspect ratio. In our experiments, we set this parameter to 0.5. Optimization in the X -direction is similarly formulated.

If the constraint graphs G_H and G_V do not contain multiple critical paths, then the reshaping of block b_i according to Equations (4.14-4.15) will reduce the floorplan area. If multiple critical paths exist then the floorplan area will remain unchanged.

This observation is stated in the next theorem.

Theorem 1 *The resizing process will not increase the area of the floorplan.*

Proof:

Let b_i be the block that has been resized. To show that the resizing of b_i does not increase the floorplan area, consider two cases:

Case 1: b_i belongs to the critical path of G_H and G_V

Therefore,

$$\delta_i^x = \delta_i^{x_{\text{total}}} = 0,$$

Thus,

$$w'_i = w_i,$$

$$h'_i = h_i,$$

Hence, there will be no increase in floorplan area.

Case 2: b_i is on the only critical path in G_V but not in G_H

Therefore,

$$\delta_i^{x_{\text{total}}} > 0,$$

Therefore,

$$h'_i < h_i,$$

Hence, the length of the critical path in G_V will decrease and therefore, the overall area will decrease. This case can be generalized to the case when b_i belongs to multiple critical paths in G_V . \square

The resizing process continues as long as there are candidate blocks for resizing. A key question is that will the resizing process converge?. The next theorem answers this question.

Theorem 2 *The resizing process is converging.*

Proof:

At each iteration, the area of the floorplan is reduced by a positive number. Obviously, the overall area cannot go to zero. Furthermore, a block which is selected for resizing in the current resizing direction will not get selected for resizing in the orthogonal direction (i.e., the block will not oscillate between the most critical paths in the horizontal and vertical directions); therefore, the resizing process must stop after a finite number of steps. \square

Time Complexity

The computation of the longest path can be achieved in $O(K)$ (assuming that G_H and G_V are levelized). Thus, resizing one block requires $O(K)$ where K is the number of blocks.

After completing the resizing process, the next task is the positioning of the blocks on the layout surface. The lower left corner of the floorplan is the origin at $(0,0)$. The lower left corner of block b_i is placed at (x_i, y_i) where x_i is the longest L - b_i path in G_H and y_i is the longest B - b_i path in G_V .

Finally, the blocks are enclosed inside a bounding rectangle with the desired aspect ratio ρ . The height H of the bounding rectangle is the longest B - T path in G_V , and its width W is the longest L - R path in G_H . If H/W is equal to the correct aspect ratio, ρ , then H and W are the required dimensions of the bounding rectangle; otherwise the blocks are enclosed inside the smallest bounding rectangle with an aspect ratio ρ . The area of the bounding rectangle is the area of the floorplan.

The final step is to verify the timing aspects of the resulting floorplan. The next section addresses this issue.

4.5 Timing Verification

The timing characteristics of an IC design are path oriented rather than net oriented. Thus, the path delay constraints ought to be satisfied; otherwise the design will have long path problems. On the other hand, the delay bounds on nets need not all be satisfied. In fact, satisfying all net bounds is a very strong constraint, and as our experiments demonstrated, the satisfaction of all net constraints becomes harder as the circuit gets denser. A path is safe if its total interconnect delay is less than its slack value. Recall from Chapter 3 that the interconnect delay is a function of its length. For net length estimation we assume a generalized pin located at the center of each block². We estimate the wire length, *net.Length*, of net n_j with m pins located at $(x_1, y_1), \dots, (x_i, y_i), \dots, (x_m, y_m)$ using the following procedure,

1. Let (x_c, y_c) be the net center point,

$$x_c = \frac{\sum_{i=1}^m x_i}{m}, y_c = \frac{\sum_{i=1}^m y_i}{m}$$

2. Let $(x_1, y_1), (x_2, y_2)$ be the lower left and the top right corners of the smallest bounding rectangle enclosing all the pins of the net

$$\ell_0 = \max\{x_2 - x_1, y_2 - y_1\}$$

²A more realistic assumption is to consider a pin on each side of the block.

