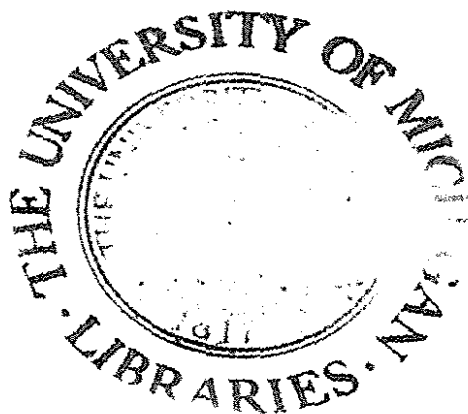


AAEL
TK
7871.99
.M44
M331
1992



LAYOUT MINIMIZATION OF CMOS CELLS

THE KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE

VLSI, COMPUTER ARCHITECTURE AND DIGITAL SIGNAL PROCESSING

Consulting Editor
Jonathan Allen

Latest Titles

- Iterative Identification and Restoration of Images*, R. L. Lagendijk, J. Biemond
ISBN: 0-7923-9097-0
- VLSI Design of Neural Networks*, U. Ramacher, U. Ruckert
ISBN: 0-7923-9127-6
- Synchronization Design for Digital Systems*, T. H. Meng
ISBN: 0-7923-9128-4
- Hardware Annealing in Analog VLSI Neurocomputing*, B. W. Lee, B. J. Sheu
ISBN: 0-7923-9132-2
- Neural Networks and Speech Processing*, D. P. Morgan, C. L. Scofield
ISBN: 0-7923-9144-6
- Silicon-on-Insulator Technology: Materials to VLSI*, J. P. Colinge
ISBN: 0-7923-9150-0
- Microwave Semiconductor Devices*, S. Yngvesson
ISBN: 0-7923-9156-X
- A Survey of High-Level Synthesis Systems*, R. A. Walker, R. Camposano
ISBN: 0-7923-9158-6
- Symbolic Analysis for Automated Design of Analog Integrated Circuits*,
G. Gielen, W. Sansen,
ISBN: 0-7923-9161-6
- High-Level VLSI Synthesis*, R. Camposano, W. Wolf,
ISBN: 0-7923-9159-4
- Integrating Functional and Temporal Domains in Logic Design: The False Path
Problem and its Implications*, P. C. McGeer, R. K. Brayton,
ISBN: 0-7923-9163-2
- Neural Models and Algorithms for Digital Testing*, S. T. Chakradhar,
V. D. Agrawal, M. L. Bushnell,
ISBN: 0-7923-9165-9
- Monte Carlo Device Simulation: Full Band and Beyond*, Karl Hess, editor
ISBN: 0-7923-9172-1
- The Design of Communicating Systems: A System Engineering Approach*,
C. J. Koomen
ISBN: 0-7923-9203-5
- Parallel Algorithms and Architectures for DSP Applications*,
M. A. Bayoumi, editor
ISBN: 0-7923-9209-4
- Digital Speech Processing: Speech Coding, Synthesis and Recognition*,
A. Nejat Ince, editor
ISBN: 0-7923-9220-5
- Assessing Fault Model and Test Quality*, K. M. Butler and M. R. Mercer
ISBN: 0-7923-9222-1
- Optimal VLSI Architectural Synthesis: Area, Performance and Testability*
C. H. Gebotys and M. I. Elmasry
ISBN: 0-7923-9223-X

LAYOUT MINIMIZATION OF CMOS CELLS

by

Robert L. Maziasz
and
John P. Hayes

The University of Michigan



Kluwer Academic Publishers
Boston/Dordrecht/London

engn

TK

7871.99

.M44

M331

1992

Distributors for North America:

Kluwer Academic Publishers

101 Philip Drive

Assinippi Park

Norwell, Massachusetts 02061 USA

Distributors for all other countries:

Kluwer Academic Publishers Group

Distribution Centre

Post Office Box 322

3300 AH Dordrecht, THE NETHERLANDS

Library of Congress Cataloging-in-Publication Data

Maziasz, Robert L., 1952-

Layout minimization of CMOS cells / by Robert L. Maziasz and John P. Hayes.

p. cm. -- (The Kluwer international series in engineering and computer science : SECS 160)

Includes bibliographical references and index.

ISBN 0-7923-9182-9 (alk. paper)

1. Metal oxide semiconductors, Complementary--Design and construction--Data processing. 2. Computer-aided design.

I. Hayes, John P. (John Patrick), 1944- . II. Title.

III. Series.

TK7871.99.M44M33 1992

621.381'52--dc20

91-34031

CIP

Copyright 1992 by Kluwer Academic Publishers

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, mechanical, photo-copying, recording, or otherwise, without the prior written permission of the publisher, Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061.

Printed on acid-free paper.

Printed in the United States of America

Enqin
728-1055
Enqin
4.10.92

To our wives

TABLE OF CONTENTS

PREFACE	xi
CHAPTER	
I. INTRODUCTION	1
1.1 Problem and Motivation	1
1.2 Layout Styles	6
1.2.1 Semi-Custom Layouts	6
1.2.2 Unstructured Methods	9
1.2.3 Programmable Logic Arrays	10
1.2.4 Gate Matrix Layouts	12
1.2.5 Functional Cells	13
1.3 Functional Cell Optimization	17
1.3.1 Layout Problem	17
1.3.2 Prior Work	21
1.4 Proposed Approach	22
1.4.1 Philosophy	22
1.4.2 Outline of the Book	24
II. FUNCTIONAL CELL LAYOUT METHODS	27
2.1 Functional Cell Design	27

2.2	Survey of Prior Methods	33
2.2.1	Static Cells	33
2.2.2	Dynamic Cells	43
2.3	Critique of Prior Work	44
III.	SERIES-PARALLEL CELL WIDTH MINIMIZATION	49
3.1	Graph Optimization Problems	50
3.2	Theory of Dual Trail Covering	55
3.3	Optimal Trail Covering without Reordering	63
3.4	Optimal Trail Covering with Reordering	67
3.5	Analysis of Complete Class of Practical Cells	68
3.6	Minimum-Width Rows of Cells	74
IV.	PLANAR CELL WIDTH MINIMIZATION	77
4.1	Nonseries-Parallel Composition	78
4.1.1	Graph Composition	79
4.1.2	Trail Covering	91
4.2	P-TrailTrace Algorithm	93
4.2.1	Description.	93
4.2.2	Design Example	96
4.2.3	Optimality.	97
4.2.4	Time Complexity.	99
4.3	Complete Study of Practical Planar Cells.	99
V.	SINGLE CELL WIDTH AND HEIGHT MINIMIZATION	105
5.1	Layout Problem.	105
5.1.1	Constraints and Assumptions.	106
5.1.2	Comparison to Other Assumptions.	107
5.2	Extension of Series-Parallel Cell Theory.	109

5.3	HR-TrailTrace Algorithm.	111
5.3.1	Description	112
5.3.2	Design Example.	115
5.3.3	Optimality	120
5.3.4	Time Complexity	123
5.4	Complete Study of Practical Cells.	124
5.5	Planar Cell Layout.	132
VI.	CELL ARRAY WIDTH AND HEIGHT MINIMIZATION	133
6.1	Layout Problem	133
6.2	HRM-TrailTrace Algorithm	135
6.2.1	Description	135
6.2.2	Design Example	137
6.2.3	Optimality	142
6.2.4	Time Complexity	148
6.3	Experimental Results	149
6.3.1	Commercial Circuits	149
6.3.2	Published Layouts	151
VII.	CONCLUSIONS	153
7.1	Contributions	153
7.2	Practical Applications and Extensions	155
BIBLIOGRAPHY		157
INDEX		167

PREFACE

The layout of an integrated circuit (IC) is the process of assigning geometric shape, size and position to the components (transistors and connections) used in its fabrication. Since the number of components in modern ICs is enormous, computer-aided-design (CAD) programs are required to automate the difficult layout process. Prior CAD methods are inexact or limited in scope, and produce layouts whose area, and consequently manufacturing costs, are larger than necessary. This book addresses the problem of minimizing exactly the layout area of an important class of basic IC structures called CMOS cells.

First, we precisely define the possible goals in area minimization for such cells, namely width and height minimization, with allowance for area-reducing reordering of transistors. We reformulate the layout problem in terms of a graph model and develop new graph-theoretic concepts that completely characterize the fundamental area minimization problems for series-parallel and nonseries-parallel circuits. These concepts lead to practical algorithms that solve all the basic layout minimization problems exactly, both for a single cell and for a one-dimensional array of such cells. Although a few of these layout problems have been solved or partially solved previously, we present here the first complete solutions to all the problems of interest.

We have implemented our algorithms by means of computer programs, and demonstrate that they efficiently generate minimum-area layouts for all possible cells of practical size. An analysis of these layouts permits us to make the first comprehensive evaluation of the quality and scope of prior layout methods. We show that most previous layout optimization algorithms are limited in scope, and cannot find optimal layouts for a large percentage of practical circuits. We also compare our optimal layouts to those of nonoptimal heuristic methods, and demonstrate that our exact algorithms produce significantly smaller cells.

This book will be of interest to designers of circuits or CAD tools, and others concerned with generating highly compact layouts for ICs. Chapter 1 introduces the subject of IC layout, and surveys traditional layout styles. Chapter 2 gives a brief tutorial on the cell layout problem, and evaluates the effectiveness of prior cell layout

techniques at minimizing area. Chapter 3 develops our general theory of cell layout and presents algorithms for generating minimum-width cells for series-parallel circuits; Chapter 4 generalizes these results for the nonseries-parallel case. Chapter 5 presents an algorithm that generates minimum-width and -height cells, and these results are extended to an array of cells in Chapter 6. Chapter 7 summarizes the book and suggests applications and extensions to this work.

Most of the material in this book was developed over the past few years as part of the first author's Ph.D. dissertation research at the University of Michigan. This research was supported by grants from the Office of Naval Research and the National Science Foundation, and by fellowships from the Burroughs (now part of Unisys) and Schlumberger Corporations. We wish to express our gratitude to these organizations. We also thank William P. Birmingham, Richard B. Brown, Philip J. Hanlon and Ronald J. Lomax of the University of Michigan for their suggestions.

LAYOUT MINIMIZATION OF CMOS CELLS

CHAPTER I

INTRODUCTION

We begin by discussing the general design problem for integrated circuits, and the role of the cell layout problem in relation to it. Next, we survey the popular cell layout styles for integrated circuits. The cell layout area minimization problem we address is defined and prior work on it is evaluated. We conclude with a discussion of our approach to exact layout optimization.

1.1 PROBLEM AND MOTIVATION

Since the introduction of the first commercial integrated circuit (IC) in 1961, the capacity of digital integrated circuits has been increasing at a rapid pace. Commercial IC chips in the 1960's had about 100 transistors. Through dramatic improvements in fabrication technology, very large scale integrated (VLSI) circuits with over ten million transistors are being fabricated at present, and the number of transistors per chip tends to double every year or two. The design of chips with such huge numbers of transistors presents major problems in design effort and cost. Computer-aided design (CAD) tools were developed to help designers cope with these problems. Such tools now play a key role in managing design complexity and improving designer productivity.

The design of a VLSI chip is a complicated process requiring many intermediate steps. Typically, several hierarchical levels of representation are used in the design process; see Fig. 1.1. One of the higher levels is the *register transfer level*, where the variables and the data operators represent the hardware registers and functional blocks of the data-processing section of a chip. The control section implements the sequencing of operations implied by the register transfer description. Next comes the *logic level*, at which the functional blocks are basic logic units called *gates*, such as the AND and NOR gates shown in Fig. 1.1(b); these gates implement the carry function of the adder in Fig. 1.1(a). At the *transistor level*, the logic gates are decomposed into transistor circuits. The NOR gate of Fig. 1.1(b) is translated into the transistor circuit of Fig. 1.1(c), where a *transistor* is a three-terminal device with

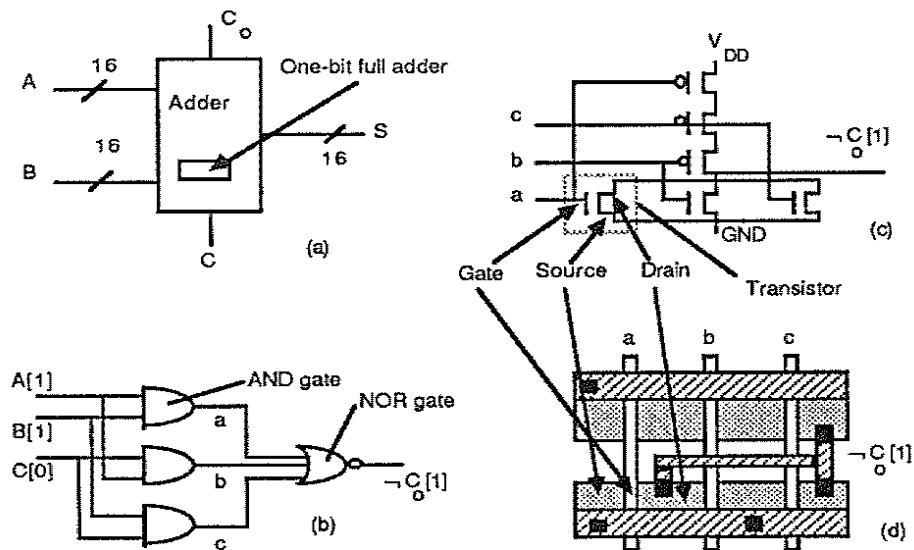


Fig. 1.1. Levels of design: (a) register transfer level (16-bit binary adder); (b) Logic level (carry circuit of a 1-bit adder); (c) transistor level (3-input NOR gate); (d) layout level (NOR gate in the CMOS functional cell style).

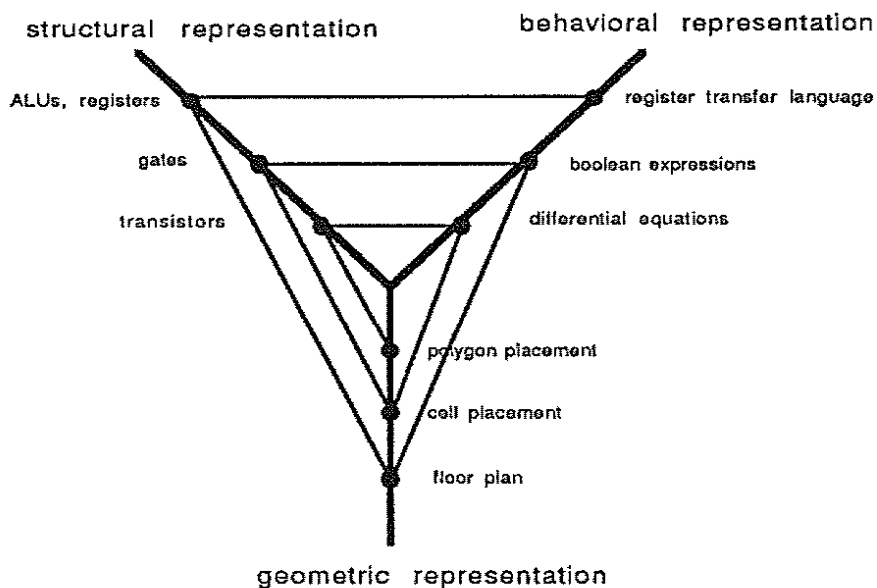


Fig. 1.2. Y chart representation of the integrated circuit design process [Ga82].

drain, source and gate terminals. The lowest design level is the *layout level*, where transistors are represented as geometric figures with dimensions such as length, width and position. The geometric representation or *layout* of a basic logic unit is called a *cell*. The layout of Fig. 1.1(d) is a cell implementing the NOR gate in Fig. 1.1(c). A large circuit may be laid out in the form of a set of interconnected cells. As in Fig. 1.1(d), the individual cells of the entire circuit are usually rectangular in shape. The layout design is transferred to mask-making facilities in this form for chip fabrication.

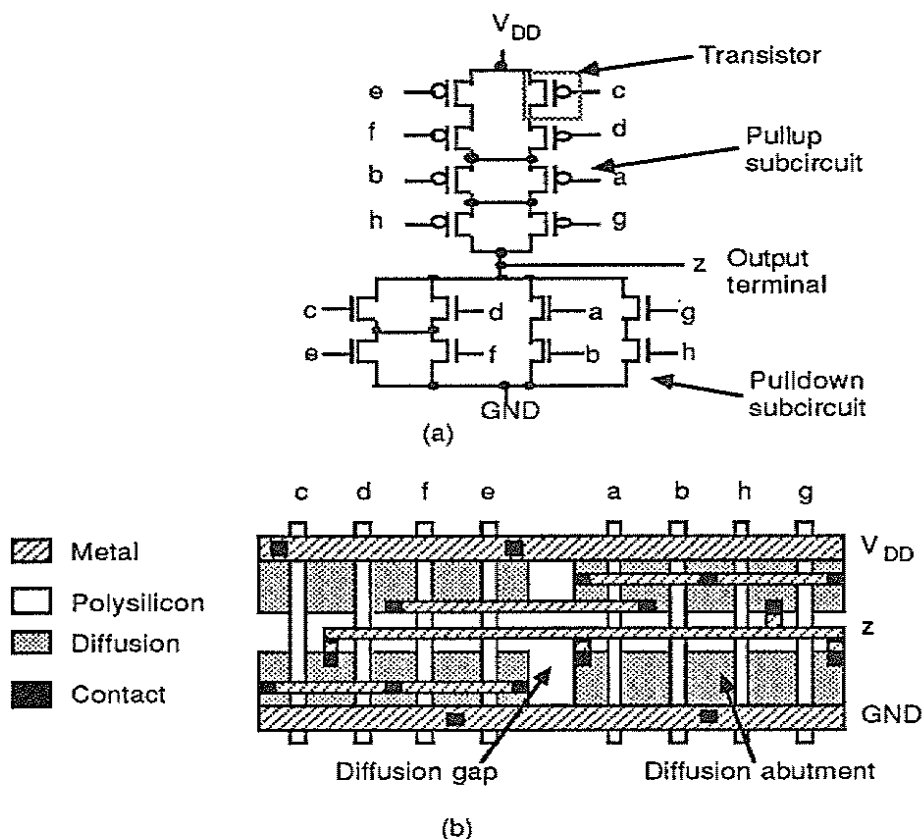


Fig. 1.3. A logic function implemented by a functional cell; (a) transistor circuit; (b) layout.

Each of the above design levels has behavioral, structural and geometric aspects. The Y chart in Fig. 1.2 illustrates these aspects, showing the order in which they are considered in a typical hierarchical, top-down design process [Ga85]. At the higher

levels, the behavioral aspects are more prominent, and the structural and geometric aspects are less important and less well defined. For example, at the register transfer level, the system is represented as registers and functional units such as an adder. These parts and their interconnection constitute the structure of the system. The behavior may be described by a register transfer language statement, such as

$$\text{register_A} := \text{register_B} + \text{register_C}.$$

Each of the functional units and interconnections at this level can be roughly sized, shaped, placed and oriented in what is called a floor plan [MC80]; see Fig. 1.2. At the transistor level, all structure is explicit; the parts are low-level design components, such as transistors and wires, and the logical behavior of each part is quite simple. At the layout level, the geometric figures composing each transistor and wire are assigned size and shape; each figure represents one of various materials used in fabricating an IC, such as diffusion, polysilicon and metal shown by different shadings in Fig. 1.3(b). Each of these materials occupies a different physical layer of the IC. At the layout level, structure and geometry are most prominent.

Each of the above design steps can be assisted or even be fully automated by CAD tools. Hardware description languages are needed at each level of representation to model the structure, behavior and geometry of the design as it is refined in top-down fashion. *Synthesis tools* are needed to translate the design from a higher level of representation, like the register transfer level, to a lower one, such as the layout level. *Analysis tools* are useful at each level of representation to analyze the quality of the design in terms of correctness and performance. Examples of CAD tools that aid IC designers are logic synthesizers, placement and routing programs, cell generators and simulators. A logic synthesizer can take a design described at the register transfer level and translate it into a set of logic (boolean) equations, accounting to some extent for speed, power and area requirements. These equations can be mapped directly into basic logic units such as AND, OR or NOT gates. A cell generator can produce a cell for each logic unit. The cells can be placed and routed on the chip by placement and routing tools. Simulation at the register transfer, gate and transistor levels can predict the performance of the design at each of these intermediate levels. The designer can use this information to decide whether the behavior of the design meets its specifications.

The *layout style* used in a chip design refers to the rules governing the transistor circuit structure (the circuit type), and the rules for mapping the circuit into the layout (the cell type), and has a profound effect on the amount of circuitry that fits on a chip of a given size. There are two major layout styles, which we discuss in detail in the next section. *Semi-custom* methods select manually designed cells from a cell library; these cells are well characterized in advance for speed and power. In

full-custom methods, on the other hand, a new type of cell is designed as needed for each logical unit. This can be done by human designers or by automatic cell generators, and requires simulation to analyze the performance of the cell.

This book is concerned with the translation of a design from the transistor circuit level to the layout level. Figure 1.3 shows the main concepts involved in this process. A transistor has three external terminals called the drain, source and gate; Figs. 1.1(c) and (d) show the correspondence between the transistor terminals and the layout. The conducting path between the drain and source terminals is open or closed depending upon the voltage level on the transistor's gate terminal; thus a transistor acts like an on-off switch. The transistor circuit of Fig. 1.3(a) is of a type called complementary MOS or CMOS, and uses two different types of transistors: p-channel MOS transistors in the pullup subcircuit and n-channel MOS transistors in the pulldown subcircuit. Each of these subcircuits has two designated terminals; the pullup subcircuit, whose designated terminals are the logical high voltage, referred to as V_{DD} or 1, and the output z ; and the pulldown subcircuit, whose designated terminals are the output z and the logical low voltage or ground, denoted as GND or 0. (The terminals to which the input signals $a, b, \dots, h$ are applied are not considered designated terminals.) The transistors composing the pullup and pulldown subcircuits create conducting paths connecting the output z either to V_{DD} or GND. Thus z implements a logical function that maps each combination of the input variables onto either 0 or 1.