3. IF $(x_2 - x_1 \geq y_2 - y_1)$ THEN

$$\ell_1 = \sum_{i=1}^m (y_c - y_i)$$

ELSE

$$\ell_1 = \sum_{i=1}^m (x_c - x_i)$$

4. $net_length = \ell_0 + \ell_1$

The above procedure constructs an approximate rectilinear *Steiner tree* for an m -pin net (Figure 4.12). For 2 and 3 pin nets, this procedure is an exact approximation. Our experiments showed that, for nets with more than 4 pins, this metric overestimates the wire length by less than 10%. A major advantage of this metric is that it never underestimates the wire length. This makes it more suitable for dense designs than the *half-perimeter* metric. The estimated net lengths³ are transformed into net delays. These net delays represent the actual interconnect delays. Next, the delays of all the interconnects along each critical path are summed and checked against its slack.

³In actual implementation, *netlength* has two components: horizontal (METAL-1) and vertical (METAL-2).

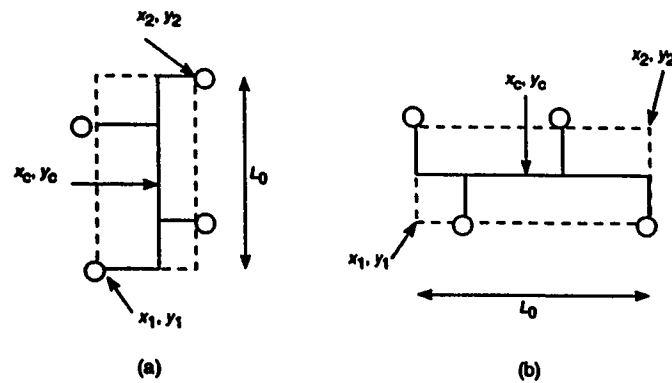


Figure 4.12: Approximate Steiner tree. (a) $L_0 = y_2 - y_1$; (b) $L_0 = x_2 - x_1$.

4.6 Discussion

In this section, we discuss advantages and disadvantages related to our floorplanning methodology. The floorplanning problem was treated as a constrained optimization problem. Two types of constraints are considered: timing constraints and geometrical constraints. During the *Cluster.Growth* step, blocks are greedily added to the partial solution based on their timing delays and connectivity. A block is selected by the maximization of a gain function G , which combines both of these criteria. Including *connectivity* in the selection minimizes connection length and improves timing on the floorplan. Also, considering connectivity information during block selection as well as when computing its target location avoids making the floorplanning process totally biased toward timing.

A major advantage of our approach is that it is not restricted to *slicing* structures.

Moreover, the execution time of the algorithm is very small. Our system was able to generate a legal floorplan for a 125-block circuit in less than 3 minutes on a 33 MHz 80386 IBM PC. Figure 4.13 shows the effect of the problem size on the execution time of our floorplanning algorithm. The graph indicates that execution time grows linearly with the number of blocks in the design. This makes the algorithm a good tool for generating good initial solutions for iterative improvement algorithms such as Genetic algorithms, Simulated Annealing, etc.

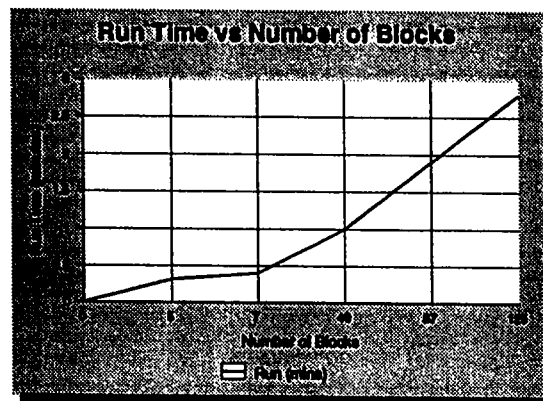


Figure 4.13: Growth in execution time.

The constructive floorplanning (*Cluster_Growth*) is a greedy algorithm. The decision made at each step is based on the partial solution (i.e., local information). To improve this point, the constructive procedure can be followed by a force relaxation procedure which will iteratively improve the global aspects of the solution.

An advantage of the floorplan sizing step is that it optimizes floorplan area while respecting all the decisions of the timing-sensitive force directed step.

At present our floorplanner does not consider routability of the floorplan. Our motivation for ignoring routability is because routability is usually not a problem for the general cell design style. This is in contrast to gate array design style where routing resources are very critical. We should mention that routability can be easily incorporated into our system. We need to integrate a global router with our system. The task of the global router is to compute routing space requirements between blocks. These space requirements can be specified as edge weights in the constraint graphs G_H and G_V . Then, the algorithm will use the edge weights (as well as node weights) to compute the correct locations of all the blocks as well as the final dimensions of the floorplan.

4.7 Experimental Results

The floorplanning approach described in this chapter has been implemented in the C language. Experiments were run on a 33 MHz 80386 IBM PC. We experimented with five test cases. The first test case is a simple 7-block example. The other two test cases are the *Fractional Multiplier* and the *traffic controller* described in the experimental section of Chapter 3. The *fractional multiplier* has 24 I/O pads and 125 cells and running at 41 ns clock period. The *Traffic Controller* has 11 I/O pads and 45 cells and running at 20 ns clock period. These test cases are implemented in

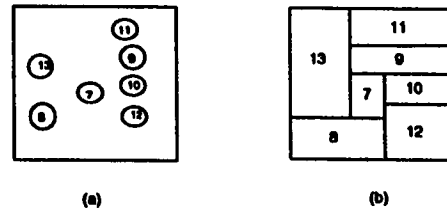


Figure 4.14: A 7-block example: (a) topological assignment; (b) after sizing.

standard cell design using a $2\mu p$ -well SCMOS technology [MCN90]. For the sake of experimentation, we treated the cells as general cells (i.e., flexible dimensions).

The first test case is used to demonstrate the quality of solution produced by the floorplan sizing step. Figure 4.14(a) shows the floorplan as obtained from the timing-sensitive force directed step. We specified an overall aspect ratio, ρ , of 1.00. Figure 4.14(b) shows the final floorplan which has minimum area.

For the computation of the *target* locations in the *Place_Block* procedure, we set the parameters α_1 and α_2 to 0.5. For the parameter β in the selection function G_i , we experimented with values 0, 1, and 4.

Table 4.4 summarizes the results for the traffic controller. When timing is included in the selection function (i.e., $\beta > 0$), the average connection length has decreased by 7% to 20% (row “*avg. net_len*” in Table 4.4). Also, the average remaining path slack (row “*avg. slack*” in Table 4.4) has improved by 40%. For this circuit, the timing analyzer reported 65 critical paths. The longest path has a slack

of 2.179 ns, while the 65th longest path had a slack of 13.358 ns. All these paths remained safe when floorplanning was guided by timing (row “*satisfied paths%*” in Table 4.4). On the other hand, even with timing driven floorplanning we could not get 100% satisfaction for net delay bounds. Achieving less than 100% net satisfaction does not mean that the design will suffer from timing problems. The reason behind this is that usually the loss on one net is accompanied by a gain on other nets. The total block area for this circuit is $88624 \lambda^2$. The floorplan area after resizing is shown in row “*chip area*” in Table 4.4. The total execution time for this example was 1:00 min. Figure 4.15 shows the floorplan of the *Traffic Controller*.

β	0	1	4
<i>avg. net_len</i>	270	251	215
<i>avg. slack</i>	5.7	8.05	8.15
<i>satisfied nets%</i>	63	89	91
<i>satisfied paths%</i>	90	100	100
<i>chip area</i>	98500	98076	97465

Table 4.4: Data from the *Traffic Controller*.

Table 4.5 summarizes the results for the *Fractional Multiplier*. When timing is included in the selection function (i.e., $\beta > 0$), the average connection length has decreased by 31% to 33% (row “*avg. net_len*” in Table 4.5). The percentage of satisfied net timing bounds has increased from 87% for $\beta = 0$ to 97.6% when β was set to 1, and to 98% when β was set to 4. Also, 100% critical path satisfaction

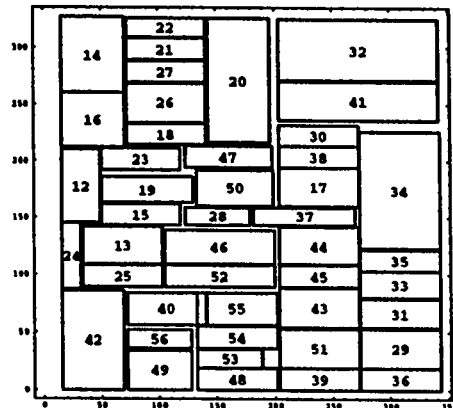


Figure 4.15: A floorplan of the *Traffic Controller*.