An important layout task is to place and orient the transistors on the chip to minimize the layout area of a circuit. Figure 1.3(b) shows a cell implementing the transistor circuit of Fig. 1.3(a) in a very area-efficient layout style called a *functional cell* [Uv81]. The various types of shading indicate the materials used to fabricate the transistor terminals and the interconnections between these terminals. The drain and source terminals of a transistor are made from diffusion and the gate terminal is polysilicon. The pullup transistors of the circuit in Fig. 1.3(a) are placed side-by-side in the top diffusion row of the cell of Fig. 1.3(b) such that their drain and source terminals are adjacent, and the pulldown transistors are placed similarly in the lower row. Transistors in the two subcircuits sharing a common gate signal are placed in the same vertical column, which makes for easy routing of the signal between them using a polysilicon column. If the transistors are placed and oriented appropriately, the width of the cell can be reduced by arranging adjacent transistors to share the same drain/source terminal via diffusion abutment, as shown in Fig. 1.3(b). This obviates the need to have separate diffusion terminals which must be isolated by an area-wasting diffusion gap, as also illustrated in Fig. 1.3(b). A judicious placement and orientation of the transistors can result in fewer metal rows for interconnections, thus reducing cell height. Therefore, it is a basic goal of a layout generation tool to lay out the transistors of a circuit and their interconnections to minimize cell area.

We have chosen to address the topic of cell layout generation because it is so important in the overall chip design process. A significant reduction in the chip area required for a given circuit can be achieved by using full-custom layout, and a dramatic increase in productivity over hand layout can be realized by automatically generating these layouts. Layout generators exist for several full-custom layout styles; these styles are surveyed in the next section. Based on the level of recent research activity, the most promising of these styles is the functional cell. However, existing functional cell methods often produce layouts that have far from optimal area. This is due to the fact that most use heuristic or approximate approaches, rather than exact algorithms. An algorithmic layout methodology is needed that achieves minimal area in the functional cell layout style.

Therefore, the general problem being addressed in this book is that of generating minimum-area cell layouts of CMOS digital integrated circuits in the functional cell style. We present a complete solution to the problem of exact layout-area minimization under precisely specified conditions. We will demonstrate that exact area minimization of all practical-sized static CMOS circuits in the functional cell style is both computationally feasible and produces significantly smaller layouts than previous methods. We analyze the properties of the important class of dual planar complementary CMOS circuits, and present efficient algorithms for obtaining the optimal area layout of these circuits. These algorithms, which we have also programmed, are based on a complete theory of cell layout, which significantly extends prior work by Uehara and vanCleemput [Uv81] and others.

1.2 LAYOUT STYLES

Digital integrated circuits are often classified according to their layout styles, which are either semi-custom, such as gate arrays and standard cells, or full-custom. In this section we survey both semi-custom methods and the following full-custom styles: unstructured methods, programmable logic arrays, gate matrix methods, and functional cells.

1.2.1 Semi-Custom Layouts

Gate arrays constitute a popular approach to the design and layout of digital circuits [Ra80]. A conventional *gate array* consists of rows and columns of cells separated by routing areas for wires interconnecting the gates; see Fig. 1.4(a). These cells are groups of unconnected transistors that can be wired together to form various types of gates, such as AND (*), OR (+), NOT (—), NAND or NOR. Fig. 1.4(b) shows the structure of a CMOS cell, which has pMOS and nMOS transistors that are interconnected by metal wires (shown by the dotted lines) to form a two-input

NAND gate implementing the function $z = \neg(a * b)$. Typically, a design is represented at the logic level using a library of these basic functions and simulated for correctness and performance. Accurate simulation models can be based on the

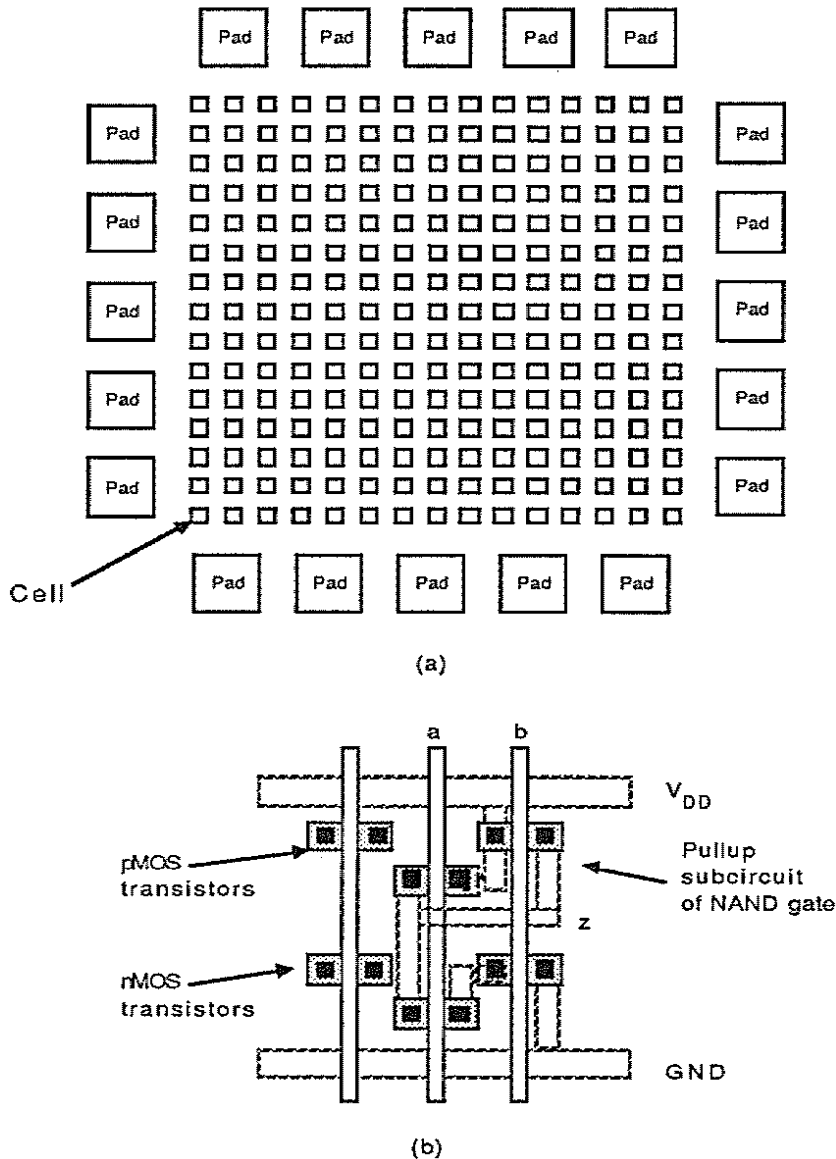


Fig. 1.4. A gate array circuit: (a) array of cells; (b) cell structure (after [RU87]).

electrical and performance characteristics of predesigned gates in the library. An alternative to a library of primitive cells is the automatic generation of complex logic circuits that are mapped onto the gate array as needed [Mi86]. Placement of the logic circuits onto the array and their interconnection can be performed automatically by CAD programs. One or more layers of metal are added to the gate array, as in Fig. 1.4(b), to form the gates within the cells and to interconnect these gates according to the particular design. A variant of the conventional gate array is the *sea-of-gates* which has no predetermined routing channels defining cells; gate interconnections are made with metal wires routed over the gates, resulting in more compact designs [Hu85].

Gate arrays can be designed more quickly and at lower cost than standard cells. However, gate arrays are not as small or as fast as standard cells or full-custom designs. They consume more power due to the larger capacitance associated with the typically longer interconnections between gates, and the cost per chip is higher than that of other methods.

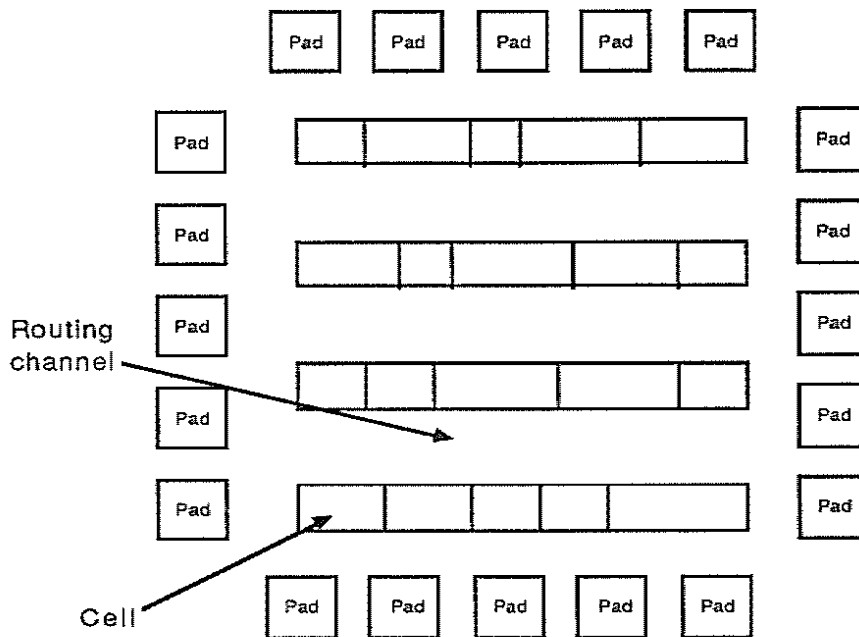


Fig. 1.5. A typical standard cell chip layout.

The layouts of basic functions, such as NAND and NOR gates, can be designed by specialists and stored in a library; each such layout is called a "standard" cell. A

system designer can then use these standard cells to implement a particular design; this is typically called the standard-cell method. Cells of similar complexity in the cell library typically have a fixed height; see Fig. 1.5. The cells vary in width, depending on their complexity, so they do not lead to the matrix structure characteristic of gate arrays. The inputs and outputs of the cells are on the top and/or bottom of the cells. The cells are placed in horizontal rows on the chip, so that a single horizontal pair of power and ground lines feeds all the cells in a row; Fig. 1.5 shows some typical rows of cells. There is a routing channel between each pair of cell rows which allows interconnection of the cells. Some interconnection is done through vertical spaces left between cells in a row and by connections in the cells. Inter-row connections also use vertical channels along the perimeter of the chip.

The advantages of the standard-cell approach are layouts that are considerably denser than gate arrays and offer higher performance. The greater density is due to the greater compactness of the basic cells and the variability of the routing channel width. This, in turn, results in lower cost per unit. The disadvantage is that the initial design cost is higher because more layers need to be customized than for gate arrays. Also the cost of designing and maintaining a cell library is high.

In addition to semi-custom styles, layout styles have been proposed which need neither a library of standard cells nor an array of configurable cells such as gate arrays. These full-custom styles allow the designer to arrange the transistors and wires in a relatively unrestricted fashion to form cells. There are two types of full-custom methods, which differ in the way they place transistors to construct a cell: unstructured methods and array methods. We survey examples of each type below.

1.2.2 Unstructured Methods

Unstructured layout methods allow the placement of transistors in any location within a cell. In addition, they allow the wires that interconnect the transistors to be routed in random patterns.

Planar Embedding is an algorithmic method of layout generation which produces nMOS circuit layouts in an unstructured style [NJ85]. Figure 1.6 shows a sample layout produced by this method. Note that the transistors are not aligned to any grid. Using a graph model of the transistor circuit, this method selects a layout that employs the fewest layer changes to interconnect the transistor terminals; in general, such layouts tend to use less area than those requiring more layer changes. However, since the number of layer changes is only indirectly related to layout area, the method does not minimize area and cannot measure how close to optimal its solutions are. The use of such indirect methods of minimizing area is typical of unstructured methods.

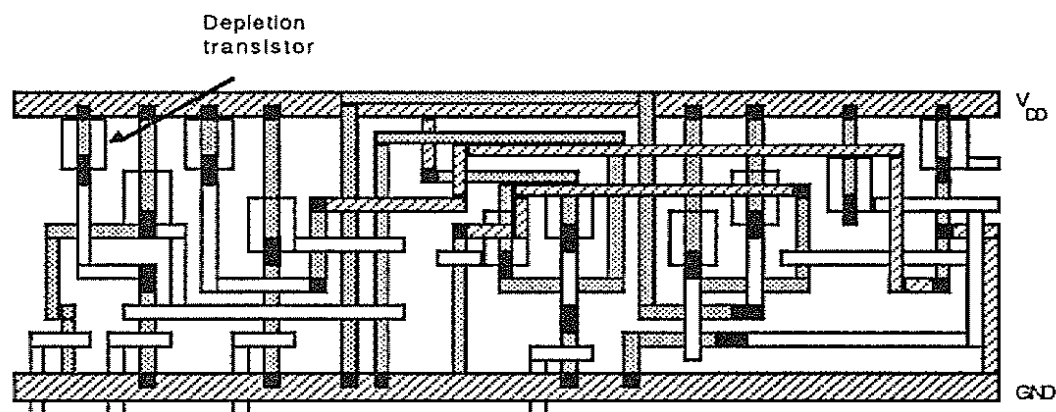


Fig. 1.6. A multicell nMOS layout generated by the Planar Embedding technique [NJ85].

The remaining layout styles we survey are called array methods, and place the transistors only at points in an array or grid pattern. These methods are the programmable logic array, the gate matrix, and the functional cell.

1.2.3 Programmable Logic Arrays

A *programmable logic array* (PLA) is a widely-used structured layout style [Sm83]. Its logical representation takes the form of a two-level, sum-of-products, multiple-output circuit. Physically, it consists of separate AND and OR arrays, with transistors placed at the grid points of the array; Fig. 1.7 shows the structure of a typical PLA. Each row of the AND array represents a logical product of various input signals, which are the columns of the AND array. These rows extend into the OR array, where the corresponding products are summed. The columns of the OR array then form the logical outputs of the PLA. There are several techniques to reduce the layout area of a PLA [Ul84]. The logic circuit can be transformed to reduce the number of rows in the array. Two inputs can be folded together so that they share the same column in the AND array, thereby reducing the number of columns of the array. Rearranging the order of the rows representing the product terms can increase the number of columns that can be shared. Another technique is the rearrangement of the output columns representing the sum terms to increase the number of rows that can be shared, thus reducing the number of rows of the array.

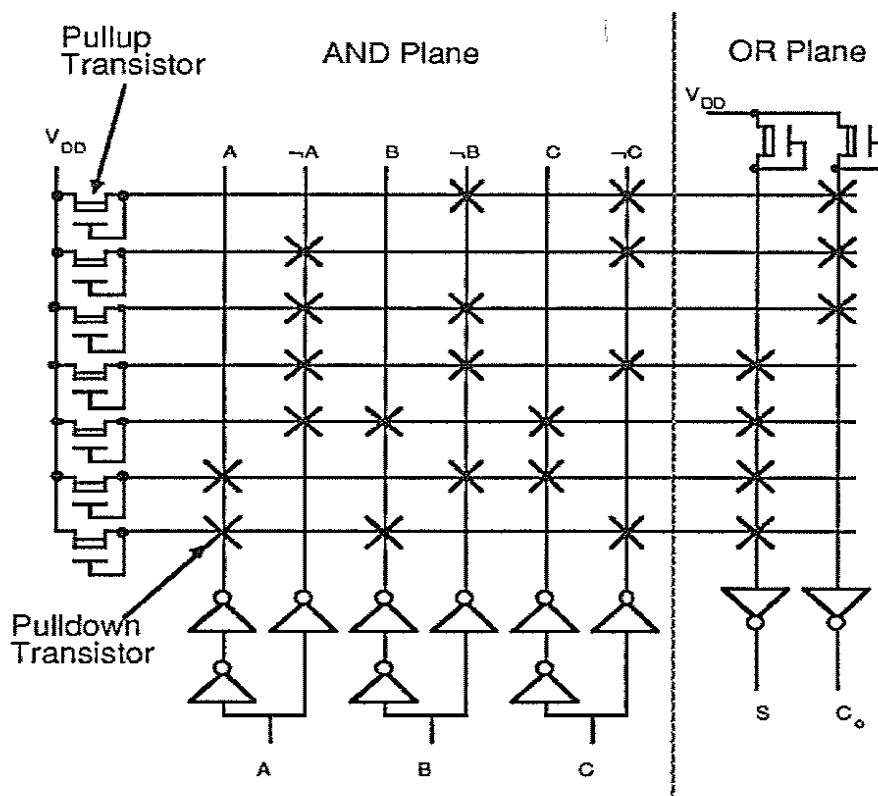


Fig. 1.7. A programmable logic array (PLA) implementing a full adder.

The standard PLA structure has several limitations. It does not handle logic functions in nonsum-of-products form; the outputs of the PLA must be routed back to the input to implement such functions, which is wasteful in area and speed. Standard PLA's maintain separate AND and OR arrays; however, the arrays often have few transistors and can be merged into a single array. The long interconnections are quite area-consuming and slow down the circuits.

Modifications of the standard PLA have been devised to remedy some of these limitations. The Associative Logic Array (ALA) proposed by Greer [Gr76] has the following features. The AND and OR arrays are merged into a single array. This allows internal feedback of intermediate functions to be made near the place where they are used as inputs, reducing long feedback paths outside the array. The merging of arrays allows the sharing of rows by two or more product terms generated within the array, by grouping the input columns of the product terms with the OR column

to which they connect. Internal feedback facilitates the implementation of *multilevel logic*, where the number of levels is, roughly speaking, the amount of parenthesis nesting in a logic expression of the following style. For example, $z = (a + b + c)$ corresponds to one level of logic, whereas $z = (a + (b * c))$ is a two-level expression. However, the ALA shares most of the other limitations of the PLA. Another modified PLA scheme was presented by Lursinsap and Gajski [LG84, LG85]. It also merges the arrays, folds pullup transistors into the array, allows pass transistors, and accepts a multilevel function description and boundary constraints. Yet another PLA variant, Path Programmable Logic, has been proposed by Smith that allows circuits like flip-flops, inverters, or pass transistors to be included within a merged AND-OR array [Sm83].

1.2.4 Gate Matrix Layouts

Next, we turn to matrix layout methods that apply primarily to CMOS technology. The gate matrix layout style is popular for static CMOS circuits [LL80], although it has also been extended to nMOS circuits, which employ n-channel transistors exclusively [SM83]. A *gate matrix* consists of equally-spaced vertical polysilicon columns, that serve the dual role of transistor gates and general interconnections, and rows of diffusion and metal, with the intersection of horizontal diffusion and vertical polysilicon forming transistors; see Fig. 1.8. This style is a generalization of the semi-custom standard-cell style with the modification that the logic and the interconnection channels are no longer kept separate, but are merged together. The full-custom gate matrix style is not to be confused with the similar-sounding gate array method, described in Section 1.2.1 under semi-custom methods.

The number of polysilicon columns in a gate matrix design is typically equal to the sum of the number of distinct inputs and output signals. The diffusions used for transistor terminals are grouped together to form rows. The interconnections required to complete the circuits are made by metal and diffusion lines, which run parallel to, and in between, the polysilicon columns, or by metal along the rows of the matrix. The columns may be ordered to reduce the number of rows. The gate matrix method has been automated using graph-theoretic algorithms which attempt to minimize the number of rows in the matrix by ordering the columns so that the rows can be shared by several gates [Wi82, Wi83, Wi85]. Boundary constraints have also been added to the gate matrix style [YK85]. A variation of the gate matrix style is called metal-oriented layout [Pi83]. It is quite similar to the gate matrix style, with the difference that rows are diffusion and columns are metal instead of polysilicon. This results in less input resistance due to the use of metal input columns. Rows and columns of the gate matrix are folded in the variation of Lin et al. [LD89].

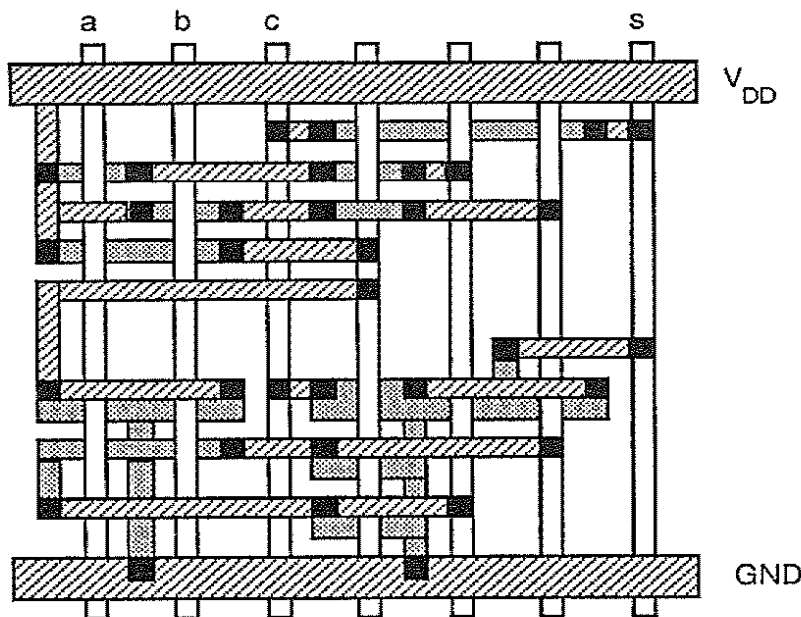


Fig. 1.8. A gate matrix layout (after [WE85]).

The gate matrix style eliminates the need for the ground columns and rows used in PLAs and ALAs. It can implement multilevel logic easily because it can route wires between gates inside the matrix. One of the limitations of the gate matrix style is that all the columns are widely spaced to allow for contacts and vertical interconnections. Unless multiple layers of metal are used for interconnection, the long polysilicon gate lines of the standard gate matrix introduce considerable parasitic resistance, and long diffusion wires add much resistance and capacitance, which reduces the circuit speed. Current automatic layout methods are not able to minimize exactly the area of gate matrix designs.

1.2.5 Functional Cells

The above full-custom styles often produce large layouts containing many logic gates. In contrast, a functional cell is a layout of a basic logic unit, such as a single gate. A functional cell can implement multilevel logic of arbitrary depth, although this depth is usually limited by performance considerations. We are primarily interested in cells for static CMOS gates, which consist of two 2-terminal subcircuits of transistors, the terminals of one, the pullup subcircuit, connecting the

cell output to the power supply (V_{DD}), and the terminals of the other, the pulldown subcircuit, connecting the cell output to ground (GND); see Fig. 1.3(a). Using a functional cell to implement a multilevel logic element instead of several standard gates, as in the standard cell style, reduces layout area and increases circuit performance. For example, consider the 8-input logic function represented in the usual infix notation

$$z = -((a*b) + ((c+d)*(e+f)) + (g*h)),$$

or, in prefix notation, where the operator precedes its operands

$$z = -(+(*ab)(*(+cd)(+ef))(*gh)).$$

This function can be implemented using five cells, each a 2-input NAND gate. Alternatively, it can be implemented by the single "complex" gate shown in Fig. 1.3(a); the layout area of its functional cell implementation in Fig. 1.3(b) is about half that of the circuit composed of five NAND gates [Uv81]. The use of such functional cells reduces the area of the overall circuit and often improves its performance in comparison with circuits using more primitive cells like NAND and NOR alone. There are two basic types of functional cells, one-dimensional and two-dimensional cells, which we examine next.