was possible when β was set to 4, and the average remaining slack was 18.5 ns (row “*avg. slack*” in Table 4.5). For this circuit, the timing analyzer reported 100 critical paths. The longest path has a slack of 10.236 ns, while the 100th longest path had a slack of 33.786 ns. All these paths remained safe after floorplanning (row “*satisfied paths%*” when $\beta = 4$ in Table 4.5). The total block area for this circuit is 225504 λ^2 . The floorplan area after resizing is shown in row “*chip area*” in Table 4.5. We should mention that the *timing-driven genetic placer* in [NAS94] was able to place the *Fractional Multiplier* with a clock period of 38 ns. On the other hand our floorplanner required 3 ns more to generate a timing-problems free floorplan (i.e., 41 ns clock period) in less than 3.0 min CPU time. This is due to two reasons:

- The timing-sensitive force directed step is a one pass constructive heuristic.

Further improvements may be achieved, if a force relaxation phase is used

β	0	1	4
<i>avg. net_len</i>	459	316	306
<i>avg. slack</i>	16.16	18.1	18.5
<i>satisfied nets%</i>	87	97.6	98
<i>satisfied paths%</i>	93	98	100
<i>chip area</i>	282997	265126	263908

Table 4.5: Data from the *Fractional Multiplier*.

chip	Description	IOPads	Cells	CLOCK	Critical Paths
adder	simple adder	5	11	10 ns	13
par1	16-bit parity	17	20	48 ns	16
par2	8-bit parity	9	30	32 ns	100

Table 4.6: Test cases statistics.

after completing the constructive pass.

- The *Genetic placer* uses the *half-perimeter* method to estimate net length. This method is optimistic in wire length estimation, especially for nets with more than 4 pins. The fractional multiplier has several 5-pin and 6-pin nets. On the other hand, our system uses a more accurate wire length metric (see Section 4.5).

We experimented with three more circuits in VPR format (Table 4.6). Table 4.7 summarizes the results for the *adder* circuit. The longest path has a delay of 7.198 ns. The total block area for this circuit is 25985 λ^2 . The aspect ratio was specified as 1.00. With $\beta = 4$ the average remaining path slack has increased by 5%

β	0	1	4
<i>avg. net.len</i>	126.27	109.64	100.45
<i>avg. slack</i>	2.02	2.06	2.12
<i>satisfied nets%</i>	72.73	81.72	82
<i>satisfied paths%</i>	84.62	100	100
<i>chip area</i>	29972	29789	29683

Table 4.7: Data from the *adder circuit*.

compared to that of $\beta = 0$.

Table 4.8 and Table 4.9 summarize the results for the *16-bit parity* and *8-bit parity* circuits. The longest path delays are 23.049 ns and 36.834 ns respectively. The total block area are $40832\lambda^2$ and $39440\lambda^2$, respectively. The reader can observe that when the timing weight, β , was set to 4, the average connection length has increased from 35.55 ($\beta = 1$) to 60 ($\beta = 4$) (Table 4.8), and increased from 65.44 ($\beta = 1$) to 109.6 ($\beta = 4$) (Table 4.8). Consequently, the average remaining path slack and the percentage of satisfied net constraints have decreased. The reason is that, increasing the value of the timing weight β beyond the circuit's requirement causes some nets to become excessively long, and hence the circuit timing gets worse.

β	0	1	4
<i>avg. net_len</i>	58	35.55	60
<i>avg. slack</i>	16.04	17.33	15.9
<i>satisfied nets%</i>	83	92	91
<i>satisfied paths%</i>	91	100	100
<i>chip area</i>	44544	44095	44584

Table 4.8: Data from the *16-bit parity checker*.

β	0	1	4
<i>avg. net_len</i>	70.37	65.44	109.6
<i>avg. slack</i>	0.89	2.1	1.58
<i>satisfied nets%</i>	75	90	89.6
<i>satisfied paths%</i>	71.6	100	100
<i>chip area</i>	47026	46878	48203

Table 4.9: Data from the *8-bit parity checker*.