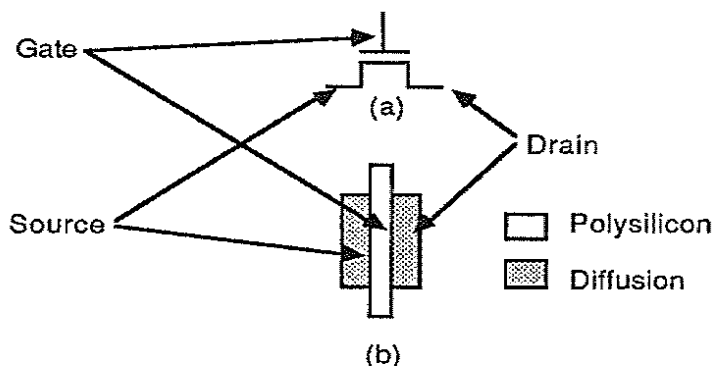


Fig. 1.9. An nMOS transistor: (a) circuit symbol; (b) layout.

One-Dimensional Cells. A *one-dimensional* functional cell is an array of transistors in which all drain and source terminals of transistors of a given type, either pMOS or nMOS, lie along a single row of diffusion. Figure 1.9 shows the correspondence between the circuit symbol of a transistor and its corresponding layout in a functional cell. A transistor is formed where a polysilicon region

crosses over a diffusion region. The polysilicon column of Fig. 1.9(b) defines the gate terminal of the transistor, which determines whether the circuit between the source and drain terminals is open or closed. Thus, the transistor acts as a switch that is on or off depending on the voltage on the gate terminal. The diffusion region on one side of the polysilicon in the figure is the source terminal and the other diffusion region is the drain terminal. Such transistor layouts are placed end-to-end in a horizontal line, with vertical polysilicon columns.

The functional cell style was introduced by Uehara and vanCleemput in 1978 [Uv78] in the context of static CMOS functional cell layout. A static one-dimensional functional cell consists of two parallel linear arrays of this sort, the lower array for the nMOS transistors implementing the pulldown subcircuit of the cell, and the upper one for the pMOS transistors implementing the pullup subcircuit. The gate terminals of transistors sharing the same column are typically connected together by polysilicon. Figure 1.10 illustrates a functional cell of this sort, showing the two diffusion rows, the pMOS and nMOS transistor regions, and the polysilicon columns. In 1985, McMullen and Otten proposed a modified version of this style for dynamic functional cells [MO85], which employ just a single linear array of transistors. We shall examine both static and dynamic one-dimensional functional cell layout styles in the next section.

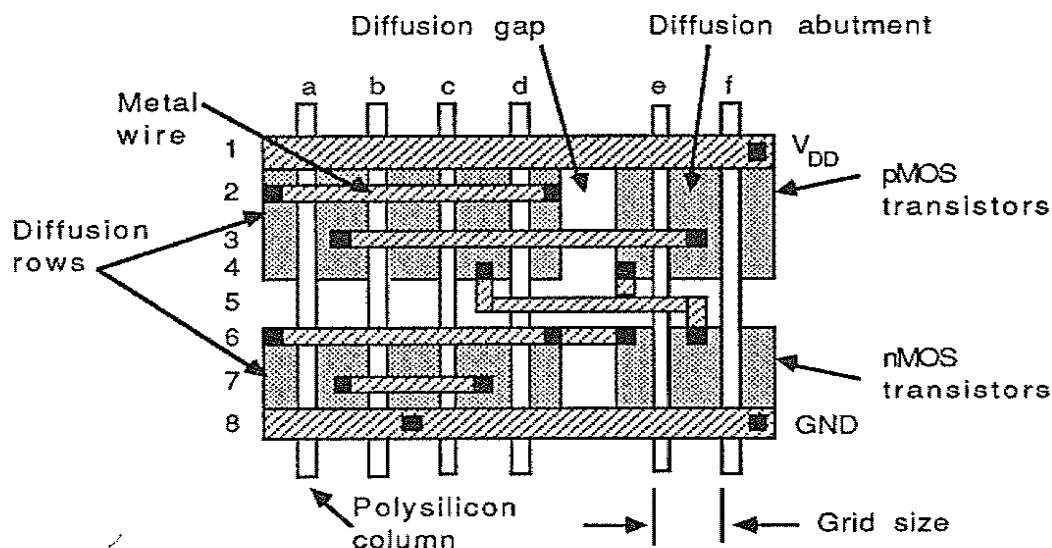


Fig. 1.10. A one-dimensional static functional cell showing diffusion regions, metal wires, and transistor types.

Two-Dimensional Cells. In contrast to one-dimensional cells, *two-dimensional* functional cells do not restrict the diffusion of the transistors of either the pullup or pulldown subcircuit to a single horizontal line; the diffusion follows the transistor circuit topology. In essence, the circuit is embedded in the two dimensions of a plane using diffusion. Figure. 1.11 illustrates a two-dimensional cell implementing the function $z = \neg(abd + \neg acd + a\neg b\neg c + \neg a\neg bc)$. Typically the gate signals are metal lines which pass over the diffusion without creating transistors, except where the metal is connected to polysilicon transistor gate terminals.

The two-dimensional functional cell style seems to have originated with Weinberger in 1967 [We67]. He proposed a layout style, now called a Weinberger array, which employs a horizontal array of nMOS functional cells with horizontal and vertical pulldown diffusion lines, and horizontal polysilicon rows which serve as the pulldown gate terminals where the two layers intersect.

Other circuit styles have been adapted to the two-dimensional cell style. One such style is cascode voltage switch logic (CVSL), in its dynamic single-ended (SCVS) [Ju90], static single-ended [Er85], and differential (DCVS) [YH85] forms. Other circuit styles implemented as two-dimensional cells are domino CMOS logic [CK89], and static CMOS logic in a related layout style called diffusion strings [TS88].

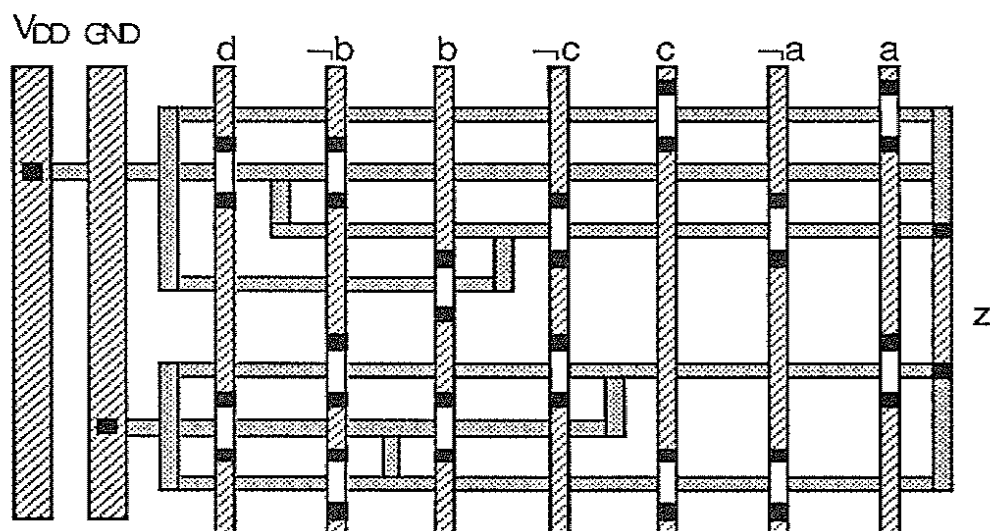


Fig. 1.11. A two-dimensional CMOS functional cell.

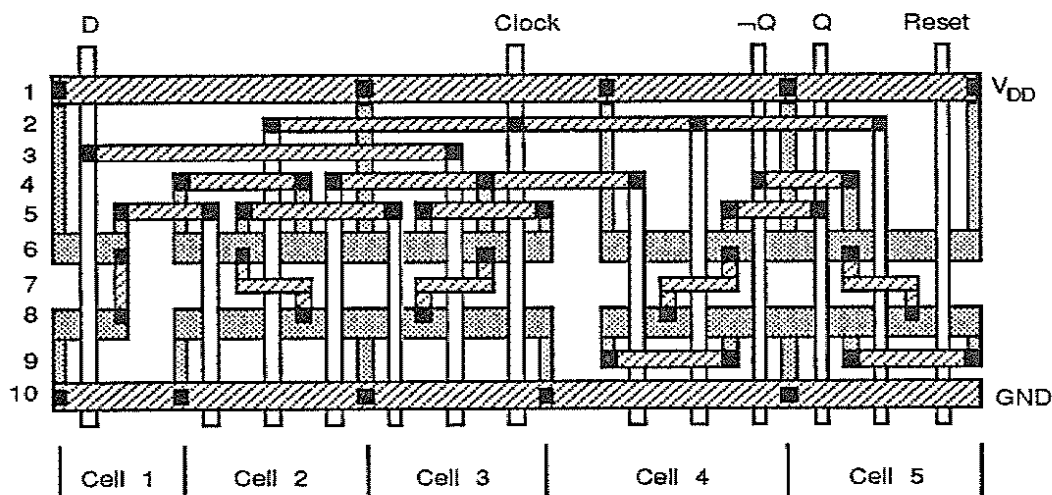


Fig. 1.12. A cell array composed of five functional cells forming a master-slave D flip-flop.

Cell Arrays. In the functional cell style of layout, logic functions are typically partitioned into multilevel logic circuits, composed of several gates. A functional cell implements each circuit. There are two ways to interconnect functional cells. One is similar to that used in the standard cell style (Section 1.2.1), where the cells are assigned to rows and the cells are interconnected through routing channels separating each pair of rows, as shown in Fig. 1.5. An alternative method groups functional cells to form a *cell array*, which is an array of interconnected functional cells. Figure 1.12 illustrates a *one-dimensional* cell array implementing a master-slave D flip-flop composed of five functional cells. A *two-dimensional* cell array consists of multiple rows of one-dimensional arrays.

1.3 FUNCTIONAL CELL OPTIMIZATION

Here we introduce the layout problem for one-dimensional CMOS functional cells. We also summarize and critique prior approaches to solving this problem.

1.3.1 Layout Problem

The primary goal in functional cell design is to minimize the cell width W , which is given by the following formula [Uv81]:

$$W = T + G + 1$$

where T is the number of dual transistor-pairs in the circuit, each pair corresponding to a polysilicon column in the cell, and G is the number of diffusion gaps. W is measured in units of the basic grid size, which is the minimum horizontal spacing between adjacent polysilicon gate columns, as illustrated in Fig. 1.10. A diffusion gap is needed to isolate transistor terminals that are physically adjacent in the cell but are not connected in the transistor circuit. Figure 1.10(b) shows transistors d and e separated by a diffusion gap because they are not connected in the pullup subcircuit. Diffusion abutment, which merges two neighboring diffusion regions, can be used if the physically adjacent terminals of the transistors in the cell are connected together in the corresponding circuit. Figure 1.10 shows transistors e and f connected by diffusion abutment because they are connected in both the pullup and pulldown subcircuits. A diffusion gap requires the separation between neighboring vertical polysilicon columns to be twice as large as that needed by diffusion abutment; see Fig. 1.10. Therefore, in the interests of reducing cell width, the preferred method of connecting physically adjacent transistors is diffusion abutment. Thus, the width minimization problem to be solved is that of finding a permutation of the polysilicon gate columns and an orientation of the transistors that minimizes W or, equivalently, maximizes the number of transistors that can be connected by abutment.

A secondary design goal is to minimize the cell height, which is determined by the number of horizontal routing rows needed for interconnecting transistor terminals that are not physically adjacent in the cell; such interconnection is done with metal wires as shown in Fig. 1.10. Transistors a and d are connected in the pullup subcircuit, but are not adjacent in the cell of Fig. 1.10; therefore, their terminals are connected by a horizontal metal wire. Figure 1.10 numbers the metal routing rows along the left side of the cell, whose height is shown to be eight rows; that of Fig. 1.12 is ten rows. The height minimization problem is to find a permutation of the polysilicon gate columns and an orientation of the transistors that minimize the number of horizontal routing rows required for interconnecting the transistors in the cell. Note that cell width and height are here defined with respect to a grid of polysilicon columns and metal rows. The corresponding width and height minimization problems also assume this grid, and layout sizes are computed with respect to the grid. If a *fixed grid* is used, then final layout size is found directly by scaling the number of grid units by the horizontal and vertical grid sizes. However, if a *virtual grid* is employed, then it may be possible to compact the cell layout by manipulating its components limited only by the layout-level design rules [WE85]; this compaction problem is beyond the scope of this book. Minimizing cell width

and height is typically called area minimization, although there may be cases where a minimum area cell has neither minimum width nor height. These functional cell area-reduction goals are in contrast with the goals of gate matrix layout, which primarily attempts to permute transistors to decrease cell height, and typically does not attempt to reduce layout width.

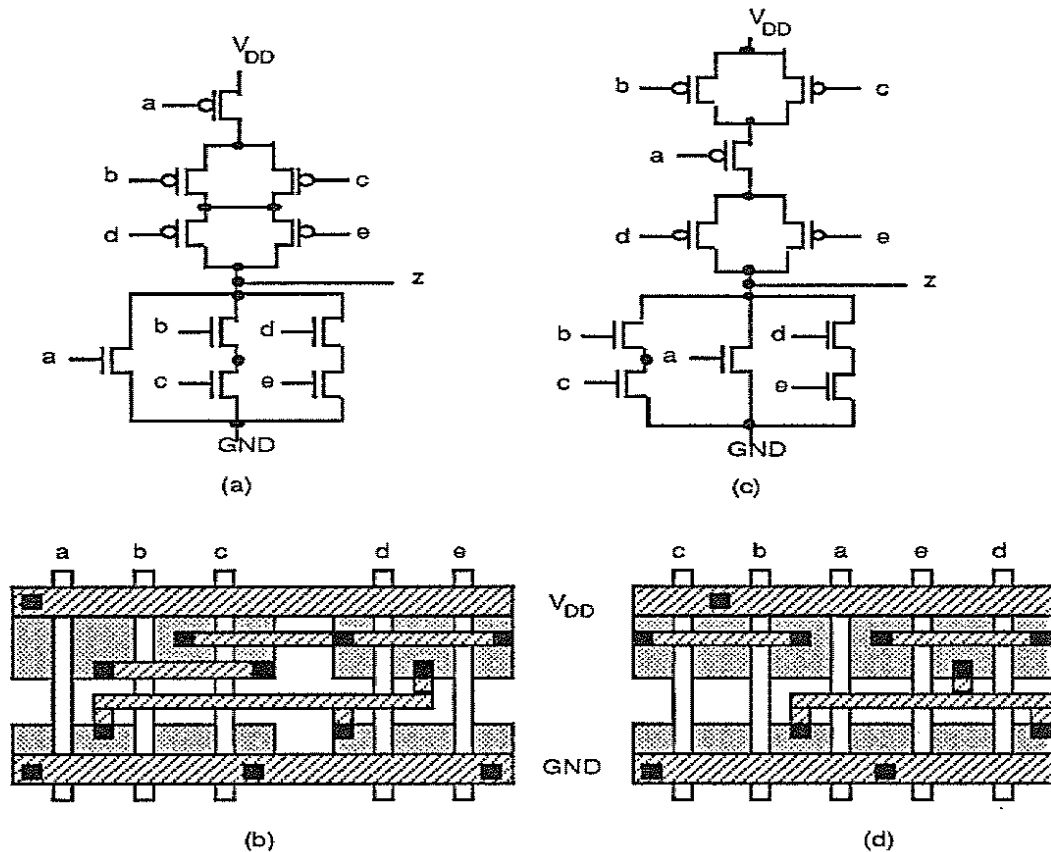


Fig. 1.13. (a) Circuit implementing function $z = \neg(+a(*bc)(*de))$ and (b) its layout; (c) Equivalent reordered circuit and (d) its smaller layout.

The transistors of a circuit can often be reordered without changing their logical function. This logical equivalence is based on the commutative property of the logical AND (OR) operation, which is implemented in the circuit by transistor subcircuits connected in series (parallel). Reordering of a circuit can affect its speed and layout area. The impact of reordering on layout area is illustrated by Fig. 1.13(a), which shows a circuit implementing the function $z = \neg(*a(+bc)(+de))$, and

an equivalent circuit (c), which reorders the series subcircuits (+bc) and *a*. The layout in Fig. 1.13(b), corresponding to the circuit (a) in the figure, is one unit (column) wider than the layout (d) of the reordered circuit (c). Therefore, another area-minimization problem is to find a reordering of the transistor circuit that minimizes the width and height of the functional cell.

These three area optimization problems, width minimization, height minimization, and circuit reordering, are related to one another for a single cell as follows. Width and height minimization are typically the primary and secondary objectives, respectively. A layout of minimum height from among all layouts of minimum width is then the overall design goal. If circuit reordering is performed, then the final goal is to find a layout of minimum height from among all layouts of minimum width for any ordering of the circuit. If reordering is not done, then such a layout is sought only for the original ordering of the circuit.

A large circuit can be composed of multiple cells, each being like the one shown in Fig. 1.13(a). The layout design of a cell array implementing such a circuit involves generating layouts for each of its component cells, such as the layout shown in Fig. 1.13(b) for the cell of Fig. 1.13(a). The problem of coordinating the layout of these component cells such that the composite layout of the cell array has minimum height from among all composite layouts of minimum width for all reorderings of the component circuits is the corresponding *multiple cell* problem.

Functional cells can implement multilevel logic circuits; however, in practice the number of logic levels is typically limited by circuit performance considerations. The transistors in a circuit act as switches, which are either in the on or off state. The transistors that are in the on state create conducting paths between the terminals of their subcircuit. Such a path connects the circuit output to the V_{DD} or GND terminals, which supply the 1 or 0 logic values, respectively. The amount of electrical resistance along a path between the terminals is directly related to the number of transistors on the path, which is defined as the path's length. This resistance determines the speed with which the circuit can switch its output from one logical state to the other. It is typical to limit the maximum-length path between either subcircuit's terminals because this path length is inversely proportional to circuit speed. This maximum length path is called the *height* of the circuit. The height of the circuit in Fig. 1.13(a) is three, since its pullup subcircuit has a path *abd* of length three and none of length four. The upper limit on circuit height defines the class of *practical-sized cells*. A typical limit on the pullup and pulldown subcircuits height is four [Uv81], although a height of up to six may be feasible for the pulldown subcircuit.

1.3.2 Prior Work

We now summarize and evaluate the prior work on one-dimensional functional cell layout methods. A more complete survey and critique of the individual methods is given in Chapter 2. There has been much recent research interest in automating the layout of circuits in this style. Most of the methods reported are heuristic in nature, and consequently are inherently nonexact. Such methods claim to find good, but not necessarily minimum-area, layouts. The few nonheuristic methods are based on algorithms of limited scope. They solve parts of the area-minimization problem exactly, but either do not address other parts or solve them heuristically. For example, Hwang et al. solve the width minimization problem but do not deal with circuit reordering and height minimization [HH89]. Nair et al. solve the width minimization and circuit reordering problem for some of the single-cell circuits of interest, but do not address the height minimization and multiple-cell problems [NB85]. Ong et al. present an exact solution to the multiple-cell width minimization problem. However, they do not deal with circuit reordering and solve the cell height minimization problem heuristically [OL89]. Madsen [Ma89] and Lefebvre and Chan [LC89] propose a solution to the width minimization and circuit reordering problems for a portion of the single-cell circuits of interest, but either do not handle the remaining circuits or lay them out heuristically. No prior method solves the width, height, and reordering problems for all single-cell circuits of interest. Furthermore, no prior method solves the width-minimization and circuit-reordering problems for multiple cells, and none solves the height minimization problem for multiple-cell layouts, with or without circuit reordering.

Another characteristic of the prior work is the way the methods are evaluated in terms of layout quality and execution speed. Some researchers report comparisons of a few of their automatically generated layouts to those designed manually. Others compare a relatively small number of their layouts to those published in the research literature. But, in most cases, the evaluations are based on relatively few comparisons; furthermore, the layouts to which comparison is made are not known to be of minimal size. No method is comprehensively evaluated by comparing its layouts to the minimum-sized layouts of all practical-sized circuits. Moreover, only a few papers report the worst-case time complexity of their methods. Some give execution times for a few examples, but times for all practical-sized circuits have not been reported. Since many of these methods may have exponential worst-case time complexity, a few examples may give little insight into their overall performance. (A procedure has *exponential* worst-case time complexity if its execution time grows as a function that is exponential in the size of the procedure's input.)

1.4 PROPOSED APPROACH

This section introduces our method of solving the one-dimensional functional cell area-minimization problems defined in Section 1.3.1.

1.4.1 Philosophy

We begin by asking the following general questions: under what circumstances should a heuristic be used to solve a hard design problem instead of an algorithm, and if a heuristic is used, what characteristics should it have? Before we discuss the answers to these questions, we define some important concepts.

An *exact algorithm* is a step-by-step procedure for solving a problem Π if, given any instance I of Π , it is guaranteed always to produce an exact solution for that instance I in finite time. For instance, where Π is a minimization problem, an exact solution must be of minimum size [GJ79]. A *heuristic algorithm* for a problem Π , by contrast, does not always find an exact solution for each instance I of Π .

For a hard problem Π , a heuristic algorithm may offer better performance guarantees than any known exact algorithm for Π . However, except in special cases, the quality of a result found by a heuristic algorithm can be arbitrarily far from optimal. In some cases, heuristic algorithms can guarantee near-optimal results [HS78]. A heuristic algorithm H is an *absolute-approximation algorithm* if for every problem instance I , its solutions $H(I)$ are never worse than the optimal solution $\text{OPT}(I)$ by more than some constant k . For example, if H is a heuristic algorithm for a minimization problem, and the solutions are single numbers, $H(I) \leq \text{OPT}(I) + k$, for all I . A heuristic algorithm is a *relative-approximation algorithm* if, for every instance I , $H(I) \leq r \times \text{OPT}(I) + k$, for some constants r and k . One example of a relative approximation algorithm for wire routing is presented by Brady and Sarrafzadeh [BS90], whose method finds a routing pattern with at most $4/3$ times the optimal area. Therefore, if a heuristic algorithm is used, and one wishes to have some guarantee about the distance between the heuristic solution and the exact solution, then either the heuristic solution must be bounded as, for example, in absolute or relative approximation algorithms, or else one must evaluate the heuristic and an exact algorithm on each problem instance in the domain of interest and measure the difference. If none of these evaluations is made, then one must be prepared to accept arbitrarily bad solutions from the heuristic algorithm.

The advantage of an exact algorithm is its guaranteed optimality. Its disadvantage is that its run (execution) time can be very long for large or hard problems. In cases when the problem size is large enough to make the run time unacceptable on the available computers, one is forced to use a heuristic that has an

acceptable run time. An exact algorithm is generally considered to be too slow or intractable if its run time on a problem of size n cannot be bounded by a polynomial function of n .

However, a gray area exists between these two extremes. What should be done when either it is known that no algorithm for Π can have polynomial worst-case time complexity, or that no polynomial-time algorithm is known for Π , and the problem size is bounded at some small or medium value? This is the type of problem that we face in the functional cell layout problem. Most CAD tool designers choose heuristics to solve such problems, whereas we choose to find exact algorithms. It is our philosophy that when this choice is faced, one should first try to find an exact solution until one is convinced that this approach is not feasible; only then should one resort to heuristics. If a heuristic is to be used, an approximation algorithm should be sought. Only if all these attempts fail, should one settle for an unbounded heuristic, which should then be thoroughly tested empirically.

The reasons for this overall approach are threefold. First, the execution time for a small problem size may be quite reasonable. Second, it may also be found that the exact solutions are greatly superior to the heuristic solutions. Third, optimality is theoretically interesting for its own sake, and results in provably minimal cost. We found all of these reasons to be true in our problem of cell layout.

This general philosophy, together with intuition gained from our study of the functional cell layout problem, led us to formulate a working hypothesis at an early stage of our work, which formed the basis for our research. It is as follows.

Hypothesis: Exact solutions to both the single- and multiple-cell width and height minimization problems over all circuit reorderings are both computationally feasible for all practical-sized static CMOS circuits and produce significantly smaller layouts than previous heuristic and limited-scope algorithmic methods.

We follow Uehara and vanCleve and others in defining the class of all practical-sized static CMOS circuits to be those with a circuit height limit of four [Uv81, LC89, TI86, AT90]. We do not precisely define the limits of computational feasibility, since these limits are changing rapidly with advances in computer technology. However, in general, we consider average execution time on the order of minutes and worst-case time on the order of hours on a small computer, such as a CAD workstation, to be within the limits of feasibility for hard problems such as these. Our goal, stated succinctly, is to demonstrate the validity of the above hypothesis.

We shall reserve the term algorithm for an exact algorithm, and we shall generally use of the terms heuristic for heuristic algorithm. Although the algorithms we present generate layouts within a specific style based upon a set of typical layout assumptions, these algorithms are also quite amenable to other layout styles, such as those that use multiple metal layers for routing.

1.4.2 Outline of the Book

The book is organized as follows. Chapter 2 introduces the subject of one-dimensional functional cell layout design. Prior methods are analyzed in detail. We develop a system of classifying all functional cell area-minimization problems, and categorize prior methods according to the problems they address. In Chapter 3, we discuss series-parallel cell-width minimization with and without reordering. We present a comprehensive theory of covering a graph model of dual series-parallel circuits. This theory forms the basis for two layout algorithms, *TrailTrace* and *R-TrailTrace*, which solve exactly the single-cell width-minimization problem for fixed and reordered circuits, respectively. We demonstrate the feasibility of these algorithms by applying them to all single-cell circuits of practical size. In this study, we also thoroughly evaluate other layout methods and analyze the properties of this class of circuits. Our analysis shows that, in the best case, circuit reordering can reduce layout width by more than 20% over optimal-width layout without reordering.

Chapter 4 generalizes the single-cell theory of Chapter 2 to handle all dual planar circuits, which include many nonseries-parallel circuits. We present a layout algorithm, *P-TrailTrace*, which solves exactly the planar cell-width minimization problem. We apply this algorithm to all nonseries-parallel circuits of practical size to demonstrate the computational feasibility of *P-TrailTrace*.

Chapter 5 extends our single-cell theory to incorporate cell-height minimization. We precisely define the height minimization problem, and present an algorithm, *HR-TrailTrace*, which solves exactly the single-cell width and height minimization problem over all circuit reorderings. Again, we apply our algorithm to all practical-sized circuits, demonstrating its feasibility. Our computed results show that height minimization can reduce layout area of a single-cell by more than 80% over width minimization and circuit reordering alone. We also show significant area improvement over most other single-cell layout methods by applying our algorithm to published layouts produced by these methods.

In Chapter 6, we broaden the single-cell theory of Chapter 5 to deal with cell arrays. We define the height minimization problem for cell arrays and present another layout algorithm, *HRM-TrailTrace*, that solves exactly the multiple-cell width and height minimization problem over all circuit reorderings. We apply this

algorithm to several published multiple-cell examples and achieve up to 80% area improvement over other heuristic methods. We also apply our algorithm to several large commercial circuits to show its computational feasibility.

Chapter 7 summarizes our results in terms of the eight functional cell area-minimization problems defined in Chapter 2, and shows how these results prove our hypothesis. It also discusses some practical applications and extensions of this work.

CHAPTER II

FUNCTIONAL CELL LAYOUT METHODS

In this chapter, we discuss one-dimensional functional cell design in depth and present a detailed survey and evaluation of all important prior methods for generating layouts in this style.

2.1 FUNCTIONAL CELL DESIGN

First we discuss the general principles of functional cell design. We begin by introducing the original set of assumptions defining this layout style, followed by an extended illustration of these fundamental ideas. Uehara and vanCleemput originally proposed the style of interest, which is illustrated by Fig. 1.3(b) [Uv78]. To simplify the resulting design problem, they made various assumptions, which we list explicitly in Fig. 2.1.

-
1. Static CMOS technology is used.
 2. The pMOS and nMOS subcircuits are series-parallel connections of transistors.
 3. The pMOS and nMOS subcircuits are geometric duals of each other.
 4. The circuit height h is limited to four for performance reasons.
 5. Each cell consists of two horizontal diffusion rows for pMOS and nMOS transistors. This implies that single p- and n-wells can be used.
 6. Complementary pMOS and nMOS transistor pairs are vertically aligned. This allows their gate terminals to be interconnected by vertical polysilicon columns without use of crossovers.
 7. Transistor drain and source terminals are connected by diffusion if the terminals are physically adjacent in the layout, or by metal if they are not adjacent. Only one metal layer is used.
-

Fig. 2.1. The assumptions defining the Uehara-vanCleemput layout style.

Figure 2.2 illustrates the first four assumptions of Fig. 2.1. A static CMOS gate is a transistor circuit composed of two 2-terminal subcircuits of MOS

transistors, a pullup subcircuit of pMOS transistors, the designated terminals of which are V_{DD} and the circuit's output z , and a pulldown subcircuit of nMOS transistors, whose terminals are z and GND. These subcircuits are functional duals, meaning that when there is a conducting path between the terminals of one subcircuit, there is not such a path between the terminals of the other subcircuit. The circuit of Fig. 2.2 is often referred to as a static CMOS "complex" gate. A pMOS transistor is shown in the pullup subcircuit. The circuit between the source and drain terminals of a pMOS transistor is closed, creating a path for current to flow, if the voltage on the gate terminal is low (at the logical 0 voltage, assuming the positive convention); otherwise the circuit is open, preventing the flow of current. Conversely, the circuit between the source and drain terminals of an nMOS transistor in the pulldown circuit is closed, if the voltage on the gate terminal is high (at the logical 1 voltage); otherwise the circuit is open. For example, if the signals c , d , a and g are low, then the path through the pMOS transistors with these signals on their gate terminals connects V_{DD} to the output terminal, and the output signal is logical 1. Note that under these conditions, no path from the output z to GND can be closed because the nMOS dual transistors labeled c , d , a and g open all possible paths to GND. If signals a and b are high, then the output is connected to GND through transistors a and b , and all paths from V_{DD} to the output are open resulting in a logical 0 output. The source terminal of an nMOS transistor in Fig. 2.2 is the

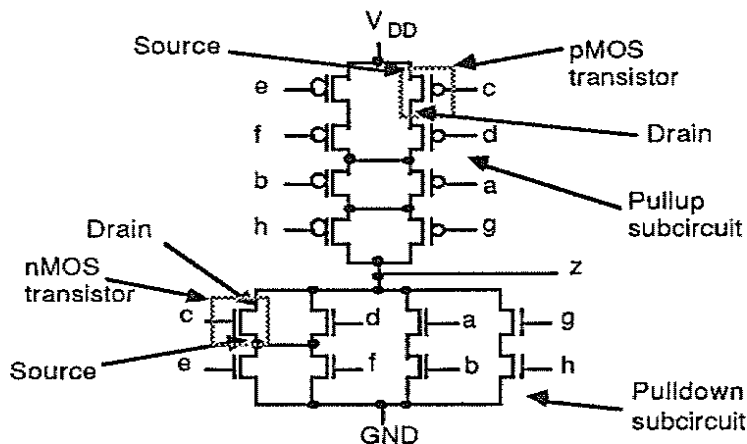


Fig. 2.2. A static CMOS complex gate.

one that is closer to GND, whereas the source terminal of a pMOS transistor is the one closer to the V_{DD} terminal; these two cases are illustrated in Fig. 2.2 for the transistors labeled c .

The second assumption of Fig. 2.1 is that the pullup and pulldown subcircuits are series-parallel. The circuit of Fig. 2.2 is of this type, where every transistor in either the pullup or pulldown subcircuit is either in series or in parallel with some other transistor. Transistors *a* and *b*, for example, are in series in the pulldown subcircuit and in parallel in the pullup subcircuit. The third assumption is that the pullup and pulldown subcircuits of the circuit are geometric duals [Ha69]. Dual subcircuits reverse the roles of the series and parallel operations; every transistor in series (parallel) with another transistor in one subcircuit is in parallel (series) with its dual transistor in the other subcircuit. Furthermore, dual transistor pairs in the subcircuits have the same signal label on their gate terminals. The pullup subcircuit of Fig. 2.2 the geometric dual of the pulldown subcircuit. The height restriction of the fourth assumption is illustrated by the longest path of transistors *c*, *d*, *a* and *g* in the pullup subcircuit. Since no path in either subcircuit is longer than this, the circuit height is four.

The remaining three assumptions relate to the layout of transistors in the functional cell. First, we briefly discuss the role of the three layers: metal, polysilicon and diffusion. Diffusion and polysilicon are primarily used for creating transistors, which occur where these layers cross, as shown in Fig. 1.9. They can also form interconnections, but they have higher resistance and capacitance than the metal layer, whose primary role is interconnection. The metal layer is on a different plane than either the diffusion or polysilicon layer, and requires a contact to make an electrical connection with these other layers. Two such contacts are shown in Fig. 2.3(b) along the bottom row of metal, interconnecting the diffusion regions of the pulldown regions to GND. Although we only use one metal layer, some IC manufacturing processes offer two or more layers.

The fifth assumption of two rows of transistors is illustrated in Fig. 2.3(b), where the pMOS transistors are placed in the upper row and the nMOS transistors in the lower row. The sixth assumption about interconnecting the gate terminals of a dual transistor pair with a vertical polysilicon column is illustrated in Fig. 2.3(a), where the pMOS and nMOS dual transistors labeled *c* in Fig. 2.2 are vertically aligned in the leftmost column also labeled *c*, and their gate terminals are connected.

Assumption 7 of Fig. 2.1 defines the methods of interconnecting those source and drain terminals in the layout that are connected in the transistor circuit. First, we outline how drain-source terminals of transistors are interconnected or electrically isolated in the functional cell, and how this affects cell size. Terminals connected in the transistor circuit are connected in the cell either by metal wires or diffusion abutment; those not connected in the circuit are isolated either by not placing such terminals adjacent to each other or, if placed adjacent, by inserting a diffusion gap between them. The position and orientation of transistors in the cell determines

which connection or isolation methods are used, and consequently determines the width and height of the cell.

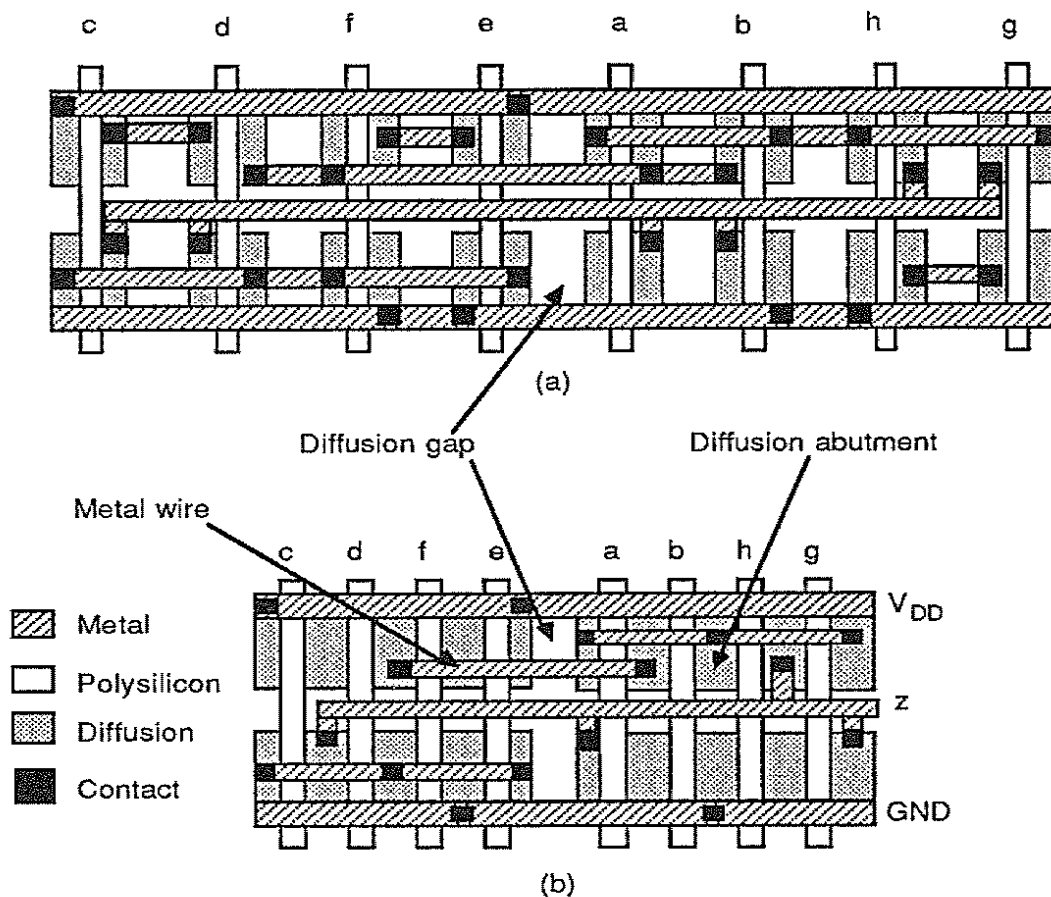


Fig. 2.3. A functional cell: (a) with no diffusion abutment; (b) with diffusion abutment.

The rules for the various interconnection methods allowed by the seventh assumption are now given. Drain-source terminals that are connected in the transistor circuit and that are also adjacent in the cell are connected by diffusion abutment; this case is illustrated in Fig. 2.3(b), where transistors *b* and *h* are connected in this way. Transistors connected in the circuit but nonadjacent in the

cell are connected by horizontal metal wires; Fig. 2.3(b) shows a metal wire in the pullup subcircuit connecting the diffusion region between columns *d* and *f* to the region between columns *a* and *b*. Drain or source terminals of two transistors not connected in the circuit but adjacent in the cell are isolated by an area-wasting diffusion gap. In Fig. 2.2, transistors *e* and *a* are not connected in the pulldown circuit, but are adjacent in the layout of Fig. 2.3(b); therefore, a diffusion gap is required between them.

We illustrate the layout process with Fig. 2.3. Assume we choose to place the dual transistor pairs in the cell in the left-to-right order of *c*, *d*, *f*, *e*, *a*, *b*, *h* and *g*. Each transistor is *oriented* in one of two ways, such that its drain terminal is either to the left or the right of its polysilicon gate terminal. First, let us consider the cell of Fig. 2.3(a) which uses only the metal layer to interconnect drain-source transistor terminals. A diffusion gap is placed between each pair of transistors, whether the adjacent terminals are to be electrically connected or not. Note that if only metal interconnections were used, all placements of transistors would result in a cell of the same width *W*. In this case, $W = T + G + 1 = 8 + 7 + 1 = 16$ units (columns). However, by choosing a good placement and orientation of the transistor pairs, and using diffusion abutment, the width of the cell can be reduced. Figure 2.3(b) shows a layout having the same transistor placement and orientation as that in Fig. 2.3(a), but using abutment wherever applicable. The cell width $W = 8 + 1 + 1 = 10$ columns. The height of the cell of Fig. 2.3(b) in terms of the number of metal rows is seven.

The height minimization problem depends on the number of metal interconnection layers used within a cell. Traditionally, one layer of metal is provided for interconnection purposes by chip fabrication processes, although more recently two or more layers have become available. Some layout styles use more than one metal layer for routing within a cell, whereas other styles employ only one metal layer in the cell even when more are available, reserving the other layers of metal for connections between cells. If more than one layer of metal is used within a cell, then in some cases cell height can be made smaller than the height of a cell using only one metal layer. The height problem is also affected by the number of contacts required between a metal layer and the other layers to ensure that a given fabrication process produces a reasonable percentage of working chips and that the total contact resistance is low enough for good circuit performance. For these reasons, multiple contacts become particularly important for submicron processes. The height of a cell may be larger if multiple contacts are required, when compared to a cell using single contacts. The vast majority of prior layout methods use a single layer of metal for cell interconnection; moreover, almost all of them use a single contact between metal layers and the other layers.

As stated in Chapter 1, there are several fundamental area-minimization goals. The primary design goal in one-dimensional functional cell layout is to minimize the cell width by ordering and orienting the transistors in the cell so that the number of neighboring transistors connected by diffusion abutment is maximized, thus minimizing the number of diffusion gaps. Likewise, the secondary goal is to order and orient the transistors in the cell to minimize cell height by minimizing the number of horizontal metal rows needed to connect the nonadjacent drain-source terminals by metal wires. The third goal is to reorder the circuit to reduce both cell width and height. A fourth goal is to solve the width, height and reordering problems for arrays of cells. We define eight layout problems based on which combinations of these four fundamental design goals are addressed; these problems are listed in Fig. 2.4. The names of the layout problems in Fig. 2.4 correspond to the optimization goals the problem address, where W, H, R and M refer to the width, height, reordering and multiple-cell design goals. For example, the WHR problem addresses the width, height and reordering problems, but not the multiple-cell problem. The relationship between these design goals is discussed in Section 1.3.1.

Layout problem	Width minimization	Height minimization	Circuit reordering	Number of cells
W	Yes	No	No	Single
WR	Yes	No	Yes	Single
WH	Yes	Yes	No	Single
WHR	Yes	Yes	Yes	Single
WM	Yes	No	No	Multiple
WRM	Yes	No	Yes	Multiple
WHM	Yes	Yes	No	Multiple
WHRM	Yes	Yes	Yes	Multiple

Fig. 2.4. Classification of area-minimization problems.

It has been shown that the WH problem is NP-hard for series-parallel dynamic circuits that possess an euler trail [Ch91]. It has also been noted that the version of this problem for static circuits is "considerably harder" than for dynamic circuits. Therefore, the WH problem for all dual series-parallel circuits is no less than NP-hard, as is true also of the WHR, WHM and WHRM problems for this broader class of circuits.

2.2 SURVEY OF PRIOR METHODS

We survey both the static and dynamic one-dimensional functional cell design methods, and classify them according to which of the layout problems of Fig. 2.4 they address.

2.2.1 Static Cells

First, we examine the design methods for static functional cells, which have two diffusion rows to accommodate the equal number of transistors contained in the pullup and pulldown subcircuits.

Uehara and vanCleave originally proposed a method for generating functional cells in the one-dimensional functional cell style [Uv78], based on the list of assumptions in Fig. 2.1. Not all methods generating layouts in the one-dimensional functional cell style make all these assumptions; some methods use them as guidelines, but do not adhere to them consistently. However, most of them are adopted by the majority of layout systems; we will note the methods that make different assumptions.

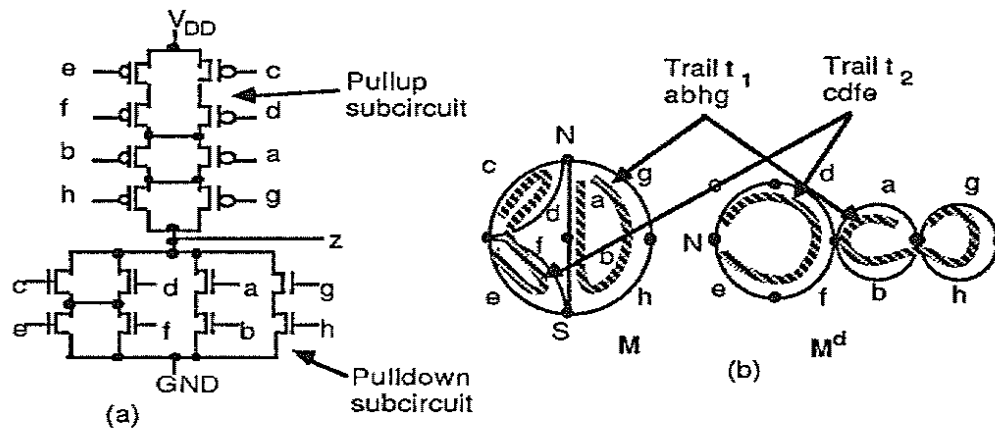


Fig. 2.5. The graph approach of Uehara and vanCleave: (a) transistor circuit; (b) graph model.