4.8 Conclusion

In this chapter we described a timing driven floorplanning program. Two types of timing data are used: a set of the α -critical paths and delay upper bounds on interconnects. The floorplanning algorithm has two major stages. The first stage constructs a timing driven topological arrangement using a constructive implementation of the force directed technique. Both timing and connectivity information are used in the derivation of block target locations. The second stage is floorplan sizing which converts the topological arrangement to a legal floorplan. This is achieved by mapping the topological arrangement into efficient topological constraint graphs. Then, a two dimensional resizer uses these graphs to reduce floorplan area and satisfy all geometrical constraints while satisfying the timing requirements and the shape and area constraints of the design. Experimental results are positive in terms of timing, area, and average connection length.

Chapter 5

Conclusion

Increase in switching speeds and decrease in feature size have made the performance of modern VLSI designs dependent on interconnect delay efficiency. It is no longer the case that the *clock* speed can be verified prior to physical design, and physical design itself should be governed with the timing requirements of the design.

To achieve its goal, a timing driven physical design tool requires two elements:

- necessary and accurate information about the temporal properties of the design, and
- a strategy to use this information during physical design.

In this thesis, we addressed both of these problems.

In Chapter 3, we addressed the issue of generating proper timing constraints prior to layout. We introduced the α -critical approach, a new prediction strategy for timing critical paths. The parameter α is interpreted as a confidence level, and is used along with the path's delay standard deviation to construct confidence intervals around the delay of the longest path. A path is critical if its total expected delay falls inside the α -critical interval. The number of critical paths increases with increasing values of α . The critical path data has been successfully used by two physical design tools: a *Genetic placer* for standard cell design style [NAS94], and a *Genetic floorplanner* for general cell design style [TAN94]. The critical paths enumerated after layout were a proper subset of the predicted paths.

We implemented *Minimax-PERT*, a procedure proposed in [YOU90]. This algorithm uses *minimax* ideas to derive maximal delay bounds for nets. The timing bounds are maximal in the sense that all slacks are distributed among the nets, thus maximizing degrees of freedom for physical design tools.

In Chapter 4, we demonstrated the usage of the generated timing data to drive the floorplanning step. The floorplanning algorithm has two major stages. The first stage constructs a timing driven topological arrangement using a force directed technique. Both timing and connectivity information are used in the derivation of

block target locations. The second stage is floorplan sizing which converts the topological arrangement to a legal floorplan. This is achieved by mapping the topological arrangement into topological constraint graphs. Then, these graphs are utilized by a resizing algorithm to reduce floorplan area and satisfy all geometric constraints without undoing any of the decisions of the timing-sensitive force directed step.

Next, we discuss some possible extensions to the work that has been done. At present, our floorplanning approach ignores routability. It needs to be integrated with a global router. The task of the global router will be to compute routing space requirements. These requirements can be interpreted as edge weights in the constraint graphs described in Chapter 4. The floorplan sizing step needs to be modified to consider the edge weights in the computation of block positions. The floorplanning approach described in this work performs the resizing process while maintaining the topological arrangement. A better approach would be to integrate the two steps to consider trade-offs between timing optimization and area optimization.

Bibliography

- [AS85] T. Asano and S. Sato. Long Path Enumeration Algorithms for Timing Verification on Large Digital Systems. *Graph Theory with Applications to Algorithms and Computer Sciences*, John Wiley, pp.25-35, 1985.
- [BRA90] D. R. Brasen and M. L. Bushnell. "MHERTZ:A New Optimization Algorithm for Floorplanning and Global Routing," *27th ACM/IEEE Design Automation Conference*, pp. 107-110, 1990.
- [BUR85] M. Burstein and M. Youssef. "Timing Influenced Layout Design," *Proceedings of the 22nd Design Automation Conference*, June 1985, pp. 124-130.
- [CD93] H. Chen and D. Du. "Path Sensitization in Critical Path Problem," *IEEE Transaction on CAD*, Vol. 12, No.2, Feb. 1993, pp. 197-207.
- [DAI87] W. Dai and E. S. Kuh. "Simultaneous Floorplanning and Global Routing for Hierarchical Building-Block Layout," *IEEE Transaction on CAD*, Vol.