We now examine the Uehara-vanCleave layout method, which is a heuristic for solving the WR problem. The input to the design process is a transistor circuit of the type represented by Fig. 2.5(a). The transistor circuit is modeled by a graph, as in Fig. 2.5(b) which represents the circuit of Fig. 2.5(a). The edges correspond to transistors in the circuit and nodes correspond to interconnections of transistor source

and drain terminals. An edge is labeled with the name of the input signal applied to its gate terminal. The graph model consists of two series-parallel multigraphs, M for the nMOS pulldown subcircuit and M^d for the pMOS pullup subcircuit. (A *multigraph* is a graph where more than one edge joins two nodes [Ha69].) The pMOS graph M^d is the geometric dual [Ha69] of the nMOS graph M , and vice versa.

The process of translating the graph model of the transistor circuit into a layout involves finding a set of trails in the graphs such that every edge of the graph model is included exactly once in some trail. (A *trail* is an alternating sequence of nodes and edges, beginning and ending with nodes, where no edge is repeated [Ha69].) Each trail in this set can be laid out as a continuous region of diffusion. For example, the trail $t_1 = abhg$ and the trail $t_2 = cdfe$ in Fig. 2.5(b) can be laid out as two diffusion regions separated from each other by a diffusion gap, as shown in Fig. 2.3(b). Therefore, the number of trails must be minimized in order to minimize the layout width.

A key assumption (number 6 of Fig. 2.1) of the Uehara-vanCleemput layout style is that the gate terminals of each nMOS and pMOS transistor-pair corresponding to dual edges in the circuit are connected by vertical polysilicon lines in the layout. This assumption constrains the type of trail that is traced through the graphs; each pair of trails, one in the pMOS graph and the other in the nMOS graph, must consist of the same sequence of dual edges. Such a trail is called a *dual trail* or *d-trail*. The trails t_1 and t_2 in Fig. 2.5(b) are both dual trails, whereas cdg is a trail in M but is not a dual trail. A dual trail that includes all the edges of a graph is a *dual-euler* or *d-euler* trail, as illustrated by the trail $cbaed$ in the circuit of Fig. 1.13(c). As Fig. 1.13(d) demonstrates, a layout corresponding to a d-euler trail contains no diffusion gaps and so has minimum width.

The circuit of Fig. 2.5 does not contain a d-euler trail. In such cases, the Uehara-vanCleemput method modifies the graph model by adding "pseudo-edges" to ensure that the modified graph will have a d-euler trail. It then rearranges the subgraphs of the modified graph in a way that preserves the logical function represented by the graph, and minimizes the number of contiguous groups of pseudo- and real edges along the d-euler trail. This d-euler trail is the basis for the layout, where real edges correspond to transistors and each contiguous group of pseudo-edges corresponds to a diffusion gap. The heuristic does not guarantee a minimal-width layout. When applied to the circuit of Fig. 2.3(a) it finds two dual trails, resulting in the layout of Fig. 2.3(b), which has a diffusion gap. We will show in Chapter 3 that our method finds the optimal layout, which in this case has no gaps.

The W Problem. Nair and his colleagues [NB85] have extended the Uehara-vanCleemput approach. Using the same layout assumptions, they have developed an optimal algorithm solving the W (width only) minimization problem, and an

algorithm that solves the WR (width minimization with circuit reordering) problem for a subset of practical circuits. The first approach finds the minimum number of dual trails in a graph model of the circuit where the order of the subgraphs is fixed. It reduces sets of dual trails found in series-parallel multigraphs to 18 "representative" graphs based upon the number of distinct trails in the graph and the terminals of the graph that are the endpoints of the trails. These representative graphs are used to keep track of the different trails as the graph is processed. Representative graphs are combined together, resulting ultimately in an optimal set of dual trails. This set of trails leads to a minimum-width layout for fixed graphs. Nair et al. have also developed an algorithm for reordering the subgraphs of the graph model. This method produces optimal results in the special case that some ordering of the graph has a d-euler trail; otherwise it yields no result. We show in Chapter 3 that only 56% of practical-sized static circuits have this property. Both methods have a time complexity that is linear in the number of transistors.

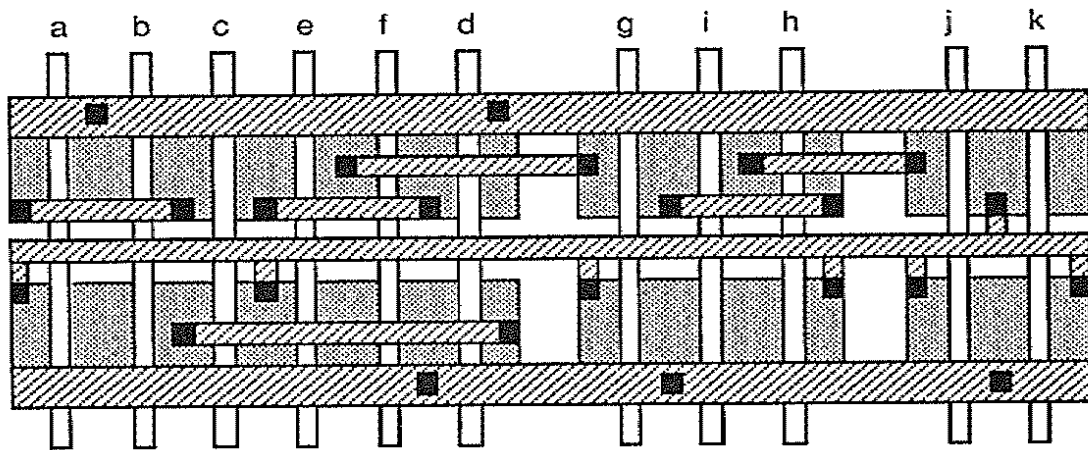


Fig. 2.6. A layout of the function $\neg(+(*hi)gjk(*ef)(*d(+c(*ab))))$ by the method of [HS88].

Huang and Sarrafzadeh have generalized the preliminary version of our dual trail-tracing serial algorithm reported in [MH87], by developing a parallel version based on our trail-covering theory [HS88]. It solves the W problem for all practical-sized series-parallel circuits. The time complexity of the method is $O(\log n)$ time using $O(n)$ processors. An example layout of minimum width produced by the algorithm of Huang and Sarrafzadeh is shown in Fig. 2.6.

The WR Problem. As discussed above, Uehara and vanCleemput were first to address the WR problem. More recently, Kwon and Kyung have reported a heuristic for the WR problem [KK88]. It yields a "nearly minimum" trail cover in $O(n^3)$ time, where n is the number of transistors in the circuit. They compare two layouts of their heuristic to those of our algorithm [MH87], but do not find any area improvement, although they do show improvement over the method of [Uv81]. The strength of this method is that it recognizes the importance of reordering and attempts to develop the best polynomial-time method that reorders the circuit. The authors predict that optimal reordering has complexity $O(n^n)$, whereas we show that its complexity is $O(h!n)$, where h is the height of the circuit, typically a very small number for practical-sized circuits.

Chen and Hou describe a heuristic for reordering and laying out a functional cell [CH88] to reduce its width, but not its height. They replace the original problem of finding the minimum number of trails to cover the dual graph model with the following simpler problem: minimize the number of vertices with odd degree by reordering the graph. This procedure indeed can improve the layout, as they show, but there are graphs all of whose vertices are even for all reorderings. Furthermore, some reorderings may require fewer trails to cover these graphs than others do; this method does not provide any guidance for reordering such graphs. Chen and Hou present five layouts produced by their method to which they compare layouts produced by our *TrailTrace* algorithm [MH87] and the heuristic of [Uv81]. They show that their reordering heuristic can produce better layouts than our algorithm that does not consider reordering. In Chapter 3, we present our *R-TrailTrace* algorithm, which optimally reorders circuits, and finds the minimum-width layout for all practical circuits, 61% of which require reordering. The earlier *TrailTrace* algorithm was designed to solve the W problem only, and so does not consider reordering.

Lin and Nakajima have reported a linear-time heuristic for the WR problem for dual series-parallel circuits [LN90]. Their method has generated optimal solutions to seven examples taken from functional cell layout papers.

The WHR Problem. Lefebvre and Chan have developed a heuristic method that appears to solve the WHR problem for a subset of practical-sized circuits [LC89]. The authors adopt the typical practicality constraint that the height of the circuit graph must be four or less. However, they also impose another constraint that the maximum number of logic levels must be three or less. We analyzed this constraint and found that only 18% of the practical-sized circuits meet it. The authors do lift assumption 6 of Fig. 2.1, allowing them to find very small layouts in special cases. In a second paper [LC90], Lefebvre presents a heuristic multicell layout method.

Madsen reports a method for functional cell synthesis that appears to solve the WHR problem under certain circumstances [Ma89]. Madsen's method also produces layouts that meet certain boundary constraints on the ordering of their inputs, and uses reordering to improve the layout area to meet boundary constraints and to improve circuit speed. He finds a d-euler trail in a circuit if one exists. However, he does not claim to find the optimal layout if a circuit cannot be covered by a single dual trail. This method makes essentially the same claim as the second algorithm in [NB85]. As we show in Chapter 3, 44% of all practical circuits do not have a d-euler trail in any reordering. Therefore, neither method cited can guarantee optimality for almost half of these circuits. It is unclear whether the Madsen method always achieves optimal height on circuits with d-euler trails. The author compares his method to those of [Uv81] and [CH88], showing his method yields better results. The layouts Madsen generates using the method of [Uv81] are better than those using the method of [CH88]; however, in [CH88], Chen and Hou show their method to be better than that of [Uv81] on all five examples! This demonstrates the inconclusive nature of evaluating a method based on a few comparisons.

Mailhot and DeMicheli have developed a linear-time heuristic method that attempts to solve the WHR problem [MD88]. The method applies to layout styles using one or two layers of metal interconnection and styles using single or multiple contacts, and can perform some width-versus-height tradeoffs. The method uses a few rules to perform limited reordering of the circuit. The authors present four sample layouts which they compare to those produced by other methods, but are not able to find the optimal layout for the one example taken from our paper [MH87].

The WM Problem. Hwang et al. present an optimal algorithm for minimizing the layout width of fixed multicell circuits with equal numbers of pMOS and nMOS transistors, thus solving the WM problem [HH89]. They model the circuit as a bipartite graph and use a branch-and-bound method, which is inherently an optimal technique. The method provides a tight lower bound on the number of trails in the optimal cover. With respect to series-parallel circuits, the results of Hwang et al. are identical to those previously produced by our first algorithm *TrailTrace*, except that our method's time complexity is linear in the number of transistors, whereas their method, which uses a branch-and-bound approach, has exponential-time complexity in the worst case. A second paper presents their method augmented by heuristics to handle cell height [HH90]; this method allows the use of multiple contacts.

The WHM Problem. Ong et al. describe a multicell layout system called GENAC that handles general CMOS circuits [OL89]. GENAC minimizes the layout width and, as a secondary optimization goal, it reduces routing channel density

and consequently cell height; it does not reorder the circuit graph. The authors do not claim that the resulting cell is of minimum height; therefore, their method addresses, but does not solve, the WHM problem. GENAC starts with pairing pMOS and nMOS transistors, followed by a hierarchical partitioning of these pairs into groups, such that only pairs within a group can potentially abut. For each such group, diffusion line-tracing is done. GENAC combines these separate diffusion lines to form a minimum-width layout. Diffusion lines are abutted so as to reduce routing channel density. Large transistors are folded, and finally the cell is routed. To evaluate the quality of the layout method in terms of area, Ong et al. have synthesized the 72 flip-flop cells in the AT&T standard cell library and compared GENAC's designs to the library cells. They show that GENAC's cells are slightly smaller in terms of width. However, GENAC's cells are usually one or two rows higher than the library cells, which are proportionally more important than diffusion gaps. Ong et al. admit that they are not able to generate all cells with equal or better area than those of the manually designed library cells. Their method is fast, taking only a few seconds to generate small cells on a medium-sized computer. Ong et al. take a mathematically rigorous approach to cell synthesis; they make some precise claims and prove them. They also generalize some important constraints, allowing general circuits, individually-sized transistors, and routing over diffusion. They give no worst-case time complexity for their method, but it seems to be exponential.

Wimer et al. report a functional cell layout algorithm that handles general CMOS circuits [Wi87], and minimizes the number of diffusion gaps for limited-size circuits. However, their method does not reorder the circuit graph and does not minimize layout height; therefore, it addresses the WHM problem. The steps of the method are these. First, p- and n-channel transistors are paired to share a vertical column. Diffusion chains are formed in an exhaustive fashion and assigned a cost reflecting routing density and performance. A set of chains is selected that minimizes the overall cost of the circuit; these chains are placed horizontally in the row so as to minimize interchain routing density. All $2^n n!$ permutations and orientations are generated if the number of chains is less than five; otherwise, a random sampling of all possible arrangements is done. The chains are then routed using a greedy channel routing heuristic. The layout program of Wimer et al. was developed for relatively small circuits with a few dozen transistors, for which it is reported to run "instantaneously." Although this method minimizes the number of diffusion gaps on small circuits, the cell width is not necessarily optimal, since the channel router may introduce additional columns into the layout that increase the cell width.

Bar-Yehuda et al. have presented a method for generating general CMOS circuits in a functional cell layout style with some additional constraints [Ba89]. Their method partially solves the WHM problem for layouts of small height. The layout

style has predefined routing channels with three horizontal tracks, one each for the n- and p-channel devices (two for transistor interconnections and one for power routing) and fixed-spacing polysilicon columns. The method minimizes the layout width for fixed circuits only; the circuit graph is not reordered. Layout height is the same for each layout, as determined by the layout constraints. The cell height constraints seem to be needed to ensure the computationally feasible of the method. The steps of the method are the following. The placement of transistors is generated in a depth-first branch-and-bound fashion, using as a bounding function some combination of (in order of importance) diffusion abutment, vertical alignment of polysilicon gates of transistors, and total wire length. A linear-time dynamic programming algorithm orients the transistors to minimize the number of diffusion gaps in the partial chain already generated. After the placement is complete, these chains are routed using a dynamic programming method that is exponential in the number of routing tracks. The method "reaches its limit" in the placement phase, so that larger circuits require partitioning. The method has been run on 300 cells of a manually generated library and compared to them. The manual cells were never smaller and often larger. The algorithm is of exponential-time complexity in the worst case.

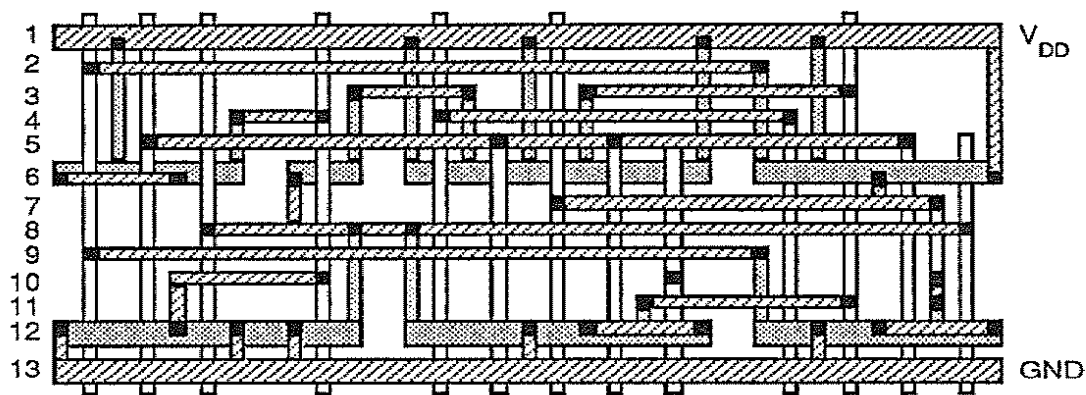


Fig. 2.7. A master-slave D flip-flop laid out by TOPOLOGIZER (after [KW85]).

Kollaritsch and Weste developed a heuristic program called TOPOLOGIZER for the layout of general multicell CMOS circuits [KW85]. They use the rule-based paradigm, which attempts to exploit the knowledge of expert designers as built-in design rules. TOPOLOGIZER allows the user to specify boundary constraints like aspect ratio and pin location, and can generate two or more rows of pMOS and

nMOS transistors. The placement of transistors is done using rules that iteratively improve an initial placement by pairwise exchange. The object is to maximize the number of short single-layer routing connections. The results from TOPOLOGIZER range from 8% better to 82% worse than production-quality manual layouts based on four comparisons. Rule-based methods are not generally amenable to precise analysis; optimality claims are almost impossible to verify. Therefore, the only way to evaluate the quality of rule-based methods is to compare the area of their layouts to those of known quality. Figure 2.7 shows the only multicell example the authors present in [KW85], a master-slave D flip-flop design, which is 3% smaller than the corresponding manual layout. In Chapter 6, we show that our optimal method produces a layout that is 39% smaller than the layout in Fig. 2.7.

Hill presents a heuristic method called Sc2 for the layout generation of arbitrary circuits in a functional cell style that uses single or multiple contacts [Hi85]. First, it splits tall transistors into shorter ones. It then pairs the pMOS and nMOS transistors and subsequently treats them as vertical neighbors. Next, it horizontally orders these pairs within a row. The transistor pairs are then oriented to increase diffusion abutment using a branch-and-bound algorithm. A heuristic method is employed for intercell routing. Sc2 produces layouts about as small as standard-cell layouts, but not as small as full-custom designs.

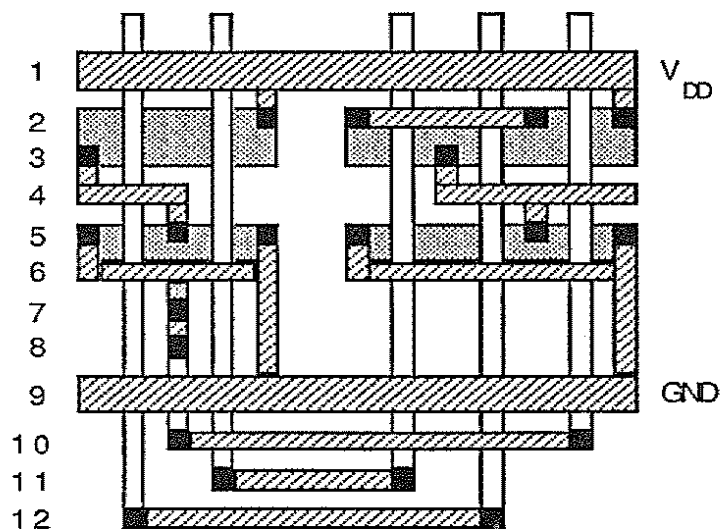


Fig. 2.8. An exclusive-OR cell generated by SOLO (after [BA88]).

Baltus and Allen report a heuristic layout method called SOLO that produces layouts in the functional cell style [BA88]. Figure 2.8 shows a layout of an

exclusive-OR gate produced by SOLO. Its most distinguishing feature is that it is more general than most of the other functional cell methods in that it can handle nonseries-parallel circuits, transmission gates, boundary constraints, and multiple functional cells. SOLO partitions a circuit into functional cells and other closely related transistor groupings. It uses general placement and routing methods. The resulting method is indeed very general. However, such generality is not without its price; a large penalty in area is paid, as we shall show in Chapter 6, where we present a layout generated by our algorithm with 80% less area than that of Fig. 2.8.

Shiraishi et al. report a heuristic layout system called MAGICAL that also handles arbitrary CMOS circuits [Sh88]. This method uses two layers of metal. Its main goals are, in order of importance, to maximize the number of vertical transistor gate connections, to maximize diffusion abutment and to minimize total wiring length; thus it addresses the WHM problem. The main steps of the method are as follows. First, a connection graph is created that represents all the connections between terminals of the transistors. Diffusion "islands" are created from the transistors: series diffusion islands and parallel diffusion islands from those transistors strictly in series or in parallel, respectively. Next, the islands of nMOS transistors are paired with those of pMOS with which they are connected to form a diffusion island couple. The transistor pairs are placed in a row so as to minimize total wiring length; they are also rotated to maximize diffusion abutment between pairs, if this rotation does not increase wiring length. Pairs that are part of the same functional cell are recognized and placed together. Routing is done in the five channels defined for the cell. Two example layouts presented in [Sh88] are slightly smaller than hand designs; however, the layouts produced are not necessarily optimal in height, width or area.

Chen and Chow present a layout heuristic for general CMOS circuits that utilizes two layers of metal and multiple contacts. It simultaneously attempts to maximize diffusion abutment and minimize wiring area [CC89]. First, transistors are paired based on a hierarchy of priorities, the most important of which is sharing a polysilicon gate column. Next, transistor pair placement is done guided by a set of graph models. A diffusion-sharing graph is transformed into an adjacency graph. The method finds all hamiltonian paths in the adjacency graph by a branch-and-bound algorithm, evaluating each for its area cost. The transistor pairs are then optimally rotated in linear time. Finally, routing is performed. The authors compare two manual layouts having 18 transistors each to those of their method. The hand-crafted design is 8% larger in one example and 1% smaller in the other.

Sun presents a heuristic layout system for high performance circuits [Su89], that does not reorder the circuit graph. This method employs multiple contacts, and two metal layers. Sun's method consists of the following steps. First, pMOS and nMOS transistors are paired. Diffusion chains are created, followed by linear

ordering of the chains in a row. Finally, some transistors are placed in under-utilized routing areas. The method's synthesized layouts range from 17% smaller to 53% larger than the ten manual examples presented.

Poirier has developed a layout generator called *Excellerator* that handles arbitrary CMOS circuits [Po89]. This method uses two levels of metal for interconnection. The optimization goals are to maximize the number of transistors sharing a gate signal, to maximize diffusion abutment, and to minimize routing density and wire length. Although the method appears to produce very compact layouts, it does not guarantee minimal width or height. *Excellerator's* layouts are from 25% smaller to 6% larger than the manual layouts to which it is compared. The reported run times range from 22 seconds to over an hour on a medium-sized computer for circuits with up to 40 transistors.

The WHRM Problem. Mano et al. [Ma85] report a layout system that partitions an input logic expression into functional cells and registers. It uses the technique of [Uv81] to generate functional cell layouts, and places and routes the cells in a standard cell layout style. We compare a layout produced by the method of Mano et al. to one produced by our optimal method in Chapter 6.

Heeb has proposed a functional cell layout system [He87], which extends the heuristic in [Uv81]. He extends the latter method to handle intracell routing of the source and drain terminals of minimum-sized transistors using only metal wires. This involves solving the single-row routing problem for functional cells [Le90]. The goal of the method is to generate a layout that can be routed only in metal and is close to optimal width. Heeb's heuristic, like that of [Uv81], does not find optimal-width layouts in a significant number of cases; the reasons for this will be discussed fully in Chapter 3. In a related paper, Heeb and Fichtner present a heuristic multicell layout scheme that uses multiple rows of functional cells [HF87], based upon their enhanced version of the Uehara-vanCleemput heuristic method.

Domic et al. describe a heuristic layout generator *CLEO* for arbitrary CMOS circuits [Do89]. *CLEO* produces layouts of one or more horizontal rows of vertically aligned transistors. This method employs multiple contacts. Its layouts are of comparable area to hand-crafted designs, and it reorders the circuit graph under unspecified circumstances. Thus *CLEO* attempts to solve the WHRM problem. The method may be summarized as follows. First, the circuits are placed in a given row using a recursive min-cut method [Le90]. A cell layout for each circuit is obtained next. A predefined library cell is used for standard logic circuits and registers, and a layout is generated for functional cells. The individual cells are manipulated to maximize the amount of diffusion abutment by selecting the best cell and orienting it to match its neighbors; gate inputs are permuted to avoid wire crossing when routing cells. A simulated annealing algorithm [Le90] is used for this stage. The

next stage involves the formation of diffusion chains by merging transistors via diffusion abutment. The chains are placed horizontally in their rows to minimize signal routing distance. Special-purpose routers that handle two layers of metal are used. CLEO processes about 75 transistors per minute for production circuits. The goals of this tool are generality and flexibility along with small area, which it reportedly achieves.

2.2.2 Dynamic Cells

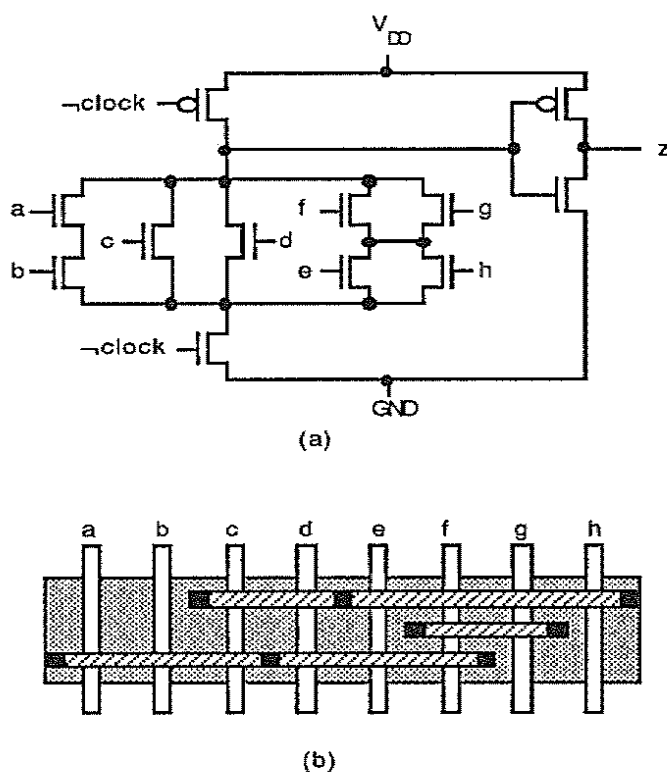


Fig. 2.9. A one-dimensional domino-logic dynamic CMOS cell: (a) transistor circuit; (b) layout of the pulldown subcircuit (after [MO88]).

Now we briefly review layout methods for one-dimensional functional cells that employ dynamic CMOS circuits. As mentioned in Section 1.2.5, dynamic circuits replace the pullup subcircuit by a fixed circuit controlled by a clock; therefore, the cell function is determined by the pulldown subcircuit. Consequently, the main focus is on layout of the pulldown circuit, which typically consists of a single row of

transistors connected by diffusion. The domino logic circuit in Fig. 2.9(a) is an example of a dynamic circuit realizing the function $z = (a * b) + c + d + ((f + g) * (e + h))$.

McMullen and Otten present an algorithm for generating the layout of the pulldown MOS subcircuit of a dynamic series-parallel CMOS circuit [MO88]. This method reorders the circuit graph to minimize the diffusion gaps, but does not handle cell height; therefore, it solves the dynamic case of the W problem. The style of layout is a one-dimensional transistor cell consisting of a horizontal row of diffusion, with vertical polysilicon columns and horizontal metal tracks, as illustrated in Fig. 2.9(b). The authors propose a linear-time layout algorithm that guarantees a minimum-width cell. In generating 1500 layouts, McMullen and Otten found that only two of them required more than three metal tracks for intracell routing.

Lengauer and Müller have presented a linear-time method for series-parallel dynamic CMOS circuits [LM88]. It is a heuristic for solving the WHR problem, but can also be classified as an exact algorithm for the WR problem. The authors present a dynamic programming algorithm that optimally reorders the circuit graph to minimize cell width; it is similar to the reordering algorithm of Nair et al. [NB85]. Lengauer and Müller do not describe their heuristic for reducing the height of their layout, but state that "in applications where channel density is the dominating issue such a (heuristic) approach may not be sufficient, and the exact solution to the corresponding optimization problem may be indicated." In Chapter 5, we present an exact solution for the related problem involving static CMOS cells.

Chakravarty et al. have reported a linear-time algorithm for the W problem for arbitrary dynamic circuits [Ch90]. They also present an algorithm that finds the minimum-height layout for a given linear placement of transistors. Carlson et al. present an algorithm for solving the WR problem for series-parallel dynamic circuits whose time complexity is linear in the number of transistors [Ca90].

2.3 CRITIQUE OF PRIOR WORK

We now evaluate and categorize the prior layout methods according to the area-minimization problems they address, in order to identify the previously unsolved or only partially solved problems; this provides motivation for our research. We conclude by discussing two trends in the layout literature that tend to discourage the search for exact solutions.

Optimality of Prior Methods. Figure 2.10 summarizes prior layout methods in terms of the problem classification of Fig. 2.4. Most of the cited methods use heuristics that do not guarantee a solution of minimum width or height.

Reference	Layout problem	Width minimization	Height minimization	Circuit reordering	Number of cells	Time complexity	Circuit type
[NB85]	W	Exact	None	No	Single	$O(n)$	SP
[HS88]	W	Exact	None	No	Single	$O(\log n)$	SP
[NB85]	WR	Exact ¹	None	Yes	Single	$O(n)$	SP
[CC87]	WR	Exact ¹	None	Yes	Single	?	Any
[Uv81]	WR	Not exact	None	Yes	Single	$O(n)$	SP
[KK88]	WR	Not exact	None	Yes	Single	$O(n^3)$	SP
[CH88]	WR	Not exact	None	Yes	Single	?	SP
[LN90]	WR	Not exact	None	Yes	Single	$O(n)$	SP
[NH85]	WH	Not exact	Not exact	No	Single	?	Any
[CC89]	WH	Not exact	Not exact	No	Single	?	Any
[Su89]	WH	Not exact	Not exact	No	Single	?	Any
[LC89]	WHR	Exact ²	Exact ?	Yes	Single	?	SP
[Ma89]	WHR	Exact ¹	Exact ?	Yes	Single	?	SP
[MD88]	WHR	Not exact	Not exact	Yes	Single	$O(n)$	SP
[HH89]	WM	Exact	None	No	Multiple	Expon.	Any
[OL89]	WHM	Exact	Not exact	No	Multiple	Expon.	Any
[HH90]	WHM	Exact	Not exact	No	Multiple	Expon.	Any
[Wi87]	WHM	Exact ?	Not exact	No	Multiple	?	Any
[Ba89]	WHM	Exact ³	Fixed	No	Multiple	Expon.	Any
[KW85]	WHM	Not exact	Not exact	No	Multiple	?	Any
[Hi85]	WHM	Not exact	Not exact	No	Multiple	?	Any
[BA88]	WHM	Not exact	Not exact	No	Multiple	?	Any
[Sh88]	WHM	Not exact	Not exact	No	Multiple	?	Any
[Po89]	WHM	Not exact	Not exact	No	Multiple	Expon.	Any
[Ma85]	WHRM	Not exact	Not exact	Yes	Multiple	?	Any
[HF87]	WHRM	Not exact	Not exact	Yes	Multiple	?	Any
[Do89]	WHRM	Not exact	Not exact	Limited	Multiple	?	Any
[LC90]	WHRM	Not Exact	Not exact	Yes	Multiple	?	Any?

(a)

Reference	Layout problem	Width minimization	Height minimization	Circuit reordering	Number of cells	Time complexity	Circuit type
[Ch90]	W	Exact	None	No	Single	$O(n)$	Any
[MO88]	WR	Exact	None	Yes	Single	$O(n)$	SP
[Ca90]	WR	Exact	None	Yes	Single	$O(n)$	SP
[LM88]	WR	Exact	None	Yes	Single	$O(n)$	SP
[LM88]	WHR	Exact	Not exact	Yes	Single	?	SP

(b)

- 1: Only if some reordering of the circuit has a d-euler trail.
 2: Restricted to series-parallel circuits with three or fewer logic levels.
 3: Only if the layout height is limited to some small number.

Fig. 2.10. Summary of prior functional cell layout methods: (a) static and (b) dynamic designs.

Layout problem	Width minimization	Height minimization	Circuit reordering	Number of cells	Previously solved for all practical-sized SP circuits
W	Exact	No	No	Single	Yes
WR	Exact	No	Yes	Single	Incompletely (static case) Yes (dynamic case)
WH	Exact	Exact	No	Single	No
WHR	Exact	Exact	Yes	Single	Incompletely
WM	Exact	No	No	Multiple	Yes
WRM	Exact	No	Yes	Multiple	No
WHM	Exact	Exact	No	Multiple	Incompletely
WHRM	Exact	Exact	Yes	Multiple	No

Fig. 2.11. Classification of area-minimization problems and their status.

The solution status of these problems, solved, partially solved, or yet unsolved by prior methods, is given in Fig. 2.11.

The prior methods listed in Fig. 2.10 that claim optimality address five of the eight problem classes. The W problem, namely, exact width minimization for fixed-circuit single cells is solved by two methods [NB85, HS88] for static circuits and by one method for dynamic circuits [Ch90]. A second group of methods addresses the WR problem; three methods [MO88, LM88, Ca90] solve this problem completely for dynamic cells, and one method [NB85] for a subset of all practical-sized static CMOS circuits. A third group partially solves the WHR problem; these methods appear to find exact width and height for a subset of all single-cell practical-sized reordered static CMOS circuits [LC89], [Ma89]. The methods of [LC89, Ma89] are limited to 18% and 56% of all practical-sized circuits, respectively. One method solves the WM problem, achieving exact width minimization for fixed-circuit multiple cells [HH89]. Another method partially solves the WHM problem, finding exact width and height for multiple cells, where the cell height is fixed at a small number [Ba89].

As summarized in Fig. 2.11, the W and WM problems are solved, the WR, WHR and WHM problems are partially solved, and the WH, WRM and WHRM problems have not been addressed by any prior method. It should be noted that a solution to a given problem also solves all its subproblems; for example, a solution to the WRM problem also solves the W, WR and WM problems, since they are restricted cases of it. This book presents exact algorithms that solve all eight problems for all practical-sized dual series-parallel circuits, and the W problem for all practical-sized dual planar circuits.

Misuse of Term "Optimal." In the research literature on the subject of functional cell layout, as in the CAD literature generally, the term "optimal" has often been used with different and misleading meanings. In their seminal paper [Uv81], whose title is "Optimal layout of CMOS functional arrays," Uehara and vanCleemput use the term to describe a layout method that has the potential to produce an optimal solution; when it does find such a solution, the method is able to recognize it as optimal. Those who have not read the paper completely may conclude that it presents an exact algorithm for area minimization. In fact, the authors emphatically state late in the paper that "this heuristic algorithm does not necessarily give the optimal layout. However, if the resulting sequence has no separation areas [i.e., diffusion gaps], it is the real optimal solution." Following Uehara and vanCleemput, several researchers have adopted this looser use of the term optimal. One example states that "in 1981, Uehara and vanCleemput presented an algorithm to *optimally* place CMOS functional cells" (emphasis ours) [He87]. Another example is [KK88], whose title is "A fast heuristic for optimal CMOS functional cell layout generation." The paper itself only claims that its heuristic finds "a *nearly* minimum number of Euler paths" (emphasis ours). In similar fashion, another paper states that Uehara and vanCleemput "concentrated on ... finding an *optimal* solution" (emphasis ours) [Ma89]. The title of this paper is "A new approach to optimal cell synthesis," but the method presented seems to find an optimal solution only under certain conditions. Another paper describes its method as "a new methodology which allows the binding of transistor connections to be delayed until an *optimal gate ordering* is determined," (emphasis ours) [CH88] but, in fact, the method is a heuristic. Other researchers in this field have maintained the traditional, more rigorous, use of the term optimal in reference to their layout methods, using it to denote that their methods *always* finds an optimal solution [LM88], [OL89].

These different uses of the term "optimal" can leave readers wondering whether exact algorithms have been found or not. Unless one reads the literature very carefully, one can easily get the erroneous impression that various layout problems have indeed been solved exactly. When almost every paper uses the term "optimal" in reference to its method, it is no wonder that true optimality is less than fully appreciated!

Overemphasis on Polynomial Methods. Another trend in CAD that discourages the pursuit of exact algorithms is overemphasis on the importance of polynomial-time complexity with the concomitant aversion to any nonpolynomial-time algorithm. This, in part, reflects the tendency of algorithm theorists to describe problems of nonpolynomial-time complexity as "intractable" [GJ79]. For instance, Uehara and vanCleemput propose an exact solution to the functional cell reordering

problem, comment on its high complexity, and consequently use it as a justification for their linear-time heuristic [Uv81]. Similarly, other researchers state in reference to this problem, "the search for an arrangement of the transistor pairs that minimizes the area (or equivalently minimizes the diffusion gaps) is a difficult problem, and *very likely intractable*. For this reason, Uehara and vanCleemput introduced a simple heuristic algorithm to align the gates" [MD88] (emphasis ours). They likewise propose a presumably polynomial-time layout heuristic. Another paper, based on a worst-case analysis, states that the exact solution to the reordering problem just discussed is of exponential-time complexity $O(n^n)$ [KK88]; consequently a polynomial-time heuristic is developed to solve it.

The impression is given that polynomial solutions are equated with efficient ones, and nonpolynomial solutions with inefficient ones. However, for any particular problem domain, depending on the limits on the problem size, and the degree and coefficients of the polynomial, a given exponential algorithm can be better or worse than a polynomial-time algorithm. Therefore, the tractability of an algorithm depends both on its time complexity and the size of the problems to which it is applied.

This pessimism is also found in design-related textbooks. One such book asserts that "In general, ... the combinatorial problems involved (are) too time consuming for us to expect to find optimal designs very often. Thus we are usually content with algorithms that produce a 'good' even if not optimal, solution..." [UI84]. Of course, in general, we agree. It also postulates that "almost all the optimization problems we encounter in VLSI design, as elsewhere, are NP-complete or worse." (It is strongly believed by algorithm theorists that an NP-complete problem cannot be solved by an algorithm with polynomial worst-case time complexity [GJ79].) This sort of pessimistic statement repeated with regularity can discourage researchers from looking for exact algorithms to VLSI problems. If one is taught that something rarely if ever exists, it is not likely one will be looking for it with any hope of success. However, we will demonstrate that our CMOS cell layout algorithms, despite their nonpolynomial-time complexity, are indeed computationally feasible for practical-sized layout problems.

CHAPTER VI

CELL ARRAY WIDTH AND HEIGHT MINIMIZATION

In this chapter, we extend our layout theory to arrays of two or more cells. We first define a practical cell-array layout style and present an algorithm *HRM-TrailTrace* that generates cell arrays optimal in width and height in our layout style. We prove this optimality claim and present experimental results to substantiate it [MH91].

6.1 LAYOUT PROBLEM

We define the cell-array layout style and a corresponding layout optimization problem in this section. A one-dimensional *cell array* is a linear array of two or more interconnected functional cells. The cell-array layout style is illustrated by the two-cell example in Fig. 6.1. This style is defined by the 16 assumptions of Fig. 5.2 plus two new ones, *assumptions 17* and *18*, stated as follows:

17. Assign all intercell connections to metal wires in the U-region.
18. Group all transistors of each cell together.

We discuss below these assumptions, along with other key assumptions of Fig. 5.2, in the context of cell-array layout.

According to assumption 7, diffusion is used to make vertical interconnections between source or drain transistor terminals and horizontal metal. Hence, in our cell-array style, vertical diffusion is used to connect the output of a cell to horizontal metal for intercell routing. This is illustrated in Fig. 6.1, where vertical diffusion is used between columns *d* and *e* in row 3 to route the output of cell 1 (signal *i*) from row 4 to row 3. As in Chapter 5, we emphasize that vertical diffusion could be replaced with metal in a two-metal routing scheme. Intercell routing is done solely in the U-region according to assumption 17, and is shown in rows 2 and 3 in Fig.

6.1. Diffusion abutment of neighboring functional cells via the V_{DD} and GND terminals is used whenever possible, thus reducing the width of the combined cell by one column width. The cells of Fig. 6.1 abut in this way between columns f and g in rows 4, 5 and 7. The 12 transistors of cell 1 in Fig. 6.1 are grouped together to the left of the eight transistors forming cell 2 in the same figure, according to assumption 18.

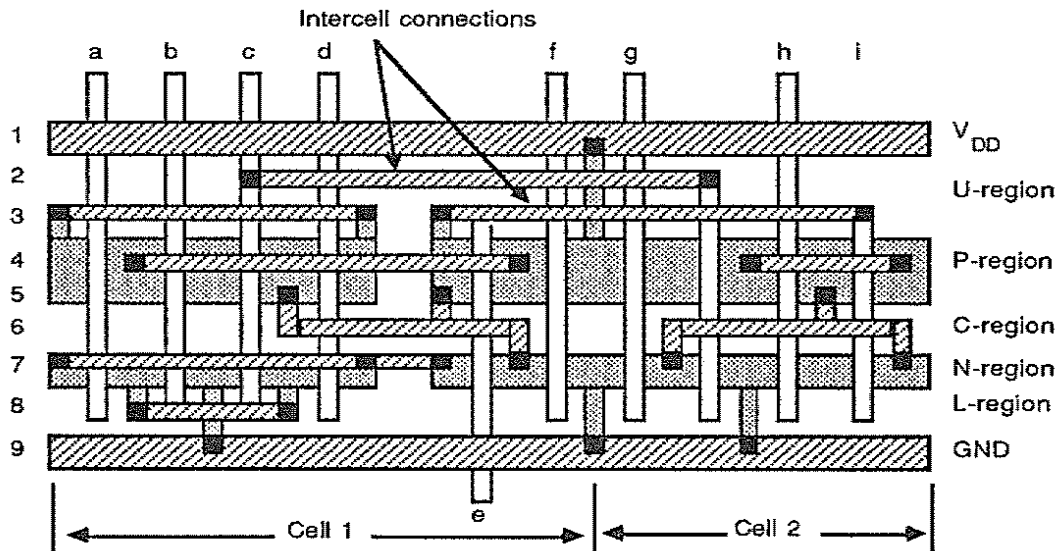


Fig. 6.1. An illustration of our cell-array layout style.

We now discuss the justification for assumption 17 in the context of other functional cell layout methods. Although no other method limits intercell routing solely to the U-region, many use this region extensively, for example the standard cell styles of [Uv81, Ma85, MD88, Ma89, CH88]. Others use the C-region solely for vertical intracell connections, as we do, relegating all horizontal connections to the U- and L-regions [HH90, BA88]. TOPOLOGIZER uses both the U- and C-regions for intercell routing [KW85]. Sc2 confines all intracell and intercell routing to the C-region [Hi85]; this is a precedent for using a single routing region in a cell-array layout style. Therefore, this assumption is in keeping with the spirit of many other layout systems.

Based on the foregoing style constraints, we now define the corresponding cell-array layout optimization problem formally. The cell-array height H is given by the following equation

$$H = \max\{U_i\} + P + C + N + \max\{L_j\}$$

where P , C and N are the number of rows in the P-, C- and N-regions of the entire cell array. These numbers are the same for all cells, according to assumption 10 of Fig. 5.2; this is shown in Fig. 6.1. $\max\{U_i\}$ and $\max\{L_j\}$ are the numbers of rows in cells i and j with the largest U- and L-regions, respectively, as shown in Fig. 6.1. The cell-array width W is given by the equation

$$W = T + G + 1$$

where T is the number of transistors in either the pullup or pulldown subcircuit of all the functional cells in the array, and G is the number of diffusion gaps present. We define the *WHRM layout minimization problem* as follows: given a set of dual series-parallel circuits with circuit graphs $\{M_1, \dots, M_n\}$ and placement P of the n cells in the cell array, find a layout in the style of Fig. 6.1 with minimum H from among all layouts of minimum W , for any set of reorderings $\{M'_1, \dots, M'_n\}$ of $\{M_1, \dots, M_n\}$.

6.2 HRM-TRAILTRACE ALGORITHM

In this section we present the algorithm *HRM-TrailTrace*, a generalization of *HR-TrailTrace* for multiple cells, that solves the WHRM optimization problem exactly. First, a brief description of the algorithm is given, followed by a design example. We then prove the optimality of *HRM-TrailTrace*. Finally, we analyze the algorithm's time complexity.

6.2.1 Description

The algorithm that computes the optimal one-dimensional cell array as described above for a given placement of functional cells is given in Fig. 6.2.