CAD-6, No.5, Sep. 1987, pp. 828-837.

- [DON89] S-K. Dong, J. Cong, and C. L. Liu. "Constrained Floorplan Design for Flexible Blocks," in *Dig. Int. Conf. Computer-Aided Design*, Nov.1989, pp. 488-491.
- [DUN84] A. E. Dunlop et.al. "Chip Layout Optimization Using Critical Path Weighting," *Proc. of 21st Design Automation Conf.*, pp. 142-146, 1987.
- [FCW67] C. J. Fisk, D. L. Caskey, and L. E. West. "Automated Circuit Card Etching Layout," *Proc. IEEE*, November 1967.
- [FRA92] Jon Frankle. "Iterative and Adaptive Slack Allocation for Performance-Driven Layout and FPGA Routing," *29th ACM/IEEE Design Automation Conference*, pp. 536-541, 1992.
- [HAU87] P. S. Hauge, R. Nair, and E. J. Yoffa. "Circuit Placement for Predictable Performance," *Proc. of ICCAD*, pp. 34-37, 1987.
- [HIT82] Robert B. Hitchcock, Sr., Gordon L. Smith, and David D. Cheng. "Timing Analysis of Computer Hardware," *IBM J. Res. Develop.*, Vol. 26, No.1, pp. 100-116, 1982.
- [HOR78] Ellis Horowitz and Sartaj Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, Inc., Rockville, Maryland 20850, 1978.

- [HWA76] M. Hanan, P.K. Wolf, and B. J. Agule. "A Study of Placement Techniques," *Journal of Design Automation and Fault-Tolerant Computing*, pp. 28-61, October 1976.
- [JAC89] M. Jackson and E.S. Kuh. "Performance-Driven Placement of Cell Based IC's ," in *Proc. of 26th Design Automation Conf.*, pp. 370-375, 1989.
- [JG83] D. Jepsen and C. Gelatt. "Macro Placement by Montecarlo Annealing," *Proc. of ICCAD*. 3:495-498. 1983.
- [JOU87] Norman P. Jouppi. "Timing Analysis and Performance Improvement of MOS VLSI Design," *IEEE Transaction on CAD*, Vol. CAD-6, No.4, pp. 650-665, July 1987.
- [KAN83] S. Kang. "Linear Ordering and Application to Placement," in *Proc. of 20th Design Automation Conf.*, pp. 457-464, 1983.
- [KGV83] S. Kirkpatrick, C. Gelatt, and M. Vechi. "Optimization by Simulated Annealing," *Science* 220, 4598:671-680, May 1983.
- [KIC87] Bernhard Kick. "Timing Correction in Logic Synthesis," *Proc. of ICCAD*, pp. 299-302, 1987.
- [KIRK66] Kirkpatrick, T. I. and Clark, N. R. "PERT as an aid to logic design," *IBM J. Res. Develop.*, vol. 10, no.2, pp. 135-141, March 1966.

- [LAP86] D. P. La Potin, S. W. Director. "Mason: A Global Floorplanning Approach for VLSI Design," *IEEE Transaction on CAD*, Vol. CAD-5, No.4, October 1986, pp. 477-489.
- [LAI88] Y. Lai, and S. M. Leinwand. "Algorithms for Floorplan Design Via Rectangular Dualization," *IEEE Transaction on CAD*, Vol. 7, No.12, December 1988, pp. 1278-1289.
- [MCN90] MCNC Group. *OASIS 2.0 Reference Manual*, 1990.
- [MIC86] Alexander Miczo. *Digital Logic Testing and Simulation*. Harper & Row Publishers, New York, 1986.
- [MS86] M. Masud and Sadiq M. Sait. "Universal AHPL - A language for VLSI design automation," *IEEE Circuits and Devices Magazine*, 2:8-13, September 1986.
- [MSL89] M. Marek-Sadowska and S. P. Lin. "Timing Driven Placement," *Proc. of ICCAD*, pp. 94-97, 1989.
- [NAI89] Ravi Nair et al. "Generation of Performance Constraints for layout," in *IEEE Transaction on CAD*, Vol. CAD-8, No.8, August 1989, pp. 860-874.
- [NAS94] Khalid Nassar. *Timing Driven Placement Algorithm for Standard-Cell Design*. MS. Thesis, Dept. of COE, KFUPM, Dhahran, June 1994.

- [OG84] R. Otten and L. Van Ginneken. "Floorplan Design Using Simulated Annealing," *Proc. of ICCAD*. 3, 1984.
- [Orb92] Orbit Semiconductor Inc., Sunnyvale, California. *Foresight Manual*, July 1992.
- [PRE88] B. Preas and M. Lorenzetti. *Physical Design Automation of VLSI Systems*. The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA94025, 1988.
- [SM91] K. Shahookar and P. Mazumder. "VLSI Cell Placement Techniques," *ACM Computing Surveys*, Vol. 23, No. 2, June 1991, pp. 143-219.
- [SRI92] A. Srinivasan, K. Chaudhary, and E. S. Kuh. "Ritual: A Performance-Driven Placement Algorithm." *IEEE Transactions on Circuits and Systems - II*, 39(11):825-840, November 1992.
- [SUT90] S. Sutanthavibul, E. Shragowitz, and J. B. Rosen. "An Analytical Approach to Floorplan Design and Optimization," in *the 27th Design Automation Conference*, 1990, pp. 93-98.
- [SUT93] S. Sutanthavibul, E. Shragowitz, and Rung-Bin Lin. "An Adaptive Timing Driven Placement for High Performance VLSI's." *IEEE Transaction on CAD*, 12(10):1488-1498, October 1993.

- [SY94] Sadiq M. Sait and Habib Youssef. *VLSI Design Automation: Theory and Practice*. McGraw-Hill Book Co., Europe, 1995.
- [TAN94] Shahid, Tanvir. *Genetic Algorithm for Timing Influenced Floorplanning of VLSI Designs*. MS. Thesis, Dept. of COE, KFUPM, Dhahran, December 1994.
- [UED85] K. Ueda, H. Kitazawa, and I. Harada. "CHAMP: Chip Floorplan for Hierarchical VLSI Layout Design," *IEEE Transaction on CAD*, Vol. CAD-4, No.1, January 1985, pp. 12-22.
- [VIJ91] G. Vijayan and R. Tsay. "A New Method for Floor Planning Using Topological Constraint Reduction," *IEEE Transaction on CAD*, Vol. 10, No.12, December 1991, pp. 1494-1501.
- [WIM89] S. Wimer, I. Koren, and I. Cederbaum. "Optimal Aspect Ratios of Building Blocks in VLSI," *IEEE Transaction on CAD*, Vol. 8, No.2, February 1989, pp. 139-145.
- [WL86] D. F. Wong and C. L. Liu. "A New Algorithm for Floorplan Design," *Proc. of the 23rd DAC*, pp. 101-107, 1986.
- [WT89] D. F. Wong and Khe-Sing The. "An Algorithm for Hierarchical Floorplan Design." *Proc. of the ICCAD*, pp. 484-487, 1989.

- [YEN89] H. C. Yen, S. Ghanta, and H. C. Du. "Efficient Algorithms for Extracting the K Most Critical Paths in Timing Analysis," *26th ACM/IEEE Design Automation Conference*, pp. 649-654, 1989.
- [YIN89] C. Ying, J. S. Wong. "An Analytical Approach to Floorplanning for Hierarchical Building Blocks Layout," *IEEE Transaction on CAD*, Vol.8, No.4, April 1989, pp. 403-412.
- [YOU90] Habib Youssef. *Timing Analysis of Cell Based VLSI Designs*. Computer and Information Sciences, University of Minnesota, Ph.D. Thesis, January 1990.
- [YOU92] H. Youssef, Rung-Bin Lin, and E. Shragowitz. "Bounds on Net Delays for VLSI Circuits." *IEEE Transactions on Circuits and Systems - II*, 39(11):815-824, November 1992.

Vitae

- **Khalid Jawdat Kamel Al-Farra**
- **Born in 1967 at Taif, Saudi Arabia.**
- **Received Bachelor of Science degree in Computer Engineering from the King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia in January 1990.**
- **Completed Master's degree requirements at KFUPM, Dhahran, Saudi Arabia in June,1995.**