A brief description of *HRM-TrailTrace* follows. First, the composition expression for each cell is read. Next, the input and output signals that cross the left and right boundaries of each cell are assigned to the proper cell boundary, thus decomposing the cell-array layout problem into several single-cell layouts with boundary constraints. The procedure `Generate_min_height_layouts2(cell,H)` generates each nondominated optimal cover and layout for a given cell. This procedure is called for each cell in the array. It is essentially the same as the *HR-TrailTrace* procedure in Fig. 5.5, with the following difference. `Generate_cell_layouts` in *HRM-TrailTrace* calls the procedure `Generate_min_height_layouts2` instead of `Generate_min_height_layouts`, which *HR-TrailTrace* uses. `Generate_min_height_layouts2` is `Generate_min_height_layouts` shown in Fig. 5.5 with the addition of statement 1. This new

procedure permutes the isomorphic subgraphs affecting layout height in the cell-array style of Fig. 6.1. (A complete description of *Generate_min_height_layouts* is given in Chapter 5.)

```

procedure Find_optimal_cell_array(HV,Smallest_H);
begin
  for each j do {initialize MHV} MHV:= HV1,j;
    for cell:= 2 to n do
      begin
        for each j do
          for each k do {Calculate next vector}
            begin
              HVtemp := lub{MHVk, HVcell,j};
              Save HVtemp in MHVtemp if not dominated or equalled by a vector in
              MHVtemp;
            end;
          for each vector in MHVtemp do {Update MHV} MHV:= MHVtemp;
        end;
      Smallest_H:= ∞;
      for each k do {Find smallest overall layout}
        if MHVk < Smallest_H then Smallest_H:= MHVk;
      end; { Find_optimal_cell_array}

procedure Generate_cell_layouts(Cell,HV); {This is essentially HR-TrailTrace}
begin
  Construct composition tree T from cell's M;
  Identify_isomorphic_subgraphs;
  Graph_CTC:= Node_CTC(T);
  if any minimum size cover has a VDD/GND terminal pair then
    Min_covers:= Retain for layout all trail covers of min. size from Graph_CTC;
  else Min_covers:= Retain for layout all trail covers of min. size and min. size+1
    from Graph_CTC;
  Generate_min_height_layout2(Cell, Min_covers, HV);
end; { Generate_cell_layouts}

procedure HRM-TrailTrace;
begin
  Input_cells;
  Calculate_boundary_signals;
  for each Cell do Generate_cell_layouts(Cell,HV)
  Find_optimal_cell_array(HV,Smallest_H);
end; { HRM-TrailTrace }

```

Fig. 6.2. The *HRM-TrailTrace* algorithm.

Finally, the height vectors (U_i, L_i) produced for each cell by `Generate_cell_layouts` are processed by `Find_optimal_cell_array`, which computes the optimal cell-array layouts from the height vectors of the individual cells. In `Find_optimal_cell_array`, the following least upper bound (lub) operation is performed on two adjacent layouts to determine the minimum height of the layout resulting from combining them. Assume the height pair of one layout is (A, B) and that of the other is (C, D) . Then their least upper bound is

$$\text{lub}\{(A, B), (C, D)\} = (\max\{A, C\}, \max\{B, D\})$$

Let H_{ij} be the j th height-pair of cell i . Let H_j be the j th height-pair of a undominated layout composed of the $i-1$ cells to the left of cell i . The overall layout with least height (smallest_H) for the n functional cells is calculated by procedure `Find_optimal_cell_array`, as shown in Fig. 6.2.

6.2.2. Design Example

We now illustrate the *HRM-TrailTrace* algorithm by an example. Figure 6.3 shows a circuit composed of three series-parallel subcircuits C_1 , C_2 and C_3 and their graph models M_1 , M_2 and M_3 , respectively. The algorithm's input data consists of these three circuits, each of which will be implemented as a cell in a 3-cell array. The procedure `Calculate_boundary_signals` of Fig. 6.2 assigns signals shared by two or more cells to the appropriate side(s) of the cells. For example, cell 1 implementing C_1 shares signal b with cell 2 implementing C_2 ; therefore, signal b is assigned to the right side of cell 1 and the left side of cell 2, as shown in Fig. 6.4.

The procedure `Generate_cell_layouts` is applied to each cell successively. This procedure is much like *HR-TrailTrace* described in Chapter 5. The composition tree T is built for the cell, and is used to identify all isomorphic subgraphs of the cell used later in the layout procedure. In the case of cell 2, for example, edges b and c are its isomorphic subgraphs. The PMCTC called `Graph_CTC` in Fig. 6.2 is calculated for the cell by procedure `Node_CTC`. `Graph_CTC` is transformed into the set `Min_covers` by pruning it of all covers of greater than minimum size, with the following exception. If no cover of minimum size is of a type with an endpoint pair (S, N) , as in the type $(S, N)/(S, S)$, which represents V_{DD} and GND in the circuit, then all covers that are of minimum size plus one with an (S, N) endpoint pair are also included in `Min_covers`. This exception ensures that the minimum height cell array for all arrays of minimum width is found, since these larger covers also can lead

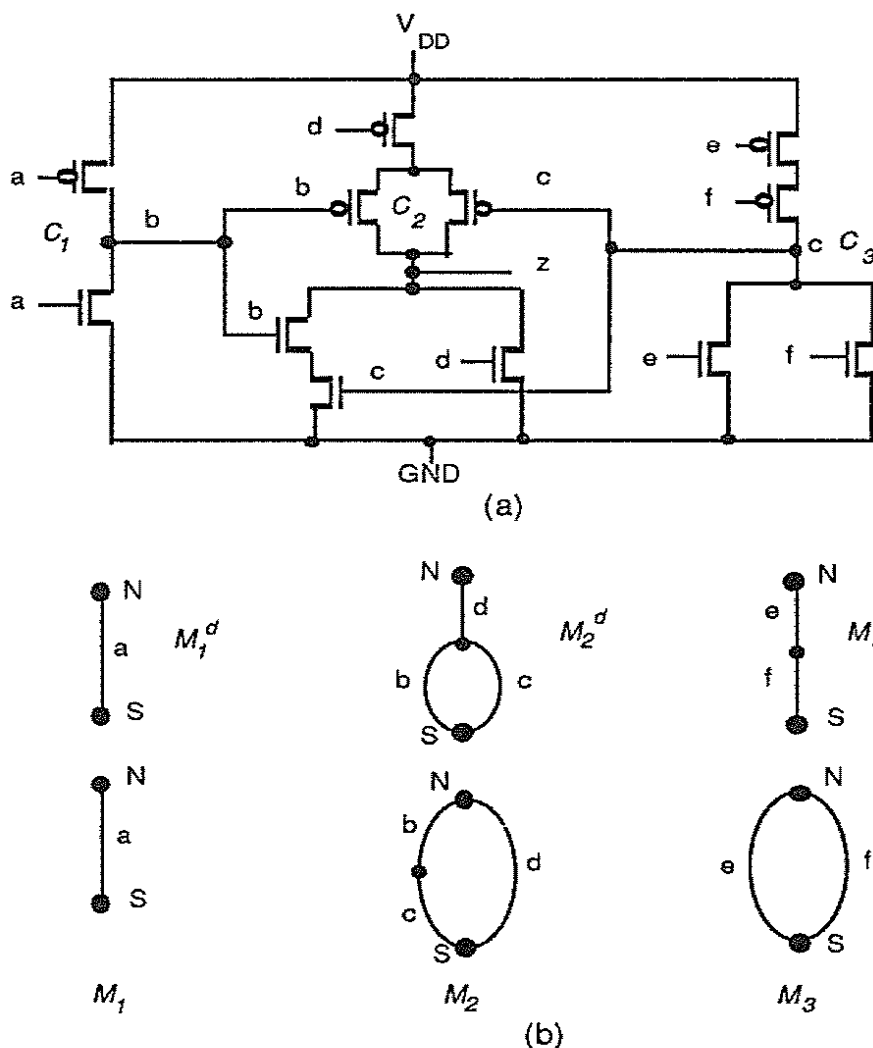


Fig. 6.3. An implementation of the function $z = -(+ d*(\neg a)\neg(+ef))$: (a) circuit and (b) graph model.

to minimum-width cell-array layouts by eliminating a diffusion gap between neighboring cells. Figure 6.5(a) shows two cells: cell 1, a layout with V_{DD} and GND contacts on its right side corresponding to the (S,N) terminals of its cover, and cell 2, a minimum-width layout of a circuit cover without the (S,N) terminals. Cells 1 and 2 of Fig. 6.5 form a cell array of Fig. 6.5(b) with the resulting gap between

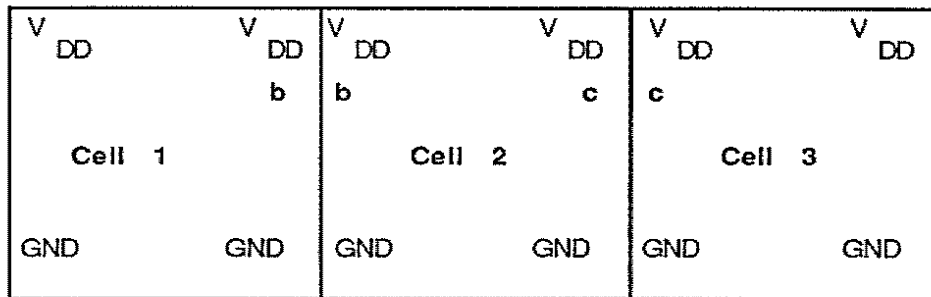


Fig. 6.4. The 3-cell array illustrating the intercell boundary signals.

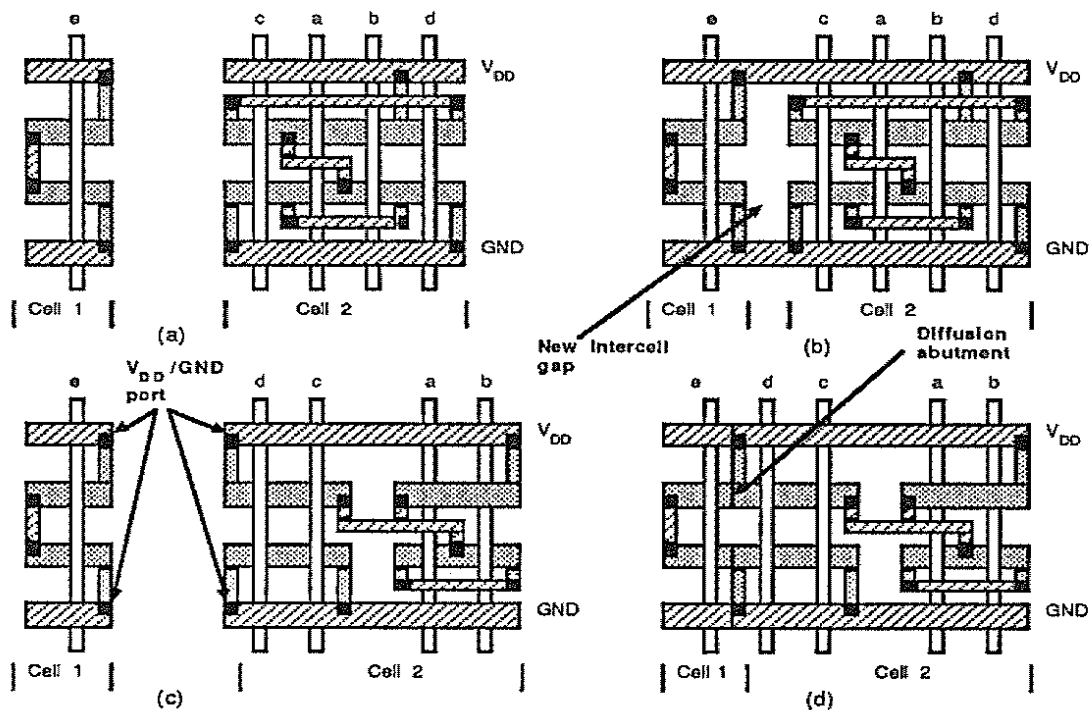


Fig. 6.5. The abutment of two cells forming an array: (a-b) intercell gap required; (c-d) diffusion abutment needing no gap.

them shown in the figure. Cell 2 in Fig. 6.5(c) is a layout of the same circuit as cell 2 in Fig. 6.5(a), but its cover has the (S,N) terminals defined by the V_{DD} and GND contacts on its left boundary. Note that cell 2 of Fig. 6.5(c) is wider by one column

than the cell 2 in Fig. 6.5(a). However, the terminals of cell 1 abut the terminals of cell 2, forming the 2-cell array of Fig. 6.5(d), with the same width as the array in Fig. 6.5(b).

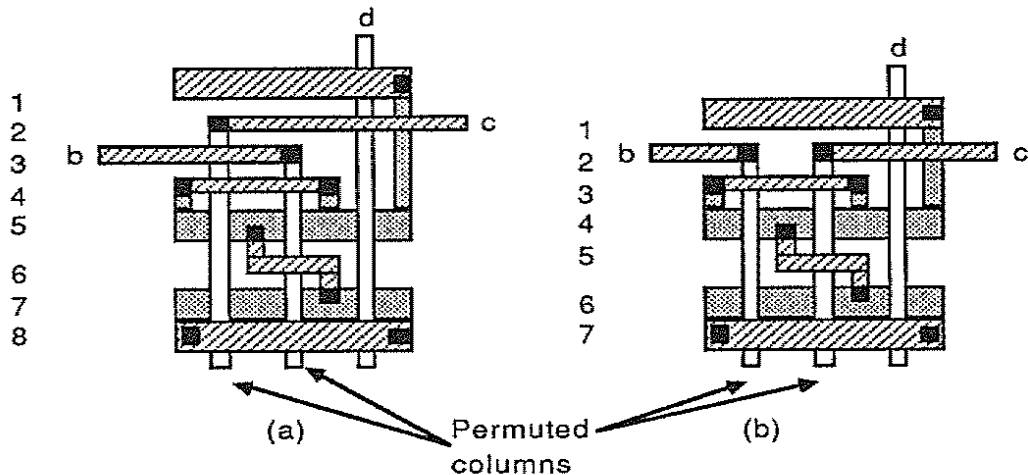


Fig. 6.6. Two layouts of circuit C_2 of Fig. 6.3(a) showing the permutation of isomorphic subgraphs and its affect on cell height: (a) cell of height 8; (b) cell of height 7.

At this point, the procedure `Generate_min_height_layouts2` is applied to the covers of the circuit; an outline of this procedure appears in Fig. 5.5. Here we emphasize its differences with `Generate_min_height_layouts`. Layouts are generated for each cover of `Min_covers` by breaking closed trails, and permuting and reflecting the trails, as in *HR-TrailTrace*, with the following exception. Isomorphic subgraphs, which are not permuted when covering the reorderings of the circuit in procedure `Node_CTC`, are permuted at this point. Figure 6.6(a) illustrates this permutation for edges b and c of C_2 . It shows that the layout of cell 2 has a height vector $(U, L) = (4, 2)$ if b and c are not permuted, but the layout of Fig. 6.6(b) has the smaller height vector $(3, 2)$ when b and c are swapped. *HRM-TrailTrace* permutes all isomorphic subgraphs in the layout generation procedure in an efficient manner, recognizing cases where such permutations cannot affect cell height. For example, if the edges b and c of Fig. 6.6(b) were connected to the same side of the cell 2, instead of different sides, then their permutation could not affect cell height; therefore, these edges would not be permuted. This analysis keeps the number of such permutations small. The set HV_i of nondominated height vectors is calculated over all the layouts of a cell i .

Height vector	HV ₁	HV ₂	HV ₃
1.	(3, 2)	(4, 2)	(3, 2)
2.	(2, 3)	(3, 4)	(2, 3)

Fig. 6.7. A set HV of height vectors for the three cells of Fig. 6.4.

After the procedure `Generate_cell_layouts` has calculated HV for each cell, `Find_optimal_cell_array` selects layouts from all the cells that together form the optimal layout for the cell array. Assume HV containing the nondominating height vectors for the three cells of Fig. 6.4 is as shown in Fig. 6.7. An optimal selection of layouts represented by these height vectors in Fig. 6.7 for the 3-cell array is calculated as follows. The set of optimal height vectors for the subarray of cells 1 through n is denoted MHV_n . For the case of cell 1, $MHV_1 = HV_1$. For MHV_n , the lub is calculated, pairing each vector of MHV_{n-1} with each vector of HV_n . For example, for $n = 2$,

$$\begin{aligned}(4, 2) &= \text{lub} \{(3, 2), (4, 2)\} \\ (3, 4) &= \text{lub} \{(3, 2), (3, 4)\} \\ (4, 3) &= \text{lub} \{(2, 3), (4, 2)\} \\ (3, 4) &= \text{lub} \{(2, 3), (3, 4)\}\end{aligned}$$

Since $(4, 2)$ dominates $(4, 3)$, MHV_2 is $\{(4, 2), (3, 4)\}$, where neither vector dominates the other. For MHV_3 , the calculation is as follows

$$\begin{aligned}(4, 2) &= \text{lub} \{(4, 2), (3, 2)\} \\ (4, 3) &= \text{lub} \{(4, 2), (2, 3)\} \\ (3, 4) &= \text{lub} \{(3, 4), (3, 2)\} \\ (3, 4) &= \text{lub} \{(3, 4), (2, 3)\}\end{aligned}$$

Therefore, MHV_3 is $\{(4, 2), (3, 4)\}$. The height vector $(4, 2)$ represents the smaller of the two cell-array layouts. This minimum-height cell array consists of the layouts corresponding to height vector 1 of each cell of Fig. 6.7. These three cell layouts result in minimum height for the cell array, namely $4 + 1 + 2 = 7$, since the C-region is always of height one. The final optimal cell-array layout is shown in Fig. 6.8, where cells 2 and 3 are connected by diffusion abutment.

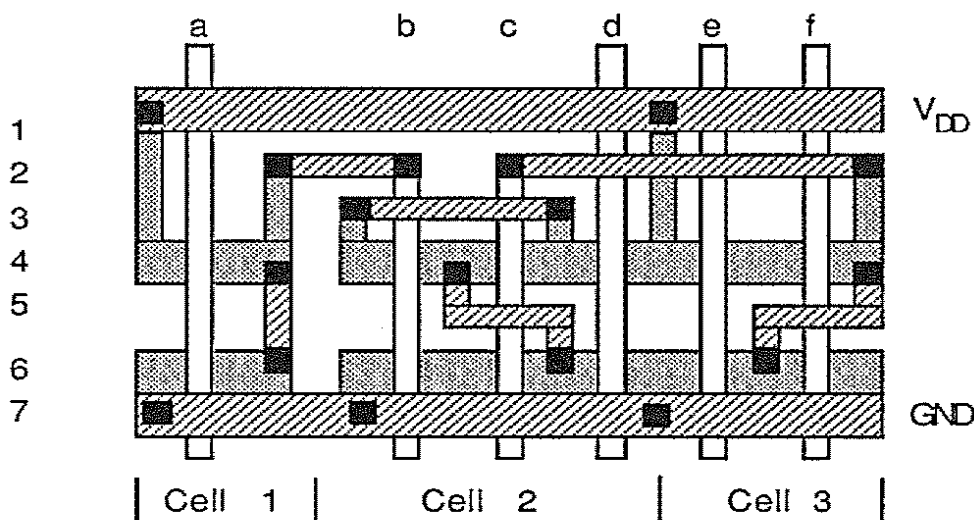


Fig. 6.8. Cell-array layout of the circuit of Fig. 6.3(a).

6.2.3 Optimality

We prove the optimality of the *HRM-TrailTrace* algorithm in this section. Three lemmas are developed as the basis for proving Theorem 6.1, which asserts the main optimality claim.

Lemma 6.1: The algorithm *HRM-TrailTrace* computes the set HV containing one layout of each nondominating height vector (U, L) from among all layouts of minimum width W , for any reordering M' of each cell M of the cell array.

Proof: The proof is by induction, and the argument is essentially the same as that of Lemma 5.3, with one minor change, which we now discuss. The procedure *GenerateMinHeightLayout2* of *HRM-TrailTrace* differs from *GenerateMinHeightLayout* of *HR-TrailTrace* only in the inclusion of statement 1 in Fig. 5.5. This procedure selects each minimum-sized cover tc in $PMCTC(M')$ and generates systematically all layouts for each tc , each such layout being of minimum width. The trails of tc are permuted and oriented, the dual closed trails are broken in each place possible, and the trail edges of the isomorphic subgraphs of M' of tc are permuted systematically. This permutation of isomorphic subgraph edges ensures that each layout, whose height can be affected by the permutation of its metal rows

in the U-region that are connected to polysilicon columns, is generated and its height is measured. Figure 6.6 illustrates cell height reduction by the permutation of subgraphs b and c of the graph model M_2 in Fig. 6.3(b) of circuit C_2 from Fig. 6.3(a). $\square$

Lemma 6.2: The procedure `Find_optimal_cell_array(HV, smallest_H)` in Fig. 6.2 computes a layout of minimum height H for the cell array, given a set of nondominated height vectors for each cell of a cell array, and a placement of the cells.

Proof: A height vector (U, L) of a layout of a cell contains the heights of each of its independent routing regions. The procedure `Find_optimal_cell_array` selects the layouts of the cells that result in the cell array of minimum height. This procedure begins with cell 1. At each step, it applies the lub operation pairwise to the nondominating vectors of the cell subarray composed of cell 1 through cell $i-1$ and the nondominating vectors of cell i . The lemma is proved by induction. For the one-cell case, the procedure selects the minimum-height vector from the set HV of height vectors for that cell. This vector is surely an element of HV_1 according to Theorem 5.1. Assume that `Find_optimal_cell_array` correctly computes the case of $n-1$ cells. We now prove that it does so correctly for n cells. The procedure applies the least upper bound operation pairwise, pairing each vector in MHV_{n-1} , the set of nondominated height vectors for a cell array of the leftmost $n-1$ cells, with each vector of cell n in HV_n .

The least upper bound is used by `Find_optimal_cell_array` because it corresponds to expanding the smaller of the corresponding routing regions to align it with the larger, so that their tops and bottoms coincide. The resulting vector corresponds to the cell array that has routing regions, each of which is tall enough to accommodate all the rows of horizontal wires of any of its constituent cells, because the lub operation applied to the vectors of neighboring cells is an *upper* bound. Figure 6.1 illustrates this alignment. For instance, the L-region of cell 1 has the two rows 8 and 9, whereas the L-region of cell 2 only needs to be one row high, since row 8 is empty. But when these cells are placed side-by-side in a cell array, the L-region of cell 2 is expanded so that the tops and bottoms of the L-regions of both cells coincide. This proves that the routing regions defined by the height vector are large enough for the cell interconnections. It is also true that each of the routing regions is no larger than is necessary to accommodate all the routing rows of each of the cells in the array. This follows from the fact that the lub operation that determines the size of the routing regions of the cell array is a *least* upper bound. Therefore, the cell array is as tall as, but no taller than, required by the nondominating vectors of each of the cells in the cell array.

Let the k th height vector of cell n be $(U_{n,k}, L_{n,k})$. The vectors of MHV_{n-1} represent the nondominated vectors of the cell subarray composed of the leftmost $n-1$ cells. Find_optimal_cell_array applies the least upper bound operation to every vector of MHV_{n-1} paired with every vector of cell n as follows.

$$\text{lub}\{U_{MHV_i}, L_{MHV_i}, (U_{n,k}, L_{n,k})\} = (\max(U_{MHV_i}, U_{n,k}), \max(L_{MHV_i}, L_{n,k}))$$

Each resulting vector is placed in MHV_temp if no vector in MHV_temp dominates or equals it. Upon completion of the procedure, MHV_temp contains all nondominated height vectors for the array of n cells. This proves that Find_optimal_cell_array computes all nondominating layouts of the cell array. The height vector of minimal height can be selected from MHV_temp according to Theorem 5.1. This inductive proof applies to a cell array of any size. Therefore, the minimum-height argument applies to the entire array. $\square$

Lemma 6.3: Given a set of nondominated height vectors for each cell of a cell array, and a placement for the cells, the *HRM-TrailTrace* algorithm selects a layout for each cell that yields a cell-array layout of minimum height for all layouts of minimum width.

Proof: The proof is again by induction. Assume that the algorithm calculates a $PMCTC(M')$ for each cell M according to Lemma 5.1. The algorithm works correctly for a cell array consisting of one cell according to Theorem 5.1.

Consider an array of several cells. Assume that the algorithm has combined the leftmost m cells such that the MHV containing the nondominated height vectors for all minimal width layouts for those m cells has been found. We now show that the algorithm combines the layouts for cell $m+1$ such that the MHV contains the nondominated height vectors for all layouts of minimal width W or of width $W + 1$ for those $m+1$ cells.

A cell layout may have a pair of contacts between a transistor source diffusion region and the V_{DD} and GND lines along the cell side. The left side of cell 1 in Fig. 6.8 has contacts connecting metal to diffusion in rows 1 and 7. Such a pair of contacts along the side of a cell layout is called a *V_{DD}/GND port*, as shown in Fig. 6.9(a). A port can be a *left* or *right* port depending on which side of the cell the corresponding pair of contacts appears. This pair of contacts is not to be confused with the V_{DD} and GND boundary signals, shown in Fig. 6.4, that every cell has on both its right and left sides.

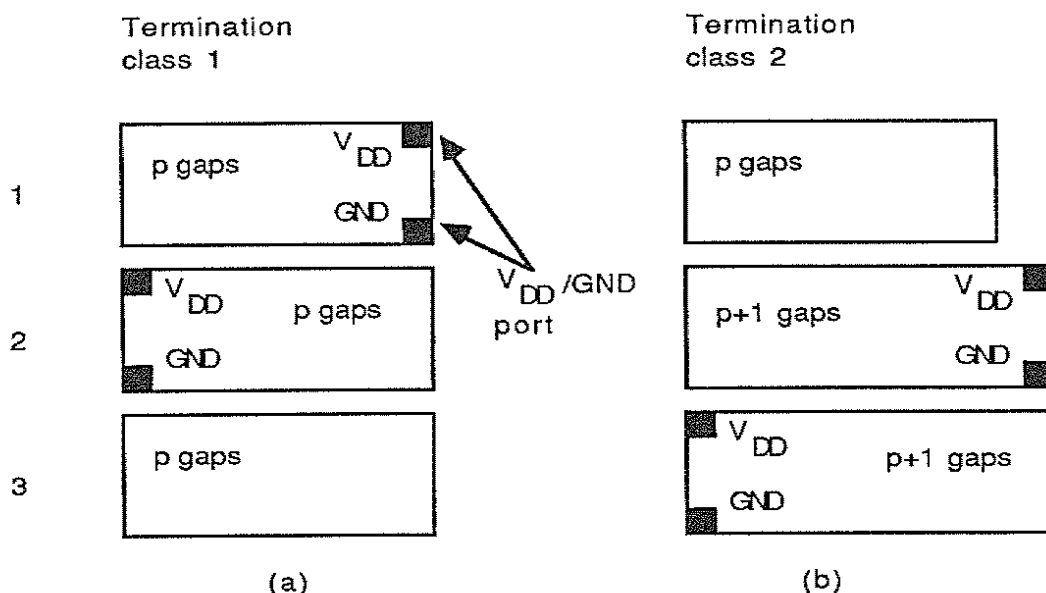


Fig. 6.9. The two basic termination classes: (a) class 1 and (b) class 2.

Two *termination classes* are based on the width of a circuit's narrowest layout with a V_{DD}/GND port. A circuit is of *termination class 1* if at least one of its possible layouts with a V_{DD}/GND port is of minimum width w_{min} . An HV of a circuit of termination class 1 has vectors of layouts only of width w_{min} . Termination class 1 is illustrated in Fig. 6.9(a), where all layouts in the HV are the same width, or alternatively, the layouts have the same number of diffusion gaps, since, for a given circuit, its layout width is directly related to the number of diffusion gaps. A circuit is of *termination class 2* if none of its layouts with a V_{DD}/GND port is of minimum width w_{min} . A circuit of termination class 2 has an HV in which no layout with a V_{DD}/GND port has minimum width. This circuit's HV contains height vectors corresponding to layouts of two widths, the minimal width w_{min} , and the near-minimum width $w_{min} + 1$. These cases correspond to layouts with p gaps or $p + 1$ gaps, respectively. Layouts of both widths for a circuit of termination class 2 must be examined in constructing a cell array to ensure that the array has minimum height for *all* those of minimum width; see Fig. 6.5. This is because some minimum-width cell arrays can result from cells of width $w_{min} + 1$ due to diffusion abutment occurring between layouts of neighboring cells, as illustrated in Fig. 6.5. The 2-cell array in Fig. 6.5(b) is formed by cell 1 and 2 of Fig. 6.5(a), both minimum-width cells, whereas the array of Fig. 6.5(d), which is the same width as the one in Fig. 6.5(b), is formed from cell 1 and cell 2 of Fig. 6.5(c);

the latter cell is of width $w_{min} + 1$. Termination class 2 is illustrated by Fig. 6.9(b), where layouts are included in the HV having p gaps and no V_{DD}/GND port, and layouts with $p + 1$ gaps and a V_{DD}/GND port.

Case	Termination class of cell array	Termination class of right cell
1.	1	1
2.	1	2
3.	2	1
4.	2	2

Fig. 6.10. The four cases of combining cells in Lemma 6.3.

Since the cell subarray formed by the leftmost m cells and cell $m+1$ can be of either termination class, we must consider each of the four cases listed in Fig. 6.10. Figure 6.11 shows in detail how a cell combines with a cell on its right, illustrating the four cases of Fig. 6.10. The left and right cells have a minimum of p and q gaps, respectively. Figure 6.11 gives the cell width in terms of gaps and port type of the two cells, along with the resulting width and port of the combined cell shown in the righthand column of the same figure. The proof of the lemma is by exhaustive enumeration of each relevant combination of cells.

Case 1: The right and left cells are both of termination class 1. Figure 6.11(a) shows the two subcases resulting in a cell array of $p+q$ or $p+q+1$ gaps. We describe the $(p+q)$ -gap subcase. The left cell of class 1 having p gaps and a right V_{DD}/GND port is combined with a right cell of class 1, whose V_{DD}/GND port is on its left side. The resulting 2-cell array has $p+q$ gaps, with no new gaps created by this combination because the V_{DD}/GND ports of the two cells align to allow the cells to connect by diffusion abutment. In cases where the alignment of ports does not allow abutment, a new diffusion gap is created, increasing the width by one gap of the resulting array, as in the second subcase in Fig. 6.11.

Case 2: The right cell is of termination class 1 and left cell is of class 2. The three subcases resulting in cell arrays with $p+q+1$ or $p+q+2$ gaps are illustrated in Figure 6.11(b).

Case 3: The right cell is of termination class 2 and left cell is of class 1. The three possible combinations of these classes is shown in Fig. 6.11(c), and result only in

$p+q+1$ gaps. Note that only the resulting arrays of this case are of termination class 1, since some of the minimum-width layouts have a V_{DD}/GND port.

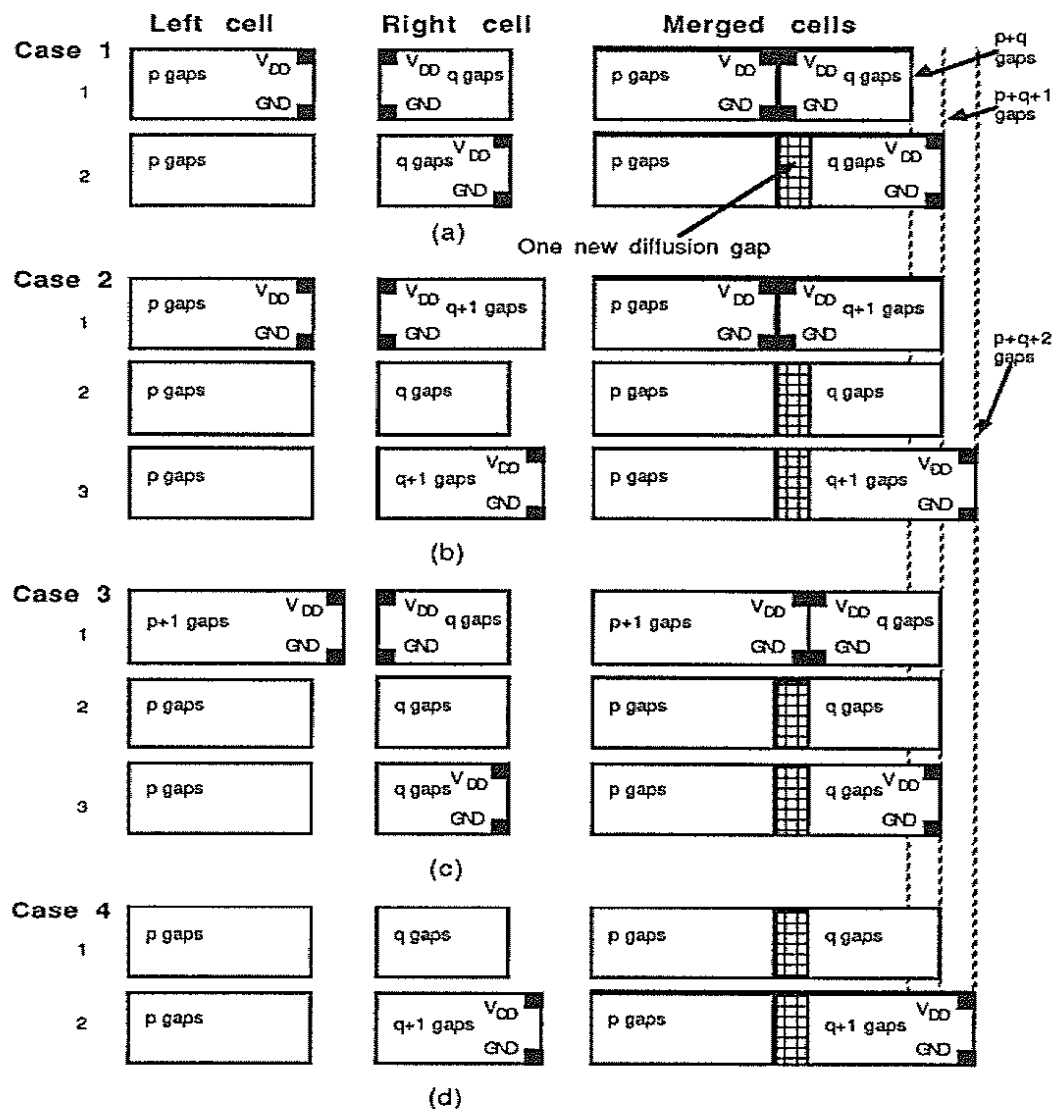


Fig. 6.11. An illustration of how two cells combine to form minimum-width cell arrays.

Case 4: Both the right and left cells are of termination class 2. Figure 6.11(d) shows the two subcases resulting in arrays having $p+q+1$ and $p+q+2$ gaps, respectively.

The above four cases include all possible layouts of minimum width and minimum width plus one for the cell array of $m+1$ cells. The height vectors of the cell array of m cells and those of cell $m+1$ are combined according to Lemma 6.2, resulting in an MHV with all nondominated height vectors for each port type and array width for the cell array of $m+1$ cells. When all cells in the cell array have been combined, a layout of minimum height is selected from all those of minimum width, as discussed in the proof of Theorem 5.1. $\square$

Theorem 6.1: The algorithm *HRM-TrailTrace* solves the WHRM layout minimization problem exactly for dual series-parallel circuits.

Proof: Lemma 5.1 proves that function *Node_CTC* computes the $PMCTC(M')$ for any reordering M' of M . Lemma 6.1 proves that *HRM-TrailTrace* generates the set HV containing one layout of each nondominating height vector (U, L) from among all layouts of minimum width W for any reordering M' of each cell M , including the permuting of isomorphic subgraphs of M . Lemma 6.2 demonstrates that the procedure *Find_optimal_cell_array* combines the height vectors of each cell with those of the other cells to compute the nondominated height vectors of the entire cell array. The minimum-height cell-array layout corresponds to one of these height vectors as shown in the proof of Theorem 5.1. Lemma 6.3 proves that from the HV_i of each cell M_i , *HRM-TrailTrace* computes a minimum-height layout from among all layouts of minimum width. Therefore, *HRM-TrailTrace* solves the WHRM layout minimization problem exactly. $\square$

6.2.4 Time Complexity

In this section, we analyze the time complexity of *HRM-TrailTrace*. First, we analyze the complexity as a function both of the height of the circuit graph and of the number of cells in the array. We also analyze the complexity as a function of the number of cells alone.

The WHRM layout problem is at least NP-hard [Ch91]. Figure 6.12 briefly outlines the *HRM-TrailTrace* algorithm along with the worst-case time complexity of its main procedures. It takes time $O(nE)$ to enter the n circuit graphs representing an n -cell array. It requires time $O(nE)$ to decompose the problem by calculating which signals cross the left and right cell boundaries. The for loop in the main procedure of *HRM-TrailTrace* executes *Generate_cell_layouts* once for each cell. This procedure is essentially the *HR-TrailTrace* procedure analyzed in Chapter 5, with the

modification noted above. The time complexity of this **for** loop is $O(n! \text{Min_covers}^{4^{E/4} 2^E E^{E+2} (h!)^{h+1}}) \approx O(nh^{4h^2})$. Finally, the height vectors of all the cells are combined to find the optimal layout of the cell array. The time complexity of this procedure is $O(nE^2)$. The time complexity of the entire *HRM-TrailTrace* algorithm is the complexity of the above-mentioned **for** loop, which is $O(nh^{4h^2})$. An analysis of the *Generate_cell_layouts* procedure is found in Chapter 5.

<i>HRM-TrailTrace</i>	Complexity
procedure <i>HRM-TrailTrace</i> ;	
Input_cells;	$O(nE)$
Calculate_boundary_signals;	$O(nE)$
for each of n cells do	$O(n! \text{Min_covers}^{4^{E/4} 2^E E^{E+2} (h!)^{h+1}}) \approx O(nh^{4h^2})$
Generate_cell_layouts(n , HV);	
Find_optimal_cell_array;	$O(nE^2)$
end; (<i>HRM-TrailTrace</i>)	

Fig. 6.12. Worst-case time complexities of components of the *HRM-TrailTrace* algorithm.

Since the height h of a functional cell is bounded by four, the time complexity of the algorithm, when restricted to the domain of practical circuits, becomes independent of h , and therefore of E . Hence, the algorithm's time complexity $O(nh^{4h^2})$ reduces to $O(n)$. Consequently, *optimal* layouts of large cell arrays consisting of an arbitrary number of functional cells can be generated in *linear* time.

6.3 EXPERIMENTAL RESULTS

We have implemented *HRM-TrailTrace* in a 9000-line Pascal program. Now we present the results of running this program on several commercial circuits, and various circuits found in the layout literature.

6.3.1 Commercial Circuits

We have generated layouts in the style of Fig. 6.1 for some commercial CMOS circuits found in IC manufacturers' data books [Si86]. The results are summarized in Fig. 6.13. This list includes a representative sample of the circuits for which logic

circuits are provided and that do not contain transmission gates. The transistor circuits used by *HRM-TrailTrace* are approximated by mapping logic gates into standard transistor realizations. The actual layouts for these circuits are not available from the manufacturers; the main reason for generating their layouts is to demonstrate that the execution time of *HRM-TrailTrace* is reasonable for medium to large-sized commercial circuits. Note that the circuits of Fig. 6.1 have up to 44 functional cells and up to 190 transistors. The execution time of *HRM-TrailTrace* depends largely on the particular cells that constitute an array. The same dependency is observed in *HR-TrailTrace*. As shown in Chapter 5, *HR-TrailTrace* generates layouts for 98% of all practical-sized cells in less than 20 minutes on a MicroVAX II, but a few require more time. Typically, *HRM-TrailTrace* processes these cells in roughly the same amount of time as *HR-TrailTrace*. Ten of the twelve circuits took 15 minutes or less, with the other two requiring a few hours. This range of times on a small computer is well within the scope of feasible computation effort specified by our hypothesis in Chapter 1. As noted already, relatively inexpensive workstations are presently available that are roughly two orders of magnitude faster than the MicroVAX II; on such computers, even the worst-case run times of this algorithm should be completely satisfactory for practical applications.

IC part number	Function	Number of functional cells	Number of transistors	Execution time
n/a	Gated SR latch	2	16	8 sec.
n/a	Full adder	2	26	1 min.
n/a	2-bit comparator	5	40	30 sec.
7476	JK flip-flop	4	40	1 min.
74183	Carry-save full adder	5	42	2 min.
7482	2-bit full adder	7	66	10 min.
74182	Carry look-ahead generator	7	88	78 min.
74148	8-to-3 priority encoder	22	114	150 min.
74248	BCD-to-seven segment decoder	15	138	11 min.
74181	4-bit ALU	44	142	15 min.
74280	9-bit parity generator	17	144	2 min.
74259	8-bit addressable latch	21	190	2 min.

Fig. 6.13. Execution times of *HRM-TrailTrace* on some commercial circuits.

6.3.2 Published Layouts

We have collected several layout examples published in the literature on one-dimensional cell arrays. From these, we have selected arrays of dual series-parallel functional cells that do not contain transmission gates or dynamic logic. Most have a layout style similar to ours, which facilitates the comparison. The cell-array area is computed by counting the number of polysilicon columns and diffusion gaps to determine cell width, and the number of metal rows to determine height. This allows us to compare quite directly the size of the various layouts independent of the specific IC technology used. The results of the comparison are given in Fig. 6.14.

Reference	Figure	Function	Published layout size		<i>HRM-TT</i> layout size		% area improvement	Run time (s)
			Height in rows (<i>U,C,L</i>)	Width in columns	Height in rows (<i>U,C,L</i>)	Width in columns		
[BA88]	7-7	XOR	(3,1,8)	6	(5,1,2)	5	80	2
[BA88]	8-5	Full adder	(6,1,6)	18	(7,1,3)	14	52	18
[KW85]	13	MSFF	(6,4,3)	16	(6,1,3)	15	39	22
[Hi85]	3	2-bit adder	(3,7,3)	38	(8,1,3)	31	33	44
[Ma85]	14	Random	(5,1,3)	16	(4,1,3)	14	29	12
[Uv81]	2c	XOR	(4,1,2)	6	(5,1,2)	5	5	2

Fig. 6.14. Comparison of *HRM-TrailTrace*'s layouts to published cell-array layouts.

The area differences between the published layouts and our optimal layouts can be quite large. For instance, the layout of the 2-bit full adder generated by the Sc2 program [Hi85] is 33% larger than that of *HRM-TrailTrace*. Another example is the master-slave D flip-flop from TOPOLOGIZER [KW85], shown in Fig. 2.7, whose layout is 39% larger than our layout in Fig. 6.15. Other differences are even greater, and highlight the importance of evaluating heuristics by using optimal layouts as standards of comparison. Without the performance guarantees that approximation algorithms offer, unbounded heuristics can produce results arbitrarily far from the optimum [HS78]. Even comparing heuristics to manual layouts can give false confidence that near-optimality is achieved; the flip-flop layout of TOPOLOGIZER

is only 3% larger than the manual design to which it is compared in [KW85], yet it is 39% larger than our optimal layout. These results underline the value of conducting a broad search over a slightly constrained space, compared to searching heuristically over a less constrained space. Often very general layout tools cannot evaluate many different transistor placements because the associated routing problem is too complicated. We can examine a larger number of placements with *HRM-TrailTrace* because we can evaluate the associated simple routing problem quickly.

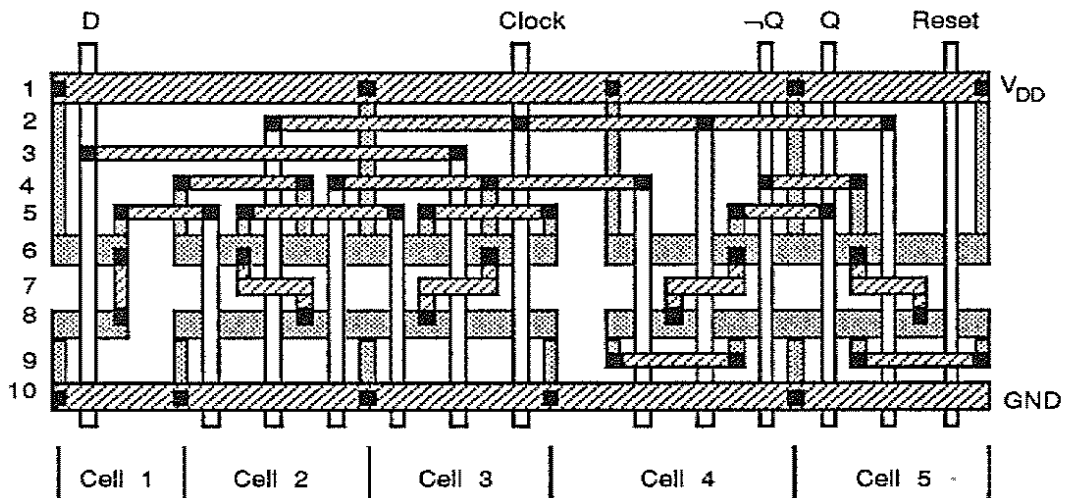


Fig. 6.15. A cell-array layout by *HRM-TrailTrace* for the master-slave D flip-flop of [KW85].